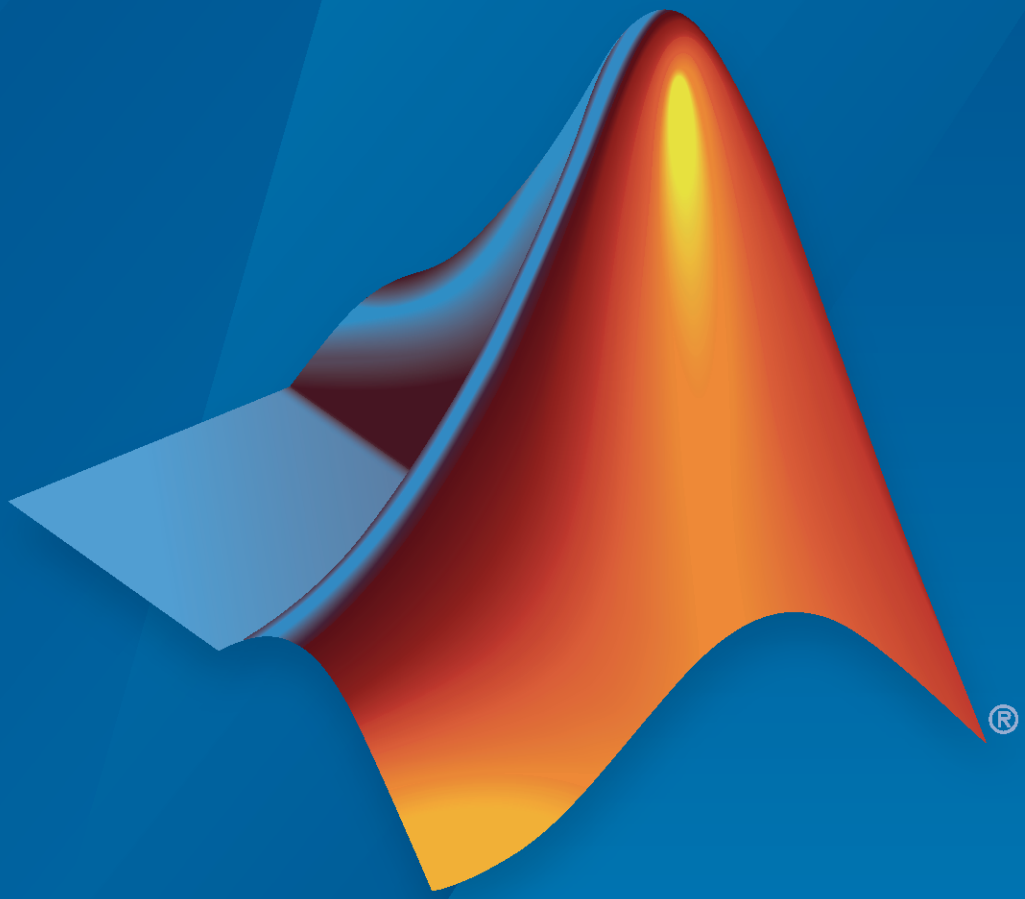
MATLAB® Production Server™

Server Management Guide



MATLAB®

R2023a

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

Sales and services: `www.mathworks.com/sales_and_services`

User community: `www.mathworks.com/matlabcentral`

Technical support: `www.mathworks.com/support/contact_us`

Phone: 508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

MATLAB® Production Server™ Management Guide

© COPYRIGHT 2012–2023 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See `www.mathworks.com/trademarks` for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

MathWorks products are protected by one or more U.S. patents. Please see `www.mathworks.com/patents` for more information.

Revision History

March 2014	Online only	New for Version 1.2 (Release 2014a)
October 2014	Online only	Revised for Version 2.0 (Release 2014b)
March 2015	Online only	Revised for Version 2.1 (Release 2015a)
September 2015	Online only	Revised for Version 2.2 (Release 2015b)
March 2016	Online only	Revised for Version 2.3 (Release 2016a)
September 2016	Online only	Revised for Version 2.4 (Release 2016b)
March 2017	Online only	Revised for Version 3.0 (Release 2017a)
September 2017	Online only	Revised for Version 3.0.1 (Release R2017b)
March 2018	Online only	Revised for Version 3.1 (Release R2018a)
September 2018	Online only	Revised for Version 4.0 (Release R2018b)
March 2019	Online only	Revised for Version 4.1 (Release R2019a)
September 2019	Online only	Revised for Version 4.2 (Release R2019b)
March 2020	Online only	Revised for Version 4.3 (Release R2020a)
September 2020	Online only	Revised for Version 4.4 (Release R2020b)
March 2021	Online only	Revised for Version 4.5 (Release R2021a)
September 2021	Online only	Revised for Version 4.6 (Release R2021b)
March 2022	Online only	Revised for Version 5.0 (Release R2022a)
September 2022	Online only	Revised for Version 5.1 (Release R2022b)
March 2023	Online only	Revised for Version 5.2 (Release R2023a)

Contents

1 Server Management

Server Overview **1-2**
What Is a Server Instance? **1-2**
How Does a Server Manage Work? **1-2**

Create Server Instance Using Command Line **1-4**
Prerequisites **1-4**
Procedure **1-4**

Configure Server Using Configuration File **1-6**
Edit Server Configuration File **1-6**
Common Customizations **1-6**

Install and Configure MATLAB Runtime **1-8**
Download MATLAB Runtime Installer **1-8**
Install MATLAB Runtime Interactively **1-8**
Default Install Folder **1-9**
Install MATLAB Runtime Noninteractively **1-10**
Install MATLAB Runtime without Administrator Rights **1-11**
Install Multiple MATLAB Runtime Versions on Single Machine **1-12**
MATLAB and MATLAB Runtime on Same Machine **1-12**
Uninstall MATLAB Runtime **1-12**

Specify Default MATLAB Runtime for New Server Instances **1-14**
Run mps-setup in Non-Interactive Mode for Silent Install **1-14**

Specify MATLAB Runtime for Server Instance Using Command Line . . **1-15**

Start Server Instance Using Command Line **1-16**
Prerequisites **1-16**
Procedure **1-16**

Support Multiple MATLAB Runtime Versions **1-17**
Configure Server to Use Multiple MATLAB Runtime Instances **1-17**
How Server Instances Select Which MATLAB Runtime to Use **1-18**
Changes to Worker Management **1-18**

Control Worker Restarts **1-20**
Restart Workers Based on Up Time **1-20**
Restart Workers Based on Amount of Memory in Use **1-20**

Configure Server Instance as Windows Service **1-22**
Create New Server Instance as Windows Service **1-22**
Make Existing Server Instance a Windows Service **1-22**
Recovery Options for Server Instance Running as Windows Service **1-22**

Manage Instances Using Dashboard 1-23
Work With Network Drives 1-23

Create Server Instance Using Dashboard 1-24
Set MATLAB Runtime Location 1-24

Set MATLAB Runtime Location Using Dashboard 1-26

Specify Server Instance License Options Using Dashboard 1-27

Start Server Instance Using Dashboard 1-28
Prerequisites 1-28
Start Server Instance From Server Information Page 1-28
Start Server Instance From Server Instance Page 1-28

2 Persistence

Data Caching Basics 2-2
Typical Workflow for Data Caching 2-2
Configure Server to Use Redis 2-2
Example: Increment Counter Using Data Cache 2-3

Manage Application State in Deployed Archives 2-5
Step 1: Write MATLAB Code that uses Persistence Functions 2-5
Step 2: Run Example in Testing Workflow 2-9
Step 3: Run Example in Deployment Workflow 2-10

Handle Custom Routes and Payloads in HTTP Requests 2-14
Write MATLAB Function for Web Request Handler 2-14
Configure Server for URL Routes 2-15
End-to-End Setup for Web Request Handler 2-16

3 Secure a Server

Enable HTTPS 3-2
Acquire and Copy SSL Certificate and Key 3-2
Edit Configuration File 3-2

Configure Client Authentication 3-4

Specify Access to MATLAB Programs 3-5

Adjust Security Protocols 3-6

Improve Startup Time When Security Is Activated 3-7

Application Access Control **3-8**
Access Control Configuration File **3-8**
Access Control Policy File **3-10**

Use Kerberos Authentication and Kerberos Delegation **3-14**
Kerberos Authentication and Kerberos Delegation Setup on Windows Systems **3-14**
Kerberos Setup on Linux Systems **3-15**

4 Troubleshooting

Verify Server Status **4-2**
Procedure **4-2**
License Server Status Information **4-3**

Diagnose a Server Instance **4-4**

Diagnose a Corrupted MATLAB Runtime **4-5**

Server Diagnostic Tools **4-6**
Log Files **4-6**
Process Identification Files (PID Files) **4-6**
Endpoint Files **4-6**

Manage Log Files **4-7**
Best Practices for Log Management **4-7**
Log Retention and Archive Settings **4-7**
Log Detail Levels **4-8**

Common Error Messages and Resolutions **4-9**
(404) Not Found **4-9**
Error: Bad MATLAB Runtime Instance **4-9**
Error: Server Instance not Specified **4-9**
Error: invalid target host or port **4-9**
Error: HTTP error: HTTP/x.x 404 Component not found **4-9**
Server failed to start (error code = 16): license checkout failed **4-9**

5 Impact of Server Configurations on Processing Asynchronous Requests

Impact of Server Configurations on Processing Asynchronous Requests **5-2**

6 Set Up MATLAB Production Server Dashboard

Set Up and Log In to MATLAB Production Server Dashboard **6-2**
Set Up Dashboard **6-2**
Start Dashboard **6-3**
Log In to Dashboard **6-4**
Reset Administrator Password **6-4**

Remove MATLAB Production Server Dashboard **6-6**

7 Commands

8 Configuration Properties

9 Cloud Deployment

Azure Deployment for MATLAB Production Server (BYOL) **9-2**
Provision Cloud Resources **9-2**
Upload License File **9-7**
Connect and Log In to Dashboard **9-8**

Azure Deployment for MATLAB Production Server (PAYG) **9-9**
Provision Cloud Resources **9-9**
Connect and Log In to Dashboard **9-14**

Manage MATLAB Production Server (BYOL) **9-15**
Connect to Dashboard **9-15**
Log In to Dashboard **9-15**
View Information About Server **9-16**
Upload MATLAB Application **9-16**
View HTTPS Server Endpoint **9-17**
Edit Server Configuration **9-17**
Use the Azure Cache for Redis for Data Persistence **9-17**
Set Up Access Control for Applications Using OAuth 2.0 Providers **9-18**
Set Up Access Control for Dashboard Using OAuth 2.0 Providers **9-19**

Manage MATLAB Production Server (PAYG) **9-20**
Connect to Dashboard **9-20**
Log In to Dashboard **9-20**
View Information About Server **9-21**
Upload MATLAB Application **9-21**

View MATLAB Execution Endpoint 9-22
Edit Server Configuration 9-22
Use the Azure Cache for Redis for Data Persistence 9-22
Set Up Access Control for Applications Using OAuth 2.0 Providers 9-23
Set Up Access Control for Dashboard Using OAuth 2.0 Providers 9-24

Manage MATLAB Production Server Using the Dashboard 9-25
Connect to Dashboard 9-25
Log In to Dashboard 9-25
View Information About Server 9-26
Upload MATLAB Application 9-27
View MATLAB Execution Endpoint 9-27
Edit Server Configuration 9-27
Use the Azure Cache for Redis for Data Persistence 9-28
Set Up Access Control for Applications Using OAuth 2.0 Providers 9-29
Set Up Access Control for Dashboard Using OAuth 2.0 Providers 9-29

Manage Azure Resources for MATLAB Production Server (BYOL) 9-30
Change the Number of Virtual Machines 9-30
Change SSL Certificate to Application Gateway 9-30
Upload MATLAB Applications 9-30
View Logs 9-30
Delete Resource Group 9-31

Manage Azure Resources for MATLAB Production Server (PAYG) 9-32
Change the Number of Virtual Machines 9-32
Change SSL Certificate to Application Gateway 9-32
View Logs 9-32
Upload MATLAB Applications 9-33
Delete Resource Group 9-33

Manage Azure Resources for MATLAB Production Server Reference Architecture 9-34
Change the Number of Virtual Machines 9-34
Change SSL Certificate to Application Gateway 9-34
View Logs 9-34
Upload MATLAB Applications 9-35
Delete Resource Group 9-35

Architecture and Resources on Azure 9-36
MATLAB Production Server (BYOL) Architecture on Azure 9-36
Azure Resources 9-36

Architecture and Resources on Azure 9-39
MATLAB Production Server (PAYG) Architecture on Azure 9-39
Azure Resources 9-39

Architecture and Resources on Azure 9-42
MATLAB Production Server Reference Architecture on Azure 9-42
Azure Resources 9-42

Application Access Control 9-45
Configure Identity Provider and Specify Access Control Policy Rules 9-45
Enable Application Access Control 9-45
Generate Access Token 9-46

Application Access Control **9-47**
Configure Identity Provider and Specify Access Control Policy Rules **9-47**
Enable Application Access Control **9-47**
Generate Access Token **9-48**

Application Access Control **9-49**
Configure Identity Provider and Specify Access Control Policy Rules **9-49**
Enable Application Access Control **9-49**
Generate Access Token **9-50**

Dashboard Access Control **9-51**
Dashboard User Roles **9-51**
Configure Identity Provider and Specify Access Control Policies **9-51**
Enable Access Control **9-52**

Dashboard Access Control **9-53**
Dashboard User Roles **9-53**
Configure Identity Provider and Specify Access Control Policies **9-53**
Enable Access Control **9-54**

Dashboard Access Control **9-55**
Dashboard User Roles **9-55**
Configure Identity Provider and Specify Access Control Policies **9-55**
Enable Access Control **9-56**

Execute MATLAB Functions on MATLAB Production Server (BYOL) . . . **9-57**
Use MATLAB Execution Endpoint URL **9-57**
Download Client Libraries **9-57**
Work with Self-Signed Certificate **9-57**
Manage HTTP Cookie **9-58**

Execute MATLAB Functions on MATLAB Production Server (PAYG) . . . **9-59**
Use MATLAB Execution Endpoint URL **9-59**
Download Client Libraries **9-59**
Work with Self-Signed Certificate **9-59**
Manage HTTP Cookie **9-60**

Execute MATLAB Functions on MATLAB Production Server Reference Architecture **9-61**
Use MATLAB Execution Endpoint URL **9-61**
Download Client Libraries **9-61**
Work with Self-Signed Certificate **9-61**
Manage HTTP Cookie **9-62**

Configure Dashboard Access Control Using Azure AD **9-63**
Configure Identity Provider **9-63**
Specify Dashboard Access Control Policy **9-64**
Enable Dashboard Access Control **9-64**

Configure Dashboard Access Control Using Google Identity **9-66**
Configure Google Identity **9-66**
Specify Dashboard Access Control Policy **9-67**
Enable Dashboard Access Control **9-67**

Configure Dashboard Access Control Using PingFederate Identity Provider **9-68**
Prerequisites **9-68**
Configure PingFederate Identity Provider **9-68**
Specify Dashboard Access Control Policy **9-69**
Enable Dashboard Access Control **9-69**

Configure Dashboard Access Control Using Other OpenID Connect Providers **9-71**
Configure Identity Provider **9-71**
Specify Dashboard Access Control Policy **9-72**
Enable Dashboard Access Control **9-73**

Configure Dashboard Access Control Using Azure AD **9-75**
Configure Identity Provider **9-75**
Specify Dashboard Access Control Policy **9-76**
Enable Dashboard Access Control **9-76**

Configure Dashboard Access Control Using Google Identity **9-78**
Configure Google Identity **9-78**
Specify Dashboard Access Control Policy **9-79**
Enable Dashboard Access Control **9-79**

Configure Dashboard Access Control Using PingFederate Identity Provider **9-80**
Prerequisites **9-80**
Configure PingFederate Identity Provider **9-80**
Specify Dashboard Access Control Policy **9-81**
Enable Dashboard Access Control **9-81**

Configure Dashboard Access Control Using Other OpenID Connect Providers **9-83**
Configure Identity Provider **9-83**
Specify Dashboard Access Control Policy **9-84**
Enable Dashboard Access Control **9-85**

Configure Dashboard Access Control Using Azure AD **9-87**
Configure Identity Provider **9-87**
Specify Dashboard Access Control Policy **9-88**
Enable Dashboard Access Control **9-88**

Configure Dashboard Access Control Using Google Identity **9-90**
Configure Google Identity **9-90**
Specify Dashboard Access Control Policy **9-91**
Enable Dashboard Access Control **9-91**

Configure Dashboard Access Control Using PingFederate Identity Provider **9-92**
Prerequisites **9-92**
Configure PingFederate Identity Provider **9-92**
Specify Dashboard Access Control Policy **9-93**
Enable Dashboard Access Control **9-93**

Configure Dashboard Access Control Using Other OpenID Connect Providers **9-95**
Configure Identity Provider **9-95**
Specify Dashboard Access Control Policy **9-96**
Enable Dashboard Access Control **9-97**

Configure Application Access Control Using Azure AD **9-99**
Register Application in Azure Portal **9-99**
Configure Identity Provider **9-101**
Specify Access Control Policy Rules **9-101**
Enable Application Access Control **9-101**
Generate Access Token **9-102**

Configure Application Access Control Using Google Identity **9-103**
Register Application in Google Cloud Platform Console **9-103**
Configure Identity Provider in Dashboard **9-103**
Specify Access Control Policy Rules **9-103**
Enable Application Access Control **9-104**
Generate Access Token **9-104**

Configure Application Access Control Using PingFederate **9-105**
Prerequisites **9-105**
Configure PingFederate in Dashboard **9-105**
Specify Access Control Policy Rules **9-105**
Enable Application Access Control **9-106**
Generate Access Token **9-106**

Configure Application Access Control Using Other Identity Providers **9-107**
Register Application With Identity Provider **9-107**
Configure Identity Provider in Dashboard **9-107**
Specify Access Control Policy Rules **9-107**
Enable Application Access Control **9-108**
Generate Access Token **9-108**

Configure Application Access Control Using Azure AD **9-109**
Register Application in Azure Portal **9-109**
Configure Identity Provider **9-111**
Specify Access Control Policy Rules **9-111**
Enable Application Access Control **9-111**
Generate Access Token **9-112**

Configure Application Access Control Using Google Identity **9-113**
Register Application in Google Cloud Platform Console **9-113**
Configure Identity Provider in Dashboard **9-113**
Specify Access Control Policy Rules **9-113**
Enable Application Access Control **9-114**
Generate Access Token **9-114**

Configure Application Access Control Using PingFederate **9-115**
Prerequisites **9-115**
Configure PingFederate in Dashboard **9-115**
Specify Access Control Policy Rules **9-115**
Enable Application Access Control **9-116**
Generate Access Token **9-116**

Configure Application Access Control Using Other Identity Providers **9-117**
Register Application With Identity Provider **9-117**
Configure Identity Provider in Dashboard **9-117**
Specify Access Control Policy Rules **9-117**
Enable Application Access Control **9-118**
Generate Access Token **9-118**

Configure Application Access Control Using Azure AD **9-119**
Register Application in Azure Portal **9-119**
Configure Identity Provider **9-121**
Specify Access Control Policy Rules **9-121**
Enable Application Access Control **9-121**
Generate Access Token **9-122**

Configure Application Access Control Using Google Identity **9-123**
Register Application in Google Cloud Platform Console **9-123**
Configure Identity Provider in Dashboard **9-123**
Specify Access Control Policy Rules **9-123**
Enable Application Access Control **9-124**
Generate Access Token **9-124**

Configure Application Access Control Using PingFederate **9-125**
Prerequisites **9-125**
Configure PingFederate in Dashboard **9-125**
Specify Access Control Policy Rules **9-125**
Enable Application Access Control **9-126**
Generate Access Token **9-126**

Configure Application Access Control Using Other Identity Providers **9-127**
Register Application With Identity Provider **9-127**
Configure Identity Provider in Dashboard **9-127**
Specify Access Control Policy Rules **9-127**
Enable Application Access Control **9-128**
Generate Access Token **9-128**

Run MATLAB Production Server on Azure Using Reference Architecture **9-129**
Requirements **9-129**
Run from GitHub **9-129**

Run MATLAB Parallel Server and MATLAB Production Server on Azure **9-130**
Deploy MATLAB Production Server **9-130**
Deploy MATLAB Parallel Server **9-130**
Upload License File **9-130**
Connect to Cluster from MATLAB **9-131**
Package MATLAB Application into Deployable Archive **9-131**

10 AWS Deployment

AWS Deployment for MATLAB Production Server (PAYG) **10-2**
Software Costs **10-2**
Prepare AWS Account **10-2**
Subscribe to MATLAB Production Server (PAYG) Software **10-3**
Deploy to Existing or New VPC **10-3**
Launch CloudFormation Console **10-3**
Create Stack **10-3**
Specify Stack Details **10-3**
Configure Stack Options **10-7**
Review **10-7**
Connect and Log In to Dashboard **10-7**

Manage MATLAB Production Server (PAYG) **10-9**
Connect to Dashboard **10-9**
Log In to Dashboard **10-9**
View Information About Server **10-10**
Upload MATLAB Application **10-11**
View MATLAB Execution Endpoint **10-11**
Edit Server Configuration **10-12**
Use Amazon ElastiCache for Redis for Data Persistence **10-12**
Set Up Access Control for Applications Using Azure Active Directory . . **10-13**
Set Up Access Control for Dashboard Using Azure Active Directory . . . **10-13**

Manage AWS Resources for MATLAB Production Server (PAYG) **10-14**
Change Number of Virtual Machines **10-14**
Import SSL Certificate **10-14**
Change SSL Certificate **10-14**
View Logs **10-15**
Handle Timeouts **10-15**
Delete Stack **10-16**

Architecture and Resources **10-17**
MATLAB Production Server (PAYG) Architecture on AWS **10-17**
AWS Resources **10-17**

Application Access Control **10-20**
Configure Identity Provider and Specify Access Control Policy Rules . . **10-20**
Enable Application Access Control **10-20**
Generate Access Token **10-21**

Configure Application Access Control Using Azure AD **10-22**
Register Application in Azure Portal **10-22**
Configure Identity Provider **10-24**
Specify Access Control Policy Rules **10-24**
Enable Application Access Control **10-24**
Generate Access Token **10-25**

Configure Application Access Control Using Google Identity **10-26**
Register Application in Google Cloud Platform Console **10-26**
Configure Identity Provider in Dashboard **10-26**
Specify Access Control Policy Rules **10-26**

Enable Application Access Control 10-27
Generate Access Token 10-27

Configure Application Access Control Using PingFederate 10-28
Prerequisites 10-28
Configure PingFederate in Dashboard 10-28
Specify Access Control Policy Rules 10-28
Enable Application Access Control 10-29
Generate Access Token 10-29

Configure Application Access Control Using Other Identity Providers 10-30
Register Application With Identity Provider 10-30
Configure Identity Provider in Dashboard 10-30
Specify Access Control Policy Rules 10-30
Enable Application Access Control 10-31
Generate Access Token 10-31

Dashboard Access Control 10-32
Dashboard User Roles 10-32
Configure Identity Provider and Specify Access Control Policies 10-32
Enable Access Control 10-33

Configure Dashboard Access Control Using Azure AD 10-34
Configure Identity Provider 10-34
Specify Dashboard Access Control Policy 10-35
Enable Dashboard Access Control 10-35

Configure Dashboard Access Control Using Google Identity 10-37
Configure Google Identity 10-37
Specify Dashboard Access Control Policy 10-38
Enable Dashboard Access Control 10-38

Configure Dashboard Access Control Using PingFederate Identity Provider 10-39
Prerequisites 10-39
Configure PingFederate Identity Provider 10-39
Specify Dashboard Access Control Policy 10-40
Enable Dashboard Access Control 10-40

Configure Dashboard Access Control Using Other OpenID Connect Providers 10-42
Configure Identity Provider 10-42
Specify Dashboard Access Control Policy 10-43
Enable Dashboard Access Control 10-44

Execute MATLAB Functions on MATLAB Production Server (PAYG) . . 10-46
Use MATLAB Execution Endpoint URL 10-46
Download Client Libraries 10-46
Work with Self-Signed SSL Certificate 10-46
Manage HTTP Cookie 10-46

Run MATLAB Production Server on Amazon Web Services Using Reference Architecture 10-48
Requirements 10-48

Run from GitHub 10-48

Manage MATLAB Production Server Using the Dashboard 10-49
Connect to Dashboard 10-49
Log In to Dashboard 10-49
View Information About Server 10-50
Upload MATLAB Application 10-51
View MATLAB Execution Endpoint 10-51
Edit Server Configuration 10-51
Use Amazon ElastiCache for Redis for Data Persistence 10-52
Set Up Access Control for Applications Using OAuth 2.0 Providers 10-52
Set Up Access Control for Dashboard Using OAuth 2.0 Providers 10-53

Manage AWS Resources for MATLAB Production Server 10-54
Change Number of Virtual Machines 10-54
Import SSL Certificate 10-54
Change SSL Certificate 10-54
Upload MATLAB Applications 10-55
View Logs 10-55
Handle Timeouts 10-56
Delete Stack 10-56

Dashboard Access Control 10-57
Dashboard User Roles 10-57
Configure Identity Provider and Specify Access Control Policies 10-57
Enable Access Control 10-58

Configure Dashboard Access Control Using Azure AD 10-59
Configure Identity Provider 10-59
Specify Dashboard Access Control Policy 10-60
Enable Dashboard Access Control 10-60

Configure Dashboard Access Control Using Google Identity 10-62
Configure Google Identity 10-62
Specify Dashboard Access Control Policy 10-63
Enable Dashboard Access Control 10-63

Configure Dashboard Access Control Using PingFederate Identity Provider 10-64
Prerequisites 10-64
Configure PingFederate Identity Provider 10-64
Specify Dashboard Access Control Policy 10-65
Enable Dashboard Access Control 10-65

Configure Dashboard Access Control Using Other OpenID Connect Providers 10-67
Configure Identity Provider 10-67
Specify Dashboard Access Control Policy 10-68
Enable Dashboard Access Control 10-69

Application Access Control 10-71
Configure Identity Provider and Specify Access Control Policy Rules . . 10-71
Enable Application Access Control 10-71
Generate Access Token 10-72

Configure Application Access Control Using Azure AD **10-73**
Register Application in Azure Portal **10-73**
Configure Identity Provider **10-75**
Specify Access Control Policy Rules **10-75**
Enable Application Access Control **10-75**
Generate Access Token **10-76**

Configure Application Access Control Using Google Identity **10-77**
Register Application in Google Cloud Platform Console **10-77**
Configure Identity Provider in Dashboard **10-77**
Specify Access Control Policy Rules **10-77**
Enable Application Access Control **10-78**
Generate Access Token **10-78**

Configure Application Access Control Using PingFederate **10-79**
Prerequisites **10-79**
Configure PingFederate in Dashboard **10-79**
Specify Access Control Policy Rules **10-79**
Enable Application Access Control **10-80**
Generate Access Token **10-80**

Configure Application Access Control Using Other Identity Providers **10-81**
Register Application With Identity Provider **10-81**
Configure Identity Provider in Dashboard **10-81**
Specify Access Control Policy Rules **10-81**
Enable Application Access Control **10-82**
Generate Access Token **10-82**

Architecture and Resources **10-83**
MATLAB Production Server Architecture on AWS **10-83**
AWS Resources **10-83**

Execute MATLAB Functions on MATLAB Production Server **10-86**
Use MATLAB Execution Endpoint URL **10-86**
Download Client Libraries **10-86**
Work with Self-Signed SSL Certificate **10-86**
Manage HTTP Cookie **10-86**

Server Management

- “Server Overview” on page 1-2
- “Create Server Instance Using Command Line” on page 1-4
- “Configure Server Using Configuration File” on page 1-6
- “Install and Configure MATLAB Runtime” on page 1-8
- “Specify Default MATLAB Runtime for New Server Instances” on page 1-14
- “Specify MATLAB Runtime for Server Instance Using Command Line” on page 1-15
- “Start Server Instance Using Command Line” on page 1-16
- “Support Multiple MATLAB Runtime Versions” on page 1-17
- “Control Worker Restarts” on page 1-20
- “Configure Server Instance as Windows Service” on page 1-22
- “Create Server Instance Using Dashboard” on page 1-24
- “Set MATLAB Runtime Location Using Dashboard” on page 1-26
- “Specify Server Instance License Options Using Dashboard” on page 1-27
- “Start Server Instance Using Dashboard” on page 1-28

Server Overview

In this section...
“What Is a Server Instance?” on page 1-2 “How Does a Server Manage Work?” on page 1-2

What Is a Server Instance?

A server instance is considered to be one unique configuration of the MATLAB Production Server product. Each configuration has its own server configuration file (`main_config`) and diagnostic files (`log`, `pid`, and `endpoint` files).

In addition, each server has its own `auto_deploy` folder, where you upload the deployable archives (CTF files) that you want the server to host for clients.

You can create any number of server instances using MATLAB Production Server software. Each server instance can host any number of deployable archives containing MATLAB code. You may find it helpful to create one server for all archives relating to a particular application. You can also create one server to host code strictly for testing, and so on.

The server also manages the MATLAB Runtime (MATLAB Compiler), which enables deployed MATLAB code to execute. The settings in `main_config` determine how each server interacts with the MATLAB Runtime to process clients requests. You can set these parameters according to your performance requirements and other variables in your IT environment.

How Does a Server Manage Work?

A server processes a transaction using these steps:

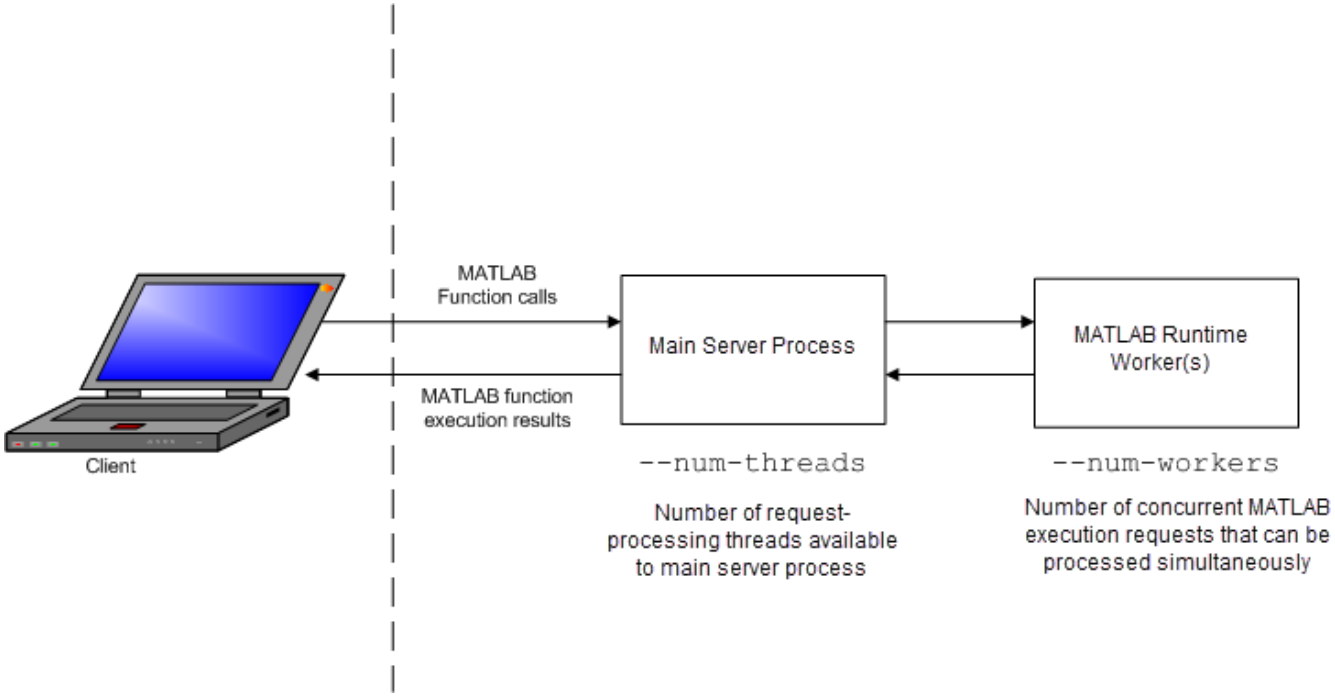
1. The client sends MATLAB function calls to the main server process.
2. The main server process passes the MATLAB function calls to one or more MATLAB Runtime workers.
3. The MATLAB Runtime workers execute the MATLAB functions deployed to the server.
4. The MATLAB Runtime workers pass the results of MATLAB function execution back to the main server process.
5. The main server process passes the results of MATLAB function execution back to the client for processing.

The server is the intermediary in the MATLAB Production Server environment. It simultaneously accepts connections from clients, and then dispatches MATLAB Runtime workers—MATLAB sessions —to process client requests to the MATLAB Runtime. By defining and adjusting the number of workers and threads available to a server, you tune capacity and throughput, respectively.

- Workers (capacity management) (num-workers) — The number of MATLAB Runtime workers available to a server.

  Each worker dispatches one MATLAB execution request to the MATLAB Runtime, interacting with one client at a time. By defining and tuning the number of workers available to a server, you set the number of concurrent MATLAB execution requests that can be processed simultaneously. `num-workers` should roughly correspond to the number of cores available on the local host.

- Threads (throughput management) (num-threads) — The number of threads (units of processing) available to the main server process.



MATLAB Production Server Data Flow from Client to Server and Back

The server does not allocate a unique thread to each client connection. Rather, when data is available on a connection, the required processing is scheduled on a pool of threads. `--num-threads` sets the size of that pool (the number of available request-processing threads) in the main server process. The threads in the pool do not execute MATLAB code directly. Instead, there is a single thread within each MATLAB Runtime worker process that executes MATLAB code on behalf of the client.

See Also

mcr-root | `mps-setup`

More About

- “Create Server Instance Using Command Line” on page 1-4
- “Support Multiple MATLAB Runtime Versions” on page 1-17

Create Server Instance Using Command Line

Before you can deploy MATLAB code with MATLAB Production Server, you must create a server instance to host your deployable archive. A server instance is one unique configuration of the MATLAB Production Server product. Each configuration has its own parameter settings file (`main_config`), as well as its own set of diagnostic files.

Prerequisites

Before creating a server instance using the command line, ensure you have:

- installed MATLAB Production Server on your machine. For more information, see “Install MATLAB Production Server Product”.
- added the `script` folder to your system `PATH` environment variable. Doing so enables you to run server commands such as `mps-new` from any folder on your system.

You can also run server commands from the *`$MPS_INSTALL`*`\script` folder, where *`$MPS_INSTALL`* is the location where MATLAB Production Server is installed. For example, on Windows, the default location is `C:\Program Files\MATLAB\MATLAB Production Server\`*`ver`*`\script`. *`ver`* is the version of MATLAB Production Server.

Procedure

To create a server instance from the command line, enter the `mps-new` command from the system prompt. Specify the name of the server that you want to create as an argument to the `mps-new` command.

```
mps-new [path\]server_name [-v]
```

- *`path`* — path to the server instance and configuration that you want to create for use with the MATLAB Production Server product.

  If you want to create a server instance in the current folder, you do not need to specify a full path; only specify the server name.
- *`server_name`* — name of the server instance and configuration that you want to create.
- `-v` — enable verbose output to get the information and status about each folder created in the server configuration.

For example, to create a server instance with the name `prod_server_1` located in `C:\tmp` and use the verbose mode, run the following on your system command prompt.

```
C:\tmp>mps-new prod_server_1 -v
```

The command generates the following output.

```
prod_server_1\.mps_version...ok
prod_server_1\config\main_config...ok
prod_server_1\.mps_socket...ok
prod_server_1\auto_deploy...ok
prod_server_1\endpoint...ok
prod_server_1\log...ok
prod_server_1\old_logs...ok
prod_server_1\pid...ok
```

```
prod_server_1\x509...ok
The UUID of the newly created instance is 4876f876-56a6-40ef-a4e3-96a69b39cb49
```

For more information on the folders created in a server configuration, see “Server Diagnostic Tools” on page 4-6.

For creating server instances on the cloud, see “Cloud Deployment”.

See Also

`mps-service` | `mps-new`

More About

- “Configure Server Instance as Windows Service” on page 1-22
- “Start Server Instance Using Command Line”

Configure Server Using Configuration File

In this section...
“Edit Server Configuration File” on page 1-6 “Common Customizations” on page 1-6

Each MATLAB Production Server instance has its own server configuration file, `main_config`. To change the server properties for an on-premises server instance managed using the command line, edit the `main_config` file that corresponds to your specific server instance. The server configuration file is located at *`server_instance_name`*`/config`. You must restart the server instance for the changes to take effect.

For on-premises server instances created using the dashboard interface and server deployments on the cloud, use the dashboard, and cloud dashboard, respectively, to edit the server properties.

Edit Server Configuration File

When editing the `main_config` server configuration file, remember these coding considerations:

- Each server instance has its own server configuration file.
- Enter only one configuration property and its options per line. Each configuration property entry starts with two dashes (`--`).
- The server ignores any line beginning with a pound sign (`#`) and considers it as a comment.
- The server ignores lines of white space.

Common Customizations

Set Default Port Number for Client Requests

Use the `http` property to set the default port number on which the server listens for client requests.

Set Number of Available Workers

Use the `num-workers` property to set the number of concurrent MATLAB execution requests that can be processed simultaneously.

Set Number of Available Threads

Use the `num-threads` property to set the number of request-processing threads available to the main server process.

Note For .NET Clients, the HTTP 1.1 protocol restricts the maximum number of concurrent connections between a client and a server to two.

This restriction only applies when the client and server are connected remotely. A local client/server connection has no such restriction.

To specify a higher number of connections than two for remote connection, use the .NET classes `System.Net.ServicePoint` and `System.Net.ServicePointManager` to modify maximum concurrent connections.

For example, to specify four concurrent connections, use the following code:

```
ServicePointManager.DefaultConnectionLimit = 4;
MWClient client = new MWHttpClient(new MyConfig());
MPSClient mpsExample = client.CreateProxy(
            new Uri("http://user01:9910/mpsexample"));
```

See Also

https

More About

- “Create Server Instance Using Command Line” on page 1-4
- “Control Worker Restarts” on page 1-20

Install and Configure MATLAB Runtime

Supported Platforms: Windows®, Linux®, macOS

MATLAB Runtime contains the libraries needed to run compiled MATLAB applications on a target system without a licensed copy of MATLAB.

Download MATLAB Runtime Installer

Download the MATLAB Runtime installer using one of the following options:

- Download the MATLAB Runtime installer at the latest update level for the selected release from the website at https://www.mathworks.com/products/compiler/matlab-runtime.html. This option is best for end users who want to run deployed applications.
- Use the MATLAB function `compiler.runtime.download` to download the MATLAB Runtime installer matching the version and update level of MATLAB from where the command is executed. If the installer has already been downloaded to the machine, it returns the path to the MATLAB Runtime installer. If the machine is offline, it returns a URL to the MATLAB Runtime installer. This option is best for developers who want to create application installers that contain MATLAB Runtime.

Note If you want to install MATLAB Runtime to a shared network drive, see “Run Applications Using a Network Installation of MATLAB Runtime” (MATLAB Compiler).

Install MATLAB Runtime Interactively

To install MATLAB Runtime:

1. Extract the archive containing the MATLAB Runtime installer. The release part of the installer file name (*_R2023a_*) changes from one release to the next.

Platform	Steps
Windows	Unzip the MATLAB Runtime installer. Right-click the ZIP file `MATLAB_Runtime_R2023a_win64.zip` and select **Extract All**.
Linux	Unzip the MATLAB Runtime installer at the terminal using the `unzip` command. For example, if you are unzipping the R2023a MATLAB Runtime installer, at the terminal, type: `unzip MATLAB_Runtime_R2023a_glnxa64.zip`
macOS	Unzip the MATLAB Runtime installer at the terminal using the `unzip` command. For example, if you are unzipping the R2023a MATLAB Runtime installer, at the terminal, type: `unzip MATLAB_Runtime_R2023a_maci64.zip`

2 Start the MATLAB Runtime installer.

Platform	Steps
Windows	Double-click the file `setup.exe` from the extracted files to start the installer.

Linux	At the terminal, type:
	`sudo -H ./install`
	`sudo` is only required if you install to a directory that you do not have write access to.
	Note You may need to allow the root user to access the running X server:
	`xhost +SI:localuser:root`
	`sudo -H ./install`
	`xhost -SI:localuser:root`

macOS	At the terminal, type: `./install` **Note** You may need to enter an administrator user name and password after you run `./install`.

3 When the MATLAB Runtime installer starts, it displays a dialog box. Read the information and then click **Next** to proceed with the installation.

4 In the **Folder Selection** dialog box, specify the folder where you want to install MATLAB Runtime.

Note You can have multiple versions of MATLAB Runtime on your computer, but only one installation for any particular version. If you have an existing installation of the same version, the MATLAB Runtime installer does not display the **Folder Selection** dialog box because it overwrites the existing installation in the same folder.

5 Confirm your choices and click **Next**.

The MATLAB Runtime installer starts copying files into the installation folder.

6 On Linux and macOS platforms, after copying files to your disk, the MATLAB Runtime installer displays the **Product Configuration Notes** dialog box. This dialog box contains information necessary for setting your path environment variables. Copy the path information from this dialog box, save it to a text file, and then click **Next**.

For instructions on setting the path environment variables, see “Set MATLAB Runtime Path for Deployment” (MATLAB Compiler).

7 Click **Finish** to exit the installer.

Default Install Folder

The default MATLAB Runtime installation folders for R2023a are specified in the following table:

Operating System	MATLAB Runtime Installation Folder
Windows	`C:\Program Files\MATLAB\MATLAB Runtime\R2023a`
Linux	`/usr/local/MATLAB/MATLAB_Runtime/R2023a`
macOS	`/Applications/MATLAB/MATLAB_Runtime/R2023a`

Install MATLAB Runtime Noninteractively

If you have many installations to perform, you can specify installation arguments as command-line arguments or in an installer control file to save time and prevent errors. When you specify installation arguments, the MATLAB Runtime installer runs as a background task and does not display any dialog boxes.

When running noninteractively, the installer overwrites the installation location.

Caution On Linux and macOS systems, the installer displays information necessary for setting your environment variables in the **Product Configuration Notes** dialog box. If you use the installer noninteractively, you must locate your MATLAB Runtime installation directory in order to set the library path after installation. For more information, see “Set MATLAB Runtime Path for Deployment” (MATLAB Compiler).

Run Installer in Silent Mode

To install MATLAB Runtime in silent mode:

1. Extract the contents of the MATLAB Runtime installer archive to a temporary folder.
2. In your system command prompt, navigate to the folder where you extracted the installer.
3. Run the MATLAB Runtime installer, specifying the `-agreeToLicense yes` option on the command line. If you do not include `-agreeToLicense yes` as the first option, the installer will not install MATLAB Runtime.

Note On most platforms, the installer is located at the root of the folder into which the archive was extracted. On 64-bit Windows, the installer is located in the archive `bin` folder.

Platform	Command
Windows	`setup -agreeToLicense yes`
Linux	`sudo ./install -agreeToLicense yes` **Note** `sudo` is only required if you install to a directory you do not have write access to.
macOS	`./install -agreeToLicense yes`

Note To install MATLAB Runtime R2022a and earlier, you must also specify `-mode silent` in the command.

4 View a log of the installation.

- On Windows systems, the installer creates a log file named `mathworks_`*`username`*`.log`, where *`username`* is your Windows login name, in the location defined by your `TEMP` environment variable.
- On Linux and macOS systems, the installer displays the log information at the command prompt. It also saves it to a file if you use the `-outputFile` option.

Customize Noninteractive Installation

When run noninteractively, the installer uses the default values unless you specify otherwise. Like the MATLAB installer, the MATLAB Runtime installer accepts a number of options that modify the default installation properties.

To specify additional options on the command line, separate each option and its value with a space. For example, on Linux:

```
./install -agreeToLicense yes \
-outputFile myapp_log.txt -destinationFolder ~/MW/MATLAB_Runtime
```

Option	Description
`-agreeToLicense`	Agree to the MATLAB Runtime license.
`-destinationFolder`	Specifies where MATLAB Runtime is installed.
`-outputFile`	Specifies where the installation log file is written.

`-inputFile`	Specifies an installer control file that contains your command-line options and values. Omit the dash before each option, and put each option and value pair on a separate line. For example:
	
	`agreeToLicense=yes`
	`outputFile=myapp_log.txt`

Note The MATLAB installer archive includes an example installer control file called `installer_input.txt`. This file contains all of the options available for a full MATLAB installation. The MATLAB Runtime installer only accepts the options listed in this section.

Install MATLAB Runtime without Administrator Rights

On Linux, to install MATLAB Runtime without `sudo` privileges, select a folder that you have write access to during installation.

On Windows, to install MATLAB Runtime as a user without administrator rights:

1 Install MATLAB Runtime on a Windows machine where you have administrator rights.
2 Copy the folder where MATLAB Runtime was installed to the machine without administrator rights. You can compress the folder into a zip file for distribution.
3 On the machine without administrator rights, add the *<MATLAB_RUNTIME_INSTALL_DIR>*`\runtime\`*`arch`* directory to the user's `PATH` environment variable. For more information, see "Set MATLAB Runtime Path for Deployment" (MATLAB Compiler).

Install Multiple MATLAB Runtime Versions on Single Machine

`MCRInstaller` supports the installation of multiple versions of MATLAB Runtime on a target machine. This capability allows applications compiled with different versions of MATLAB Runtime to execute side by side on the same machine.

If you do not want multiple MATLAB Runtime versions on the target machine, you can remove the unwanted ones. On Windows, run **Add or Remove Programs** from the Control Panel to remove a specific version. On Linux, manually delete the unwanted MATLAB Runtime directories. You can remove unwanted versions before or after installation of a more recent version of MATLAB Runtime because versions can be installed or removed in any order.

Note Installing multiple versions of MATLAB Runtime on the same machine is not supported on macOS.

MATLAB and MATLAB Runtime on Same Machine

To test your deployed component on your development machine, you do not need an installation of MATLAB Runtime. The MATLAB installation that you use to compile the component can act as a MATLAB Runtime replacement.

You can install MATLAB Runtime for debugging purposes.

Modify Path

If you install MATLAB Runtime on a machine that already has MATLAB on it, you must adjust the system library path according to your needs.

To run deployed MATLAB code against MATLAB Runtime rather than MATLAB, ensure that your library path lists the MATLAB Runtime directories before any MATLAB directories. For information on setting environment variables, see “Set MATLAB Runtime Path for Deployment” (MATLAB Compiler).

Alternatively, you can specify the location of MATLAB Runtime using the generated shell script for your compiled application.

Uninstall MATLAB Runtime

Windows

1. Start the uninstaller.

   From the Windows Start menu, search for the **Add or Remove Programs** control panel, and double-click MATLAB Runtime in the list.

   You can also start the MATLAB Runtime uninstaller from the `<MATLAB_RUNTIME_INSTALL_DIR>\bin\<arch>` folder, where `<MATLAB_RUNTIME_INSTALL_DIR>` is your MATLAB Runtime installation folder and `<arch>` is an architecture-specific folder, such as `win32` or `win64`.
2. Select MATLAB Runtime from the list of products in the Uninstall Products dialog box and click **Next**.

3 Click **Finish**.

Linux

1 Close all instances of MATLAB and MATLAB Runtime.
2 Enter this command at the Linux terminal:

```
rm -rf <MATLAB_RUNTIME_INSTALL_DIR>
```

Caution Be careful when using the `rm` command, as deleted files cannot be recovered.

macOS

1 Close all instances of MATLAB and MATLAB Runtime.
2 Navigate to your MATLAB Runtime installation folder. For example, the installation folder might be named `MATLAB_Compiler_Runtime.app` in your Applications folder.
3 Drag your MATLAB Runtime installation folder to the trash, and then select **Empty Trash** from the Finder menu.

See Also

`compiler.runtime.download`

More About

- About MATLAB Runtime (MATLAB Compiler)
- “MATLAB Runtime Startup Options” (MATLAB Compiler)
- “Set MATLAB Runtime Path for Deployment” (MATLAB Compiler)

Specify Default MATLAB Runtime for New Server Instances

Each server instance that you create with MATLAB Production Server has its own configuration file that defines various server management criteria. Use the `mps-setup` command to set the default MATLAB Runtime for all on-premises server instances that you create. The `mps-setup` command line wizard searches your machine for installed MATLAB Runtime instances and sets the default path to the MATLAB Runtime for all server instances.

If you do not have MATLAB Runtime installed on your machine, you must install it first. For more information, see “Supported MATLAB Runtime Versions for MATLAB Production Server”.

To set the default MATLAB Runtime:

1. Open a system command prompt with `administrator` privileges.
2. From the command prompt, navigate to the MATLAB Production Server `script` folder and run `mps-setup`.

   Alternatively, add the `script` folder to your system `PATH` environment variable to run `mps-setup` from any folder on your system. The `script` folder is located at *`$MPS_INSTALL`*`\script`, where *`$MPS_INSTALL`* is the location in which MATLAB Production Server is installed. For example, on Windows, the default location for the `script` folder is `C:\Program Files\MATLAB\MATLAB Production Server\`*`ver`*`\script\mps-setup`, where *`ver`* is the version of MATLAB Production Server.
3. Follow the instructions in the command line wizard.

   The wizard searches your system for installed MATLAB Runtime instances and displays them.
4. Enter `y` to confirm or `n` to specify a default MATLAB Runtime for all server instances.

   If `mps-setup` cannot locate an installed MATLAB Runtime on your system, the wizard prompts you to enter a path name to a valid instance.

Run mps-setup in Non-Interactive Mode for Silent Install

You can also run `mps-setup` without interactive command input for silent installations. To do so, specify the path name of the MATLAB Runtime as a command line argument.

For example, on Windows, run the following at the system command prompt.

```
C:\>mps-setup "C:\Program Files\MATLAB\MATLAB Runtime\mcrver"
```

`mcrver` is the version of the MATLAB Runtime to use.

See Also

mcr-root | `mps-setup`

More About

- “Specify MATLAB Runtime for Server Instance Using Command Line” on page 1-15
- “Support Multiple MATLAB Runtime Versions” on page 1-17

Specify MATLAB Runtime for Server Instance Using Command Line

You can use the `main_config` server configuration file to specify the MATLAB Runtime location for an on-premises installation of MATLAB Production Server. For a server environment deployed in the cloud, the deployment specifies the MATLAB Runtime locations for you.

For an on-premises server installation, set the mcr-root property in the server configuration file to specify the MATLAB Runtime locations for the server to use.

1. Stop the server instance, if it is running.
2. Open the configuration file for the instance in a text editor. The configuration file is located at *`instanceRoot`*`/config/main_config`.
3. Locate the entry for the mcr-root property.

   ```
   --mcr-root mCRuNsETtOKEN
   ```
4. Modify the `mcr_root` property to point to the installed MATLAB Runtime you want to work with.

   For example:

   ```
   --mcr-root C:\Program Files\MATLAB\MATLAB Runtime\vnnn
   ```

   Note You must specify the MATLAB Runtime version number (v*nnn*). The MATLAB Runtime versions that you specify must be compatible with MATLAB Production Server.
5. Restart the server instance.

See Also

`mps-start` | `mps-setup` | `mcr-root`

More About

- `mps-new`
- “Specify Default MATLAB Runtime for New Server Instances” on page 1-14
- “Configure Server Using Configuration File” on page 1-6
- “Support Multiple MATLAB Runtime Versions” on page 1-17
- “Set MATLAB Runtime Location Using Dashboard” on page 1-26

Start Server Instance Using Command Line

In this section...
“Prerequisites” on page 1-16 “Procedure” on page 1-16

Follow the procedure below to start a server instance from the command line in an on-premises installation of MATLAB Production Server.

Prerequisites

Before attempting to start a server, verify that you have:

- Installed the MATLAB Runtime. For more information, see “Supported MATLAB Runtime Versions for MATLAB Production Server”.
- Specified the default MATLAB Runtime for the server instance. For more information, see “Specify Default MATLAB Runtime for New Server Instances”.
- Created a server instance. For more information, see “Create Server Instance Using Command Line”.

Procedure

To start a server instance, complete the following steps:

1. Open a system command prompt.
2. Enter the `mps-start` command:

   ```
   mps-start [-C path/]server_name [-f]
   ```

   where:

   - `-C` *`path/`* — Path to the server instance that you want to start. *`path`* should end with the server name.
   - *`server_name`* — Name of the server instance you want to start.
   - `-f` — Force command to succeed, regardless of whether the server is already started or stopped.

After you start a server, you can verify that the server is running by using the `mps-status` command.

See Also

`mps-start` | `mps-service` | `mps-new` | `mps-stop`

More About

- “Configure Server Instance as Windows Service” on page 1-22
- “Create Server Instance Using Command Line”
- “Deploy Archive to MATLAB Production Server”

Support Multiple MATLAB Runtime Versions

MATLAB Production Server instances can host deployable archives compiled using multiple versions of MATLAB Compiler SDK™. To do so, you must configure on-premises server instances to use multiple MATLAB Runtime versions using the mcr-root property. For a server environment deployed in the cloud, the deployment sets the `mcr-root` property to support multiple MATLAB Runtime versions.

Note An installation of MATLAB Production Server supports MATLAB Runtime versions up to six releases back.

Configure Server to Use Multiple MATLAB Runtime Instances

If you do not have a MATLAB Runtime installation on your on-premises server, you must install it first. For more information, see MATLAB Runtime.

Note

- Configure a server instance to use MATLAB Runtime roots on a local file system. Otherwise, a network partition might cause worker processes to fail.
- All paths to the MATLAB Runtime installations must belong to the same operating system and hardware combination.

Specify Multiple MATLAB Runtime Instances Using Command-Line

If you manage server instances using the command-line, add multiple `mcr-root` properties to the `main_config` configuration file to support multiple MATLAB Runtime instances.

1. Stop the server instance, if it is running.
2. Open the configuration file for the server instance in a text editor.

   The configuration file is at *`instanceRoot`*`/config/main_config`.
3. Locate the entry for the `mcr-root` property in the configuration file.

   ```
   --mcr-root mCRuNsETtOKEN
   ```
4. For each version of MATLAB Runtime that the instance supports, add an instance of the `mcr-root` property. Order the MATLAB Runtime versions from the latest to the oldest.

   For example, to configure the instance to use the v98 and v97 versions of the MATLAB Runtime, specify the following.

   ```
   --mcr-root C:\Program Files\MATLAB\MATLAB Runtime\v98
   --mcr-root C:\Program Files\MATLAB\MATLAB Compiler Runtime\v97
   ```
5. Restart the server instance.

Specify Multiple MATLAB Runtime Instances Using Dashboard

If you manage server instances using the dashboard, log in to the dashboard and update the server settings to support multiple MATLAB Runtime instances.

1. Stop the server instance, if it is running.
2. Under **Settings > Core**, locate the **MATLAB Runtime** field.
3. For each version of MATLAB Runtime that the instance supports, specify the path to the MATLAB Runtime installations separated by a comma. Order the MATLAB Runtime versions from the latest to the oldest.

   For example, to configure the instance to use the v98 and v97 versions of the MATLAB Runtime, specify the following.

   ```
   C:\Program Files\MATLAB\MATLAB Runtime\v98,C:\Program Files\MATLAB\MATLAB Runtime\v97
   ```
4. Restart the server instance.

How Server Instances Select Which MATLAB Runtime to Use

After you configure the server instance to use multiple versions of MATLAB Runtime, to process a server request, the server scans the list of MATLAB Runtime versions in the configuration file in order from first to last, and chooses the first MATLAB Runtime installation capable of processing the request. A MATLAB Runtime installation can process a request if it is compatible with the version of MATLAB used to create the deployable archive containing the function being evaluated.

Note Since the server instance always chooses the first compatible version of MATLAB Runtime, configuring the server instance with multiple instances of the same MATLAB Runtime version has no effect on performance.

Changes to Worker Management

Configuring a server instance to use multiple MATLAB Runtime versions changes the management of the workers that process requests.

When using a single MATLAB Runtime installation, the server instance starts the workers as needed, until a number of workers specified by the `num-workers` property are running. Once running, workers can restart depending on the `worker-restart-interval` property or the `worker-restart-memory-limit` property. Workers are never fully stopped.

Once a server instance starts using multiple MATLAB Runtime versions, it dynamically manages the worker pool. The server instance starts new workers as needed until `num-workers` workers are running. The worker instances are spread out over the different MATLAB Runtime versions. Once `num-workers` workers are running, the server instance returns workers to the pool of available workers based on the `worker-memory-trigger` property and the `queue-time-trigger` property. Once a worker returns to the pool, the server can allocate it to process new requests using any of the configured MATLAB Runtime versions.

See Also

`num-workers` | `worker-restart-interval` | `worker-restart-memory-limit`

More About

- “Specify Default MATLAB Runtime for New Server Instances” on page 1-14
- “Specify MATLAB Runtime for Server Instance Using Command Line” on page 1-15

- “Configure Server Using Configuration File” on page 1-6
- “Start Server Instance Using Command Line” on page 1-16

Control Worker Restarts

In this section...
“Restart Workers Based on Up Time” on page 1-20
“Restart Workers Based on Amount of Memory in Use” on page 1-20

Restart Workers Based on Up Time

As worker processes evaluate MATLAB functions, the MATLAB workspace accumulates saved state and other data. This accumulated data can occasionally cause a worker process to fail. One way to avoid random worker failures is to configure the server instances to restart worker processes when they have been running for set period.

1. If the server instance is running, stop it.
2. Open the configuration file for the instance in a text editor.

   The configuration file is at *`instanceRoot`*`/config/main_config`.
3. Locate the entry for the `worker-restart-interval` property.

   ```
   --worker-restart-interval 12:00:00
   ```
4. Change the value to the desired restart interval.

   For example, restart workers at intervals of 1 hour, 29 minutes, 5 seconds.

   ```
   --worker-restart-interval 1:29:05
   ```
5. Restart the server instance.

Restart Workers Based on Amount of Memory in Use

As worker processes evaluate MATLAB functions, the MATLAB workspace accumulates saved state and other data. This accumulated data can occasionally cause a worker process to fail. One way to avoid random worker failures is to configure the server instances to restart worker processes when they begin consuming a predefined amount of memory.

This is done by adjusting three configuration properties:

- worker-memory-check-interval— Interval at which workers are polled for memory usage
- worker-restart-memory-limit — Size threshold at which to consider restarting a worker
- worker-restart-memory-limit-interval — Interval for which a worker can exceed its memory limit before restart

To adjust memory-based restart thresholds:

1. If the server instance is running, stop it.
2. Open the configuration file for the instance in a text editor.

   The configuration file is at *`instanceRoot`*`/config/main_config`.
3. Locate the entry for the `worker-memory-check-interval` property.

   ```
   --worker-memory-check-interval 0:00:30
   ```
4. Change the value to the desired restart interval.

For example, restart workers at intervals of 1 hour, 29 minutes, 5 seconds.

```
--worker-memory-check-interval 1:29:05
```

5 Add an entry for the `worker-restart-memory-limit` property.

For example, consider restarting workers when they consume 1 GB of memory.

```
--worker-restart-memory-limit 1GB
```

6 Add an entry for the `worker-restart-memory-limit-interval` property.

For example, restart workers when they exceed the memory limit for 1 hour.

```
worker-restart-memory-limit-interval 1:00:00
```

7 Restart the server instance.

See Also

num-workers

More About

- “Configure Server Using Configuration File” on page 1-6
- “Support Multiple MATLAB Runtime Versions” on page 1-17

Configure Server Instance as Windows Service

In this section...
“Create New Server Instance as Windows Service” on page 1-22
“Make Existing Server Instance a Windows Service” on page 1-22
“Recovery Options for Server Instance Running as Windows Service” on page 1-22
“Manage Instances Using Dashboard” on page 1-23
“Work With Network Drives” on page 1-23

You can configure an on-premises MATLAB Production Server instance to start when the host machine starts by registering the server instance as a Windows service.

Create New Server Instance as Windows Service

To create a new on-premises server instance and register it as a Windows service, run the `mps-new` command with the `--service` option at the system command prompt.

```
mps-new C:\TEMP\server --service
```

You can assign a different name, description, and user for the Windows service than the defaults by providing arguments to `mps-new`.

The Windows service created for the server instance does not start automatically. To start the server instance, start the service, or start the server at the command prompt using the `mps-start`. You can edit the configuration for the instance before starting it.

The Windows service created for the server instance starts when the host machine starts. When the host machine restarts, the server instance restarts with it.

Make Existing Server Instance a Windows Service

To create a new Windows service for an existing on-premises server instance, use the `mps-service` command with the `create` option at the system command prompt.

```
mps-service -C C:\TEMP\server create
```

You can change the name, description, and user for the Windows service by providing arguments to `mps-service`.

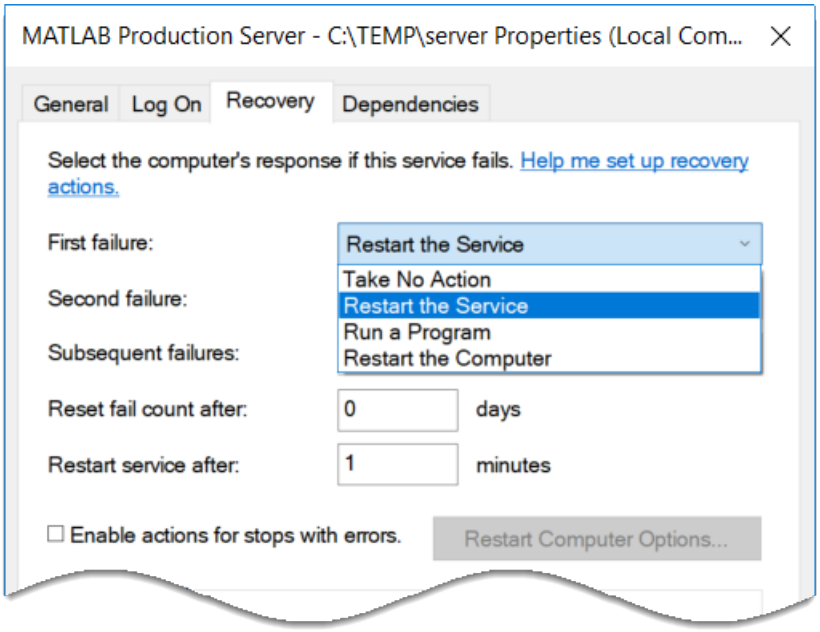
The Windows service created for the server instance is configured to start when the machine starts. When the host machine is restarted, the server instance restarts with it.

Recovery Options for Server Instance Running as Windows Service

You can specify how your system responds if the server instance running as a Windows service fails.

1. Open Service Control Manager in Windows.
2. Locate and double-click the server instance service that you want to configure for failure recovery.

3 Specify recovery options in the **Recovery** tab.



Manage Instances Using Dashboard

If you use the MATLAB Production Server dashboard to create and start your server instances, you must use only the dashboard and not command-line scripts to stop or delete those instances. For example, for server instances that you create using the dashboard, it is not recommended that you use command-line scripts such as `mps-service list` or `mps-service delete` to list and delete server instances.

Work With Network Drives

If you want to use network drives with a server instance that runs as a Windows service, it is recommended that you specify UNC paths to the network drives instead of mapped drive letters. When you configure a server instance to run as a Windows service, the default user account for the service is SYSTEM. However, the user account that starts the server, either using `mps-start` at the command line or by starting the Windows service, is not SYSTEM, but is a dedicated domain account—the user account that is logged in to the computer. Using UNC paths makes the network drives accessible to both the user accounts.

Depending on the permissions on the network share, you might choose to configure the Windows service to run under a dedicated domain account and grant to the dedicated domain account necessary permissions on the network share.

See Also

`mps-service` | `mps-new` | `mps-start`

More About

- “Start Server Instance Using Command Line” on page 1-16

Create Server Instance Using Dashboard

After you set up and log in to the dashboard in an on-premises server instance, you can create a server instance. For details on setting up the dashboard, see “Set Up and Log In to MATLAB Production Server Dashboard” on page 6-2.

1. In the dashboard, navigate to the server machine for the new instance.

   For example: **Servers** > **localhost**
2. Select **Create New**.
3. In the **Name** field, enter a name for the instance.

   - The name can be any combination of characters and numbers without spaces.
   - The name indicates the purpose of the server. For example, a server instance hosting trading functions for use by east coast traders could be named `trading_east`.
   - It must be unique to the server machine.
4. In the **Description** field, provide an optional description for the server instance.

   The description describes the function of the server instance. For example, the description for `trading_east` may be:

   ```
   Instance hosting arbitrage functions joe, fred, and arlo. This instance
   uses strong encryption, and access to the functions is limited to a
   select group of clients.
   ```
5. Click **Create**.

The new server instance is added to the dashboard in a stopped state. You must manually start it before it can process requests.

Note To start your instance, you will need to complete two additional steps:

1. Set the MATLAB Runtime location. For more information, see “Set MATLAB Runtime Location Using Dashboard” on page 1-26.
2. Specify license information. For more information, see “Specify Server Instance License Options Using Dashboard” on page 1-27.

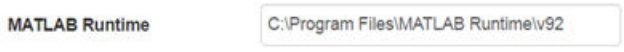
Failure to complete these steps will generate errors.

Set MATLAB Runtime Location

1. In the dashboard, select the server instance from the leftmost navigation pane.

   For example: **Servers** > **localhost** > **myinstance**
2. Select the **Settings** tab.
3. Expand the **Core** area.
4. Replace the value `mCRrOOTuNsET` in the field **MATLAB Runtime** with the location of your MATLAB Runtime.

   For example:



5 Click **Save**.
6 Start or restart the server instance.

If you do not set the location of the MATLAB Runtime, you will get the following error message :

```
Instance cannot be started: please update the MATLAB Runtime option in instance
settings by replacing the "mCRrOOTuNsET" with the full path to MATLAB runtime
directory
```

See Also

Related Procedures

- “Set MATLAB Runtime Location Using Dashboard” on page 1-26
- “Specify Server Instance License Options Using Dashboard” on page 1-27
- “Start Server Instance Using Dashboard” on page 1-28

Set MATLAB Runtime Location Using Dashboard

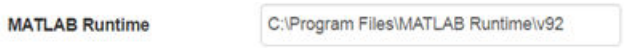
After you create an on-premises MATLAB Production Server instance using the dashboard, you can specify the location of MATLAB Runtime for the server to use.

If you use the command line to manage your on-premises server instance, to specify the location of MATLAB Runtime, see “Specify MATLAB Runtime for Server Instance Using Command Line” on page 1-15

1. In the dashboard, select the server instance from the leftmost navigation pane.

   For example: **Servers > localhost > myinstance**
2. Select the **Settings** tab.
3. Expand the **Core** area.
4. Replace the value `mCRrOOTuNsET` in the field **MATLAB Runtime** with the location of your MATLAB Runtime.

   For example:

5. Click **Save**.
6. Start or restart the server instance.

If you do not set the location of the MATLAB Runtime, you will get the following error message :

```
Instance cannot be started: please update the MATLAB Runtime option in instance
settings by replacing the "mCRrOOTuNsET" with the full path to MATLAB runtime
directory
```

See Also

Related Procedures

- “Specify Server Instance License Options Using Dashboard” on page 1-27
- “Start Server Instance Using Dashboard” on page 1-28
- “Create Server Instance Using Dashboard” on page 1-24

Specify Server Instance License Options Using Dashboard

After you create an on-premises server instance using the dashboard, you can specify licensing options. For details on creating a server instance, see “Create Server Instance Using Dashboard” on page 1-24.

1. Select the server instance from the leftmost navigation pane.
2. Select the **Settings** tab.
3. Expand the **License** area.
4. Set **License** to the server, license files, or both.

   You can specify multiple license servers including port numbers (*`port_number`*`@`*`license_server_name`*), as well as license files. List where you want the product to search in order of precedence, using semi-colons (;) as separators on Windows or colons (:) as separators on Linux.

   For example, on a Linux system, you specify this value for `license`:

   ```
   27000@hostA:/opt/license/license.dat:27001@hostB:./license.dat
   ```

   The system searches these resources in this order:

   a. `27000@hostA:` (`hostA` configured on port `27000`)
   b. `/opt/license/license.dat` (local license data file)
   c. `27001@hostB:` (`hostB` configured on port `27001`)
   d. `./license.dat` (local license data file)
5. Set **License Grace Period** to the maximum length of time MATLAB Production Server responds to HTTP requests after the license server heartbeat has been lost.
6. Set **License Poll Interval** to the interval of time that must pass:

   - After license server heartbeat has been lost and MATLAB Production Server stops responding to HTTP requests
   - Before license server is polled, to verify and check out a valid license
7. Click **Save**.
8. Restart the server instance.

See Also

license | license-grace-period | `license-poll-interval`

Start Server Instance Using Dashboard

In this section...
“Prerequisites” on page 1-28 “Start Server Instance From Server Information Page” on page 1-28 “Start Server Instance From Server Instance Page” on page 1-28

The MATLAB Production Server dashboard for an on-premises installation provides two ways to start a server instance that you have created. For information on creating a server instance, see “Create Server Instance Using Dashboard” on page 1-24.

Prerequisites

Before you start your server instance, you must complete two additional steps:

1. Set the MATLAB Runtime location. For more information, see “Set MATLAB Runtime Location Using Dashboard” on page 1-26.
2. Specify license information. For more information, see “Start Server Instance Using Dashboard” on page 1-28.

Failure to complete these steps will generate errors.

Start Server Instance From Server Information Page

1. In the dashboard, from the navigation tree, select the server machine.

   For example: **Servers** > **localhost**
2. Locate the server instance in the instance list.
3. Click the green arrow start button in the **Actions** column.

   For example:

Start Server Instance From Server Instance Page

1. In the dashboard, from the navigation tree, select the server instance.
2. Click the green arrow start button at the top.

Note To start your instance, you will need to complete two additional steps:

1. Set the MATLAB Runtime location. For more information, see “Set MATLAB Runtime Location Using Dashboard” on page 1-26.
2. Specify license information. For more information, see “Specify Server Instance License Options Using Dashboard” on page 1-27.

Failure to complete these steps will generate errors.

See Also

Related Examples

- “Set MATLAB Runtime Location Using Dashboard” on page 1-26
- “Specify Server Instance License Options Using Dashboard” on page 1-27
- “Create Server Instance Using Dashboard” on page 1-24

Persistence

Data Caching Basics

Persistence provides a mechanism to cache data between calls to MATLAB code running on a server instance. A *persistence service* runs separately from the server instance and can be started and stopped manually. A *connection name* links a server instance to a persistence service. A persistence service uses a *persistence provider* to store data. Currently, Redis is the only supported persistence provider. The connection name is used in MATLAB application code to create a *data cache* in the linked persistence service.

Typical Workflow for Data Caching

Steps	Command Line	Dashboard
1. Create file `mps_cache_config`	Manually create a JSON file and place it in the `config` folder of the server instance. Do not include the `.json` extension in the filename.	Automatically created.
2. Start persistence service	Use `mps-cache` to start a persistence service. For testing purposes, you can create a persistence service controller object using `mps.cache.control`.	• Create a persistence service. • Add the persistence service to a server instance using a connection name. • Start the persistence service. • Attach the connection associated with a persistence service to a server instance.
3. Create a data cache	Use `mps.cache.connect` to create a data cache.	Use `mps.cache.connect` to create a data cache.

Configure Server to Use Redis

Create Redis Configuration File

Before starting a persistence service for an on-premises server instance from the system command prompt, you must create a JSON file called `mps_cache_config` (no `.json` extension) and place it in the `config` folder of the server instance. If you use the dashboard to manage an on-premises server instance and for server deployments on the cloud, the `mps_cache_config` file is automatically created on server creation.

`mps_cache_config`

```
{
  "Connections": {
    "<connection_name>": {
      "Provider": "Redis",
      "Host": "<hostname>",
      "Port": <port_number>,
      "Key": <access_key>
    }
  }
}
```

Specify the <*connection_name*>, <*hostname*>, and <*port_number*> in the JSON file. The host name can either be `localhost` or a remote host name obtained from an Azure® Redis cache resource. If you use Azure Cache for Redis, you must specify an access key. To use an Azure Redis cache, you need a Microsoft® Azure account.

You can specify multiple connections in the file `mps_cache_config`. Each connection must have a unique name and a unique (host, port) pair. If you are using the persistence service through the dashboard, the file `mps_cache_config` is automatically created in the `config` folder of the server instance.

Install WSL for Server Instances Running on Windows

If your MATLAB Production Server instance runs on a Windows machine, you require additional configuration. The following configuration is not required for server instances that run on Linux and macOS.

- You need to install Windows Subsystem for Linux (WSL). For details on installing WSL, see Microsoft documentation.
- If the MATLAB Production Server software is installed on a network drive, you must mount the network drive in WSL.

Example: Increment Counter Using Data Cache

This example shows you how to use persistence to increment a counter using a data cache. The example presents two workflows: a testing workflow that uses the MATLAB and a deployment workflow that requires an active server instance.

Testing Workflow in MATLAB Compiler SDK

1. Create a persistence service that uses Redis as the persistence provider and start the service.

```
ctrl = mps.cache.control('myRedisConnection','Redis','Port',4519)
start(ctrl)
```

2. Write MATLAB code that creates a cache and then updates a counter using the cache. Name the file `myCounter.m`

myCounter.m

```
function x = myCounter(cacheName,connectionName)

% create a data cache
c = mps.cache.connect(cacheName,'Connection',connectionName);

% if the key 'count' doesn't exist yet, initialize it
if isKey(c,'count') == false
    put(c,'count',0)
else
    value = get(c,'count');
    % increment the counter
    put(c,'count', value+1);
end
x = get(c,'count');
```

3. Test the counter.

```
for i = 1:5
    y(i) = myCounter('myCache','myRedisConnection');
```

```
end
y
```

```
y =

     0     1     2     3     4
```

Deployment Workflow Using MATLAB Production Server

Before you deploy code that uses persistence to a server instance, start the persistence service and attach it to the server instance. You can start the persistence service from the system command line using `mps-cache` or follow the steps in the dashboard. This example assumes your server instance uses the default host and port: `localhost:9910`.

1. Package the file `myCounter.m` using the **Production Server Compiler** app or `mcc`.
2. Deploy the archive (`myCounter.ctf` file) to the server.
3. Test the counter. You can make calls to the server using the "RESTful API for MATLAB Function Execution" from the MATLAB desktop.

```
rhs = {['myCache'],['myRedisConnection']};
body = mps.json.encoderequest(rhs,'Nargout',1);

options = weboptions;
options.ContentType = 'text';
options.MediaType = 'application/json';
options.Timeout = 30;

for i = 1:5
response = webwrite('http://localhost:9910/myCounter/myCounter', body, options);
x(i) = mps.json.decoderesponse(response);
end
x = [x{:}]
```

```
x =

     0     1     2     3     4
```

As expected, the results from the testing environment workflow and the deployment environment workflow are the same.

See Also

`mps.cache.Controller` | `mps.cache.DataCache` | `mps.sync.TimedMATFileMutex` | `mps.sync.TimedRedisMutex` | `mps.cache.control` | `mps.cache.connect` | `mps.sync.mutex`

More About

- "Manage Application State in Deployed Archives" on page 2-5

Manage Application State in Deployed Archives

This example shows how to manage persistent data in application archives deployed to MATLAB Production Server. It uses the MATLAB Production Server “RESTful API for MATLAB Function Execution” and JSON to connect one or more instances of a MATLAB app to an archive deployed on the server.

MATLAB Production Server workers are stateless. Persistence provides a mechanism to maintain state by caching data between multiple calls to MATLAB code deployed on the server. Multiple workers have access to the cached data.

The example describes two workflows.

1. A testing workflow for testing the functionality of the application in a MATLAB desktop environment before deploying it to the server.
2. A deployment workflow that uses an active MATLAB Production Server instance to deploy the archive.

To demonstrate how to use persistence, this example uses the traveling salesman problem, which involves finding the shortest possible route between cities. This implementation stores a persistent MATLAB `graph` object in the data cache. Cities form the nodes of the graph and the distances between the cities form the weights associated with the graph edges. In this example, the graph is a complete graph. The testing workflow uses the local version of the route-finding functions. The deployment workflow uses route-finding-functions that are packaged into an archive and deployed to the server. The MATLAB app calls the route-finding functions. These functions read from and write graph data to the cache.

The code for the example is located at *`$MPS_INSTALL`*`/client/matlab/examples/persistence/TravelingSalesman`, where *`$MPS_INSTALL`* is the location where MATLAB Production Server is installed.

To host a deployable archive created with the **Production Server Compiler** app, you must have a version of MATLAB Runtime installed that is compatible with the version of MATLAB you use to create your archive. For more information, see “Supported MATLAB Runtime Versions for MATLAB Production Server”.

1. “Step 1: Write MATLAB Code that uses Persistence Functions” on page 2-5
2. “Step 2: Run Example in Testing Workflow” on page 2-9
3. “Step 3: Run Example in Deployment Workflow” on page 2-10

Step 1: Write MATLAB Code that uses Persistence Functions

1. Write a function to initialize persistent data

   Write a function to check whether a graph of cities and distances exists in the data cache. If the graph does not exist, create it from an Excel® spreadsheet that contains the distance data and write it to the cache. Because only one MATLAB Production Server worker at a time can perform this write operation, use a synchronization lock to ensure that data initialization happens only once.

   Connect to the cache that stores the distance data or create it if it does not exist using `mps.cache.connect`. Acquire a lock on a mutex using `mps.sync.mutex` for the duration of the write operation. Release the lock once the data is written to the cache.

Initialize the distance data using the `loadDistanceData` function.

```
function tf = loadDistanceData(connectionName, cacheName)
    c = mps.cache.connect(cacheName,'Connection',connectionName);
    tries = 0;

    while isKey(c,'Distances') == false && tries < 6
        lk = mps.sync.mutex('DistanceData','Connection',connectionName);
        if acquire(lk,10)
            if isKey(c,'Distances') == false
                g = initDistanceData('Distances.xlsx');
                c.Distances = g;
            end
            release(lk);
        end
        tries = tries + 1;
    end
    tf = isKey(c,'Distances');
end
```

2 Write functions to read persistent data

Write a function to read the distance data graph from the data cache. Because reading data from the cache is an idempotent operation, you do not need to use synchronization locks. Connect to the cache using `mps.cache.connect` and then retrieve the graph.

Read the graph from the cache and convert it into a cell array using the `listDestinations` function.

Calculate the shortest possible route using the `findRoute` function. Use the nearest neighbor algorithm, by starting at a given city and repeatedly visiting the next nearest city until all cities have been visited.

```
function destinations = listDestinations()
    c = mps.cache.connect('TravelingSalesman','Connection','ScratchPad');
    if loadDistanceData('ScratchPad','TravelingSalesman') == false
        error('Failed to load distance data. Cannot continue.');
    end

    g = c.Distances;
    destinations = table2array(g.Nodes);
end

function [route,distance] = findRoute(start,destinations)
    c = mps.cache.connect('TravelingSalesman','Connection','ScratchPad');
    if loadDistanceData('ScratchPad','TravelingSalesman') == false
        error('Failed to load distance data. Cannot continue.');
    end

    g = c.Distances;
    route = {start};
    distance = 0;
    current = start;

    while ~isempty(destinations)
        minDistance = Inf;
        nextSegment = {};
        for n = 1:numel(destinations)
```

```
            [p,d] = shortestpath(g,current,destinations{n});
            if d < minDistance
                nextSegment = p(2:end);
                minDistance = d;
            end
        end

        current = nextSegment{end};
        distance = distance + minDistance;
        destinations = setdiff(destinations,current);
        route = [ route nextSegment ];
    end
end
```

3 Write a function to modify persistent data

Write a function to add a new city. Adding a city modifies the graph stored in the data cache. Because this operation requires writing to the cache, use the `mps.sync.mutex` function described in Step 1 for locking. After adding a city, check that the graph is still complete by confirming that the distance between every pair of cities is known.

Add a city using the `addDestination` function. Adding a city adds a new graph node `name` along with new edges connecting this node to all existing nodes in the graph. The weights of the newly added edges are given by the vector `distances`. `destinations` is a cell array of character vectors that has the names of other cities in the graph.

```
function count = addDestination(name, destinations, distances)
    count = 0;
    c = mps.cache.connect('TravelingSalesman','Connection','ScratchPad');
    if loadDistanceData('ScratchPad','TravelingSalesman') == false
        error('Failed to load distance data. Cannot continue.');
    end

    lk = mps.sync.mutex('DistanceData','Connection','ScratchPad');
    if acquire(lk,10)
        g = c.Distances;
        newDestinations = setdiff(g.Nodes.Name, destinations);
        if ~isempty(newDestinations)
            error('MPS:Example:TSP:MissingDestinations', ...
                  'Add distances for missing destinations: %s', ...
                strjoin(newDestinations,', '));
        end

        src = repmat({name},1,numel(destinations));
        g = addedge(g, src, destinations, distances);
        c.Distances = g;
        release(lk);
        count = numnodes(g);
    end
end
```

4 Write a MATLAB app to call route-finding functions

Write a MATLAB app that wraps the functions described in Steps 2 and 3 in their respective proxy functions. The app allows you to specify a host and a port. For testing, invoke the local version of the route-finding functions when the host is blank and the port has the value 0. For the deployment workflow, invoke the deployed functions on the server running on the specified host and port. Use the `webwrite` function to send HTTP POST requests to the server.

For more information on how to write an app, see “Create and Run a Simple App Using App Designer” (MATLAB).

Write the proxy functions `findRouteProxy`, `addDestinationProxy`, and `listDestinationProxy` for the `findRoute`, `addDestination`, and `listDestination` functions, respectively.

```
function destinations = listDestinationsProxy(app)
    if isempty(app.HostEditField.Value) && ...
            app.PortEditField.Value <= 0
        destinations = listDestinations();
        return;
    end

    listDestinations_OPTIONS = weboptions('MediaType','application/json','Timeout',60,'ContentType','raw');
    listDestinations_HOST = app.HostEditField.Value;
    listDestinations_PORT = app.PortEditField.Value;
    noInputJSON = '{ "rhs": [], "nargout": 1 }';
    destinations_JSON = webwrite(sprintf('http://%s:%d/TravelingSalesman/listDestinations', ...
        listDestinations_HOST,listDestinations_PORT), noInputJSON, listDestinations_OPTIONS);
    if iscolumn(destinations_JSON), destinations_JSON = destinations_JSON'; end
    destinations_RESPONSE = mps.json.decoderesponse(destinations_JSON);
    if isstruct(destinations_RESPONSE)
        error(destinations_RESPONSE.id,destinations_RESPONSE.message);
    else
        if nargout > 0, destinations = destinations_RESPONSE{1}; end
    end
end

function [route,distance] = findRouteProxy(app,start,destinations)
    if isempty(app.HostEditField.Value) && ...
            app.PortEditField.Value <= 0
        [route,distance] = findRoute(start,destinations);
        return;
    end
    findRoute_OPTIONS = weboptions('MediaType','application/json','Timeout',60,'ContentType','raw');
    findRoute_HOST = app.HostEditField.Value;
    findRoute_PORT = app.PortEditField.Value;
    start_destinations_DATA = {};
    if nargin > 0, start_destinations_DATA = [ start_destinations_DATA { start } ]; end
    if nargin > 1, start_destinations_DATA = [ start_destinations_DATA { destinations } ]; end
    route_distance_JSON = webwrite(sprintf('http://%s:%d/TravelingSalesman/findRoute', ...
            findRoute_HOST,findRoute_PORT), ...
            mps.json.encoderequest(start_destinations_DATA,'nargout',nargout), findRoute_OPTIONS);
    if iscolumn(route_distance_JSON), route_distance_JSON = route_distance_JSON'; end
    route_distance_RESPONSE = mps.json.decoderesponse(route_distance_JSON);
    if isstruct(route_distance_RESPONSE)
        error(route_distance_RESPONSE.id,route_distance_RESPONSE.message);
    else
        if nargout > 0, route = route_distance_RESPONSE{1}; end
        if nargout > 1, distance = route_distance_RESPONSE{2}; end
    end
end

function count = addDestinationProxy(app, name, destinations,distances)
    if isempty(app.HostEditField.Value) && ...
            app.PortEditField.Value <= 0
        count = addDestination(name, destinations,distances);
        return;
    end

    addDestination_OPTIONS = weboptions('MediaType','application/json','Timeout',60,'ContentType','raw');
    addDestination_HOST = app.HostEditField.Value;
    addDestination_PORT = app.PortEditField.Value;
    name_destinations_distances_DATA = {};
    if nargin > 0, name_destinations_distances_DATA = [ name_destinations_distances_DATA { name } ]; end
    if nargin > 1, name_destinations_distances_DATA = [ name_destinations_distances_DATA { destinations } ]; end
    if nargin > 2, name_destinations_distances_DATA = [ name_destinations_distances_DATA { distances } ]; end
    count_JSON = webwrite(sprintf('http://%s:%d/TravelingSalesman/addDestination', ...
        addDestination_HOST,addDestination_PORT), ...
        mps.json.encoderequest(name_destinations_distances_DATA,'nargout',nargout), addDestination_OPTIONS);
```

```
    if iscolumn(count_JSON), count_JSON = count_JSON'; end
    count_RESPONSE = mps.json.decoderesponse(count_JSON);
    if isstruct(count_RESPONSE)
        error(count_RESPONSE.id,count_RESPONSE.message);
    else
        if nargout > 0, count = count_RESPONSE{1}; end
    end
end
```

Step 2: Run Example in Testing Workflow

Test the example code in the MATLAB desktop environment. To do so, copy the all the files located at *$MPS_INSTALL*`/client/matlab/examples/persistence/TravelingSalesman` to a writable folder on your system, for example, `/tmp/persistence_example`. Start the MATLAB desktop and set the current working directory to `/tmp/persistence_example` using the `cd` command.

For testing purposes, control a persistence service from the MATLAB desktop with the `mps.cache.control` function. This function returns an `mps.cache.Controller` object that manages the life cycle of a local persistence service.

1. Create an `mps.cache.Controller` object for a local persistence service that uses the Redis persistence provider.

   ```
   >> ctrl = mps.cache.control('ScratchPad', 'Redis', 'Port', 8675);
   ```

   When active, this controller enables a connection named `ScratchPad`. Connection names link caches to storage locations in persistence services. The `mps.cache.connect` function requires connection names to create data caches. The MATLAB Production Server administrator sets connection names in the cache configuration file `mps_cache_config`. For details, see “Configure Server to Use Redis”. By using the same connection names in MATLAB desktop sessions, you enable your code to move from development through testing to production without change.

2. Start the persistence service using `start`.

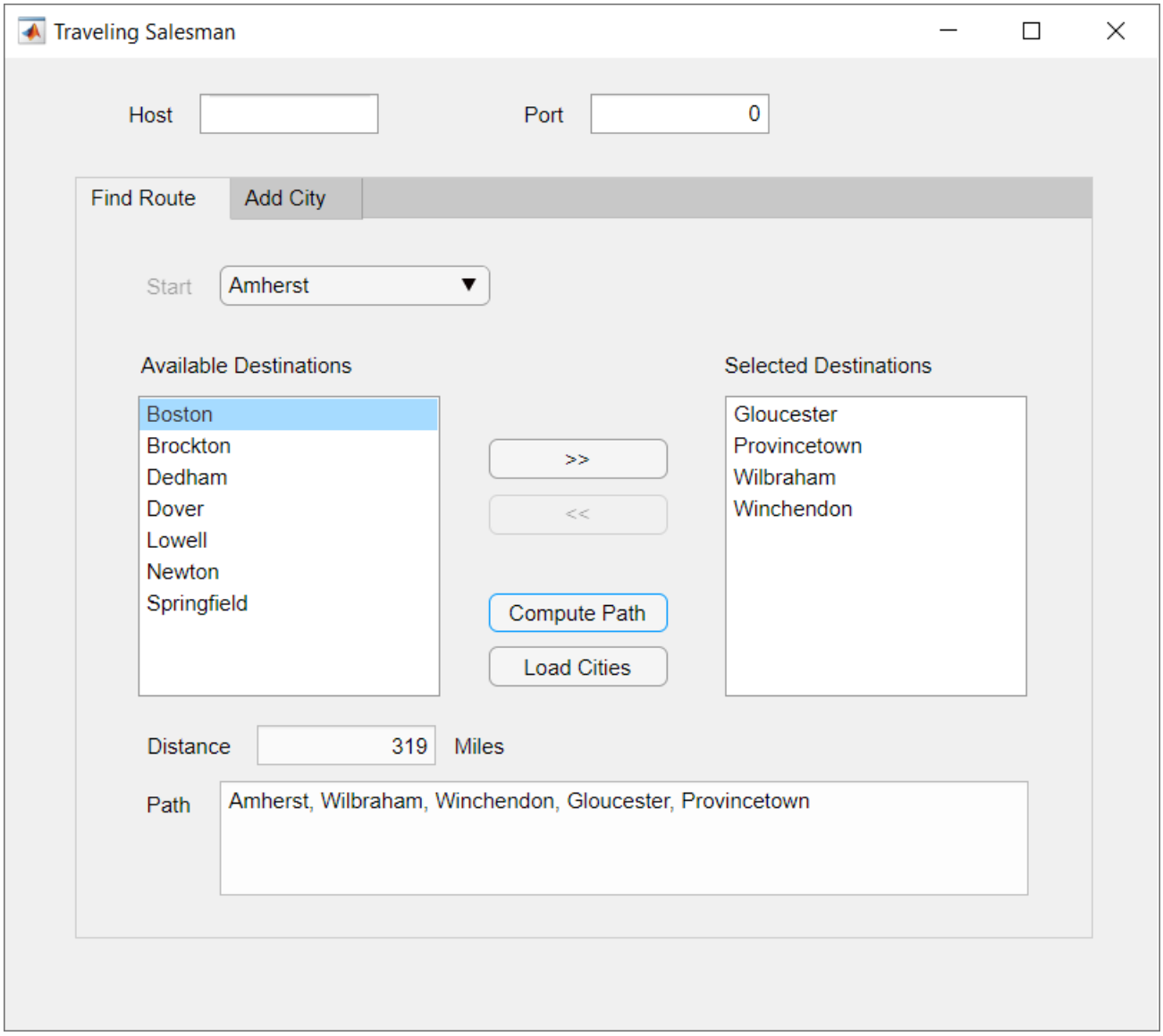
   ```
   >> start(ctrl);
   ```

3. Start the `TravelingSalesman` route-finding app that uses the persistence service.

   ```
   >> TravelingSalesman
   ```

   The app starts with default values for **Host** and **Port**.

   Click **Load Cities** to load the list of cities. Use the **Start** menu to set a starting location and the **>>** and **<<** buttons to select and deselect cities to visit. Click **Compute Path** to display a route that visits all the cities.



4 When you close the app, stop the persistence service using `stop`. Stopping a persistence service will delete the data stored by that service.

```
>> stop(ctrl);
```

Step 3: Run Example in Deployment Workflow

To run the example in the deployment workflow, copy the all the files located at *`$MPS_INSTALL`*`/client/matlab/examples/persistence/TravelingSalesman` to a writeable folder on your system, for example, `/tmp/persistence_example`. Start the MATLAB desktop and set the current working directory to `/tmp/persistence_example` using the MATLAB `cd` command.

The deployment workflow manages the lifetime of a persistence service outside of a MATLAB desktop environment and invokes the route-finding functions packaged in an archive deployed to the server.

1 Create a MATLAB Production Server instance

Create a server from the system command line using `mps-new`. For more information, see “Create Server Instance Using Command Line” on page 1-4. If you have not already set up your server environment, see `mps-setup` for more information.

Create a new server `server_1` located in the folder `tmp`.

```
mps-new /tmp/server_1
```

Alternatively, use the MATLAB Production Server dashboard to create a server. For more information, see “Set Up and Log In to MATLAB Production Server Dashboard” on page 6-2.

2. Create a persistence service connection

   The deployable archive requires a persistence service connection named `ScratchPad`. Use the dashboard to create the `ScratchPad` connection or copy the file `mps_cache_config` from the example directory to the config directory of your server instance. If you already have an `mps_cache_config` file in your config directory, edit it to add the `ScratchPad` connection as specified in the example `mps_cache_config`.

3. Create a deployable archive with the Production Server Compiler App and deploy it to the server

   1. Open **Production Server Compiler** app

      - MATLAB toolstrip: On the **Apps** tab, under **Application Deployment**, click **Production Server Compiler**.

      - MATLAB command prompt: Enter `productionServerCompiler`.

   2. In the **Application Type** menu, select **Deployable Archive**.
   3. In the **Exported Functions** field, add `findRoute.m`, `listDestinations.m` and `addDestination.m`.
   4. Under **Archive information**, rename the archive to `TravelingSalesman`.
   5. Under **Additional files required for your archive to run**, add `Distances.xlsx`.
   6. Click **Package**.
   7. The generated deployable archive `TravelingSalesman.ctf` is located in the `for_redistribution` folder of the project. Copy the `TravelingSalesman.ctf` file to the `auto_deploy` folder of the server, `/tmp/server_1/auto_deploy` in this example, for hosting.

4. Start the server instance

   Start the server from the system command line using `mps-start`.

   ```
   mps-start -C /tmp/server_1
   ```

   Alternatively, use the dashboard to start the server.

5. Start the persistence service

   Start the persistence service from the system command line using `mps-cache`.

   ```
   mps-cache start -C /tmp/server_1 --connection ScratchPad
   ```

   Alternatively, use the dashboard to start and attach the persistence service.

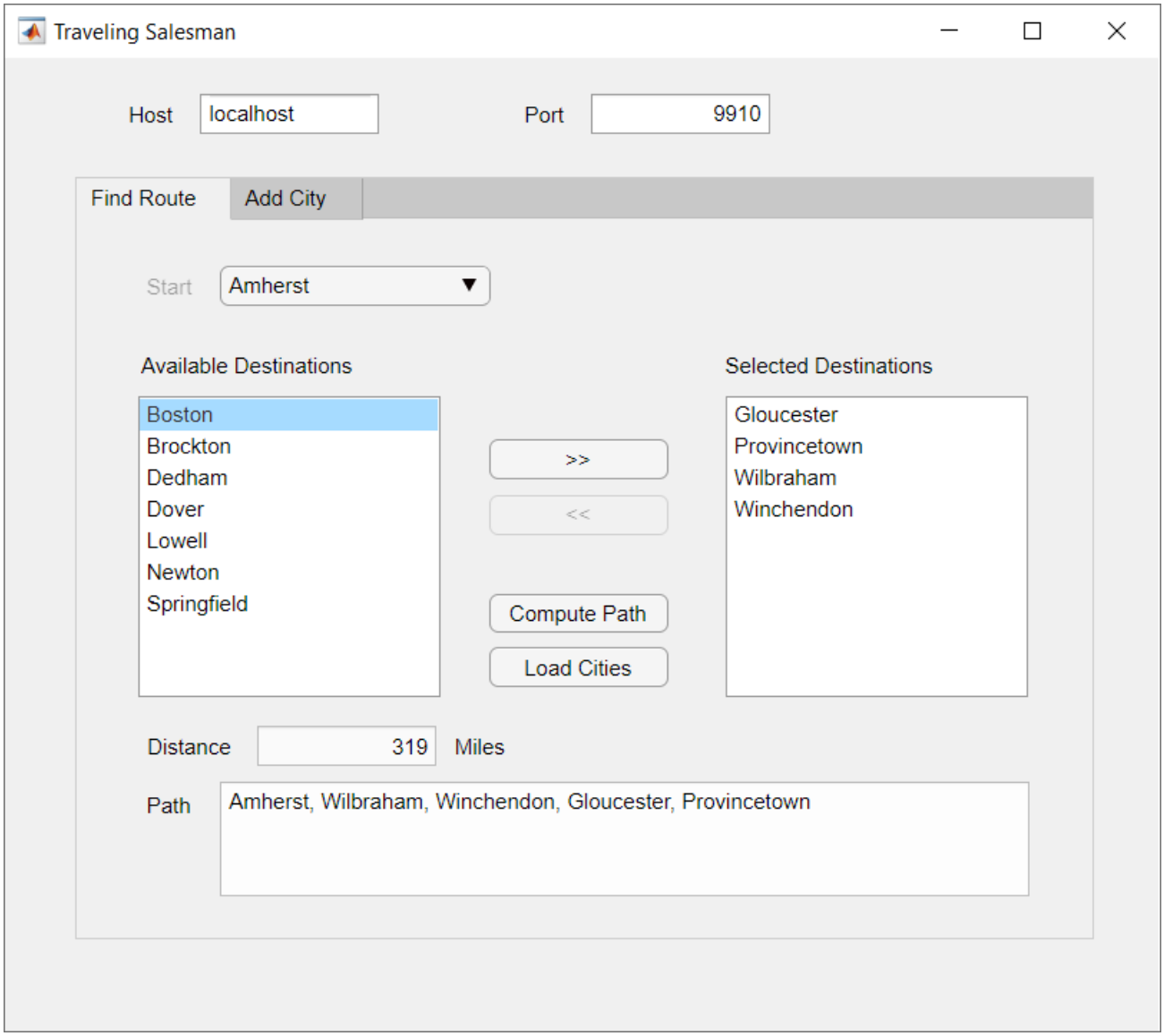
6. Test the app

   Start the `TravelingSalesman` route-finding app that uses the persistence service.

   ```
   >> TravelingSalesman
   ```

The app starts with empty values for **Host** and **Port**. Refer to the server configuration file `main_config` located at *`server_name`*`/config` to get the host and port values for your MATLAB Production Server instance. For this example, find the config file at `/tmp/server_1/config`. Enter the host and port values in the app.

Click **Load Cities** to load the list of cities. Use the **Start** menu to set a starting location and the **>>** and **<<** buttons to select and deselect cities to visit. Click **Compute Path** to display a route that visits all the cities.



The results from the testing environment workflow and the deployment environment workflow are the same.

See Also

`mps.cache.Controller` | `mps.cache.DataCache` | `mps.sync.TimedMATFileMutex` | `mps.sync.TimedRedisMutex` | `mps.cache.control` | `mps.cache.connect` | `mps.sync.mutex`

More About

- “Data Caching Basics” on page 2-2

Handle Custom Routes and Payloads in HTTP Requests

Web request handlers for MATLAB Production Server provide flexible client-server communication.

- Client programmers can send custom HTTP headers and payloads in RESTful requests to the server.
- Server administrators can provide flexible mapping of the request URLs to deployed MATLAB functions.
- Server administrators can provide static file serving.
- MATLAB programmers can return custom HTTP headers, HTTP status codes, HTTP status messages, and payloads in functions deployed to MATLAB Production Server.

To use web request handlers, you write the MATLAB function that you deploy to the server in a specific way and specify custom URL routes in a JSON file on the server.

Write MATLAB Function for Web Request Handler

To work as a web request handler, the MATLAB function that you deploy to the server must accept one input argument that is a scalar structure array, and return one output argument that is a scalar structure array.

The structure in the function input argument provides information about the client request. Clients can send custom HTTP headers and custom payloads. There are no data format restrictions on the payload that the deployed function can accept. For example, the function can accept raw data in binary or ASCII formats, CSV data, or JSON data that is not in the schema specified by the MATLAB Production Server RESTful API. Clients can also use the `Transfer-Encoding: chunked` header to send data in chunks. In chunked transfer encoding, though the server receives payload in chunks, the input structure receives payload data in entirety.

The structure in the function input argument contains the following fields:

Field Name	Data Type	Dimensions	Description
`ApiVersion`	`double`	1 x 3	Version of the input structure schema in the format *<major> <minor> <fix>*
`Body`	`uint8`	1 x N	Request payload
`Headers`	`cell`	N x 2	HTTP request headers Each element in the cell array represents a header. Each element is a key-value pair, where the key is of type `char` and the value can be of type `char` or `double`.
`HttpVersion`	`double`	1 x 2	HTTP version in the format *<major> <minor>*

Field Name	Data Type	Dimensions	Description
`Method`	`char`	1 x N	HTTP request method
`Path`	`char`	1 x N	Path of request URL

Since the deployed MATLAB function can accept custom headers and payloads in RESTful requests, you can vary the behavior of the MATLAB function depending on the request header data. You can use the structure in the function output argument to return a response with custom HTTP headers and payload. Server processing errors, if any, override any custom HTTP headers that you might set. If a MATLAB error occurs, the server returns an HTTP `500 Internal Server Error` response. All fields in the structure are optional.

The structure in the output argument can contain the following fields:

Field Name	Data Type	Dimensions	Description
`ApiVersion`	`double`	1 x 3	Version of the output structure schema in the format *<major> <minor> <fix>*
`Body`	`uint8`	1 x N	Response payload
`Headers`	`cell`	N x 2	HTTP response headers Each element in the cell array represents a header. Each element is a key-value pair, where the key is of type `char` and the value can be of type `char` or `double`.
`HttpCode`	`double`	1 x 1	HTTP status code
`HttpMessage`	`char`	1 x N	HTTP status message

Configure Server for URL Routes

Custom URL routes allow the server to map the path in request URLs to any deployable archive and MATLAB function deployed on the server.

To specify the mapping of a request URL to a deployed MATLAB function, you use a JSON file present on the server. The default name of the file is `routes.json` and its default location is in the *$MPS_INSTALL*`/config` directory. You can change the file name and its location by changing the value of the `--routes-file` property in the `main_config` server configuration file. You must restart the server after making any updates to `routes.json`.

When the server starts, it tries to read the `routes.json` file. If the file does not exist or contains errors, the server does not start, and writes an error message to the `main.log` file present in the directory that the log-root property specifies.

The default `routes.json` contains a `version` field with a value of `1.0.0`, and an empty `pathmap` field. `version` specifies the schema version of the file. You do not need to change its value. To allow custom routes, edit the file to specify mapping rules in the `pathmap` array. In the `pathmap` array, you can specify multiple objects, where each object corresponds to a URL route.

Following is the schema of `routes.json`.

```
{
  "version": "1.0.0",
   "pathmap": [
       {
           "match": "<regular_expression>",
           "webhandler": {
               "component": "<name_of_deployable_archive>",
               "function": "<name_of_deployed_function>"
           }
       },
    {
           "match": "<regular_expression>",
           "webhandler": {
               "component": "<name_of_deployable_archive>",
               "function": "<name_of_deployed_function>"
           }
       }
    ]
}
```

To specify a URL mapping rule, use the `match` and `webhandler` fields from the `pathmap` array.

- In the `match` field, specify a regular expression that uses ECMAScript grammar to match the path in a request URL.
  - If the request URL matches multiple regular expressions in the `match` field, the first match starting from the beginning of the file is selected.
  - The regular expression patterns are considered a match if any substring of the request URL is a match. For example, the pattern `a/b` matches `a/b`, `/a/b`, and `/x/a/b/y`. However, you can use the regular expression anchors, `^` and `$`, to match positions before or after specific characters. For example, the pattern `^a/b$` only matches `a/b`.
  - You can specify regular expressions that match query parameters in the request URL. However, asynchronous request execution using the MATLAB Production Server RESTful API is not supported. Request execution is synchronous. For more information about the MATLAB Production Server RESTful API, see “RESTful API for MATLAB Function Execution”.
- In the `webhandler` field, use the `component` field to specify the name of the deployable archive and the `function` field to specify the name of the deployed function for the request URL to execute.

End-to-End Setup for Web Request Handler

This example assumes you have a server instance running at the default host and port, localhost:9910. For information on starting a server, see “Start Server Instance Using Command Line” on page 1-16.

1 Write a MATLAB function for the web request handler.

The following code shows a MATLAB function that uses an input argument structure `request`, whose fields provide information about the request headers and body. The function also constructs and returns a structure `response`, whose fields contain a success HTTP code and status message, custom headers, and a message body.

```
function response = hellowh(request)
    disp(request);
    disp('request.Headers:');
    disp(request.Headers);
    bodyText = char(request.Body);
    disp('request.Body:');
    if length(bodyText) > 100
        disp(bodyText(1:100));
        disp('...');
    else
        disp(bodyText);
    end

    response = struct('ApiVersion', [1 0 0], ...
                      'HttpCode', 200, ...
                      'HttpMessage', 'OK', ...
                      'Headers', {{ ...
                        'Server' 'WebFunctionTest/1'; ...
                        'X-MyHeader' 'foobar'; ...
                        'X-Request-Body-Len' sprintf('%d', length(request.Body)); ...
                        'Content-Type' 'text/plain'; ...
                      }},...
                      'Body', uint8('hello, world'));

    disp(response);
    disp('response.Headers:');
    disp(response.Headers);
end
```

2. Package the function into a deployable archive.

   The following command compiles the `hellowh.m` function into a deployable archive, `whdemo.ctf`. For other ways to create deployable archives, see “Create Deployable Archive for MATLAB Production Server”.

   ```
   mcc -v -U -W 'CTF:whdemo' hellowh.m
   ```

3. Deploy the archive, `whdemo`, to the server. For more information, see “Deploy Archive to MATLAB Production Server”.
4. Edit the `routes.json` file on the server to map a client request to the deployed function. Restart the server instance for the changes to take effect. See mps-restart.

   The following file maps any client request that contains `MyDemo` in the request URL to the `hellowh` function in the `whdemo` archive deployed to the server.

   ```
   {
     "version": "1.0.0",
       "pathmap": [
           {
               "match": "^/MyDemo/.*",
               "webhandler": {
                   "component": "whdemo",
                   "function": "hellowh"
               }
           }
       ]
   }
   ```

5. Use a client of your choice to invoke the deployed function.

   The following command uses cURL to invoke the deployed function from the system command line.

   ```
   curl -v http://localhost:9910/MyDemo/this/could/be/any/path?param=YES
   ```

You see the following output at the system command line:

```
*   Trying ::1...
* TCP_NODELAY set
* Connected to localhost (::1) port 9910 (#0)
> GET /MyDemo/this/could/be/any/path?param=YES HTTP/1.1
> Host: localhost:9910
> User-Agent: curl/7.55.1
> Accept: */*
>
< HTTP/1.1 200 OK
< Server: WebFunctionTest/1
< X-MyHeader: foobar
< X-Request-Body-Len: 0
< Content-Type: text/plain
< Content-Length: 12
< Connection: Keep-Alive
<
hello, world* Connection #0 to host localhost left intact
```

See Also

files-root

Related Examples

- “Test Web Request Handlers” (MATLAB Compiler SDK)
- “Create Deployable Archive for MATLAB Production Server”
- “Deploy Archive to MATLAB Production Server”

Secure a Server

- “Enable HTTPS” on page 3-2
- “Configure Client Authentication” on page 3-4
- “Specify Access to MATLAB Programs” on page 3-5
- “Adjust Security Protocols” on page 3-6
- “Improve Startup Time When Security Is Activated” on page 3-7
- “Application Access Control” on page 3-8
- “Use Kerberos Authentication and Kerberos Delegation” on page 3-14

Enable HTTPS

MATLAB Production Server uses HTTPS to establish secure connections between server instances and clients. HTTPS provides certificate-based authentication for the client to validate the connection to the server. Optionally, you can configure HTTPS such that the server can provide certificate-based authentication of the client. For more information on configuring client authentication, see “Configure Client Authentication” on page 3-4. HTTPS also provides an encrypted data path between the clients and server instances.

Acquire and Copy SSL Certificate and Key

To set up HTTPS on a server instance, you must save an SSL certificate and the corresponding private key to the *<instance_root>/x509/* folder of your server instance. The SSL certificate and private key must be in PEM format.

To generate a self-signed SSL certificate, you can use the following `openssl` command:

```
openssl req -x509 -nodes -newkey rsa:4096 -keyout private_key.pem -out cert_chain.pem -days 365
```

The command generates a self-signed certificate `cert_chain.pem` with a private key `private_key.pem`. The certificate is valid for 365 days. For more information, see OpenSSL.

Self-signed SSL certificates are suitable for use in testing environments as they offer encryption but do not offer authentication. SSL certificates signed by a certificate authority (CA) are suitable for production environments.

Edit Configuration File

To configure HTTPS, specify the following properties in the `main_config` configuration file of the server instance:

- https: HTTPS port
- x509-cert-chain: Valid certificate in a PEM-format certificate chain
- x509-private-key: Valid private key in PEM format

When you set the `https` property on the server, you must set both the `x509-cert-chain` and `x509-private-key` properties; otherwise, the server fails to start. For more information about the server configuration file, see “Configure Server Using Configuration File” on page 1-6.

The following configuration excerpt configures a server instance to accept secure connections on port *port*, using the certificate stored in `./x509/cert_chain.pem` and the unencrypted private key stored in `./x509/private_key.pem`.

```
...
--https port
--x509-cert-chain ./x509/cert_chain.pem
--x509-private-key ./x509/private_key.pem
...
```

In production settings that require greater security than that provided by an unencrypted private key, use an encrypted private key. You specify the passphrase for decrypting the private key in a file with owner-read-only access, and use the x509-passphrase property to tell the server instance about it.

```
...
--https port
--x509-cert-chain ./x509/cert_chain.pem
--x509-private-key ./x509/private_key.pem
--x509-passphrase ./x509/key_passphrase
...
```

You must set either the http property, the `https` property or both properties for the server to start. To ensure that clients communicate with the server using only HTTPS and not HTTP, you must disable the `http` property. If both the `https` and `http` properties are enabled, clients can communicate with the server using both HTTPS and HTTP. It is recommended that you enable the `https` property unless HTTP support is required.

See Also

ssl-protocols | client-credential-delegation | ssl-tmp-ec-param

More About

- “Configure Client Authentication” on page 3-4
- “Specify Access to MATLAB Programs” on page 3-5
- “Adjust Security Protocols” on page 3-6

Configure Client Authentication

To ensure that only trusted client applications have access to a MATLAB Production Server instance that uses HTTPS, configure the server instance to require client authentication by setting the following properties in the `main_config` server configuration file:

1. Set the ssl-verify-peer-mode configuration property to `verify-peer-require-peer-cert`.
2. Configure the server instance to use the system provided certificate authority (CA) store, a server specific CA store, or both.

   Use these configuration properties to control the CA stores used by the server instance:

   - x509-ca-file-store specifies a PEM-format CA store to authenticate clients.
   - x509-use-system-store directs the server instance to use the system CA store to authenticate clients.

   Note `x509-use-system-store` does not work on Windows.
3. Optionally configure the server instance to respect any certificate revocation lists (CRLs) in the CA store.

   Specify this behavior by setting the x509-use-crl property in the server configuration. If you do not set this property, the server instance ignores the CRLs and may authenticate clients using revoked credentials.

   Caution You must add a CRL list to the CA store of the server before adding the `x509-use-crl` property. If the CA store does not include a CRL list, the server crashes.

This configuration excerpt configures a server instance to authenticate clients using the system CA store and to respect CRLs:

```
...
--https port
--x509-cert-chain ./x509/my-cert.pem
--x509-private-key ./x509/my-key.pem
--x509-passphrase ./x509/my-passphrase
--ssl-verify-peer-mode verify-peer-require-peer-cert
--x509-use-system-store
--x509-use-crl
...
```

The server must be configured to use HTTPS in order to configure client authentication.

See Also

https | x509-cert-chain | x509-use-crl | x509-private-key | ssl-verify-peer-mode | x509-use-system-store | x509-ca-file-store | enable-ssl-renegotiation

More About

- “Enable HTTPS” on page 3-2
- “Adjust Security Protocols” on page 3-6
- “Specify Access to MATLAB Programs” on page 3-5

Specify Access to MATLAB Programs

By default, server instances allow all clients to access all hosted MATLAB programs. MATLAB Production Server provides a certificate-based authorization mechanism for restricting access to specific programs. The ssl-allowed-client property uses this mechanism to specify the MATLAB programs that a client can access. The property specifies a comma-separated list of clients, identified by their certificate's common name, that are allowed to access MATLAB programs. You also use the property to list specific MATLAB programs that a client is allowed to access.

If you do not specify the `ssl-allowed-client` property, the server instance does not restrict access to the hosted MATLAB programs. After you add an entry for the `ssl-allowed-client` property, the server instance authorizes only the listed clients to access the hosted MATLAB programs.

For example, to only authorize clients with the common names `jim`, `judy`, and `ash` to use the MATLAB programs hosted on a server instance, add this configuration excerpt:

```
--ssl-allowed-client jim,judy,ash
```

You can restrict access further by only authorizing specific clients to have access to specific MATLAB programs. Do this by adding `:`*`allowedPrograms`* to the value of the `ssl-allowed-client` property. *`allowedPrograms`* is a comma-separated list of program names.

For example, to allow clients with the common name `jim` access to all hosted programs, allow clients with the common name `judy` access to the programs `tail` and `zap`, and allow clients with the common name `ash` or `joe` access to the programs `saw` and `travel`, add this configuration excerpt:

```
--ssl-allowed-client jim
--ssl-allowed-client judy:tail,zap
--ssl-allowed-client ash,joe:saw,travel
```

The server must be configured to use HTTPS in order to use the property.

See Also

https | x509-cert-chain | x509-private-key

More About

- "Enable HTTPS" on page 3-2
- "Configure Client Authentication" on page 3-4
- "Adjust Security Protocols" on page 3-6

Adjust Security Protocols

The default security settings for MATLAB Production Server enable all security protocols and cipher suites, except for the eNULL cipher suite. Use the ssl-protocols and ssl-ciphers properties to adjust the level of security.

By default, MATLAB Production Server instances try to use TLSv1.2 to secure connections between client and server. The server supports connections using TLSv1, TLSv1.1, and TLSv1.2. Use the `ssl-protocols` property to specify a list of allowed SSL protocols.

For example, to disable the TLSv1.1 and TLSv1.2 protocols, add this configuration excerpt:

```
--ssl-protocols TLSv1
```

Because TLSv1.1 and TLSv1.2 are not included in the list, the server instance does not enable the protocols.

Set the `ssl-ciphers` property in the server instance configuration to restrict the cipher suites used by the server instance.

For example, to enable only high-strength cipher suites, add this configuration excerpt:

```
--ssl-ciphers HIGH
```

See Also

ssl-tmp-ec-param | x509-private-key

More About

- “Enable HTTPS” on page 3-2
- “Configure Client Authentication” on page 3-4
- “Specify Access to MATLAB Programs” on page 3-5

Improve Startup Time When Security Is Activated

When a server instance is configured to use HTTPS, it generates an ephemeral DH key at startup. Generating the DH key at startup provides more security than reading it from a file on disk. However, this can add a couple of minutes to a server instance's startup time.

If you need the server instance to start up without delay and are not concerned about the loss of security, you can configure the server instance to read the ephemeral DH key from a file using the ssl-tmp-dh-param configuration property. The `ssl-tmp-dh-param` property specifies the file storing the DH key in PEM format.

See Also

ssl-tmp-ec-param | https | ssl-ciphers

More About

- "Enable HTTPS" on page 3-2
- "Configure Client Authentication" on page 3-4
- "Specify Access to MATLAB Programs" on page 3-5
- "Adjust Security Protocols" on page 3-6

Application Access Control

MATLAB Production Server integrates with OAuth2 providers such as Azure Active Directory (Azure AD) and PingFederate®, and uses JSON Web Tokens (JWTs) to restrict user access to applications or archives deployed to the server. Application access control is available only for clients that use the MATLAB Production Server "RESTful API for MATLAB Function Execution" to communicate with the server. Clients must send an access token in the HTTP authorization header when they make a request to the server. The format for this header is `Authorization:Bearer <access token>`. To set up access control for an on-premises server instance, you must define an access control configuration file and an access control policy file, and set the access-control-provider property to `OAuth2` in the `main_config` server configuration file.

Access Control Configuration File

To help verify the identity of each user that sends requests to an on-premises server instance, the server administrator must define an access control configuration file in JSON format. The server uses the parameters in the access control configuration file to validate the JWT that a client request contains. The default path and name for the configuration file is `./config/jwt_idp.json`. You can change the path and name of the configuration file by setting the access-control-config property. If you change the path, name, or contents of the configuration file, you must restart the server for the changes to take effect.

The access control configuration file contains the following parameters:

Parameter	Required or Optional	Description
version	Required	Version number of the configuration file schema. The version number is a JSON string that has the format *`major#.minor#.patch#`*, where each number is a nonnegative integer. Set `version` to `1.0.0`.
jwtIssuer	Required	JWT issuer metadata of the identity provider. Specify the metadata as a JSON string. The metadata string must match the `iss` claim in the JWT. For example, for Azure AD, set the `jwtIssuer` to `https://sts.windows.net/`*`Azure AD tenant id/`*.

Parameter	Required or Optional	Description
appId	Required	Intended recipient of the JWT. Specify the recipient as a JSON string. The recipient helps in validating the `aud` claim in the JWT. For example, for Azure AD, the `appId` is the application ID of the registered MATLAB Production Server server app. For information on registering apps on Azure, see Quickstart: Register an application with the Microsoft identity platform.
jwksUri	Required	URI to retrieve the JSON Web Key Set (JWKS). Specify the URI as a JSON string. The JWKS stores public keys that are used to verify the JWT. For example, for Azure AD, set the `jwksUri` to `https://login.microsoftonline.com/common/discovery/keys`.
jwksStrictSSL	Optional	Flag to verify the peer certificate of the server specified by jwksUri during HTTPS communication with the JWKS server. Specify the choice as a JSON boolean. The default selection is `true`. Specifying `true` verifies the peer certificate. If the JWKS server uses a self-signed certificate, set this value to `false`.
jwksTimeOut	Optional	Maximum time allowed for a request to retrieve the JWKS. Specify the time as nonnegative JSON integer. The default timeout value is 120 seconds.
userAttributeName	Optional	JWT claim name that uniquely identifies a user. Specify the claim name as a JSON string. The default value for `userAttributeName` is `sub`. For Azure AD, set `userAttributeName` to `upn`.

Parameter	Required or Optional	Description
groupAttributeName	Optional	JWT claim name that lists the groups that a user belongs to. Specify the claim name as a JSON string. The default value for `groupAttributeName` is `groups`.

You can specify the `userAttributeName`, the `groupAttributeName`, or both. If you want to configure access control for specific users, specify the `userAttributeName`. If you want to configure access control for groups of users, specify the `groupAttributeName`.

An example of a configuration file for which Azure AD is the identity provider follows.

```
{
  "version": "1.0.0",
  "jwtIssuer": "https://sts.windows.net/54ss4lk1-8428-7256-5fvh-d5785gfhkjh6/",
  "appId": "j21n12bg-3758-3r78-v25j-35yj4c47vhmt",
  "jwksUri": "https://login.microsoftonline.com/common/discovery/keys",
  "jwksStrictSSL": true,
  "jwksTimeOut": 120,
  "userAttributeName": "upn",
  "groupAttributeName": "groups"
}
```

Access Control Policy File

To set up application access control for MATLAB Production Server, the server administrator must define an access control policy file in JSON format. The default path and name for the policy file is `./config/ac_policy.json`. You can change the default values by setting the access-control-policy property.

When the server starts, it tries to read the policy file. If the file does not exist or contains errors, the server does not start, and writes an error message to the `main.log` file present in the directory that the log-root property specifies.

After the server starts, it scans the policy file every five seconds, if application access control is enabled. If the policy file is absent or contains errors, the server continues to run, but denies all requests and writes an error message to the `main.log` file.

The policy file has a single JSON object that defines the schema version and a *policy block*. The policy block consists of a list of policies. Each policy contains a *rule block* that defines a set of rules for a policy. The rule block consists of a *subject block*, a *resource block*, and an *action block*.

The schema version is a JSON string in the format *major#.minor#.patch#*, where each number is a nonnegative integer. Set the schema version to `1.0.0`.

Policy Block

The policy block contains a list of policies required for application access control. Currently, the policy file supports specifying only a single policy.

```
"policy" : [
   {
     "id": "policy_id",
     "description": "policy_description",
     <rule_block>
   }
]
```

Each policy requires an `id` value. *`policy_id`* is a JSON string and must be unique for each policy. Any leading or trailing white spaces are removed.

The `description` is optional for a policy.

Rule Block

The rule block contains a list of rule objects.

```
"rule":[
  {
    "id": "rule_id",
    "description": "rule_description",
    <subject_block>,
    <resource_block>,
    <action_block>
  }
]
```

The rule block supports multiple rules, for example, `"rule": [<rule>, <rule>, ...]`.

Each rule requires an `id` value. *`rule_id`* is a JSON string and must be unique for each rule. Any leading or trailing white spaces are removed.

The `description` is optional for a rule.

Subject Block

The subject block of a rule defines who can access the resources. You can control access for a list of users, user groups, or both.

```
"subject" : {"groups": ["group_id", "group_id", ...],
             "users": ["user_id", "user_id", ...]}
```

Use the `users` attribute to specify a list of users identified by their unique *`user_id`* values. Use the `groups` attribute to specify user groups identified by their unique *`group_id`* values. You can specify the `users` attribute, `groups` attribute, or both.

Resource Block

The resource block of a rule specifies the resource that a server request wants to access. Currently, the only resource that a server request can access is a deployable archive (CTF file). You can specify multiple deployable archives.

```
"resource" : {"ctf": ["archive_name", "archive_name", ...]}
```

You can use the wildcard character * to specify multiple archives. For example, if you want to access all archives whose names start or end with `test`, specify *`archive_name`* as `test`* or *`test`, respectively. Use * as the *`archive_name`* to access all archives deployed to the server.

Action Block

The action block of a rule describes the action that a server request wants to perform on the resource. Currently, `execute` is the only supported action.

```
"action" : ["execute"]
```

Example of JSON Policy File

The following example defines an access control policy with three rules.

- Users `aaa@xyz.com` and `bbb@xyz.com` can execute the deployable archive `magic`.
- All users that belong to groups with IDs `aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa` and `bbbbbbbb-bbbb-bbbb-bbbb-bbbbbbbbbbbb`, as well as user `ccc@xyz.com`, can execute the deployable archives `monteCarlo` and `fastFourier`.
- All users that belong to quality engineering group `cccccccc-cccc-cccc-cccc-cccccccccccc` can execute all deployable archives that have names that start with `test`.

The server denies access for all other requests.

```
{
  "version": "1.0.0",
  "policy" : [
    {
      "id": "policy1",
      "description": "Access Control policy for XYZ Corp.",
      "rule": [
        {
          "id": "rule1",
          "description": "Users aaa@xyz.com and bbb@xyz.com can execute deployable archive magic",
          "subject": { "users": ["aaa@xyz.com", "bbb@xyz.com"] },
          "resource": { "ctf": ["magic"] },
          "action": ["execute"]
        },
        {
          "id": "rule2",
          "description": "Group A and B and user ccc@xyz.com can execute deployable archives monteCarlo and fastFourier",
          "subject": { "groups": ["aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa", "bbbbbbbb-bbbb-bbbb-bbbb-bbbbbbbbbbbb"],
                       "users": ["ccc@xyz.com"]  },
          "resource": { "ctf": ["monteCarlo", "fastFourier"] },
          "action": ["execute"]
        },
        {
          "id": "rule3",
          "description": "QE group C can execute any deployable archive whose name starts with test",
          "subject": { "groups": ["cccccccc-cccc-cccc-cccc-cccccccccccc"] },
          "resource": { "ctf": ["test*"] },
          "action": ["execute"]
        }
      ]
    }
  ]
}
```

See Also

access-control-policy | access-control-config | access-control-provider

External Websites

- Microsoft Graph API

Use Kerberos Authentication and Kerberos Delegation

To authenticate user access to a MATLAB Production Server instance, you need to configure Kerberos. To delegate a client's credential to a next hop web server or a database server that is protected by Kerberos, you need to configure Kerberos delegation. Configuring Kerberos and Kerberos delegation require domain administrator privileges.

- You can use Kerberos authentication and Kerberos delegation with MATLAB Production Server instances running on Windows Server® operating systems with a Windows Key Distribution Center (KDC).
- You can use only Kerberos authentication with MATLAB Production Server instances running on Linux operating systems. For Linux server instances, you can use a Windows KDC or the Linux-based MIT Kerberos 5 KDC.

To configure Kerberos authentication and Kerberos delegation, consult your IT or system administrator.

Kerberos Authentication and Kerberos Delegation Setup on Windows Systems

Configure Kerberos Authentication

- Set up a service account for the MATLAB Production Server and register a *service principal name* for MATLAB Production Server service instance.
- Enable Kerberos in the MATLAB Production Server configuration file (`main_config`). For more information, see http-authentication-method.

Configure Kerberos Delegation

- Set up a service account for the MATLAB Production Server and register a *service principal name* for MATLAB Production Server service instance.
- Configure constrained delegation without protocol transition for the service account.
- Configure the local security privilege for the MATLAB Production Server service account.
- Enable Kerberos authentication and Kerberos delegation in the MATLAB Production Server configuration file (`main_config`). For more information, see http-authentication-method and client-credential-delegation.

Only the following MATLAB functions within a deployable archive (CTF file) support using Kerberos delegation:

- `webread`
- `webwrite`
- "Use HTTP with MATLAB" (MATLAB) functions
- Database Toolbox™ functions (requires an ODBC driver)

Note If you use persistent database connections when using Kerberos delegation on a MATLAB Production Server instance that uses Database Toolbox functions, the credentials of the user that opens the connection are used for every subsequent database request, regardless of the user making the requests.

All other functions within a deployable archive (CTF file) are executed using the credential of the MATLAB Production Server instance.

Supported Environment

Option	Requirement
Operating system	Windows Server
Key distribution center	Windows Server 2003 or later
Client	• RESTful client over HTTP/HTTPS (HTTP 1.1) with JSON payload • The RESTful client must be one that supports SPNEGO/Kerberos—for example, `curl` with the `--negotiate` option or .NET `HttpClient`
Kerberos delegation	Constrained delegation without protocol transition
MATLAB Runtime	MATLAB Runtime R2019b or later
Deployable archive packaging	MATLAB Compiler SDK R2019b or later
Database server	Microsoft SQL Server® 2012 or later
Database driver	Microsoft SQL Server ODBC driver version 11 or later

Kerberos Setup on Linux Systems

- Set up a service account for the MATLAB Production Server and register a service principal name for MATLAB Production Server service instance.
- Create a keytab for the MATLAB Production Server service ticket principal. For more information, see the MIT Kerberos documentation for keytab, kvno, and ktutil.
- Create a credential cache for the MATLAB Production Server service ticket principal. For more information, see the MIT Kerberos documentation for credential cache and kinit.
- Set the environment variable `KRB5_KTNAME` to point to the keytab location.
- Enable Kerberos in the MATLAB Production Server configuration file (`main_config`). For more information, see http-authentication-method.

Supported Environment

Option	Requirement
Operating system	For supported Linux distributions, see System Requirements for MATLAB.
Key distribution center	• Windows Server 2003 or later for a KDC • MIT Kerberos 5 KDC for a Linux KDC

Option	Requirement
Client	• RESTful client over HTTP/HTTPS (HTTP 1.1) with JSON payload • The RESTful client must be one that supports SPNEGO/Kerberos—for example, `curl` with the `--negotiate` option or .NET `HttpClient`

See Also

http-authentication-method | client-credential-delegation

More About

- “Enable HTTPS” on page 3-2

External Websites

- MIT Kerberos Documentation

Troubleshooting

- “Verify Server Status” on page 4-2
- “Diagnose a Server Instance” on page 4-4
- “Diagnose a Corrupted MATLAB Runtime” on page 4-5
- “Server Diagnostic Tools” on page 4-6
- “Manage Log Files” on page 4-7
- “Common Error Messages and Resolutions” on page 4-9

Verify Server Status

In this section...
“Procedure” on page 4-2 “License Server Status Information” on page 4-3

Use the `mps-status` command to verify the status of an on-premises MATLAB Production Server instance. You can use `mps-status` after issuing commands such as `mps-start`, `mps-stop`, and `mps-restart` to check if the server has started or stopped. The output of `mps-status` contains information about the status of the MATLAB Production Server instance and also information about the license server that the MATLAB Production Server instance uses.

Procedure

1. Open a system command prompt.
2. Enter the following command:

   ```
   mps-status [-C path/]server_name
   ```

   where:

   - `-C` *`path/`* — Path to the server instance. *`path`* should end with the name of the server to be queried for status.
   - *`server_name`* — Name of the server to be queried for status.

Example

To verify the status of a server instance `prod_server_1` located at `\tmp\prod_server_1`, type the following at the system command prompt:

```
mps-status -C \tmp\prod_server_1
```

Depending on whether the server is able to check out a valid license, `mps-status` returns different responses.

- If `prod_server_1` is running and operating with a valid license, it returns the following response.

  ```
  \tmp\prod_server_1 STARTED
  License checked out
  ```

- If `prod_server_1` is unable to check out a valid license, the server returns either a warning when it is within the license checkout grace period or returns an error if it is past the grace period. For details, see `license-grace-period`.

  ```
  \tmp\prod_server_1 STARTED
  WARNING: lost connection to license server -
  request processing will be disabled at 2019-Jun-27
  15:40:31.002137 Eastern Daylight Time unless
  connection to license server is restored.
  ```

  or

  ```
  \tmp\prod_server_1 STARTED
  ERROR: lost connection to license server -
  request processing disabled.
  ```

License Server Status Information

In addition to the status of the MATLAB Production Server instance, `mps-status` also displays the status of the license server associated with the server you are querying. The following table lists different license server status messages from the output of `mps-status`.

License Server Status Message	Message Description
`License checked out`	The server is operating with a valid license. The server is communicating with the License Manager, and is able to check out the required number of license keys.
`WARNING: lost connection to license server - request processing will be disabled at` *`time`* `unless connection to license server is restored`	The server has lost communication with the License Manager, but the server is still fully operational and will remain operational until the specified *`time`*. At *`time`*, if the server is still unable to connect to the license server, the server will be unable to process requests until licensing is reestablished.
`ERROR: lost connection to license server - request processing disabled`	The server has lost communication with the License Manager for a period of time exceeding the grace period. See `license-grace-period`. Request processing is suspended, but the server is actively attempting to reestablish communication with the License Manager. Request processing resumes if the sever is able to reestablish communication with the License Manager.

See Also

`mps-status` | `mps-stop` | `mps-restart` | `mps-start` | `license-grace-period`

More About

- “Health Check”

Diagnose a Server Instance

To diagnose a problem with a server instance or configuration of MATLAB Production Server, do the following, as needed:

- Check the logs for warnings, errors, or other informational messages.
- Check Process Identification Files (PID files) for information relating to problems with MATLAB Runtime worker processes.
- Check Endpoint Files for information relating to problems relating to the server's bound external interfaces — for example, a problem connecting a client to a server.
- Use server diagnostic tools, such as `mps-which`, as needed.

Diagnose a Corrupted MATLAB Runtime

This example shows a typical diagnostic procedure you might follow to solve a problem starting server `prod_server_x`.

After you issue the command:

```
mps-start prod_server_x
```

from within the server instance folder (`prod_server_x`), you get the following error:

```
Server process exited with return code: 4
(check logs for more information)
Error while waiting for server to start: The I/O operation
has been aborted because of either a thread exit
or an application request
```

To solve this issue, you might check the `log` files for more detailed messages, as follows:

1. Navigate to the server instance folder (`prod_server_x`) and open the `log` folder.
2. Open `main.err` with any text editor. Note the following message listed under `Server startup error:`

   ```
   Dynamic exception type: class std::runtime_error
   std::exception::what: bad MATLAB Runtime installation:
   C:\Program Files\MATLAB\MATLAB Runtime\v82
   (C:\Program Files\MATLAB\MATLAB Runtime\v82\bin\
   win64\mps_worker_app could not be found)
   ```
3. The message indicates the installation of the MATLAB Runtime is incomplete or has been corrupted. To solve this problem, reinstall the MATLAB Runtime.

Server Diagnostic Tools

In this section...
“Log Files” on page 4-6 “Process Identification Files (PID Files)” on page 4-6 “Endpoint Files” on page 4-6

Log Files

Each server writes a log file containing data from both the main server process, as well as the workers, named *`server_name`*`/log/main.log`. You can change the primary log folder name from the default value (`log`) by setting the option log-root in the `main_config` server configuration file.

The primary log folder contains the `main.log` file, as well as a symbolic link to this file with the auto-generated name of `main_`*`date_fileID`*`.log`.

The `stdout` stream of the main server process is captured as `log/main.out`.

The `stderr` stream of the main server process is captured as `log/main.err`.

For information on viewing logs for a server deployment in the cloud, see “View Logs” on page 9-30.

Process Identification Files (PID Files)

In an on-premise MATLAB Production Server installation, each process that the server runs generates a Process Identification File (PID File) in the folder identified as pid-root in `main_config`.

The main server PID file is `main.pid`; for each MATLAB Runtime worker process, it is `worker-`*`n`*`.pid`, where *`n`* is the unique identifier of the worker.

PID files are automatically deleted when a process exits.

Endpoint Files

In an on-premises MATLAB Production Server installation, endpoint files are generated to capture information about the server’s bound external interfaces. The files are created when you start a server instance and deleted when you stop it.

`server_name``/endpoint/http` contains the IP address and port of the clients connecting to the server. This information can be useful in the event that zero is specified in `main_config`, indicating that the server bind to a free port.

See Also

profile | `mps-profile`

More About

- “Configure Server Using Configuration File” on page 1-6

Manage Log Files

MATLAB Production Server generates several log files when it starts. You can set server configuration properties to specify where and when to store and archive logs, as well as the level of details to log.

Best Practices for Log Management

- Avoid placing log files and archived log files on different physical file systems.
- Place log files on local drives, not on network drives.
- Send MATLAB output to `stdout`. Develop an appropriate, consistent logging strategy following MATLAB coding best practices.

Log Retention and Archive Settings

The `main_config` server configuration file of a server instance contains properties that you can set to manage log files. The `main.log` file contains logs of server and worker processes for as long as a specific server instance is active, or until midnight.

- Specify the location of `main.log` by setting the log-root property in `main_config`.
- When you restart the server, log data is written to an archive log. Specify the location of the archive log folder by setting the log-archive-root property in `main_config`.

You can specify when a server instance archives `main.log` by setting the following properties in `main_config`:

- log-rotation-size — When `main.log` reaches this size, the active log is written to an archive log (located in the folder specified by `log-archive-root`).
- log-archive-max-size — When the combined size of all files in the archive folder (location defined by `log-archive-root`) reaches this limit, archive logs are purged until the combined size of all files in the archive folder is less than `log-archive-max-size`. Oldest archive logs are deleted first.

Specify values for these properties using the following units and notations:

Unit of Measurement	Notation	Example
Byte	`b`	`900b`
Kilobyte (1024 bytes)	`k`	`700k`
Megabytes (1024 kilobytes)	`m`	`40m`
Gigabytes (1024 megabytes)	`g`	`10g`
Terabytes (1024 gigabytes)	`t`	`2t`
Petabytes (1024 terabytes)	`p`	`1p`

Note The minimum value you can specify for `log-rotation-size` is 1 megabyte.

On Windows 32-bit systems, values larger than 2^{32} bytes are not supported. For example, specifying `5g` is not valid on Windows 32-bit systems.

Log Detail Levels

Log levels provide different levels of information for troubleshooting. Set log levels using the log-severity server configuration property. `log-severity` supports the following values:

- `error` — Notification of problems or unexpected results
- `warning` — Events that might lead to problems if unaddressed
- `information` — High-level information about major server events
- `trace` — Detailed information about the internal state of the server

Before you call technical support, set the log level to `trace`.

See Also

log-root | log-rotation-size | log-archive-max-size | log-archive-root | log-severity

Common Error Messages and Resolutions

(404) Not Found

This error is commonly caused when a client requests a component that is not deployed on the server, or tries to call a function that is not exported by the given component.

To resolve this error, verify that the name of the deployable archive specified in your `Uri` is the same as the name of the deployable archive hosted in the `auto_deploy` folder of your server instance.

Error: Bad MATLAB Runtime Instance

This error is commonly caused when you do not specify the correct path to the MATLAB Runtime. You must include the MATLAB Runtime version number in the path. For example, you need to specify:

`C:\Program Files\MATLAB\MATLAB Runtime\v`*`n.n`*

not

`C:\Program Files\MATLAB\MATLAB Runtime`

Error: Server Instance not Specified

This error is caused when the server instance folder cannot be located.

To resolve this error, ensure that you enter commands either from the folder containing the server instance, or use the `-C` argument to specify the location of the server instance.

For example, if you create `server_1` in `C:\tmp\server_1`, run the `mps-start` command from within that folder to avoid specifying a path with the `-C` argument:

```
cd c:\tmp\server_1
mps-start server_1
```

For more information, see “Start Server Instance Using Command Line” on page 1-16.

Error: invalid target host or port

This error is caused when the port number specified has not been properly defined on your computer. To resolve the error, define a valid port and retry the command.

Error: HTTP error: HTTP/x.x 404 Component not found

This error can be caused by a number of reasons. For more information about possible causes, see “Log Files” on page 4-6.

Server failed to start (error code = 16): license checkout failed

This error indicates that either the license manager is not running, or the server instance cannot locate the license file or the license server.

To resolve the issue, if your setup uses a license manager, ensure that it is running. Ensure that the license property in the `main_config` server configuration file is set. The `license` property requires either a full path to the license file or a path to the license server.

See Also

More About

- “Manage Log Files” on page 4-7
- “Verify Server Status” on page 4-2

5

Impact of Server Configurations on Processing Asynchronous Requests

Impact of Server Configurations on Processing Asynchronous Requests

MATLAB Production Server supports asynchronous execution of client requests. The following configurations in the server's `main_config` file impact the how the server supports this functionality:

- request-timeout
- server-memory-threshold
- server-memory-threshold-overflow-action

The `request-timeout` configuration parameter specifies the duration after which a request in a terminal states times out and gets deleted.

The `server-memory-threshold` configuration parameter specifies the size threshold of the server process at which point action needs to be taken to manage the responses. The size threshold includes both the size of the base server process plus any growth in the server process resulting from processing a client request.

The `server-memory-threshold-overflow-action` configuration parameter specifies the action to be taken when the memory size threshold of server process has been breached. The possible actions are that the responses be archived to disk or the request be purged.

Setting too small a `request-timeout` can lead to a request being timed out before a client fetches the response.

Since the `server-memory-threshold` includes both the size of the base server process plus any growth in the server process resulting from processing client requests, setting too small a `server-memory-threshold` can lead to responses being archived or purged before being retrieved.

Since the operating system governs memory management, the memory footprint size of the base server process may not return to its original size even after a response has been archived or purged. The size of the base server process in most cases ends up being larger than its original size. As a result, subsequent requests to the server may have a much smaller range of memory to work with before reaching the `server-memory-threshold`.

Setting the `server-memory-threshold` to be too large will result in a large server process footprint which may not be required.

These configuration parameters need to be set appropriately and carefully balanced in order to provide a suitable contract between a client and a server.

See Also

cors-allowed-origins | response-archive-root | response-archive-limit

More About

- "Configure Server Using Configuration File" on page 1-6

Set Up MATLAB Production Server Dashboard

- “Set Up and Log In to MATLAB Production Server Dashboard” on page 6-2
- “Remove MATLAB Production Server Dashboard” on page 6-6

Set Up and Log In to MATLAB Production Server Dashboard

In this section...
“Set Up Dashboard” on page 6-2 “Start Dashboard” on page 6-3 “Log In to Dashboard” on page 6-4 “Reset Administrator Password” on page 6-4

Follow these instructions to set up, start, and log in the dashboard for an on-premises installation of MATLAB Production Server. You can use the dashboard to create and manage server instances.

Set Up Dashboard

Warning You must have administrator privileges on Windows to complete the setup.

1. Open a Terminal or Command Window with administrator privileges, and navigate to the `dashboard` directory in the MATLAB Production Server installation directory.

Platform	Default Directory Where MATLAB Production Server Dashboard Is Installed
Windows *(Administrator)*	`C:\Program Files\MATLAB\MATLAB Production Server\R2023a\dashboard`
Linux	`/usr/local/MATLAB/MATLAB_Production_Server/R2023a/dashboard`

2. Execute the `mps-dashboard` script with the `setup` option, and when prompted, specify the directory for dashboard setup.

   You must have write privileges to the directory where MATLAB Production Server is installed and to the workspace directory where the dashboard is going to be set up.

Platform	Script for Dashboard Setup

Windows *(Administrator)*	`> mps-dashboard.bat setup`
	For example:
	`> mps-dashboard.bat setup`
	`Specify a workspace directory for MATLAB Production Server Dashboard: C:\mps\dashboard`

Linux	`$ ./mps-dashboard.sh setup`
	For example:
	`$ ./mps-dashboard.sh setup`
	`Specify a workspace directory for MATLAB Production Server Dashboard: /opt/mps/dashboard`

   You receive a message acknowledging that the dashboard has been successfully setup.

   Tip To directly specify a directory when setting up the dashboard, use the `-C` option after the `setup` option and provide a directory name.

For example, at the Windows command prompt, type: `mps-dashboard.bat setup -C D:\mps\dashboard`

For example, in the Linux terminal, enter: `./mps-dashboard.sh setup -C /opt/mps/dashboard`

Note To view the list of options that the `mps-dashboard` script accepts, pass a `?` as an option to the `mps-dashboard` script.

For example, at the Windows command prompt, type:

```
mps-dashboard.bat ?
```

For example, in the Linux terminal, type:

```
./mps-dashboard.sh ?
```

The complete list of options are as follows:

```
setup | start | stop | remove | reset_admin_password
```

Tip If you see garbled text in the dashboard logs, verify that the server machine uses UTF-8 encoding. You must execute `mps-dashboard.bat setup` again after setting the locale.

For information about setting the locale, see “Set Locale on Microsoft Windows Platforms” (MATLAB) and “Set Locale on Linux Platforms” (MATLAB).

Start Dashboard

1. Open a Terminal or Command Window with a user account that does not have administrator privileges, then navigate to the `dashboard` directory in the MATLAB Production Server installation directory.

   To start the dashboard, it is recommended that you run the run the `mps-dashboard` script with a least privilege user account that has write privileges to the MATLAB Production Server Dashboard workspace directory.

Platform	Default Directory Where MATLAB Production Server Dashboard Is Installed
Windows	`C:\Program Files\MATLAB\MATLAB Production Server\R2023a\dashboard`
Linux	`/usr/local/MATLAB/MATLAB_Production_Server/R2023a/dashboard`

2. Execute the `mps-dashboard` script with the `start` option to start the dashboard.

Platform	Script to Start Dashboard
Windows	`mps-dashboard.bat start`
Linux	`./mps-dashboard.sh start`

   After the dashboard starts, you see a message at the terminal indicating the host and port where the dashboard is running. The default host and port are `localhost` and `9090`, respectively.

Tip Windows only: To run the dashboard instance as a background process in Windows, precede the `mps-dashboard` script with command `start /B`.

For example: `start /B mps-dashboard.bat start`

Note You can change the default port used by dashboard by editing the `--node_server_port` option in `config.txt` file. You can find the `config.txt` file here:

Platform	Location of `config.txt` File
Windows	`C:\Program Files\MATLAB\MATLAB Production Server\R2023a\dashboard\config\config.txt`
Linux	`/usr/local/MATLAB/MATLAB_Production_Server/R2023a/dashboard/config/config.txt`

To make other customizations to the setup process, you can edit the relevant parts of the `config.txt` file.

3. Open a web browser, and type the host and port number that were displayed in the previous step.

   For example:

   ```
   http://localhost:9090
   ```

Log In to Dashboard

To log in to MATLAB Production Server Dashboard follow this procedure:

1. Open a web browser, and type the host and port number that were displayed at the end of the install process.

   For example:

   ```
   http://localhost:9090
   ```

2. Type the following information at the login screen for the username and password:

   Username: `admin`

   Password: `admin`

You are now logged into the MATLAB Production Server Dashboard.

Reset Administrator Password

You can use the `mps-dashboard` script with the option `reset_admin_password` to change the administrator password.

Platform	Script to Reset Administrator Password
Windows	`mps-dashboard.bat reset_admin_password`
Linux	`./mps-dashboard.sh reset_admin_password`

Warning Do not execute the `reset_admin_password` option while the dashboard is still running. First, stop dashboard execution using the `mps-dashboard` script with the `stop` option, then reset the administrator password.

See Also

Related Examples

- “Remove MATLAB Production Server Dashboard” on page 6-6

Remove MATLAB Production Server Dashboard

Follow these steps to remove the MATLAB Production Server dashboard in an on-premises server installation.

1. Open a Terminal or Command Window, and navigate to the `dashboard` folder in the MATLAB Production Server installation directory.

Platform	Default Directory Where MATLAB Production Server Dashboard Is Installed
Windows *(Administrator)*	`C:\Program Files\MATLAB\MATLAB Production Server\R2023a\dashboard`
Linux	`/usr/local/MATLAB/MATLAB_Production_Server/R2023a/dashboard`

2. Execute the `mps-dashboard` script with the `stop` option.

Platform	Script to Stop Dashboard
Windows *(Administrator)*	`mps-dashboard.bat stop`
Linux	`./mps-dashboard.sh stop`

   Note You need to complete this step only if the dashboard is running.

3. Execute the `mps-dashboard` script with the `remove` option.

Platform	Script to Remove Dashboard
Windows *(Administrator)*	`mps-dashboard.bat remove`
Linux	`./mps-dashboard.sh remove`

   You receive a message acknowledging that the dashboard was successfully removed.

   Note Attempting to remove the dashboard while it is still running will result in an error.

The procedure will remove the following directories and files from the directory where the dashboard was set up:

```
data
mps_workspace
.pid
```

If you run into any issues while removing the dashboard, manually delete the `.pid` file and re-run the `mps-dashboard` script with the `remove` option.

Note In Linux, if you started the dashboard using the & control operator, you do not need to open a new Terminal. The & control operator makes commands run in the background.

In Windows, if the dashboard is running, you will not have access to the command prompt. Therefore, you need to open a new Command Window to stop any running dashboard instances.

Note Removing the dashboard does not uninstall it from the system. It removes the dashboard instance that was set up. The dashboard remains installed as part of MATLAB Production Server. If you want to set up the dashboard again, use the `mps-dashboard` script with the `setup` option.

See Also

Related Examples

- “Set Up and Log In to MATLAB Production Server Dashboard” on page 6-2

Commands

mps-check

Test and diagnose server instance for problems from command line on Windows, Linux, and macOS systems

Syntax

```
mps-check host:port [--bearer-token bearer token] [--help|-h] [--timeout seconds] [--verbose|-v]
```

Description

`mps-check` sends a request to a MATLAB Production Server instance and receives a status report that you can use to identify issues that cause the product to run less than optimally.

Information reported by `mps-check` to `stdout` includes:

- Status of the server instance
- Port the HTTP interface is listening on
- Deployed archives for a server instance

Input Arguments

host*:*port

Specify the host name of the machine running the server instance and the port number on which the server instance listens for requests.

`--bearer-token` *`bearer token`*

This argument is required if application access control is enabled on the server instance. To use this argument, the access control policy file must contain the following rule to allow specific users or user groups to execute the `mps-check` command.

```
{
    "id": "<rule_id>",
    "subject": { "users": ["<user_id>"] ,
                 "groups": ["<group_id>"] },
    "resource": { "ctf": ["mps_check*"] },
    "action": ["execute"]
}
```

If you do not add the rule, you get a `403 NotAuthorized` error when the `mps-check` command tries to verify the installed MATLAB Runtime versions. For information on enabling access control and the access control policy file, see “Application Access Control” on page 3-8.

`--help | -h`

Display the list of options for using the `mps-check` command. Specifying the host name and port number is optional when you use this argument.

`--timeout` *`seconds`*

Specify the time in *`seconds`* to wait for a response from the server before timing out. The default is two minutes.

`--verbose | -v`

Display detailed system messages related to the server status report.

Examples

- Display diagnostic information for the server instance running on port 9910 of the local computer.

  Type the follow at the system command prompt:

  ```
  mps-check localhost:9910
  ```

  ```
  Checking the MATLAB Runtime version 9.9.0 at C:\Program Files\MATLAB\
  Check completed successfully
  ```

- Display diagnostic information for the server instance running on port 9910 of the local computer with the verbose option.

  Type the follow at the system command prompt:

  ```
  C:\tmp>mps-check -v localhost:9910
  ```

  ```
  -- Resolve the hostname
  -- Start the mps-check client
  -- Retrieving the list of installed MATLAB Runtimes from the MATLAB Production Server
  -- start timer 00:02:00
  -- Connecting to the peer server
  -- Conncted. Start receiving TCP data
  -- Send HTTP request
  -- Send TCP data
  -- TCP data sent succeeded
  -- HTTP request sent
  -- TCP data received
  -- HTTP/1.1 200 OK
  -- HTTP response completed
  -- Cancel timer
  -- HTTP/1.1 200 OK
  Content-Length: 126
  Connection: Keep-Alive


  -- Retrieving the MATLAB Runtime list succeeded
  -- Shutdown TCP client
  -- Socket closed
  Checking the MATLAB Runtime version 9.9.0 at C:\Program Files\MATLAB\
  -- start timer 00:02:00
  -- Connecting to the peer server
  -- Conncted. Start receiving TCP data
  -- Send HTTP request
  -- Send TCP data
  -- TCP data sent succeeded
  -- HTTP request sent
  -- TCP data received
  ```

```
-- HTTP/1.1 200 OK
-- HTTP response completed
-- Cancel timer
-- Verify http response against the input
-- Input = 25921
-- Output = 161
-- Verification succeeded
Check completed successfully
-- Shutdown TCP client
-- Socket closed

-- Terminated
```

Version History

Introduced in R2012b

See Also

`mps-profile` | `mps-status` | `mps-support-info` | `mps-which`

mps-license-reset

Force server instance to immediately attempt license checkout from command line on Windows, Linux, and macOS systems

Syntax

`mps-license-reset [-C path/]server_name`

Description

`mps-license-reset [-C path/]server_name` triggers the server to checkout a license immediately, regardless of the current license status. License keys that are currently checked out are checked in first.

Input Arguments

`-C path/`

Specify a path to the server instance. If this option is omitted, the current working folder and its parents are searched to find the server instance.

`server_name`

Server checking out license

Examples

Create a new server instance and display the status of each folder in the file hierarchy, as the server instance is created.

Type the following at the system command prompt:

```
mps-license-reset -C /tmp/server_2
```

Version History

Introduced in R2012b

See Also

`mps-status`

Topics

“Manage Licenses for MATLAB Production Server”

mps-new

Create server instance from command line on Windows, Linux, and macOS systems

Syntax

```
mps-new [path/]server_name [-v] [--service] [--service-name name] [--service-description description] [--service-user user] [--service-password password] [--noprompt]
```

Description

`mps-new [path/]server_name [-v] [--service] [--service-name name] [--service-description description] [--service-user user] [--service-password password] [--noprompt]` makes a new folder at *`path`* and populates it with the default folder hierarchy for a server instance.

Input Arguments

`path`

Path to server instance.

`server_name`

Name of the server instance to create.

If you are creating a server instance in the current working folder, you do not need to specify a full path; specify only the server name.

`-v`

Display the status of each folder in the file hierarchy created to form a server instance

`--service`

On Windows, register the server instance as a Windows service.

The Windows service default settings are:

- Service Display Name: MATLAB Production Server – *`path\server_name`*
- Service Description: MATLAB Production Server running instance *`path\server_name`*
- Service User: `LocalSystem`

The Windows service is configured to start when the machine starts, not at creation of the service. After you have made configuration changes, start the server instance using `mps-start`.

`--service-name` ***`name`***

Display name for the Windows service associated with the server instance

`--service-description` *`description`*

Informational statement describing the Windows service associated with the server instance

`--service-user` *`user`*

Windows account under which the service associated with the server instance should run. The user account must have read, write, and, delete permissions for the instance directory as well read and execute permissions for the MATLAB Production Server installation directory.

`--service-password` *`password`*

Password for the service user account

`--noprompt`

Indicates that no prompts are generated

Examples

Create Server Instance

To create a new server instance, and display the status of each folder in the file hierarchy as the server instance is created, type the following at the system command prompt:

```
mps-new /tmp/server_1 -v
```

```
server_1/.mps-version...ok
server_1/config/...ok
server_1/config/main_config...ok
server_1/endpoint/...ok
server_1/auto_deploy/...ok
server_1/.mps-socket/...ok
server_1/log/...ok
server_1/pid/...ok
```

Create Windows Service

To create a new server instance, and register it as a Windows service, type the following at the Windows command prompt:

```
mps-new /tmp/server_1 --service
```

Tips

- Before creating a server instance, ensure that no file or folder with the specified *`path`* currently exists on your system.
- After issuing `mps-new`, issue `mps-start` to start the server instance.

Version History

Introduced in R2012b

See Also

`mps-status` | `mps-start` | `mps-service`

Topics

“Create Server Instance Using Command Line” on page 1-4
“Configure Server Instance as Windows Service” on page 1-22
“Server Overview” on page 1-2

mps-profile

Log profile information for server instance from command line on Windows, Linux, and macOS systems

Syntax

```
mps-profile [-C [path/]server_name] [state] [object...]
```

Description

`mps-profile` starts or stops the logging of server profile information in the main log based on *`state`*. *`object`* specifies the information to log. You can specify multiple objects. You can log information about server requests, such as the IP address of the client, the archives requested by the client, and the worker pool.

To set up or update the profiling options for an already running server without restarting the server, use the `mps-profile` command. To set up profiling options when you configure a server, specify the profile property. Running `mps-profile` overrides any profiling options set using the `profile` property.

Note Activating profiling has a negative impact on performance.

Input Arguments

`path`

Path to server instance. If you omit this option, the current working folder and its parents are searched to find the server instance.

`server_name`

Name of the server instance to profile.

`state`

Flag to control whether the server writes profile information to the main log. Valid states are:

- `on` — Log profile information.
- `off` — Do not log profile information.

`object`

Information to log. Valid objects are:

- `server` — Information about server requests and workers.
- `server.request` — Information about server requests, which includes information about requested archives and clients that make the requests
- `server.request.archive` — Information about archives in the request

- `server.request.client` — Information about clients that make the request
- `server.worker` and `server.worker.pool` — Information about workers

The objects are hierarchical. For example, specifying `server.request` implies specifying `server.request.archive` and `server.request.client`.

If you do not specify an object, the server logs profile messages for all objects.

Examples

- Log profile information for server requests and the worker pool.

  Type either of the following at the system command prompt:

  ```
  mps-profile on
  ```

  Or:

  ```
  mps-profile on server
  ```
- Log profile information for server requests without turning on worker pool profiling.

  Type the following at the system command prompt:

  ```
  mps-profile on server.request
  ```
- Log profile information about archives in the server requests and worker pool.

  Type the following at the system command prompt:

  ```
  mps-profile on server.request.archives server.worker.pool
  ```
- Stop the logging of all profile information.

  ```
  mps-profile off
  ```
- Stop the logging of worker pool information only.

  ```
  mps-profile off server.worker.pool
  ```

The following is an excerpt of the main log that contains profiling information for all objects.

```
93 [2020.03.19 13:05:56.554236] [profile] [client:[::1]:62736] [component:mymagic] [connection_id:2]
[function:magic] [mode:sync] [request_id:0:1:1][service:http-connection] [type:request_arrive]
Request arrived and was placed in the queue.
94 [2020.03.19 13:05:56.554236] [profile]
Request to allocate next available worker
95 [2020.03.19 13:05:56.555240] [profile]
Lease created for worker-1
96 [2020.03.19 13:05:56.555240] [profile] [client:[::1]:62736] [request_id:0:1:1] [type:request_start] [worker_id:1]
Request started executing on worker 1
...
99 [2020.03.19 13:05:56.558233] [profile] [client:[::1]:62736] [request_id:0:1:1] [type:request_finish] [worker_id:1]
Request completed with HTTP status 200
100 [2020.03.19 13:05:56.558233] [profile]
Lease terminated for worker-1
101 [2020.03.19 13:05:56.558233] [profile]
worker-1 PASSED health check; returning to the pool
```

Version History

Introduced in R2015b

R2020b: `requests` and `worker_pool` objects will be removed
Not recommended starting in R2020b

The objects `requests` and `worker_pool` will be removed in a future release. Use `server.request` instead of `requests` and `server.worker.pool` instead of `worker_pool` when running this command.

See Also

profile

Topics

“Log Files” on page 4-6

mps-restart

Stop and start server instance from command line on Windows, Linux, and macOS systems

Syntax

```
mps-restart [-C [path/]server_name] [-f]
```

Description

`mps-restart [-C [path/]server_name] [-f]` stops a server instance, then restarts the same server instance. Issuing `mps-restart` is equivalent to issuing the `mps-stop` and `mps-start` commands in succession.

Input Arguments

`-C path/`

Specify a path to the server instance. If this option is omitted, the current working folder and its parents are searched to find the server instance. If you are restarting a server instance in the current working folder, you do not need to specify a full path. Only specify the server name.

`server_name`

Name of the server to be restarted.

`-f`

Force success even if the server instance is stopped. Restarting a stopped instance returns an error.

Examples

To restart a server instance named `server_1`, located in folder `tmp`, and forcing successful completion of `mps-restart`, type the following at the system command prompt:

```
mps-restart -f -C /tmp/server_1
```

Tips

- After issuing `mps-restart`, issue the `mps-status` command to verify the server instance has started.
- If you are restarting a server instance in the current working folder, you do not need to specify a full path. Only specify the server name.

Version History

Introduced in R2012b

See Also

`mps-start` | `mps-stop` | `mps-status`

mps-service

Create or modify Windows service for server instance from command line on Windows systems

Syntax

```
mps-service [-C [path\]server_name] create [--name name] [--description
description] [--user user] [--password password] [--noprompt]

mps-service [-C [path\]server_name] update [--name name] [--description
description] [--user user] [--password password] [--instance-root new_path]
[--noprompt]

mps-service [-C [path\]server_name] delete
mps-service delete service_name [[--force]|[-f]]
mps-service clean [[--force]|[-f]][[--verbose]|[-v]]

mps-service [-C [path\]server_name] undelete

mps-service [-C [path\]server_name]
mps-service list
```

Description

`mps-service [-C [path\]server_name] create [--name name] [--description description] [--user user] [--password password] [--noprompt]` creates a Windows service for the server instance.

The Windows service default settings are:

- Service Display Name: MATLAB Production Server – *`path\server_name`*
- Service Description: MATLAB Production Server running instance *`path\server_name`*
- Service User: `LocalSystem`

The Windows service is configured to start when the machine starts, not at creation of the service. After you have made configuration changes, start the server instance using `mps-start`.

`mps-service [-C [path\]server_name] update [--name name] [--description description] [--user user] [--password password] [--instance-root new_path] [--noprompt]` updates the Windows service entry for the server instance.

`mps-service [-C [path\]server_name] delete` deletes the Windows service entry for the server instance. If you use the dashboard to create and start server instances, do not use `mps-service` to delete server instances.

`mps-service delete service_name [[--force]|[-f]]` deletes the Windows service entry by name. If you use the dashboard to create and start server instances, do not use `mps-service` to delete server instances.

`mps-service clean [[--force]|[-f]][[--verbose]|[-v]]` deletes invalid Windows service entries.

Invalid Windows service entries are entries where either the target version of MATLAB Production Server is not present or the associated server instance no longer exists.

`mps-service [-C [path\]server_name] undelete` restores the deleted Windows service entry for the server instance.

`mps-service [-C [path\]server_name]` displays the Windows service entry for the server instance.

`mps-service list` lists the Windows service entries for all server instances.

Input Arguments

`-C` *`path`*`\`

Path to server instance

`server_name`

Name of the server instance

`--name` *`name`*

Display name for the Windows service associated with the server instance

`--description` *`description`*

Informational statement describing the Windows service associated with the server instance

`--user` *`user`*

Windows account under which the service associated with the server instance should run. The user account must have read, write, and, delete permissions for the instance directory as well read and execute permissions for the MATLAB Production Server installation directory.

`--password` *`password`*

Password for the service user account

`--instance-root` *`new_path`*

Updated path to server instance

`--noprompt`

Indicate that no prompts are generated

`--force`, `-f`

Force deletion without prompting

`--verbose`, `-v`

Include details about why the service is not valid.

Examples

Create Windows Service for Server Instance

To create a default Windows service for the server instance `server_1` located in a folder `tmp`, type the following at the system command line:

```
mps-service -C tmp\server_1 create
```

Delete Windows Service for Server Instance

To delete the Windows service entry for the server instance `server_1` located in a folder `tmp`, type the following at the system command line:

```
mps-service -C tmp\server_1 delete
```

List Existing Windows Services

To list the Windows service entries for all the server instances installed on the local machine, type the following at the system command line:

```
mps-service list

Service Name:  MATLAB Production Server {01234567-89ab-cdef-0123-456789abcdef}
Display Name:  MATLAB Production Server - My Custom Name
Description:   My Description
Instance Root: C:\instances\instance1
MPS Root:      C:\Program Files\MATLAB\MATLAB Production Server\R2014b
Status:        Started

Service Name:  MATLAB Production Server {01234567-89ab-cdef-0123-456789abcdef}
Display Name:  MATLAB Production Server - c:\instances\instance2
Description:   MATLAB Production Server running instance C:\instances\instance2
Instance Root: C:\instances\instance2
MPS Root:      C:\Program Files\MATLAB\MATLAB Production Server\R2015a
Status:        Stopped
```

Version History

Introduced in R2015a

See Also

`mps-start` | `mps-new` | `mps-stop`

Topics

“Configure Server Instance as Windows Service” on page 1-22

mps-setup

Set up server environment from command line on Windows, Linux, and macOS systems

Syntax

`mps-setup [`*`mcrroot`*`]`

Description

`mps-setup [`*`mcrroot`*`]` sets the location of the MATLAB Runtime and other start up options for server instances.

`mps-setup` sets the default path to the MATLAB Runtime instances for all server instances that you create. This is equivalent to presetting the mcr-root property in the `main_config` configuration file of each server instance.

The MATLAB Runtime instance locations that you specify when running `mps-setup` overwrite any previous values that might be present in the `mcrroot` file located in the *`server_install_location`*`/config/` folder.

Tips

- Run `mps-setup` from the *`server_install_location`*`/script` folder. Alternatively, add the `script` folder to your system `PATH` environment variable to run `mps-setup` from any folder on your system.
- If you run `mps-setup` without arguments, it searches your system for MATLAB Runtime instances that you may want to use with MATLAB Production Server.
- You can specify the path to the MATLAB Runtime as an argument when running `mps-setup`. This method is ideal for non-interactive (silent) installations.
- Before creating server instances using the dashboard, you can run `mps-setup` to specify the default MATLAB Runtime instances to use. This might be useful if you use a script to launch the dashboard. Typically, you must use either the command line interface or the dashboard to manage server instances.

Input Arguments

`mcrroot`

Path to MATLAB Runtime if running `mps-setup` in non-interactive or silent mode.

Examples

Run `mps-setup` non-interactively, by specifying the path to the MATLAB Runtime instance that you want MATLAB Production Server to use.

Type the following at the system command prompt:

```
mps-setup "C:\Program Files\MATLAB\MATLAB Runtime\mcrver"
```

mcrver is the version of the MATLAB Runtime to use.

Version History

Introduced in R2012b

See Also

`mps-start` | `mps-new` | `mps-status` | mcr-root

Topics

“Specify Default MATLAB Runtime for New Server Instances” on page 1-14
“Specify MATLAB Runtime for Server Instance Using Command Line” on page 1-15
“Configure Server Using Configuration File” on page 1-6
“Start Server Instance Using Command Line” on page 1-16

mps-start

Start server instance from command line on Windows, Linux, and macOS systems

Syntax

`mps-start [-C [`*`path/`*`]`*`server_name`*`] [-f]`

Description

`mps-start [-C [`*`path/`*`]`*`server_name`*`] [-f]` starts a server instance.

Input Arguments

`-C` ***`path/`***

Specify a path to the server instance.

If you omit this option, the system searches the current working folder and its parents to find the server instance. If you start a server instance in the current working folder, you do not need to specify a full path, specify the server name only.

`server_name`

Name of the server to be started.

`-f`

Force success even if the server instance is currently running. Starting a running server instance is considered an error.

Examples

Start a server instance named `server_1`, located in folder `tmp`, and force successful completion of `mps-start`.

Type the following at the system command prompt:

```
mps-start -f -C /tmp/server_1
```

Tips

- After issuing `mps-start`, issue the `mps-status` command to verify the server instance has started.

Version History

Introduced in R2012b

See Also

`mps-stop` | `mps-restart` | `mps-new` | `mps-status` | `mps-service`

Topics

“Start Server Instance Using Command Line” on page 1-16
“Server Overview” on page 1-2

mps-status

Display status of server instance from command line on Windows, Linux, and macOS systems

Syntax

```
mps-status [-C [path/]server_name][--statistics|-s [sample_interval]] [--
json|-j]
```

Description

`mps-status [-C [path/]server_name][--statistics|-s [sample_interval]] [--json|-j]` displays the status of the server (`STARTED`, `STOPPED`), along with a full path to the server instance. Additionally, it can display performance statistics about the server including:

- sample interval in milliseconds
- CPU utilization
- number of active worker processes
- number of requests in queue
- memory usage
- request throughput per second
- total queue time in milliseconds

Input Arguments

`-C path/`

Specify a path to the server instance. If you omit this option, the current working folder and its parents are searched to find the server instance.

`server_name`

Server to be queried for status

`--statistics [sample_interval], -s [sample_interval]`

Specify that statistics are to be collected and displayed.

The optional *`sample_interval`* allows you to specify the interval, in milliseconds, over which statistics are collected. The default is 500.

Note If you specify a sample interval of 0, only one sample is taken. Two samples are required to compute some statistics such as CPU utilization and throughput.

`--json, -j`

Specify that statistics are output in JSON format:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Instance Status",
    "description": "Status and Statistics for a MATLAB Production
                    Server Instance",
    "type": "object",
    "properties": {
        "instancePath": {
            "description": "Filesystem path for the server
                            instance",
            "type": "string"
        },
        "started": {
            "type": "boolean"
        },
        "license": {
            "type": "object",
            "properties": {
                "status": {
                    "enum": [ "CHECKED_OUT", "IN_GRACE_PERIOD",
                              "GRACE_PERIOD_EXPIRED" ]
                },
                "type": {
                    "enum": [ "INVALID", "UNKNOWN", "COMPILED",
                              "TRIAL", "EDU", "COMM" ]
                },
                "number": {"type": "string"}
            },
            "required": ["status"]
        },
        "statistics": {
            "type": "object",
            "properties": {
                "sampleIntervalMS": {
                    "description": "The difference in upTime
                                    between the two samples, 0 if
                                    only a single sample was
                                    taken",
                    "type": "number"
                },
                "localTime": {
                    "description": "Local Time at server in format
                                    YYYY.MM.DD HH.MM.SS.SSSSSS",
                    "type": "string"
                },
                "upTime": {
                    "description": "Time since server start in
                                    fractional seconds",
                    "type": "number"
                },
                "cpuTime": {
                    "description": "CPU time consumed by all server
                                    processes in fractional
                                    seconds",
                    "type": "number"
                },
                "cpuPercentage": {
                    "description": "CPU utilzation, computed using
```

```
                                change in cpuTime and upTime
                                between two samples",
                        "type": "number"
                    },
                    "totalRequestsReceived": {
                        "description": "The number of valid requests
                                received",
                        "type": "integer"
                    },
                    "totalRequestsStarted": {"type": "integer"},
                    "totalRequestsFailedToStart": {
                        "description": "The number of requests that
                                could not be started",
                        "type": "integer"
                    },
                    "totalRequestsFinishedHttpSuccess": {
                        "type": "integer"
                    },
                    "totalRequestsFinishedHttpError": {
                        "description": "Note: does not includes
                                requests that failed to start",
                        "type": "integer"
                    },
                    "memoryWorkingSet": {
                        "description": "Amount of memory resident in
                                physical memory for all
                                processes (KiB)",
                        "type": "number"
                    },
                    "throughput": {
                        "description": "Requests retired per second,
                                computed using the number of
                                requests finished or failed to
                                start over two samples",
                        "type": "number"
                    },
                    "totalQueueTimeMS": {
                        "description": "Sum of the wait times for
                                currently queued requests",
                        "type": "number"
                    }
                }
            }
        },
        "required": ["instancePath", "started"]
}
```

Examples

Check if Server is Running

Display status of server instance `server_1`, residing in `tmp` folder.

Type the following at the system command prompt:

```
mps-status -C /tmp/server_1
```

Output if server is running and running with a valid license:

```
'/tmp/server_1' STARTED
license checked out
```

Output if server is not running:

```
'/tmp/server_1' STOPPED
```

Report Statistics in Human Readable Format

Display statistics for the server instance `server_1`, residing in `tmp` folder.

Type the following at the system command prompt:

```
mps-status -C /tmp/server_1 -s
```

If server is running and running with a valid license:

```
'/tmp/server_1' STARTED
license checked out
Statistics:
Sample Interval (ms):     500
CPU Utilization (%):      40
Active Worker Processes:  2
Requests in Queue:        1
Memory Usage (KiB):       1024
Throughput (requests/s):  10
Total Queue Time (ms):    100
```

Report Statistics in JSON Format

Display statistics for the server instance `server_1`, residing in `tmp` folder.

Type the following at the system command prompt:

```
mps-status -C /tmp/server_1 -s -j
```

If server is running and running with a valid license:

```
{
    "instancePath":"L:\\MPS\\stats",
    "license":{
        "number":"unknown",
        "status":"CHECKED_OUT",
        "type":"COMM"
    },
    "started":true,
    "statistics":{
        "cpuPercentage":0,
        "cpuTime":1.7628113000000001,
        "localTime":"2015.04.28 16:52:49.874483",
        "memoryWorkingSet":393468,
        "sampleIntervalMS":500.31748899999951,
        "throughput":0,
```

```
        "totalQueueTimeMS":0,
        "totalRequestsFailedToStart":0,
        "totalRequestsFinishedHttpError":0,
        "totalRequestsFinishedHttpSuccess":0,
        "totalRequestsReceived":0,
        "totalRequestsStarted":0,
        "upTime":6.9780032949999997
    }
}
```

Version History

Introduced in R2012b

See Also

`mps-start` | `mps-stop` | `mps-restart` | `mps-which`

Topics

“Verify Server Status” on page 4-2
“Start Server Instance Using Command Line” on page 1-16
“Server Overview” on page 1-2
“Metrics Service”

mps-stop

Stop server instance from command line on Windows, Linux, and macOS systems

Syntax

```
mps-stop [-C [path/]server_name] [-f] [-p | --purge] [-k | --kill] [-v] [--
timeout hh:mm:ss]
```

Description

`mps-stop [-C [path/]server_name] [-f] [-p | --purge] [-k | --kill] [-v] [--timeout hh:mm:ss]` closes the HTTP server socket and all open client connections immediately. When you issue `mps-stop`, all function requests that the server was executing are allowed to complete before the server shuts down.

If you want to delete a server instance directory, you can delete it after the instance has stopped. You can issue the `mps-status` command to verify that the server instance has stopped.

Input Arguments

`-C path/`

Specify a path to the server instance.

If you want to stop a server instance in the current working folder, you do not need to specify a full path; only specify the server name. If you omit this option, the system searches the current working folder and its parents to find the server instance.

`server_name`

Name of the server to stop.

`-f`

Force success even if the server instance is not currently stopped. Stopping a stopped instance is considered an error.

`-p | --purge`

Remove working files in the instance directory. These files are usually removed during a graceful shutdown.

`-k | --kill`

Immediately and forcibly terminate any running processes for this instance. Use this option if a graceful shutdown has failed.

If you specify both `-k|--kill` and `--timeout` *`hh`*:*`mm`*:*`ss`* options, all running server instance processes are forcibly terminated at *`hh`*:*`mm`*:*`ss`*; if they have not already stopped.

To forcibly terminate server instance processes within the duration specified by the `server-termination-grace-period` property, do not specify the `-k` option.

You cannot forcibly terminate a server instance that is running as a Windows service.

`-v`

Display system messages.

`--timeout` *`hh`*`:`*`mm`*`:`*`ss`*

Set a limit on how long `mps-stop` runs before returning either success or failure. If you specify the `--timeout` option, the server tries to gracefully shut down and release any checked out licenses.

For example, if you specify `--timeout 00:02:00`, then `mps-stop` exits with a message if the server takes longer than two minutes to shut down. The server instance continues to attempt to terminate even if `mps-stop` times out. If you do not specify this option, the default behavior is to wait as long as necessary (infinity) for the instance to stop.

If you specify both `-k|--kill` and `--timeout` *`hh`*`:`*`mm`*`:`*`ss`* options, all running server instance processes are forcibly terminated at *`hh`*`:`*`mm`*`:`*`ss`*; if they have not already stopped.

Examples

Stop a server instance `server_1` located in the `tmp` folder.

- Force successful completion of `mps-stop` using `-f` option. Use `--timeout` option to return with a message, if `mps-stop` takes longer than three minutes to complete. Specify the verbose `-v` option to produce an output status message.

  Type the following at the system command line:

  ```
  mps-stop -f -v -C /tmp/server_1 --timeout 00:03:00

  waiting for stop... (timeout = 00:03:00)
  ```
- Immediately terminate all running server instance processes by force using the `-k` option.

  Type the following at the system command line:

  ```
  mps-stop -k /tmp/server_1
  ```
- To wait as long as necessary to stop server instance processes, do not specify the `--timeout` option or set the server-termination-grace-period property in the `main_config` server configuration file.

  Type the following at the system command line:

  ```
  mps-stop /tmp/server_1
  ```

Version History

Introduced in R2012b

See Also

`mps-start` | `mps-service` | `mps-restart` | `mps-new` | `mps-status` | server-termination-grace-period

mps-support-info

Display licensing and configuration information of server instance from command line on Windows, Linux, and macOS systems

Syntax

```
mps-support-info [-C [path/]server_name]
```

Description

`mps-support-info` displays licensing and configuration information of a MATLAB Production Server instance.

Input Arguments

- *`path`* — The path to where the server instance is installed.
- *`server_name`* — The name of the server instance to locate in the current folder.

Examples

Display licensing and configuration information of server instance `fred`, residing in `/` folder.

Type the following at the system command prompt:

```
mps-support-info -C /fred
```

```
Instance Version:           1.0
License Number:             UNKNOWN -- MPS stopped
MPS Version:                UNKNOWN -- MPS stopped
Available License Number:   857812
Client Version:             1.0.1 R2013a
Operating System:           Microsoft Windows 7 Enterprise Edition (build 7601), 64-bit
Number of CPU cores:        8
CPU Info:                   Intel(R) Xeon(R) CPU           W3550  @ 3.07GHz 64-bit Compatible
Memory:                     11.9915 GB ( 1.2574e+007 KB )
```

Version History

Introduced in R2012b

See Also

`mps-status` | `mps-check` | `mps-license-reset`

mps-which

Display path to server instance that is currently using configured port from command line on Windows, Linux, and macOS systems

Syntax

`mps-which [-C [path/]server_name]`

Description

`mps-which [-C [path/]server_name]` is useful when running multiple server instances on the same machine. If you attempt to start two server instances on the same port, the latter server instance fails to start, displaying an `address-in-use` error. `mps-which` identifies which server instance is using the port.

Input Arguments

`-C path/`

Specify a path to the server instance. If this option is omitted, the current working folder and its parents are searched to find the server instance.

`server_name`

Server to be queried for path.

Examples

`server_1` and `server_2`, both residing in folder `tmp`, are configured to use to same port, defined by the `http` configuration property.

Run `mps-which` from the system command prompt for both servers:

```
mps-which -C /tmp/server_1
```

```
mps-which -C /tmp/server_2
```

In both cases, the server that has allocated the configured port displays (`server_1`):

```
/tmp/server_1
```

Tips

- If you are creating a server instance in the current working folder, you do not need to specify a full path. Specify only the server name.

Version History

Introduced in R2012b

See Also

`mps-status`

mps-cache

Control persistence service from command line on Windows, Linux, and macOS systems

Syntax

```
mps-cache [operation] [-C server_path] [--connection connection_name] [--all]
[--configFile provider_config_file_path] [--key cache_access_key_string] [--
timeout seconds] [--verbose | -v] [--help |-h]
```

Description

`mps-cache [operation] [-C server_path] [--connection connection_name] [--all] [--configFile provider_config_file_path] [--key cache_access_key_string] [--timeout seconds] [--verbose | -v] [--help |-h]` controls the persistence service based on the specified *operation*. The supported operations are `start`, `stop`, `restart`, `ping`, `attach`, and `detach`.

The option *connection_name* is obtained from the JSON file `mps_cache_config`. This file must be created by an administrator and placed in the `config` folder of the server instance. The JSON structure of the `mps_cache_config` file is:

```
{
  "Connections": {
    "<connection_name>": {
      "Provider": "Redis",
      "Host": "<hostname>",
      "Port": <port_number>
    }
  }
}
```

<connection_name>, *<host_name>* and *<port_number>* are the only fields that can be set by the administrator and *<port_number>* has to be a non-SSL port. Currently, Redis is the only supported persistence service provider. You can have multiple connections to the persistence provider.

Input Arguments

operation

`start | stop | restart | ping | attach | detach`

- `start` — Start a persistence service.
- `stop` — Stop a persistence service.
- `restart` — Restart a persistence service.
- `ping` — Test whether the persistence service is reachable.
- `attach` — Connect persistence service to server instance process.
- `detach` — Disconnect persistence service from server instance process.

Note You cannot start, stop, or restart a remote persistence service.

`-C` *`server_path`*

Path to the server instance.

`--connection` *`connection_name`*

Name of connection to persistence service. Specify either `--all` or `--connection` *`connection_name`*, not both.

`--all`

Connect to all the persistence services specified in `mps_cache_config` file. Specify either `--all` or `--connection` *`connection_name`*, not both.

`--configFile` *`provider_config_file_path`*

Path to the persistence provider configuration file.

`--key` *`cache_access_key_string`*

Access key string to connect to an Azure Redis Cache instance obtained from the Azure portal. For an example, see “Ping Remote Persistence Service” on page 7-34.

`--timeout` *`ss`*

Set a limit on how long `mps-cache` will run before returning either success or failure. The default duration is `30` seconds. For example, specifying `--timeout 15` indicates that `mps-cache` should exit with an error status if it takes longer than `15` seconds to access the service.

`--verbose | -v`

Displays system messages relating to controlling the persistence service.

`--help | -h`

Displays options for using the `mps-cache` command.

Examples

Start Persistence Service

Start a persistence service on Windows assuming a connection name `myConnection` has been defined in the file `mps_cache_config`.

Type the following at the system command line:

```
mps-cache start -C "h:\server\mps_instance" --connection myRedisConnection
mps-cache ping -C "h:\server\mps_instance" --connection myRedisConnection

Sending ping to Redis on localhost:9710.
Redis service running on localhost:9710.
```

The corresponding `mps_cache_config` file for the example follows:

```
{
  "Connections": {
    "myRedisConnection": {
      "Provider": "Redis",
      "Host": "localhost",
      "Port": 9710
    }
  }
}
```

Ping Remote Persistence Service

Assuming an Azure Redis Cache instance has been setup in the Azure portal and a connection name `myRemoteAzureRedisCacheConnection` has been defined in the file `mps_cache_config`.

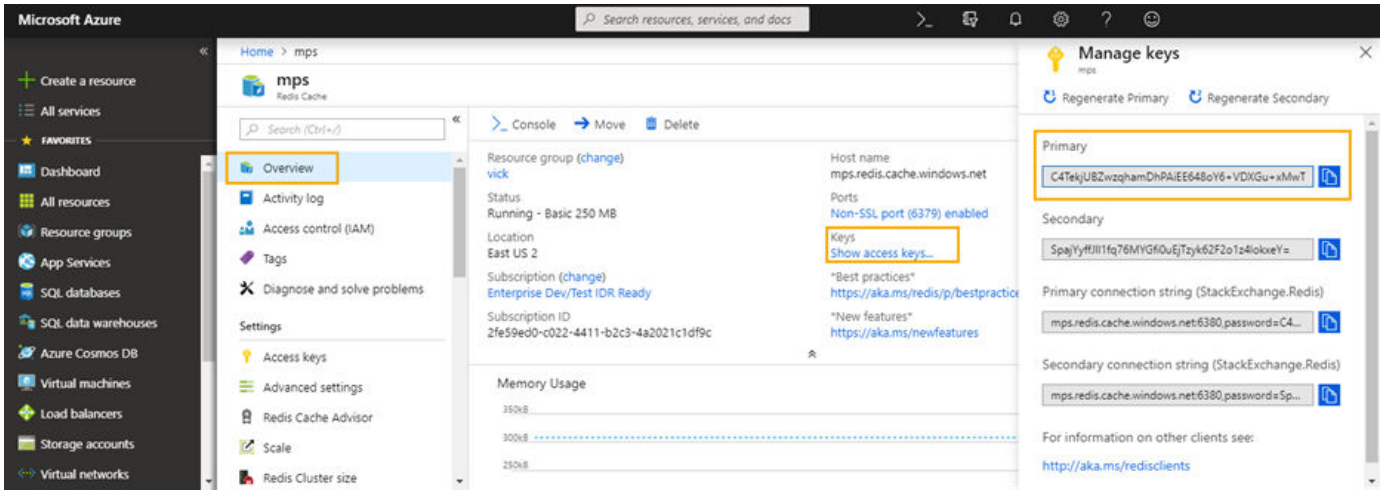
```
mps-cache ping -C "h:\server\mps_instance"
               --connection myRemoteAzureRedisCacheConnection
               --key +WcI8pU0YodDMsw1LLC7gInkjtrjamLBRoq9rQQdMTU=

Sending ping to Redis on azure.redis.cache.windows.net:6379.
Redis service running on azure.redis.cache.windows.net:6379.
```

The corresponding `mps_cache_config` file for the example follows:

```
{
  "Connections": {
    "myRedisConnection": {
      "Provider": "Redis",
      "Host": "localhost",
      "Port": 9710
    },
    "myRemoteAzureRedisCacheConnection": {
      "Provider": "Redis",
      "Host": "azure.redis.cache.windows.net",
      "Port": 6379
    }
  }
}
```

Tips

- To retrieve an access key to connect to an Azure Redis Cache instance:
  - Log in to your Azure portal and select your Azure Redis Cache instance.
  - Select **Overview** and under **Keys** click **Show access keys**.
  - In the resulting blade, copy the access key string listed under **Primary**.



Version History

Introduced in R2018b

See Also

Topics

“Data Caching Basics”

Configuration Properties

access-control-provider

Identity management service provider name

Syntax

```
--access-control-provider provider
```

Description

`--access-control-provider` enables access control using generic identity providers for applications deployed to a server.

You can set either the `--access-control-provider` property or the `ssl-allowed-client` property. If you set both, the server fails to start.

Parameters

`provider` specifies the identity provider that MATLAB Production Server uses for application access control. Supported values for *`provider`* include `OAuth2`.

Examples

- For on-premises server instances created using the command line, update this server configuration property in the `main_config` server configuration file.
- For on-premises server instances created using the dashboard and for server deployments in the cloud, use the dashboard and cloud dashboard, respectively, to update the server configuration property.

Configure Using Command Line

Enable Access Control Using OAuth2

In the `main_config` file, set the `access-control-provider` property to the following:

```
--access-control-provider OAuth2
```

Configure Using Dashboard

Enable Access Control Using OAuth2

In the dashboard, in the **Settings** tab of your server instance, under **Advanced**, enter the following for the **Additional Options** property:

```
--access-control-provider OAuth2
```

Version History

Introduced in R2018b

R2021a: Access control provider value will be renamed
Not recommended starting in R2021a

Use `OAuth2` instead of `AzureAD` when specifying the *`provider`* for this property. Specifying `OAuth2` includes support for `AzureAD`.

See Also

access-control-config | access-control-policy

Topics

“Application Access Control” on page 3-8

access-control-config

Path to the identity management service provider configuration file

Syntax

`--access-control-config` *`path`*

Description

`--access-control-config` *`path`* specifies the path to the identity provider specific configuration file. Specifying this property is optional. The default path for the OAuth2 identity provider is `./config/jwt_idp.json`.

If you enable access control on a server instance by specifying the access-control-provider property, the access control configuration file must exist in *`path`*; otherwise, the server instance fails to start.

Parameters

`path` specifies the path to access the configuration file.

Examples

Specify the path to the OAuth2 configuration file.

```
--access-control-config ./config/jwt_idp.json
```

Version History

Introduced in R2018b

R2021a: `azure_ad.json` file name will be removed
Not recommended starting in R2021a

The `azure_ad.json` file name will be removed in a future release. The default name for the access control configuration file is `jwt_idp.json` instead of `azure_ad.json`, if the `access-control-provider` property is set to `OAuth2`.

See Also

access-control-provider | access-control-policy

Topics

“Access Control Configuration File” on page 3-8

access-control-policy

Path to access control policy file

Syntax

`--access-control-policy` *`path`*

Description

`--access-control-policy` *`path`* specifies the path to the access control policy file on a server instance. Setting this property is optional. The default path is `./config/ac_policy.json`.

If you enable access control by specifying the access-control-provider property, the access control policy file must exist in *`path`*; otherwise, the server instance fails to start.

After a server starts, it scans the policy file every five seconds for any changes, if access control is enabled. If the policy file is absent or contains errors, the server continues to run, but denies all requests and logs an error message in the `main.log` file.

Parameters

`path` specifies the path to the policy file.

Examples

Specify the path to the access control policy JSON file.

```
--access-control-policy ./config/ac_policy.json
```

Version History

Introduced in R2018b

See Also

access-control-provider | access-control-config

Topics

“Access Control Policy File” on page 3-10

async-deploy-on-startup

Deploy archives after the server starts

Syntax

```
--async-deploy-on-startup
```

Description

`async-deploy-on-startup` delays deployment of the deployable archives until after the server starts. After starting, the server scans the folder specified by the auto-deploy-root property for archives to deploy, then deploys them. Due to this delayed deployment, the archives are accessible only after the server finishes deploying them and are not accessible immediately after the server starts. If you have several archives or large archives to deploy, then set this property to avoid server timeout on startup.

If you do not set this property, the server instance automatically unpacks and deploys the archives placed in the folder specified by the `auto-deploy-root` property when it starts.

Examples

Deploy archives after the server starts.

```
--async-deploy-on-startup
```

Version History

Introduced in R2020a

See Also

auto-deploy-root | `mps-start`

Topics

“Server Overview” on page 1-2
“Deploy Archive to MATLAB Production Server”
“Modifying Deployed Functions”

auto-deploy-root

Folder that the server instance scans for deployable archives

Syntax

`--auto-deploy-root` *`path`*

Description

`--auto-deploy-root` *`path`* specifies the folder that the server instance scans for deployable archives.

The server instance automatically unpacks and deploys archives placed in this folder when it starts. Starting in R2020a, if you set the async-deploy-on-startup, the server instance deploys the archives only after it starts.

You do not need to restart the server after a deployable archive is added, updated, or removed. Multiple server instances can share a single `auto-deploy-root` folder. Using this folder allows near-simultaneous hot deployment to multiple instances. The server scans the folder every five seconds for any changes.

Parameters

`path`

Path to the folder that the server instance scans for deployable archives. The path is relative to the server instance root folder.

Examples

Scan the `auto_deploy` folder for deployable archives to hot deploy.

```
--auto-deploy-root ./auto_deploy
```

See Also

async-deploy-on-startup | `mps-start`

Topics

“Server Overview” on page 1-2
“Deploy Archive to MATLAB Production Server”

client-credential-delegation

Client credential delegation method name

Syntax

`--client-credential-delegation` *`method`*

Description

`--client-credential-delegation` *`method`* specifies the client credential delegation method that the server uses. Currently, `kerberos-without-protocol-transition` is the only supported method. If you set `client-credential-delegation` to `kerberos-without-protocol-transition`, then you must set http-authentication-method to `spnego`; otherwise, the server fails to start.

Parameters

`method`

Name of the client credential delegation method. `kerberos-without-protocol-transition` is the only supported method.

Examples

Use `kerberos-without-protocol-transition` as the client credential delegation method.

```
--client-credential-delegation kerberos-without-protocol-transition
```

Version History

Introduced in R2019b

See Also

http-authentication-method

Topics

“Use Kerberos Authentication and Kerberos Delegation” on page 3-14

cors-allowed-origins

Specify the domain origins from which clients are allowed to make requests to the server

Syntax

```
--cors-allowed-origins [ LIST | * ]
```

Description

`cors-allowed-origins` specifies the set of domain origins from which clients are allowed to make requests to a MATLAB Production Server instance. Cross-Origin Resource Sharing or CORS defines a way in which client-side web applications and a server can interact to safely determine whether or not to allow a cross-origin request. Most clients such as browsers use the `XMLHttpRequest` object to make a cross-domain request. This is especially true for client code written using JavaScript®. For MATLAB Production Server to support such requests, you must enable `cors-allowed-origins` on the server.

Parameters

*

Requests from any domain origin are allowed access to the sever.

LIST

Requests from a list of comma-separated domain origins are allowed access to the server.

Examples

- For on-premises server instances created using the command line, update this server configuration property in the `main_config` server configuration file.
- For on-premises server instances created using the dashboard and for server deployments in the cloud, use the dashboard and cloud dashboard, respectively, to update the server configuration property.

Configure Using Command Line

Allow Requests from Any Domain Origin to Access Server

In the `main_config` file, set the `cors-allowed-origins` property to the following:

```
--cors-allowed-origins *
```

Allow Requests from Specific List of Domain Origins to Access Server

In the `main_config` file, set the `cors-allowed-origins` property to the following:

```
--cors-allowed-origins http://www.w3.org, https://www.apache.org
```

Configure Using Dashboard

Allow Requests from Any Domain Origin to Access Server

In the `main_config` file, set the `cors-allowed-origins` property to the following:

*

Allow Requests from Specific List of Domain Origins Access to Server

In the dashboard, in the **Settings** tab of your server instance, under **HTTP**, enter the following for the **CORS Allowed ORigins** property:

```
http://www.w3.org, https://www.apache.org
```

See Also

`http`

disable-control-c

Disable keyboard interruptions for server instance

Syntax

```
--disable-control-c
```

Description

`disable-control-c` disables keyboard interruption for the server instance. The server instance does not respond to **Ctrl+C**.

If you set both `disable-control-c` and enable-graceful-shutdown properties, then the server ignores the terminal interrupt event **Ctrl+C**. The server gracefully shuts down only on receiving the program termination signal or when the system shuts down or restarts.

Examples

Disable the **Ctrl+C** keys.

```
--disable-control-c
```

See Also

enable-graceful-shutdown | `mps-stop`

Topics

“Configure Server Using Configuration File” on page 1-6

enable-graceful-shutdown

Gracefully shut down server processes after receiving a terminal interrupt signal or program termination signal

Syntax

```
--enable-graceful-shutdown
```

Description

`enable-graceful-shutdown` gracefully shuts down server processes if they are interrupted by a terminal interrupt signal or terminated by the program termination signal. When you set this property, the server releases any checked out licenses when the system that runs the server shuts down or restarts.

If you set both the disable-control-c and `enable-graceful-shutdown` properties, then the server ignores the terminal interrupt event **Ctrl+C**. The server gracefully shuts down only on receiving the program termination signal or when the system shuts down or restarts.

Examples

Enable graceful server shutdown.

```
--enable-graceful-shutdown
```

Version History

Introduced in R2020a

See Also

disable-control-c | `mps-stop`

Topics

“Configure Server Using Configuration File” on page 1-6

endpoint-root

Folder used to store server named endpoints

Syntax

`--endpoint-root` *`path`*

Description

`--endpoint-root` *`path`* specifies the location for storing server named endpoints. Each interface used to communicate with the outside world generates an endpoint file in this folder. Normally that means:

- http - The HTTP function execution interface.
- control - The local control interface used by the scripting commands.

These files contain the `host:post` portion of the URL used to communicate with the named service.

Note While modifying this location is allowed, each instance must have a unique endpoint directory; otherwise behavior is undefined.

Parameters

`path`

Path to the folder used to store endpoint files relative to the server instance's root folder.

Examples

Store endpoint files in the `endpt` folder.

```
--endpoint-root ./endpt
```

extract-root

Root folder used to store contents of deployed archives

Syntax

`--extract-root` *`path`*

Description

`--extract-root` *`path`* specifies the root folder used to store the expanded contents of the deployable archives deployed on the server instance. Deployable archives are unpacked to a hidden subdirectory of `extract-root`.

Parameters

`path`

Path to the root folder used to store contents of deployable archives relative to the server instance's root folder.

Examples

Extract deployable archives into the `archives` folder.

```
--extract-root ./archives
```

files-root

Serve static content over HTTP or HTTPS

Syntax

`--files-root` *`path`*

Description

`--files-root` *`path`* enables MATLAB Production Server to serve static content over HTTP or HTTPS.

To access static file content from a web browser, provide the file name *`file_name.ext`* (where ext is the file extension), host name *`server_host_name`*, and port `server_port` in the format *`http(s):server_host_name:server_port/~files/file_name.ext`*.

Parameters

`path`

Path to the folder containing static files.

Examples

- For on-premises server instances created using the command line, update this server configuration property in the `main_config` server configuration file.
- For on-premises server instances created using the dashboard and for server deployments in the cloud, use the dashboard and cloud dashboard, respectively, to update the server configuration property.

Serve Static Content

Assume that you have a server running at the host name `my.server.com` and port number 9910.

Configure Using Command Line

1. Assume that you have a text file, `my_file.txt`, stored in a folder called `static_files`, and `static_files` is located in the root folder of your server instance.
2. In the `main_config` server configuration file, specify the path to the `static_files` folder as the parameter to the `files-root` property.

```
--files-root ./static_files
```

Configure Using Dashboard

1. Assume that you have a text file, `my_file.txt`, stored in a folder called `static_files`, and `static_files` is located in the `C:\tmp` folder.
2. In the dashboard, in the **Settings** tab of your server instance, under **Advanced**, enter the following for the **Additional Options** property.

```
--files-root C:\tmp\static_files
```

To access `my_file.txt` from a web browser, specify the following in the address bar of your browser, `http:my.server.com:9910/~files/my_file.txt`.

See Also

Topics

“Handle Custom Routes and Payloads in HTTP Requests” on page 2-14

hide-matlab-error-stack

Hide MATLAB error stack from clients

Syntax

```
--hide-matlab-error-stack
```

Description

`hide-matlab-error-stack` controls whether the server sends the MATLAB error stack to the client. You can choose to send the error stack during development and debug phases, but can turn it off in production.

Examples

- Do not transmit the error stack to clients.

  Uncomment the following property in the `main_config` server configuration file.

  ```
  --hide-matlab-error-stack
  ```

  Example of the error that a REST client receives:

  ```
  {
      "error": {
          "id": "MATLAB:invalidConversion",
          "message": "Conversion to double from struct is not possible.",
          "type": "matlaberror"
      }
  }
  ```

- Transmit the error stack to clients.

  Add a # to comment the following property in the `main_config` server configuration file.

  ```
  #--hide-matlab-error-stack
  ```

  Example of the error that a REST client receives:

  ```
  {
      "error": {
          "id": "MATLAB:invalidConversion",
          "message": "Conversion to double from struct is not possible.",
          "stack": [
              {
                  "file": "C:\\Program Files\\MATLAB\\MATLAB Runtime\\v99\\mcr\\toolbox\\matlab\\elmat\\magic.m",
                  "line": 9,
                  "name": "magic"
              },
              {
                  "file": "J:\\server20b\\.mps_deployed\\mpsTestData_6\\mpsTestData\\m\\myMagic.m",
                  "line": 7,
                  "name": "myMagic"
              }
          ],
          "type": "matlaberror"
  ```

```
        }
    }
```

See Also

Topics
“Manage Log Files” on page 4-7

http

URL for insecure connections

Syntax

```
--http host:port
```

Description

`http` specifies the interface port and optional address or host name.

Parameters

`host`

Host name or IP address of the machine running the server instance. If you do not specify the host, the server binds to any available interface.

`port`

Port number used by the server instance to accept connections. Bind to any available port by specifying `0`.

Examples

- For on-premises server instances created using the command line, update this server configuration property in the `main_config` server configuration file.
- For on-premises server instances created using the dashboard and for server deployments in the cloud, use the dashboard and cloud dashboard, respectively, to update the server configuration property.

Configure Using Command Line

Restrict Access to HTTP Interface for Local Clients Only on Port 9910

In the `main_config` file, set the `http` property to the following:

```
--http 9910
```

Bind to Any Free Port

In the `main_config` file, set the `http` property to the following. The bound address is written to *`$INSTANCE`*`/endpoint/http`.

```
--http 0
```

Bind to Specific IP Address and Port

In the `main_config` file, set the `http` property to the following:

```
--http 234.27.101.3:9920
```

Bind to Specific Host Name on Any Free Port

In the `main_config` file, set the `http` property to the following:

```
--http my.hostname.com:0
```

Configure Using Dashboard

Restrict Access to HTTP Interface for Local Clients Only on Port 9910

In the dashboard, in the **Settings** tab of your server instance, under **HTTP**, enter the following for the **HTTP** property:

```
9910
```

Bind to Any Free Port

In the dashboard, in the **Settings** tab of your server instance, under **HTTP**, enter the following for the **HTTP** property:

```
0
```

Bind to Specific IP Address and Port

In the dashboard, in the **Settings** tab of your server instance, under **HTTP**, enter the following for the **HTTP** property:

```
234.27.101.3:9920
```

Bind to Specific Host Name on Any Free Port

In the dashboard, in the **Settings** tab of your server instance, under **HTTP**, enter the following for the **HTTP** property:

```
my.hostname.com:0
```

See Also

https

http-authentication-method

HTTP authentication method name

Syntax

`--http-authentication-method` *`method`*

Description

`--http-authentication-method` *`method`* specifies the HTTP authentication method that the server uses to authenticate the client.

If you do not specify this property, the server does not perform HTTP authentication. You can still authenticate using an HTTPS client certificate. For more information on configuring client authentication, see “Configure Client Authentication” on page 3-4.

Parameters

`method`

Name of HTTP authentication method. `spnego` (Simple and Protected Negotiation Mechanism) is the only supported method.

Examples

Specify `spnego` as the HTTP authentication method.

```
--http-authentication-method spnego
```

Version History

Introduced in R2019b

See Also

client-credential-delegation

Topics

“Use Kerberos Authentication and Kerberos Delegation” on page 3-14

http-linger-threshold

Amount of data the server instance discards after an HTTP error and before the server instance closes the TCP connection

Syntax

```
--http-linger-threshold size
```

Description

`http-linger-threshold` sets the amount of data a server instance reads after an error. If an HTTP request is rejected and the server instance sends back an HTTP error response such as HTTP 404/413, the server instance does not close the TCP connection immediately. Instead it waits for the client to shut down the TCP connection. This ensures that the client receives the HTTP error response sent by the server instance. During this time, the server instance receives, and discards, data from the client, until the amount of data received equals `http-linger-threshold`. After that, the server instance resets the TCP connection.

By default, the threshold is unlimited and the server instance waits to receive the whole HTTP request.

Parameters

size

Amount of data received before the TCP connection is reset.

Examples

Set the linger threshold to be 64 MB.

```
--http-linger-threshold 64MB
```

Set the linger threshold to be 32 KB.

```
--http-linger-threshold 32KB
```

Set the linger threshold to be 1024 B.

```
--http-linger-threshold 1024
```

https

URL for secure connections

Syntax

```
--https host:port
```

Description

`https` specifies the interface port and the optional address or host name to use for secure client-server communication.

Starting in R2019b, if you set the `https` property, you must set the x509-private-key and x509-cert-chain properties; otherwise, the server fails to start.

Parameters

`host`

Host name or IP address of the machine running the server instance. If you do not specify the host, the server binds to any available interface.

`port`

Port number used by the server instance to accept connections. Bind to any available port by specifying `0`.

Examples

- For on-premises server instances created using the command line, update this server configuration property in the `main_config` server configuration file.
- For on-premises server instances created using the dashboard and for server deployments in the cloud, use the dashboard and cloud dashboard, respectively, to update the server configuration property.

Configure Using Command Line

Restrict Access to HTTPS Interface for Local Clients Only on Port 9920

In the `main_config` file, set the `https` property to the following:

```
--https 9920
```

Bind to Any Free Port

In the `main_config` file, set the `http` property to the following. The bound address is written to *`$INSTANCE`*`/endpoint/http`.

```
--https 0
```

Bind to Specific IP Address and Port

In the `main_config` file, set the `https` property to the following:

```
--https 234.27.101.3:9920
```

Bind to Specific Host Name on Any Free Port

In the `main_config` file, set the `https` property to the following:

```
--https my.hostname.com:0
```

Configure Using Dashboard

Restrict Access to HTTPS Interface for Local Clients Only on Port 9920

In the dashboard, in the **Settings** tab of your server instance, under **HTTP**, enter the following for the **HTTPS** property:

```
9920
```

Bind to Any Free Port

In the dashboard, in the **Settings** tab of your server instance, under **HTTP**, enter the following for the **HTTPS** property:

```
0
```

Bind to Specific IP Address and Port

In the dashboard, in the **Settings** tab of your server instance, under **HTTP**, enter the following for the **HTTPS** property:

```
234.27.101.3:9920
```

Bind to Specific Host Name on Any Free Port

In the dashboard, in the **Settings** tab of your server instance, under **HTTP**, enter the following for the **HTTPS** property:

```
my.hostname.com:0
```

See Also

x509-private-key | x509-cert-chain

Topics

“Enable HTTPS” on page 3-2

license

Locations for valid licenses

Syntax

`--license` *`pathList`*

Description

`license` specifies the address of the license servers or the path to the license files that a server instance uses. You can specify multiple license sources with this option.

If you do not specify a value for this property, the server searches for the license files in *`$MPS_INSTALL`*`/licenses`, where *`$MPS_INSTALL`* is the location in which MATLAB Production Server is installed.

Parameters

`pathList`

Path to one or more license servers or license files. Separate multiple entries by the appropriate path separator for the platform. Use a colon (`:`) as the path separator for Linux and semi-colon (`;`) as the path separator for Windows.

Examples

- For on-premises server instances created using the command line, update this server configuration property in the `main_config` server configuration file.
- For on-premises server instances created using the dashboard and for server deployments in the cloud, use the dashboard and cloud dashboard, respectively, to update the server configuration property.

Configure Using Command Line

Use License Server and Specify License File on Linux System

A Linux server looks for licenses using a license server hosted on port 27000 of `hostA` and in `/opt/license/license.dat`.

In the `main_config` file, set the `license` property to the following:

```
--license 27000@hostA
--license /opt/license/license.dat
```

The same configuration in one line:

```
--license 27000@hostA:/opt/license/license.dat
```

Use License Server and Specify License File on Windows System

A Windows server looks for licenses using a license server hosted on port 27000 of `hostA` and in `c:\license\license.dat`.

In the `main_config` file, set the `license` property to the following:

```
--license 27000@hostA
--license c:\license\license.dat
```

The same configuration in one line:

```
--license 27000@hostA;c:\license\license.dat
```

Configure Using Dashboard

Specify License Server Address and Specify License File on Linux System

A Linux server looks for licenses using a license server hosted on port 27000 of `hostA` and in `/opt/license/license.dat`.

In the dashboard, in the **Settings** tab of your server instance, under **License**, enter the following for the **License** property:

```
27000@hostA:/opt/license/license.dat
```

Specify License Server Address and Specify License File on Windows System

A Windows server looks for licenses using a license server hosted on port 27000 of `hostA` and in `c:\license\license.dat`.

In the dashboard, in the **Settings** tab of your server instance, under **License**, enter the following for the **License** property:

```
27000@hostA;c:\license\license.dat
```

See Also

license-grace-period | license-poll-interval | `mps-license-reset`

Topics

“Manage Licenses for MATLAB Production Server”
“Verify Server Status” on page 4-2

license-grace-period

Maximum length of time the server instance responds to HTTP requests after license server heartbeat has been lost

Syntax

`--license-grace-period` *`hr`*`:`*`min`*`:`*`sec`*`.`*`fractSec`*

Description

`license-grace-period` specifies the grace period, which starts at the first heartbeat loss event. Once the grace period expires, the server instance rejects any new incoming HTTP requests.

The default grace period is 2 hours 30 minutes. The maximum value is 2 hours 30 minutes. The minimum value is 10 minutes.

Parameters

`hr`

Hours in interval.

`min`

Minutes in interval.

`sec`

Seconds in interval.

`fractSec`

Fractional seconds in interval.

Examples

The grace period lasts for 1 hour, 29 minutes, 5 seconds.

```
--license-grace-period 1:29:05
```

The grace period lasts for 10 minutes and 250 ms.

```
--license-grace-period 00:10:00.25
```

license-poll-interval

Interval of time before license server is polled to verify and check out a valid license after the grace period expires

Syntax

```
--license-poll-interval hr:min:sec.fractSec
```

Description

`license-poll-interval` specifies interval at which the server instance polls the license server after the license server has timed out or after the grace period has expired.

The default poll interval is 10 minutes. The minimum value is 10 minutes.

Parameters

hr
: Hours in interval.

min
: Minutes in interval.

sec
: Seconds in interval.

fractSec
: Fractional seconds in interval.

Examples

Poll for licenses at intervals of 1 hour, 29 minutes, 5 seconds.

```
--license-poll-interval 1:29:05
```

Poll for licenses at intervals of 10 minutes and 250 ms.

```
--license-poll-interval 00:10:00.25
```

log-archive-max-size

Maximum size of the log archive folder

Syntax

`--log-archive-max-size` *size*

Description

`log-archive-max-size` specifies the maximum size to which the log archive folder can grow before old log files are deleted.

If this property is not specified, then the log archive grows without limit.

Parameters

size

Size, in bytes, of the archive folder.

Examples

Reap log archives when they reach 5 MB.

```
--log-archive-max-size 5MB
```

log-archive-root

Path to the folder containing archived log files

Syntax

`--log-archive-root` *`path`*

Description

`--log-archive-root` *`path`* specifies the path to directory that stores rotated log files.

Note If you omit this property, rotated logs remain in the log root directory, which grows unbounded as logs are rotated.

Parameters

`path`

Path to the folder where log files are archived relative to the server instance's root folder.

Examples

Archive logs to *`server_root`*`/old_logs`.

```
--log-archive-root ./old_logs
```

log-handler

Add custom log handler

Syntax

`--log-handler` *`format command`*

Description

`--log-handler` *`format command`* adds a log handler that writes log data to the application specified by *`command`* in the format specified by *`format`*.

The server instance launches an instance of the log handler at startup. All log events are sent to the `STDIN` stream of the log handler. The `STDOUT` and `STDERR` streams of the log handler are captured and written to *`INSTANCE_ROOT`*`/log/custom_logger_`*`N`*`.out` and *`INSTANCE_ROOT`*`/log/custom_logger_`*`N`*`.err`.

Parameters

`format`

Format used to write log events. Valid values are:

- `text/plain`
- `text/json`
- `text/xml`

`command`

Application launched to process log events.

Examples

Send log events to a custom JSON parser that prepares performance graphs.

```
--log-handler text/json perf_grapher
```

log-root

Path to folder containing log files

Syntax

`--log-root` *`path`*

Description

`--log-root` *`path`* specifies the location of the folder where a MATLAB Production Server instance writes log files. This property is available only for an on-premises server installation.

The server creates the following log files when it starts.

- `main.log` — Most recent log file.
- `main.out` — Captures standard output from the main process.
- `main.err` — Captures standard error output from the main process.
- `main__`*`DATE`*`__`*`SERIAL`*`.log` — Main process log. When a server instance restarts or the size of `main.log` reaches the limit set by the log-rotation-size property, the server renames `main.log` to `main__`*`DATE`*`__`*`SERIAL`*`.log` and archives it in the folder set by the log-archive-root property. The default archive location is the `old_logs` folder of a server instance.

Note Omitting this property disables all logging.

Parameters

`path`

Path to the folder containing log files for a server instance.

Examples

- For on-premises server instances created using the command line, update this server configuration property in the `main_config` server configuration file.
- For on-premises server instances created using the dashboard and for server deployments in the cloud, use the dashboard and cloud dashboard, respectively, to update the server configuration property.

Configure Using Command Line

Store Server Logs in Default Location in `log` Folder of Server Instance

In the `main_config` file, set the `log-root` property to the following:

```
--log-root ./log
```

See Also

log-severity | main-log-format | log-archive-root | log-rotation-size

Topics

“Manage Log Files” on page 4-7
“Server Diagnostic Tools” on page 4-6

log-rotation-size

Size at which the log is archived

Syntax

`--log-rotation-size` *size*

Description

`log-rotation-size` specifies the maximum size to which the log can grow before it is rotated into the archive area. If specified as less than 1 MB, a warning is issued and the effective size is increased to 1 MB.

No entry signifies that logs are never archived.

Parameters

size

Size, in bytes, of the log file.

Examples

Rotate logs when they reach 5 MB.

```
--log-rotation-size 5MB
```

log-severity

Severity at which messages are logged

Syntax

```
--log-severity level
```

Description

`log-severity` specifies the level of detail at which to add information to the main log.

Parameters

`level`

Severity threshold at which messages are logged. Valid values are:

- `error` — Notification of problems or unexpected results.
- `warning` — Events that could lead to problems if not addressed.
- `information` — High-level information about major server events.
- `trace` — Detailed information about the internal state of the server.

The levels are cumulative; specifying `information` implies `warning` and `error`.

Examples

- For on-premises server instances created using the command line, update this server configuration property in the `main_config` server configuration file.
- For on-premises server instances created using the dashboard and for server deployments in the cloud, use the dashboard and cloud dashboard, respectively, to update the server configuration property.

Configure Using Command Line

Enable All Log Messages

In the `main_config` file, set the `log-severity` property to the following:

```
--log-severity trace
```

Configure Using Dashboard

Enable All Log Messages

In the dashboard, in the **Settings** tab of your server instance, under **Logging**, select the following for the **Log Severity** property:

```
trace
```

See Also

Topics
“Manage Log Files” on page 4-7

main-log-format

Text format for the main log file

Syntax

`--main-log-format` *`format`*

Description

`main-log-format` specifies the text format for logging events in the `main.log` file. If you do not set this property, the server writes the log data as plain text.

Parameters

`format`

Format for writing log events. Valid values are:

- `text/plain` — Log the data in plain text.
- `text/json` — Log the data in JSON format.
- `text/xml` — Log the data in XML format.

Examples

Log events in JSON format.

```
--main-log-format text/json
```

Version History

Introduced in R2020a

See Also

log-handler | log-root

Topics

“Log Files” on page 4-6
“Manage Log Files” on page 4-7

mcr-root

Location of a MATLAB Runtime installation

Syntax

`--mcr-root` *path*

Description

`mcr-root` specifies the path to a MATLAB Runtime installation on a system that has a MATLAB Production Server installed.

You can configure a server instance to use multiple MATLAB Runtime versions by specifying multiple `mcr-root` properties. To do so, specify the `mcr-root` property with the path to each MATLAB Runtime installation on a separate line, starting from the latest version to the oldest. The server instance scans the list of specified `mcr-root` properties in order from first to last, then chooses the first MATLAB Runtime installation capable of processing a server request. A MATLAB Runtime installation can process a server request if it is compatible with the deployable archive containing the MATLAB function being evaluated. When you configure the server to use multiple MATLAB Runtime versions, the server uses dynamic worker pool management, where it starts the worker processes in response to demand, and stops them in response to system resource utilization. Specifying multiple MATLAB Runtime installations of the same version has no effect on server performance.

Note An installation of MATLAB Production Server supports MATLAB Runtime versions up to six releases back.

Note

- Specify the path to a MATLAB Runtime installation on a local file system when configuring a server instance. Specifying a path on network partition might cause worker processes to fail.
- All values for `mcr-root` must be for the same operating system and hardware combination.

For a server environment deployed in the cloud, the deployment sets the `mcr-root` property to support multiple MATLAB Runtime versions.

Parameters

path

Path to the root folder of the MATLAB Runtime installation.

Note The special value `mCRuNsETtOKEN` indicates to the `mps-start` command that a server is not configured to use a MATLAB Runtime. Running the `mps-start` command without configuring MATLAB Runtime results in an error.

Examples

- For on-premises server instances created using the command line, update the server configuration property in the `main_config` server configuration file.
- For on-premises server instances created using the dashboard, use the dashboard to update the server configuration property.

Configure Using Command Line

Use Latest Version of MATLAB Runtime

In the `main_config` file, set the `mcr-root` property to the following:

```
--mcr-root /usr/local/MATLAB/MATLAB_Runtime/R2023a
```

Use Two Versions of MATLAB Runtime

In the `main_config` file, set the `mcr-root` property to the following:

```
--mcr-root /usr/local/MATLAB/MATLAB_Runtime/R2023a
--mcr-root /usr/local/MATLAB/MATLAB_Runtime/v98
```

Configure Using Dashboard

Use Latest Version of MATLAB Runtime

In the dashboard, in the **Settings** tab of your server instance, under **Core**, for the **MATLAB Runtime** property, enter the following:

```
/usr/local/MATLAB/MATLAB_Runtime/R2023a
```

Use Two Versions of MATLAB Runtime

In the dashboard, in the **Settings** tab of your server instance, under **Core**, for the **MATLAB Runtime** property, enter the following:

```
/usr/local/MATLAB/MATLAB_Runtime/R2023a
/usr/local/MATLAB/MATLAB_Runtime/v98
```

See Also

`mps-setup`

Topics

“Specify MATLAB Runtime for Server Instance Using Command Line” on page 1-15
“Support Multiple MATLAB Runtime Versions” on page 1-17
“Supported MATLAB Runtime Versions for MATLAB Production Server”

num-threads

Number of request-processing threads within server instance

Syntax

```
--num-threads count
```

Description

`num-threads` sets the size of the thread pool available for a MATLAB Production Server instance to process client requests. Server instances do not allocate a unique thread to each client connection. Rather, when data is available on a connection, the required processing is scheduled on the pool of threads in the main server process.

The threads in this pool do not directly evaluate MATLAB functions. There is a single thread within each worker process that executes MATLAB code on behalf of the client.

Set this parameter to `1`, and increase it only if the expected load consists of a high volume of short-running requests. This strategy ensures that the available processor resources are balanced between MATLAB function evaluation and processing client-server requests. There is usually no benefit to increasing this parameter to more than the number of available cores.

Parameters

`count`

Number of threads available in the thread pool.

This value must be one or greater.

Examples

- For on-premises server instances created using the command line, update this server configuration property in the `main_config` server configuration file.
- For on-premises server instances created using the dashboard and for server deployments in the cloud, use the dashboard and cloud dashboard, respectively, to update the server configuration property.

Configure Using Command Line

Create Pool of Ten Threads for Processing Requests

In the `main_config` file, set the `num-threads` property to the following:

```
--num-threads 10
```

Configure Using Dashboard

Create Pool of Ten Threads for Processing Requests

In the dashboard, in the **Settings** tab of your server instance, under **Core**, enter the following for the **Maximum Threads** property:

```
10
```

See Also

`request-size-limit`

num-workers

Maximum number of workers allowed to process work simultaneously

Syntax

`--num-workers` *`count`*

Description

`num-workers` defines the number of concurrent MATLAB execution requests that a server instance can simultaneously process. It should correspond to the number of hardware threads available on the local host.

If you specify a single value for the mcr-root property, the `num-workers` property determines the fixed size of the worker pool. For example, if you specify a single value for `mcr-root`, and set `num-workers` to `4`, your machine has 4 workers available.

If you specify multiple MATLAB Runtime versions in your server configuration, `num-workers` specifies the maximum size of the worker subpool specific to a MATLAB Runtime version. When you specify multiple MATLAB Runtime versions, the total number of workers can exceed those specified by `num-workers`. However, at a maximum, only the number of workers that you specify in `num-workers` are allowed to process a request. For example, for a server configuration that has three MATLAB Runtime versions, and `num-workers` set to `2`, there a total of six workers. The maximum number of workers in each subpool is two, and the maximum number of workers that are active at a given time is two.

Parameters

`count`

Number of workers available to evaluate functions.

This value must be one or greater.

The maximum value is determined by the number of license keys available for MATLAB Production Server.

Examples

- For on-premises server instances created using the command line, update this server configuration property in the `main_config` server configuration file.
- For on-premises server instances created using the dashboard and for server deployments in the cloud, use the dashboard and cloud dashboard, respectively, to update the server configuration property.

Configure Using Command Line

Allow Ten Workers to Process Requests at a Time

In the `main_config` file, set the `num-workers` property to the following:

```
--num-workers 10
```

Configure Using Dashboard

Allow Ten Workers to Process Requests at a Time

In the dashboard, in the **Settings** tab of your server instance, under **Worker**, enter the following for the **Maximum Workers** property:

```
10
```

See Also

mcr-root | worker-restart-interval

Topics

“Support Multiple MATLAB Runtime Versions” on page 1-17

pid-root

Folder used to store PID files

Syntax

`--pid-root` *`path`*

Description

`--pid-root` *`path`* specifies the folder used to store PID files. PID files record the system-specific process identifiers for all processes associated with the server instance. This includes:

- `main.pid` — The process identifiers of the server's head process.
- `worker_`*`N`*`.pid` — The process identifiers of each worker process *`N`*.

In some circumstances, `worker_2.pid` may be present when `worker_1.pid` is not. This is a strong indication that `worker_1` crashed and was restarted automatically. You can confirm this by checking the main log file.

The format of these files is a single decimal integer, the process identifier.

Parameters

`path`

Path to the folder used to store PID files relative to the server instance's root folder.

Examples

Store PID files in the `pid` folder.

```
--pid-root ./pid
```

profile

Log server profile information

Syntax

```
--profile state object
```

Description

`profile` logs server profile information in the main log for an *`object`* when the *`state`* is `on`. The default *`state`* is `off`, where the server does not log any profile information. You can log profile information for multiple objects by specifying multiple profile properties.

To set up profiling options when you configure a server, specify the `profile` property. If you update the `profile` property for a server that is already running, you must restart the server for the update to take effect. To set or update the profiling options for an already running server without having to restart the server, use the `mps-profile` command. Running `mps-profile` overrides any profiling options set using the `profile` property.

Note Activating profiling has a negative impact on performance.

Parameters

`state`

Flag to control whether the server writes profile information to the main log. Valid states are:

- `on` — Log profile information.
- `off` — Do not log profile information.

`object`

Information to log. Valid objects are:

- `server` — Information about requests that the server receives and worker pool
- `server.request` — Information about server requests, which includes information about the requested archives and clients that make the requests
- `server.request.archive` — Information about archives in the request
- `server.request.client` — Information about clients that make the request
- `server.worker` and `server.worker.pool` — Information about the worker pool

The objects are hierarchical. For example, specifying `server.request` implies specifying `server.request.archive` and `server.request.client`.

If you do not specify an object, the server logs information for all the objects.

Examples

Log profile information for all the objects.

```
--profile on
```

Log profile information for all server requests only.

```
--profile on server.request
```

Log profile information for the clients in a request and workers.

```
--profile on server.request.client
--profile on server.worker
```

The following is an excerpt of the main log that contains profiling information for all objects.

```
93 [2020.03.19 13:05:56.554236] [profile] [client:[::1]:62736] [component:mymagic] [connection_id:2]
[function:magic] [mode:sync] [request_id:0:1:1][service:http-connection] [type:request_arrive]
Request arrived and was placed in the queue.
94 [2020.03.19 13:05:56.554236] [profile]
Request to allocate next available worker
95 [2020.03.19 13:05:56.555240] [profile]
Lease created for worker-1
96 [2020.03.19 13:05:56.555240] [profile] [client:[::1]:62736] [request_id:0:1:1] [type:request_start] [worker_id:1]
Request started executing on worker 1
...
99 [2020.03.19 13:05:56.558233] [profile] [client:[::1]:62736] [request_id:0:1:1] [type:request_finish] [worker_id:1]
Request completed with HTTP status 200
100 [2020.03.19 13:05:56.558233] [profile]
Lease terminated for worker-1
101 [2020.03.19 13:05:56.558233] [profile]
worker-1 PASSED health check; returning to the pool
```

Version History

R2020b: `requests` and `worker_pool` objects will be removed
Not recommended starting in R2020b

The objects `requests` and `worker_pool` will be removed in a future release. Use `server.request` instead of `requests` and `server.worker.pool` instead of `worker_pool` when specifying this property.

See Also

`mps-profile`

Topics

“Configure Server Using Configuration File” on page 1-6
“Log Files” on page 4-6
“View Logs” on page 9-30

request-size-limit

Set the maximum size of a request

Syntax

```
--request-size-limit  size
```

Description

`request-size-limit` specifies the maximum size of a request. The default request size is 64 MB. The maximum allowed request size is 2,147,483,647 bytes, which is the maximum positive value for a 32-bit signed integer.

Parameters

size

Size of the request, specified as an integer followed by an optional size unit. The default unit is bytes.

Examples

- For on-premises server instances created using the command line, update this server configuration property in the `main_config` server configuration file.
- For on-premises server instances created using the dashboard and for server deployments in the cloud, use the dashboard and cloud dashboard, respectively, to update the server configuration property.

Configure Using Command Line

Set Request Size to 128 MB

In the `main_config` file, set the `request-size-limit` property to the following:

```
--request-size-limit 128MB
```

Configure Using Dashboard

Set Request Size to 128 MB

In the dashboard, in the **Settings** tab of your server instance, under **Core**, enter the following for the **Request Size Limit** property:

```
128MB
```

See Also

`num-threads`

ssl-allowed-client

MATLAB programs a client can access

Syntax

`--ssl-allowed-client` *`client1`*`,...,`*`clientN`*`:`*`archive1`*`,...,`*`archiveN`*

Description

`ssl-allowed-client` authorizes clients based on the client certificate common name. Only authorized clients can request the evaluation of MATLAB functions deployed to the server.

If there are no archive names following the common name of the client, the client can access all the deployed archives. Otherwise, the client can access only the specified archives.

You can set either the `ssl-allowed-client` property or the access-control-provider property. If you set both, MATLAB Production Server fails to start.

Parameters

`client`

Common name of the client.

`archive`

Name of the deployed archive that the client can access.

Examples

Allow `client1` and `client2` to access `magic.ctf` and `helloworld.ctf`. Allow `client3` access to all deployed archives.

```
--ssl-allowed-client client1,client2:magic,helloworld
--ssl-allowed-client client3
```

See Also

https | x509-cert-chain | x509-private-key

Topics

“Specify Access to MATLAB Programs” on page 3-5
“Configure Client Authentication” on page 3-4

enable-ssl-renegotiation

Enable SSL renegotiation

Syntax

```
--enable-ssl-renegotiation
```

Description

`--enable-ssl-renegotiation` enables SSL renegotiation on your MATLAB Production Server instance. When server authentication requirements change, SSL renegotiation lets clients maintain their connection to the server by enabling them to re-enter their authentication details.

Caution Enabling SSL renegotiation exposes security vulnerabilities.

By default, SSL renegotiation is disabled.

Examples

In the `main_config` file, uncomment the following line.

```
--enable-ssl-renegotiation
```

Version History

Introduced in R2023a

See Also

Topics

“Configure Client Authentication” on page 3-4

ssl-ciphers

List of cipher suites used for encryption

Syntax

`--ssl-ciphers` *`ciphers`*

Description

`ssl-ciphers` provides a list of cipher suites that the server uses for encryption.

Parameters

`ciphers`

Cipher suites the server instance uses for encryption. Valid values are:

- `ALL` — Use all available cipher suites except eNULL.
- `HIGH` — Use all available high encryption cipher suites.
- *`list`* — Comma-separated list of cipher suites to use.

All OpenSSL configuration strings can be passed with the ciphers. This provides finer control over the selected cipher.

Examples

Use only high encryption cipher suites.

```
--ssl-ciphers HIGH
```

Disable the use of ADH ciphers.

```
--ssl-ciphers ALL:!ADH
```

Use the strongest available ECDHE ciphers.

```
--ssl-ciphers ALL:@STRENGTH
```

Disable the use of ADH ciphers and use the strongest available ECDHE ciphers.

```
--ssl-ciphers ALL:!ADH@STRENGTH
```

See Also

https | ssl-protocols

Topics

“Enable HTTPS” on page 3-2
“Adjust Security Protocols” on page 3-6

ssl-protocols

List of allowed SSL protocols

Syntax

`--ssl-protocols` *`protocols`*

Description

`ssl-protocols` lists the allowed SSL protocols. If you do not set this property, the server allows the use of all supported SSL protocols. Supported protocols are TLSv1, TLSv1.1, and TLSv1.2. The default server behavior is to attempt to use TLSv1.2.

Starting in R2019b, SSLv3 is no longer supported.

Parameters

`protocols`

Comma-separated list of allowed protocols. Valid entries are:

- `TLSv1`
- `TLSv1.1`
- `TLSv1.2`

Examples

- For on-premises server instances created using the command line, update this server configuration property in the `main_config` server configuration file.
- For on-premises server instances created using the dashboard and for server deployments in the cloud, use the dashboard and cloud dashboard, respectively, to update the server configuration property.

Configure Using Command Line

Allow Only TLSv1

In the `main_config` file, set the `access-control-provider` property to the following:

```
--ssl-protocols TLSv1
```

Configure Using Dashboard

Allow Only TLSv1

In the dashboard, in the **Settings** tab of your server instance, under **SSL**, enter the following for the **SSL Protocols** property:

```
TLSv1
```

See Also

https | ssl-ciphers

Topics

“Enable HTTPS” on page 3-2
“Adjust Security Protocols” on page 3-6

ssl-tmp-ec-param

Elliptic curve used for the ECDHE ciphers

Syntax

`--ssl-tmp-ec-param` *`elliptic_curve_name`*

Description

`--ssl-tmp-ec-param` *`elliptic_curve_name`* specifies the name of the elliptic curve used for the ECDHE ciphers.

Starting in R2019b, ECDHE ciphers are enabled by default. If you do not specify the elliptic curve name, ECDHE ciphers use a default elliptic curve. The default elliptic curves are in the following order: x25519, secp256r1, x448, secp521r1, secp384r1. During the SSL/TLS handshake, the client advertises the curves that it supports. Based on this client-server negotiation, one of the default curves is used to establish a secure connection for the subsequent data exchange.

For earlier releases, if this property is not specified, all ECDHE ciphers are disabled.

Parameters

`elliptic_curve_name`

Name of curve. All curves supported by OpenSSL are supported.

Examples

Use the prime256v1 curve.

```
--ssl-tmp-ec-param prime256v1
```

ssl-tmp-dh-param

File containing a pregenerated ephemeral DH key

Syntax

`--ssl-tmp-dh-param` *`path`*

Description

`ssl-tmp-dh-param` specifies the path to the pre-generated ephemeral DH key. If this parameter is not provided, the server instance automatically generates the DH key at start-up. Providing a pre-generated DH key can decrease instance start time.

Parameters

`path`

Path to the pre-generated DH key. Relative and absolute paths are valid.

Examples

The instance loads the DH key from `dh_param.pem` which is located at *`instance_root`*`/x509`.

```
--ssl-tmp-dh-param ./x509/dh_param.pem
```

ssl-verify-peer-mode

Level of client verification required by the server instance

Syntax

```
--ssl-verify-peer-mode mode
```

Description

`ssl-verify-peer-mode` specifies whether the server requires clients to present a valid certificate to connect to it. Server instances allow clients to connect to it with or without providing a valid certificate. All requests will still require authorization.

If you set `ssl-verify-peer-mode` to `verify-peer-require-peer-cert`, you must set either the x509-ca-file-store or x509-use-system-store property.

Parameters

mode

Mode used to authenticate clients. Valid values are:

- `no-verify-peer` — No peer certificate verification. The client side does not need to provide a certificate.
- `verify-peer-require-peer-cert` — The client must provide a certificate and the certificate will be verified.

The default is `no-verify-peer`.

Examples

- For on-premises server instances created using the command line, update this server configuration property in the `main_config` server configuration file.
- For on-premises server instances created using the dashboard and for server deployments in the cloud, use the dashboard and cloud dashboard, respectively, to update the server configuration property.

Configure Using Command Line

Require Clients to Provide SSL Certificate

In the `main_config` file, set the `ssl-verify-peer-mode` property to the following:

```
--ssl-verify-peer-mode verify-peer-require-peer-cert
```

Configure Using Dashboard

Require Clients to Provide SSL Certificate

In the dashboard, in the **Settings** tab of your server instance, under **SSL**, enter the following for the **SSL Verify Peer Mode** property:

`verify-peer-require-peer-cert`

See Also

https | x509-use-system-store | x509-ca-file-store | x509-use-crl

Topics

“Configure Client Authentication” on page 3-4

use-single-comp-thread

Start MATLAB Runtime with a single computational thread

Syntax

```
--use-single-comp-thread
```

Description

`--use-single-comp-thread` specifies that workers start the MATLAB Runtime with a single computational thread.

Examples

- For on-premises server instances created using the command line, update this server configuration property in the `main_config` server configuration file.
- For on-premises server instances created using the dashboard and for server deployments in the cloud, use the dashboard and cloud dashboard, respectively, to update the server configuration property.

Configure Using Command Line

Start MATLAB Runtime With Single Computational Thread

In the `main_config` file, enable the `use-single-comp-thread` property:

```
--use-single-comp-thread
```

Configure Using Dashboard

Start MATLAB Runtime With Single Computational Thread

In the dashboard, in the **Settings** tab of your server instance, under **Core**, select the **Use Single Computational Thread** property.

See Also

num-threads

worker-memory-check-interval

Interval at which workers are polled for memory usage

Syntax

```
--worker-memory-check-interval hr:min:sec.fractSec
```

Description

`worker-memory-check-interval` specifies how often to poll the memory usage of a worker process. This setting affects the behavior of all other settings that act based on worker memory usage such as `worker-memory-trigger`, `worker-memory-target`, and `worker-restart-memory-limit`.

Parameters

hr

Hours in interval.

min

Minutes in interval.

sec

Seconds in interval.

fractSec

Fractional seconds in interval.

Examples

Check memory usage every one and a half minutes.

```
--worker-memory-check-interval 0:01:30
```

See Also

`worker-restart-memory-limit-interval` | `worker-restart-memory-limit`

Topics

“Control Worker Restarts” on page 1-20

worker-restart-interval

Time interval at which server instance stops and restarts workers

Syntax

```
--worker-restart-interval hr:min:sec.fractSec
```

Description

`worker-restart-interval` specifies the time interval at which the server instance stops and restarts its worker processes. If a worker process has not completed request processing at the specified restart interval, it completes the processing and then restarts.

If you do not specify this property, the workers do not restart in response to time.

Parameters

hr

Hours in interval.

min

Minutes in interval.

sec

Seconds in interval.

fractSec

Fractional seconds in interval.

Examples

Restart workers at intervals of 1 hour, 29 minutes, 5 seconds.

Update the `main_config` server configuration file with the following:

```
--worker-restart-interval 1:29:05
```

Restart workers at intervals of 10 minutes and 250 ms.

Update the `main_config` server configuration file with the following:

```
--worker-restart-interval 00:10:00.25
```

See Also

num-workers | worker-restart-memory-limit | worker-restart-memory-limit-interval

Topics

“Control Worker Restarts” on page 1-20

worker-restart-memory-limit

Size threshold at which to consider restarting a worker

Syntax

```
--worker-restart-memory-limit size
```

Description

`worker-restart-memory-limit` sets the memory usage limit of a worker process. If a worker's working set size exceeds `worker-restart-memory-limit` for an interval of time greater than `worker-restart-memory-limit-interval`, then that worker is restarted.

Parameters

size

Amount of memory used by worker.

Examples

Restart any worker whose working set size exceeds 1 GB for more than 1 hour.

```
--worker-restart-memory-limit 1GB
--worker-restart-memory-limit-interval 1:00:00
```

See Also

`worker-restart-memory-limit-interval` | `worker-memory-check-interval`

Topics

“Control Worker Restarts” on page 1-20

worker-restart-memory-limit-interval

Interval for which a worker can exceed its memory limit before restart

Syntax

```
--worker-restart-memory-limit-interval hr:min:sec.fractSec
```

Description

`worker-restart-memory-limit-interval` sets the interval for which a worker process can exceed its memory limit before restart. If a worker's working set size exceeds `worker-restart-memory-limit` for an interval of time greater than `worker-restart-memory-limit-interval`, then that worker is restarted.

Parameters

hr

Hours in interval.

min

Minutes in interval.

sec

Seconds in interval.

fractSec

Fractional seconds in interval.

Examples

Restart any worker whose working set size exceeds 1 GB for more than 1 hour.

```
--worker-restart-memory-limit 1GB
--worker-restart-memory-limit-interval 1:00:00
```

See Also

`worker-restart-memory-limit` | `worker-memory-check-interval` | `mps-status`

Topics

“Control Worker Restarts” on page 1-20

x509-ca-file-store

File containing the server certificate authority file

Syntax

`--x509-ca-file-store` *path*

Description

`x509-ca-file-store` specifies the certificate authority (CA) file to verify peer certificates. This file contains trusted certificates and certificate revocation lists.

You can also put intermediate certificates into the CA file. An intermediate certificate in the CA file becomes a trusted certificate.

Parameters

path

Path to the certificate CA file store. Relative and absolute paths are valid.

Examples

- For on-premises server instances created using the command line, update this server configuration property in the `main_config` server configuration file.
- For on-premises server instances created using the dashboard and for server deployments in the cloud, use the dashboard and cloud dashboard, respectively, to update the server configuration property.

Configure Using Command Line

Load CA store from `ca_file.pem` which is located at *`instance_root`*`/x509`

In the `main_config` file, set the `x509-ca-file-store` property to the following:

```
--x509-ca-file-store ./x509/ca_file.pem
```

Configure Using Dashboard

Load CA store from `ca_file.pem` which is located at *`instance_root`*`/x509`

In the dashboard, in the **Settings** tab of your server instance, under **SSL**, enter the following for the **X509 CA File Store** property:

```
./x509/ca_file.pem
```

See Also

https | x509-use-system-store | x509-use-crl | ssl-verify-peer-mode

Topics

“Configure Client Authentication” on page 3-4

x509-cert-chain

File containing the server certificate chain

Syntax

```
--x509-cert-chain path
```

Description

`x509-cert-chain` specifies the server certificate chain file. It contains one or more PEM-format certificates. The chain begins with the server certificate. The server certificate is followed by a chain of untrusted certificates. To use the certificate chain file, specify the x509-private-key.

Starting in R2019b, if https is enabled on the server, you must set the `x509-cert-chain` and `x509-cert-chain` properties; otherwise, the server fails to start.

Note Do not specify trusted certificates in the certificate chain file.

Parameters

path

Path to the certificate chain file. Relative and absolute paths are valid.

Examples

- For on-premises server instances created using the command line, update this server configuration property in the `main_config` server configuration file.
- For on-premises server instances created using the dashboard and for server deployments in the cloud, use the dashboard and cloud dashboard, respectively, to update the server configuration property.

Configure Using Command Line

In the `main_config` file, set the `x509-cert-chain` property to the following:

The instance loads the certificate chain from `cert_chain.pem` which is located at *`instance_root`*`/x509`.

```
--x509-cert-chain ./x509/cert_chain.pem
```

Configure Using Dashboard

In the dashboard, in the **Settings** tab of your server instance, under **SSL**, enter the following for the **X509 Certificate Chain** property:

```
./x509/cert_chain.pem
```

See Also

https | x509-private-key

Topics

“Enable HTTPS” on page 3-2

x509-passphrase

File containing the passphrase that decodes the private key

Syntax

`--x509-passphrase` *`path`*

Description

`x509-passphrase` specifies the path to the file containing the passphrase of the encrypted private-key. This is required if `x509-private-key` is specified and the private key file is encrypted. Otherwise, the private key fails to load.

Note This file must be owner read-only.

Parameters

`path`

Path to the passphrase file. Relative and absolute paths are valid.

Examples

The instance loads the passphrase from `key_passphrase.pem` which is located at *`instance_root`*`/x509`.

```
--x509-passphrase ./x509/key_passphrase.pem
```

x509-private-key

File containing the private key in PEM format

Syntax

`--x509-private-key` *`path`*

Description

`x509-private-key` specifies the path to the private key. The key must be in PEM format.

If you do not set this property, the server instance does not load the private key or the server-side certificates.

Starting in R2019b, if https is enabled on the server, you must set the `x509-private-key` and `x509-cert-chain` properties; otherwise, the server fails to start.

Parameters

`path`

Path to the PEM-format private key file. Relative and absolute paths are valid.

Examples

- For on-premises server instances created using the command line, update this server configuration property in the `main_config` server configuration file.
- For on-premises server instances created using the dashboard and for server deployments in the cloud, use the dashboard and cloud dashboard, respectively, to update the server configuration property.

Configure Using Command Line

In the `main_config` file, set the `x509-private-key` property to the following:

The instance loads the private key from `private_key.pem`, which is located at *`instance_root`*`/x509`.

```
--x509-private-key ./x509/private_key.pem
```

Configure Using Dashboard

In the dashboard, in the **Settings** tab of your server instance, under **SSL**, enter the following for the **X509 Private Key** property:

```
./x509/private_key.pem
```

See Also

https | x509-cert-chain

Topics
“Enable HTTPS” on page 3-2

x509-use-crl

Use the certificate revocation list

Syntax

```
--x509-use-crl
```

Description

`x509-use-crl` specifies that the server instance uses the certificate revocation list (CRL). By default, instances do not use any CRLs. In this case, the CRLs in the certificate authority store are ignored.

If `x509-use-crl` is added, the CRLs are loaded and participate in the client certificate verification. If the CRL has expired, the SSL handshake is rejected.

Examples

The instance uses certificate revocation list when authenticating clients.

```
--x509-use-crl
```

See Also

https | ssl-verify-peer-mode | x509-ca-file-store | x509-use-system-store

Topics

“Configure Client Authentication” on page 3-4

x509-use-system-store

Use the certificate authority store provided by the system

Syntax

```
--x509-use-system-store
```

Description

`x509-use-system-store` specifies that the server instance uses the system provided certificate authority (CA) store. By default, the server uses the file `/etc/ssl/certs/ca-certificates.crt` as trusted CA store and searches for trusted certificates under the folder `/etc/ssl/certs`. You can override these locations by setting the environment variables `SSL_CERT_FILE` and `SSL_CERT_DIR`.

Examples

- For on-premises server instances created using the command line, update this server configuration property in the `main_config` server configuration file.
- For on-premises server instances created using the dashboard and for server deployments in the cloud, use the dashboard and cloud dashboard, respectively, to update the server configuration property.

Configure Using Command Line

Use System CA Store

In the `main_config` file, enable the `x509-ca-file-store` property to the following:

```
--x509-use-system-store
```

Configure Using Dashboard

Use System CA Store

In the dashboard, in the **Settings** tab of your server instance, under **SSL**, select the **X509 Use System Store** property.

See Also

https | x509-ca-file-store | x509-use-crl | ssl-verify-peer-mode

Topics

“Configure Client Authentication” on page 3-4

request-timeout

Duration after which the request times out and gets deleted after reaching a terminal state

Syntax

`--request-timeout` *`hr`*`:`*`min`*`:`*`sec`*`.`*`fractSec`*

Description

`request-timeout` specifies the duration after which the request times out upon reaching a terminal state. At this point, the request gets deleted unless a client process has already deleted the request.

Parameters

`hr`

Hours in interval.

`min`

Minutes in interval.

`sec`

Seconds in interval.

`fractSec`

Fractional seconds in interval.

Examples

- For on-premises server instances created using the command line, update this server configuration property in the `main_config` server configuration file.
- For on-premises server instances created using the dashboard and for server deployments in the cloud, use the dashboard and cloud dashboard, respectively, to update the server configuration property.

Configure Using Command Line

Set Request to Time Out After Two Hours

In the `main_config` file, set the `request-timeout` property to the following:

```
--request-timeout 2:00:00
```

Configure Using Dashboard

Set Request to Time Out After Two Hours

In the dashboard, in the **Settings** tab of your server instance, under **Core**, enter the following for the **Request Timeout** property:

```
2:00:00
```

Version History

Introduced in R2016b

See Also

`server-memory-threshold`

server-memory-threshold

Size threshold of server process at which action needs to be taken to manage responses

Syntax

```
--server-memory-threshold SIZE
```

Description

`server-memory-threshold` sets the memory size limit of a server process. If the size of a server process size exceeds `SIZE`, then responses need to be either archived or purged by setting `server-memory-threshold-overflow-action` to `archive_responses` or `purge_responses`, respectively. If `server-memory-threshold` is not set, then the responses will be bound to the server process causing the memory footprint of the server process to keep increasing. As a best practice, it is recommended that a client process delete a request after usage in order to prevent the memory of the server process from growing.

Parameters

`size`

Threshold size of the server process.

Examples

- For on-premises server instances created using the command line, update this server configuration property in the `main_config` server configuration file.
- For on-premises server instances created using the dashboard and for server deployments in the cloud, use the dashboard and cloud dashboard, respectively, to update the server configuration property.

Configure Using Command Line

Archive Responses if Size of Server Process in Memory Exceeds 500 MB

In the `main_config` file, set the `server-memory-threshold` and `server-memory-threshold-overflow-action` properties to the following:

```
--server-memory-threshold 500MB
--server-memory-threshold-overflow-action archive_responses
```

Purge Responses if Size of Server Process in Memory Exceeds 2 GB

In the `main_config` file, set the `server-memory-threshold` and `server-memory-threshold-overflow-action` property to the following:

```
--server-memory-threshold 2GB
--server-memory-threshold-overflow-action purge_responses
```

Configure Using Dashboard

Archive Responses if Size of Server Process in Memory Exceeds 500 MB

In the dashboard, in the **Settings** tab of your server instance, under **Server Memory**:

- Enter the following for the **Server Memory Threshold** property:

  ```
  500MB
  ```
- Select the following for the **Server Memory Threshold Overflow Action** property:

  ```
  archive_responses
  ```

Purge Responses if Size of Server Process in Memory Exceeds 2 GB

In the dashboard, in the **Settings** tab of your server instance, under **Server Memory**:

- Enter the following for the **Server Memory Threshold** property:

  ```
  2GB
  ```
- Select the following for the **Server Memory Threshold Overflow Action** property:

  ```
  purge_responses
  ```

See Also

`server-memory-threshold-overflow-action`

server-memory-threshold-overflow-action

Action taken when the memory size threshold of server process is breached

Syntax

```
--server-memory-threshold-overflow-action ACTION
```

Description

`server-memory-threshold-overflow-action` acts by either archiving responses or purging them when the size of the server process in memory set by `server-memory-threshold` is breached.

Parameters

ACTION

`archive_responses`

`purge_responses`

Examples

Archive responses if the size of the server process in memory exceeds 500 MB.

```
--server-memory-threshold 500MB
--server-memory-threshold-overflow-action archive_responses
```

Purge responses if the size of the server process in memory exceeds 2 GB.

```
--server-memory-threshold 2GB
--server-memory-threshold-overflow-action purge_responses
```

See Also

`server-memory-threshold`

server-termination-grace-period

Duration after which all server instance processes are forcibly terminated

Syntax

```
--server-termination-grace-period hr:min:sec.fractSec
```

Description

`server-termination-grace-period` *hr*:*min*:*sec*.*fractSec* specifies the time interval after which all running server and worker processes are forcibly terminated on receipt of the `mps-stop` command, if they have not already stopped within *hr*:*min*:*sec*.*fractSec*.

If you specify the `--timeout` option when running the `mps-stop` command and have also set the `server-termination-grace-period` *hr*:*min*:*sec* property, server instance processes are forcibly terminated at *hr*:*min*:*sec*.

If you specify the both the `-k` and `--timeout` options when running the `mps-stop` command and have also set the `server-termination-grace-period` *hr*:*min*:*sec* property, server instance processes are forcibly terminated within the duration specified by the `--timeout` value.

Parameters

hr

Hours.

min

Minutes.

sec

Seconds.

fractSec

Fractional seconds.

Examples

Forcibly terminate server instance processes after 1 hour, 29 minutes, and 5 seconds.

```
--server-termination-grace-period 1:29:05
```

Forcibly terminate server instance processes after 10 minutes and 250 ms.

```
--server-termination-grace-period 00:10:00.25
```

Version History

Introduced in R2020a

See Also

mps-stop

response-archive-root

Path to the location where responses are archived

Syntax

```
--response-archive-root PATH
```

Description

`response-archive-root` shows the location where responses specified by *PATH* are archived. This option must be set if the server-memory-threshold-overflow-action option is set to `archive_responses`. The server process must have read & write permissions to the *PATH*.

Parameters

PATH

Location specified as a string.

Examples

Set the archive root.

```
--response-archive-root ./.response_archive
```

See Also

`response-archive-limit`

response-archive-limit

Maximum disk space available to the server process for archiving

Syntax

```
--response-archive-limit SIZE
```

Description

`response-archive-limit` specifies the maximum disk space available to the server process for archiving. If the limit set by *`SIZE`* is reached, the archives will be deleted in a 'First-In First-Out' order until the space for the server process fall below *`SIZE`*. If this limit is not specified, the server process will assume there is no limit to disk usage for archiving.

Parameters

`size`

Size of the server process.

Examples

Set the size of the archive to be 5GB

```
--response-archive-limit 5GB
```

Version History

Introduced in R2016b

See Also

`response-archive-root`

user-data

Associate MATLAB data value with string key

Syntax

`--user-data` *`KEY VALUE`*

Description

`user-data` associates MATLAB data value with key string. *`KEY`* and *`VALUE`* are strings. Use the double quotes (") character around strings with spaces. The backslash (\) character is the escape character and is used to insert double quotes or backslash characters: `\"` `\\`. The application can retrieve the data value by using `getmcruserdata(key)`.

Parameters

`KEY VALUE`

MATLAB *`value`* to be associated with *`key`*.

Examples

Set user data with parallel profile settings.

```
--user-data ParallelProfile c:\\MPS\\myprofile.settings
```

Use quotes.

```
--user-data MyValue "Quoted string with escaped \"quotes\" and \\backslash."
```

Version History

Introduced in R2016a

Cloud Deployment

Azure Deployment for MATLAB Production Server (BYOL)

To deploy your MATLAB Production Server bring your own license (BYOL) environment on Azure, launch the MATLAB Production Server solution template from the Azure marketplace. After the deployment to Azure is complete, upload your license file using the Network License Manager for MATLAB dashboard, and configure and manage MATLAB Production Server using the MATLAB Production Server dashboard.

For information about deploying MATLAB Production Server on Azure without a license, see “Azure Deployment for MATLAB Production Server (PAYG)” on page 9-9.

Note You must have an Azure account to deploy resources on Azure and configure your server environment.

To use your solution, you need a MATLAB Production Server license. You can upload the MATLAB Production Server license file only after provisioning the cloud resources. For more information on licensing on the cloud, see “Configure MATLAB Production Server License for Use on the Cloud”.

You are responsible for the cost of the Azure services and resources that the deployment uses.

Follow these steps to deploy your server environment on Azure.

1. In the Azure marketplace, on the MATLAB Production Server (BYOL) offering page, click **Create**. Doing so launches the solution template where you provide values to configure your server environment.
2. “Provision Cloud Resources” on page 9-2. Creating resources on Azure can take up to 30 minutes.
3. “Upload License File” on page 9-7.
4. “Connect and Log In to Dashboard” on page 9-8.

To run an application on MATLAB Production Server, you need to create the application using MATLAB Compiler SDK. For more information, see “Create Deployable Archive for MATLAB Production Server”.

Provision Cloud Resources

You must have an Azure subscription before you can start deploying cloud resources for MATLAB Production Server on Azure. Launch the MATLAB Production Server solution template to configure and deploy cloud resources. The deployment process goes through the following steps. Click **OK** at the end of each step to proceed to the next step.

Basics

First, you must choose an Azure subscription, specify a resource group to hold the resources you provision, and specify a location to start your resources in.

Parameter Name	Value
Subscription	Choose an Azure subscription to use for purchasing resources.

Parameter Name	Value
Resource group	Choose a name for the resource group to hold the resources. Creating a new resource group for each deployment is recommended. Doing so enables you to delete all the resources for each deployment easily.
Region	Choose the region to start resources in. Select a region that supports your requested virtual machine (VM) instance types. For a list of resources that are supported by region, see Products available by region.

Server

Next, you configure the server VM and data persistence. Each MATLAB Production Server instance runs on a VM and each instance runs multiple MATLAB Production Server workers. To deploy a server instance, you must specify parameters for the VM, such as the size, number of VMs, and operating system.

Parameter Name	Value
Server VM Size	Specify the size of the VM to use for the server. It is recommended that you select a VM size where the number of vCPUs on your VM matches the number of MATLAB Production Server workers per VM that you plan on using. For example, if you select a `D8s V3` VM which has 8 vCPUs, use 8 MATLAB Production Server workers. You can change the number of workers by updating the **Number of Workers** in the **Settings** tab in the MATLAB Production Server dashboard. For more information about the property, see num-workers. The template defaults to `Standard_D4s_v3`. This configuration has 4 vCPUs and 16 GB of memory. For more information on Azure VMs, see Azure VM sizes.

Parameter Name	Value
Number of Server VMs	Specify the number of VMs to run MATLAB Production Server instances. The deployment template sets the default to 2 VMs for load balancing. If you have a standard 24 worker MATLAB Production Server license and select `Standard_D4s_v3 VM` (4 vCPUs) as the **Server VM Size**, you need 6 VMs to fully utilize the MATLAB Production Server workers in your license. You can always underprovision the number of VMs. If you do so, you can end up using fewer workers than you are licensed for. You can change the number of VMs after the initial deployment. For more information, see “Change the Number of Virtual Machines” on page 9-30.
Server VM Operating System	Choose the server VM operating system. Windows (Windows Server) and Linux (Ubuntu®) are the only available options. In most cases, choosing an operating system depends on your personal preference. Unless you add operating system-specific dependencies or content such as MEX files to your applications, the applications you deploy to the server do not depend on a specific operating system.
Create Redis Cache	Choose whether you want to create an Azure cache for Redis. Creating this service allows you to use the persistence functionality of the server. Persistence provides a mechanism to cache data between calls to MATLAB code running on a server instance. For more information, see “Data Caching Basics”. You can provision an Azure cache for Redis after the initial deployment. For setup instructions, see Azure documentation.

License Server

The network license manager manages the license files for MATLAB Production Server (BYOL). You can choose to deploy a license server for the Network License Manager for MATLAB or use an existing license server. For information about the Network License Manager for MATLAB, see Network License Manager for MATLAB on Microsoft Azure.

If you use an existing license server, use the MATLAB Production Server dashboard to enter the port number and IP address of the license server. For more information, see “Upload License File” on page 9-7.

Parameter Name	Value
Create License Server	Specify whether you want to deploy the Network License Manager for MATLAB to upload and mange your license files. The license manager runs on a separate Windows VM. If you want to use an existing network license manager, choose `No`. If you do not have an existing license manager, choose `Yes`.

Dashboard Login

After you deploy the server VMs, you can manage the server using the MATLAB Production Server dashboard, which provides a web-based interface to configure and manage MATLAB Production Server in the cloud. Specify the login credentials for the dashboard.

If you deploy the license manager with this deployment, use these same credentials to log in to the Network License Manager for MATLAB dashboard.

Parameter Name	Value
Admin Username	Specify the administrator user name to log in to the MATLAB Production Server dashboard.
Admin Password	Specify the administrator password to log in to the MATLAB Production Server dashboard.

Network

You can specify which IP addresses can access the dashboard, whether your solution should use a public IP address, and configure a virtual network (VNet).

Parameter Name	Value
Allow Connections from IP address	Specify the range of IP addresses that are permitted to connect to the dashboard that manages the server. Use CIDR notation, which provides the IP address before the slash and mask after the slash, when specifying the IP address range. The mask determines the number of IP addresses to include. For example, 10.0.0.1/24. You can use a CIDR calculator to determine the CIDR notation for a range of IP addresses.

Parameter Name	Value
Make Solution Available over Internet	Make your solution available over the Internet by setting this parameter to `Yes`. This will use public IP addresses. If you set this parameter to `No`, the ARM template does not assign a public IP address for the VM that hosts the dashboard. To access the dashboard, you can use a different VM located in the same virtual network as the VM that hosts the dashboard.
Virtual Network	Create a new VNet or choose an existing one. The template defaults to a creating a new VNet with predefined values. These are the values that Azure defaults to while creating a new VNet. You can use the default values or enter new values based on your network setup. The deployment opens the following ports. If you are using an existing VNet, you need to open the following ports. • `443` – Required for communicating with the dashboard. • `8000`, `8004`, `8080`, `9090`, and `9910` – Required for communication between the dashboard, MATLAB Production Server workers, and various microservices within the virtual network. These ports do not need to be accessible over the Internet. • `27000` – Required for communication between the network license manager and the MATLAB Production Server workers. • `65200–65535` – Required for the Azure application gateway health check to work. These ports need to be accessible over the Internet. For more information, see MSDN Community. • `22`, `3389` – Required for Remote Desktop functionality. You can use remote login functionality for troubleshooting and debugging. You can close these ports after deployment is complete.

Parameter Name	Value
First Subnet **Second Subnet**	Specify the subnet name and subnet address prefix for a new or existing subnet. The first subnet hosts the dashboard. The second subnet hosts the application gateway. The template defaults to creating new subnets with predefined values. You can use the default values or enter new values based on your network setup.

SSL Certificate

In this section, you specify an SSL certificate for the Azure application gateway to use. The application gateway provides an HTTPS endpoint that you use to connect to server instances and the MATLAB Production Server dashboard.

Parameter Name	Value
PFX Certificate	Upload a PFX-formatted SSL certificate.
PFX Certificate Password	If the certificate requires a password, enter it here. Otherwise, leave the field blank.

Upload License File

The network license manager manages the MATLAB Production Server license file. The MATLAB Production Server deployment template provides an option to use an existing license manager or deploy the Network License Manager for MATLAB. For information about the Network License Manager for MATLAB, see the documentation on GitHub® for Network License Manager for MATLAB on Microsoft Azure.

If you want to use an existing license manager, the VM that hosts the license server must be in the same VNet as the VM that hosts MATLAB Production Server. Use the MATLAB Production Server dashboard to enter the port number and IP address of the license server. Enter the license server details in the **Advanced** > **License Server** field on the **Settings** tab of the dashboard.

If you choose to deploy a new license server, you do not need to enter the license server details in the dashboard. The deployment template prepopulates default values for port number and IP address in the **License Server** field.

You need a fixed MAC address to get a license file from the MathWorks® License Center. The license server MAC address is available only after the deployment is complete. For more information on configuring your license for use on Azure, see “Configure MATLAB Production Server License for Use on the Cloud”.

Follow these steps to upload the license file using the Network License Manager for MATLAB dashboard, if you have deployed the license manager during the MATLAB Production Server deployment process.

1. In the Azure Portal, click **Resource groups** and select the resource group containing your cluster resources.

2. Select **Deployments** from the left pane and click the deployment name that has the text `mathworks-inc.matlabprodserver`.
3. Select **Outputs** from the left pane.
4. Copy the parameter value for **networkLicenseManagerURL** and paste it in a browser to launch the dashboard.
5. Log in using the administrator user name and password that you specified during the deployment process.
6. Follow the instructions in the Network License Manager for MATLAB dashboard to upload your MATLAB Production Server license.

Connect and Log In to Dashboard

The MATLAB Production Server dashboard is a web-based interface to configure and manage server instances in the cloud.

Note The Internet Explorer® web browser is not supported for interacting with the dashboard.

Complete these steps only after you successfully create your resource group. This workflow assumes that your solution uses public IP addresses. If your solution uses private IP addresses, you can connect to the dashboard from a VM located in the same virtual network as the VM that hosts the dashboard.

1. In the Azure Portal, click **Resource groups** and select the resource group containing your cluster resources
2. Select **Deployments** from the left pane. In the pane that opens, click the deployment name that has the text `mathworks-inc.matlabprodserver`.
3. Select **Outputs** from the left pane.
4. Copy the parameter value for **dashboardAdminURL** and paste it in a browser to launch the dashboard.
5. Log in using the administrator user name and password that you specified during the deployment process.

Configuring role-based access control for the dashboard is recommended. MATLAB Production Server integrates with OAuth 2.0 providers such as Microsoft Azure Active Directory, Google® Identity, and PingFederate for providing role-based access control. Role-based access control lets you grant users the privileges to perform tasks on the dashboard and server, based on their role. For more information on how to configure role-based access control, see “Dashboard Access Control” on page 9-51.

See Also

More About

- “Architecture and Resources on Azure” on page 9-36
- “Manage MATLAB Production Server (BYOL)” on page 9-15
- “Manage Azure Resources for MATLAB Production Server (BYOL)” on page 9-30

Azure Deployment for MATLAB Production Server (PAYG)

To deploy your MATLAB Production Server (PAYG) environment on Azure, launch the MATLAB Production Server solution template from the Azure marketplace. After the deployment to Azure is complete, configure and manage MATLAB Production Server by logging in to the MATLAB Production Server (PAYG) dashboard.

For information about deploying MATLAB Production Server on Azure with a license, see “Azure Deployment for MATLAB Production Server (BYOL)” on page 9-2.

Note You must have an Azure account to deploy resources on Azure and configure your server environment.

You are responsible for the cost of the Azure services and resources that the deployment uses.

Follow these steps to deploy your server environment on Azure.

1. In the Azure marketplace, on the MATLAB Production Server (PAYG) offering page, click **Get It Now**. Doing so launches the solution template where you provide values to configure your server environment.
2. “Provision Cloud Resources” on page 9-9. Creating resources on Azure can take up to 30 minutes.
3. “Connect and Log In to Dashboard” on page 9-14.

To run an application on MATLAB Production Server, you need to create the application using MATLAB Compiler SDK. For more information, see “Create Deployable Archive for MATLAB Production Server”.

Provision Cloud Resources

You must have an Azure subscription before you can start deploying cloud resources for MATLAB Production Server on Azure. Launch the MATLAB Production Server solution template to configure and deploy cloud resources. The deployment process goes through the following steps. Click **OK** at the end of each step to proceed to the next step.

Basics

First, you must choose an Azure subscription, specify a resource group to hold the resources you provision, and specify a location to start your resources in.

Parameter Name	Value
Subscription	Choose an Azure subscription to use for purchasing resources.

Parameter Name	Value
Resource group	Choose a name for the resource group to hold the resources. Creating a new resource group for each deployment is recommended. Doing so enables you to delete all the resources for each deployment easily.
Region	Choose the region to start resources in. Select a region that supports your requested virtual machine (VM) instance types. For a list of resources that are supported by region, see Products available by region.

Server

Next, you configure the server VM and data persistence. Each MATLAB Production Server instance runs on a VM and each instance runs multiple MATLAB Production Server workers. To deploy a server instance, you must specify parameters for the VM, such as the size, number of VMs, and operating system.

Parameter Name	Value
Server VM Size	Specify the size of the VM to use for the server. It is recommended that you select a VM size where the number of vCPUs on your VM matches the number of MATLAB Production Server workers per VM that you plan on using. For example, if you select a `D8s V3` VM which has 8 vCPUs, use 8 MATLAB Production Server workers. You can change the number of workers by updating the **Number of Workers** in the **Settings** tab in the MATLAB Production Server dashboard. For more information about the property, see num-workers. The template defaults to `Standard_D4s_v3`. This configuration has 4 vCPUs and 16 GB of memory. For more information on Azure VMs, see Azure VM sizes.
Number of Server VMs	Specify the number of VMs to run MATLAB Production Server instances. The deployment template sets the default to 2 VMs for load balancing. You can change the number of VMs after the initial deployment. For more information, see “Change the Number of Virtual Machines” on page 9-32.

Parameter Name	Value
Server VM Operating System	Choose the server VM operating system. Windows (Windows Server) and Linux (Ubuntu) are the only available options. In most cases, choosing an operating system depends on your personal preference. Unless you add operating system-specific dependencies or content such as MEX files to your applications, the applications you deploy to the server do not depend on a specific operating system.
Create Redis Cache	Choose whether you want to create an Azure cache for Redis. Creating this service allows you to use the persistence functionality of the server. Persistence provides a mechanism to cache data between calls to MATLAB code running on a server instance. For more information, see “Data Caching Basics”. You can provision an Azure cache for Redis after the initial deployment. For setup instructions, see Azure documentation.

Dashboard Login

After you deploy the server VMs, you can manage the server using the MATLAB Production Server dashboard, which provides a web-based interface to configure and manage MATLAB Production Server in the cloud. Specify the login credentials for the dashboard.

Parameter Name	Value
Admin Username	Specify the administrator user name to log in to the MATLAB Production Server dashboard.
Admin Password	Specify the administrator password to log in to the MATLAB Production Server dashboard.

Network

You can specify which IP addresses can access the dashboard, whether your solution should use a public IP address, and configure a virtual network (VNet).

Parameter Name	Value
Allow Connections from IP Address	Specify the IP address or a range of IP addresses that is permitted to connect to the dashboard that manages the server. If you specify a range of IP addresses, use CIDR notation, which provides the IP address before the slash and mask after the slash. The mask determines the number of IP addresses to include. For example, 10.0.0.1/24. You can use a CIDR calculator to determine the CIDR notation for a range of IP addresses.
Make Solution Available over Internet	Make your solution available over the Internet by setting this parameter to `Yes`. This will use public IP addresses. If you set this parameter to `No`, the ARM template does not assign a public IP address for the VM that hosts the dashboard. To access the dashboard, you can use a different VM located in the same virtual network as the VM that hosts the dashboard.

Parameter Name	Value
Virtual Network	Create a new VNet or choose an existing one. The template defaults to a creating a new VNet with predefined values. These are the values that Azure defaults to while creating a new VNet. You can use the default values or enter new values based on your network setup. The deployment opens the following ports when deploying in a new VNet. If you are using an existing VNet, you need to open the following ports. • `443` – Required for communicating with the dashboard and the MATLAB execution endpoint. • `8000`, `8004`, `880`, `9090`, and `9910` – Required for communication between the dashboard, MATLAB Production Server workers, and various microservices within the virtual network. These ports do not need to be accessible over the Internet. • `65200-65535` – Required for the Azure application gateway health check to work. These ports need to be accessible over the Internet. For more information, see MSDN Community. • `22`, `3389` – Required for Remote Desktop functionality for Windows or SSH for Linux respectively. You can use remote login functionality for troubleshooting and debugging. You can close these ports after deployment is complete.
First Subnet **Second Subnet**	Specify the subnet name and subnet address prefix for a new or existing subnet. The first subnet hosts the dashboard. The second subnet hosts the application gateway. The template defaults to creating new subnets with predefined values. You can use the default values or enter new values based on your network setup.

SSL Certificate

In this section, you specify an SSL certificate for the Azure application gateway to use. The application gateway provides an HTTPS endpoint that you use to connect to server instances and the MATLAB Production Server dashboard.

Parameter Name	Value
PFX Certificate	Upload a PFX-formatted SSL certificate.
PFX Certificate Password	If the certificate requires a password, enter it here. Otherwise, leave the field blank.

Connect and Log In to Dashboard

The MATLAB Production Server dashboard is a web-based interface to configure and manage MATLAB Production Server in the cloud.

Note The Internet Explorer web browser is not supported for interacting with the dashboard.

Complete these steps only after you successfully create your resource group. If your solution uses private IP addresses, you can connect to the dashboard from a VM located in the same virtual network as the VM that hosts the dashboard.

1. In the Azure portal, click **Resource groups**.
2. Select the resource group that you created for this deployment.
3. Select **Deployments** from the left pane. In the pane that opens, click **Microsoft.Template**.
4. Select **Outputs** from the left pane.
5. Copy the parameter value for **dashboardURL** and paste it in a browser.
6. Use the administrator user name and password that you specified in the “Dashboard Login” on page 9-11 step of the deployment process to log in.

Configuring role-based access control for the dashboard is recommended. MATLAB Production Server integrates with OAuth 2.0 providers such as Microsoft Azure Active Directory, Google Identity, and PingFederate for providing role-based access control. Role-based access control lets you grant users the privileges to perform tasks on the dashboard and server, based on their role. For more information on how to configure role-based access control, see “Dashboard Access Control” on page 9-53.

See Also

More About

- “Architecture and Resources on Azure” on page 9-39
- “Manage MATLAB Production Server (PAYG)” on page 9-20

Manage MATLAB Production Server (BYOL)

After you deploy a MATLAB Production Server bring your own license (BYOL) environment in Azure, and configure licensing in the cloud, use the MATLAB Production Server dashboard, which is a web-based interface to upload applications, edit server settings, and configure access control for the dashboard and applications.

Connect to Dashboard

Find the URL to connect to the dashboard in the Azure portal.

Note The Internet Explorer web browser is not supported for interacting with the dashboard.

Complete these steps only after you successfully create your resource group. If your solution uses private IP addresses, you can connect to the dashboard from a VM that belongs to the same virtual network as the VM that hosts the dashboard.

1. In the Azure portal, click **Resource groups**.
2. Select the resource group you created for this deployment.
3. Select **Deployments** from the left pane. In the pane that opens, click **Microsoft.Template**.
4. Select **Outputs** from the left pane.
5. Copy the parameter value for **dashboardAdminURL** and paste it in a browser.

Log In to Dashboard

There are three roles for logging in to the dashboard depending on your role in your organization — administrator, manager, or application author.

Log In to Dashboard as Administrator

If you are accessing the dashboard for the first time, or if you have not configured or not enabled dashboard access control, you must log in using the administrator credentials. These are the credentials that you entered for the dashboard during the deployment process. An administrator has access to all areas of the dashboard. The administrator can log in to server virtual machines (VMs), configure which users or groups of users can access the dashboard and applications, edit server settings, configure remote persistence services, view logs, and upload and delete applications. You cannot change the administrator credentials.

To grant access to only certain dashboard areas for specific users or groups of users, set up dashboard access control after you log in. For more information, see “Dashboard Access Control” on page 9-51.

Log In to Dashboard as Manager or Application Author

If the administrator has configured and enabled dashboard access control, you can log in to the dashboard as a manager or application author depending on your role in your organization. The login process supports single sign-on using OAuth 2.0 providers.

An application author has the privileges to upload and delete applications and view logs. A manager has the privileges to edit server settings, configure access control for applications, and configure

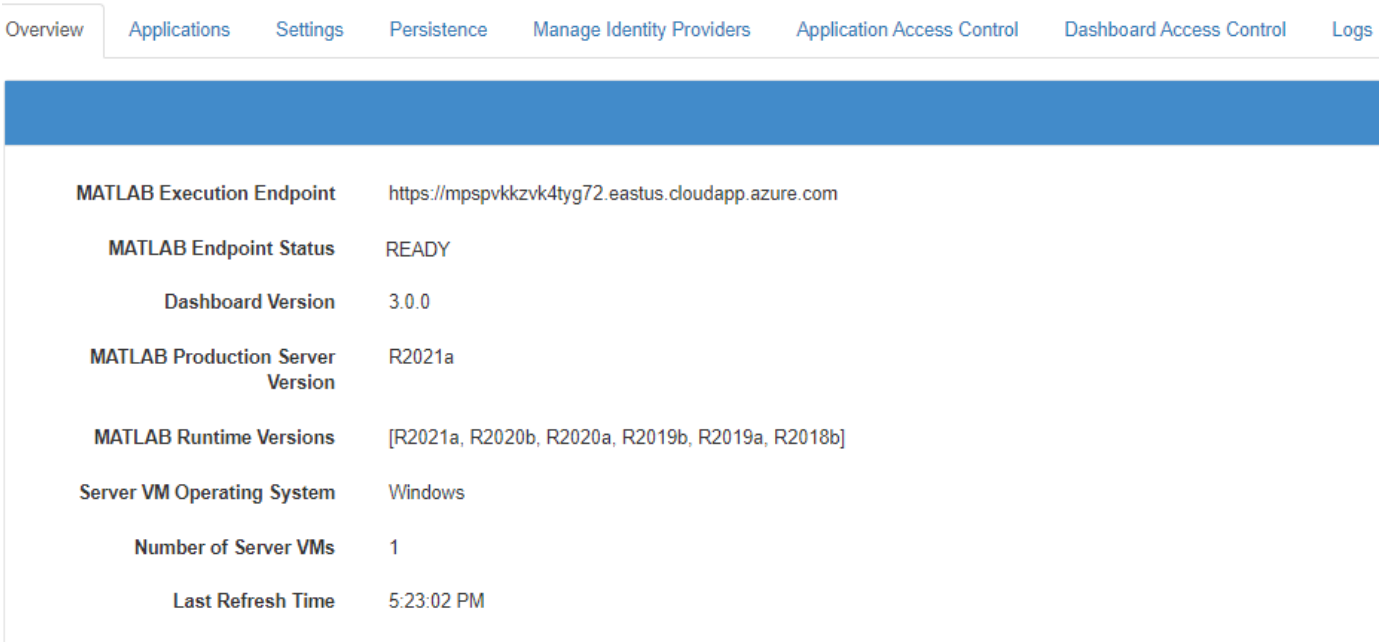
remote persistence services, as well as all the privileges of an application author, including for uploading and deleting applications and viewing logs.

The following table shows the dashboard tabs that users with these roles can access.

Role	Overview	Applications	Settings	Persistence	Manage Identity Providers	Application Access Control	Dashboard Access Control	Logs
Server administrator	✓	✓	✓	✓	✓	✓	✓	✓
Manager	✓	✓	✓	✓	✓	✓		✓
Application author	✓	✓						✓

View Information About Server

To view information about the server instance VMs deployed on Azure, click **Overview** on the dashboard navigation menu.



Upload MATLAB Application

You can run an application on MATLAB Production Server that is created using MATLAB Compiler SDK. Upload the application using the dashboard.

1. Click **+Upload**.
2. Click **Choose File**, select the file, and click **Deploy**.

For information on how to upload multiple applications, see “Upload MATLAB Applications” on page 9-30.

For information on how to create an application, see “Create Deployable Archive for MATLAB Production Server”.

View HTTPS Server Endpoint

The Azure application gateway in the MATLAB Production Server (BYOL) deployment provides an HTTPS endpoint URL to make requests to the server. The **HTTPS Server Endpoint** in the **Home** tab in the cloud console specifies the HTTPS endpoint. Use this endpoint to execute MATLAB functions deployed to the server. For example, if the HTTPS server endpoint for your server is `https://mpst4ezclcdtlcay.eastus.cloudapp.azure.com`, to use the MATLAB Production Server RESTful API to execute a MATLAB function `mymagic` located in a deployed application `myapp`, use the URL `https://mpst4ezclcdtlcay.eastus.cloudapp.azure.com/myapp/mymagic`.

For more information on working with the self-signed SSL certificate and managing cookies set by the application gateway, see “Execute MATLAB Functions on MATLAB Production Server (BYOL)” on page 9-57.

Edit Server Configuration

Edit the MATLAB Production Server configuration properties by selecting the **Settings** tab in dashboard. After you update the properties, click **Save** to apply your changes.

Note The server restarts when you click **Save**.

Only the global admin and managers have the privilege to edit the server configuration.

Some examples of server configuration properties follow.

- To allow requests from specific domains, specify parameter values for the **CORS Allowed Origins** property.

  If you want to specify multiple domains, separate them with a comma, for example, `http://www.w3.org, https://www.apache.org`.
- To set the number of MATLAB Production Server workers, specify a parameter value for the **Number of Workers** property.

  When setting the **Number of Workers** property, carefully consider your cluster setup. Each VM in the cluster runs an instance of MATLAB Production Server and each instance runs multiple MATLAB Production Server workers. Using 1 vCPU per MATLAB Production Server worker is recommended. For example, if the server size is set to *Standard_D4s_v3 Server*, which specifies 4 vCPUs, set the number of workers to no more than 4 workers per instance.

Use the Azure Cache for Redis for Data Persistence

MATLAB Production Server uses Redis for data persistence. Persistence allows caching of data between calls to MATLAB code running on the servers. Only the global admin and managers have the privilege to configure remote persistence services.

To view a persistence service that your deployment creates or to create a new remote persistence service, select **Persistence** in the MATLAB Production Server dashboard navigation pane.

Click **+Add** to create a new remote persistence service. Specify the following parameter values, then click **Create**. Creating a persistence service takes 3 minutes on a Windows VM.

Value	Description
Connection Name	Specify a name for the connection to the persistence service. Use this name in MATLAB code to pass data to the cache.
Host and **Port**	Specify the host name and port number for Azure Cache for Redis from the Azure portal. The port number has to be a non-SSL port. To retrieve the host name and port number, follow these steps. • Log in to your Azure portal and select the resource group associated with the deployment. • Select the Azure Cache for Redis resource within the resource group. This resource usually has the name `vmss<uniqueID>redis`. • Select **Overview** and copy the values listed under **Host name** and under **Ports**. You can also use an Azure Cache for Redis that you create outside of this deployment. The steps to retrieve the host name and port number are the same.
Access Key	If you use Azure Cache for Redis for persistence, you must specify an access key. To retrieve the access key, follow these steps. • Log in to your Azure portal and select the resource group associated with this deployment. • Select the Azure Cache for Redis resource within the resource group. This resource usually has the name `vmss<uniqueID>redis`. • Select **Access keys** from the left pane. • In the resulting pane, copy the access key string listed under **Primary**.

For more information on using a data cache, see “Data Caching Basics”.

Set Up Access Control for Applications Using OAuth 2.0 Providers

MATLAB Production Server integrates with OAuth 2.0 providers such as Microsoft Azure Active Directory (Azure AD), Google Identity, and PingFederate, for restricting access to deployed

applications to only certain groups of users. Only the global admin and manager can configure access control for applications.

For more information on how to configure application access control using the **Application Access Control** tab, see “Application Access Control” on page 9-45.

Set Up Access Control for Dashboard Using OAuth 2.0 Providers

MATLAB Production Server integrates with OAuth 2.0 providers such as Microsoft Azure Active Directory (Azure AD), Google Identity, and PingFederate from Ping Identity®, and uses JSON Web Tokens (JWTs) to provide role-based access control for accessing the dashboard. You can grant access to certain dashboard areas for specific users or groups of users based on the user role. Only the server administrator can configure dashboard access control. For more information on how to configure dashboard access control using the **Dashboard Access Control** tab, see “Dashboard Access Control” on page 9-51.

See Also

More About

- “Azure Deployment for MATLAB Production Server (BYOL)” on page 9-2
- “Architecture and Resources on Azure” on page 9-36
- “Manage Azure Resources for MATLAB Production Server (BYOL)” on page 9-30

Manage MATLAB Production Server (PAYG)

After you deploy the MATLAB Production Server pay as you go (PAYG) environment in Azure, use the MATLAB Production Server dashboard, which is a web-based interface to upload applications, edit server settings, and configure access control for the dashboard and applications.

Connect to Dashboard

Find the URL to connect to the dashboard in the Azure portal.

Note The Internet Explorer web browser is not supported for interacting with the dashboard.

Complete these steps only after you successfully create your resource group. If your solution uses private IP addresses, you can connect to the dashboard from a VM that belongs to the same virtual network as the VM that hosts the dashboard.

1. In the Azure portal, click **Resource groups**.
2. Select the resource group you created for this deployment.
3. Select **Deployments** from the left pane. In the pane that opens, click **Microsoft.Template**.
4. Select **Outputs** from the left pane.
5. Copy the parameter value for **dashboardURL** and paste it in a browser.

Log In to Dashboard

There are three roles for logging in to the dashboard depending on your role in your organization — administrator, manager, or application author.

Log In to Dashboard as Administrator

If you are accessing the dashboard for the first time, or if you have not configured or not enabled dashboard access control, you must log in using the administrator credentials. These are the credentials that you entered for the dashboard during the deployment process. An administrator has access to all areas of the dashboard. The administrator can log in to server virtual machines (VMs), configure which users or groups of users can access the dashboard and applications, edit server settings, configure remote persistence services, view logs, and upload and delete applications. You cannot change the administrator credentials.

To grant access to only certain dashboard areas for specific users or groups of users, set up dashboard access control after you log in. For more information, see “Dashboard Access Control” on page 9-53.

Log In to Dashboard as Manager or Application Author

If the administrator has configured and enabled dashboard access control, you can log in to the dashboard as a manager or application author depending on your role in your organization. The login process supports single sign-on using OAuth 2.0 providers.

An application author has the privileges to upload and delete applications and view logs. A manager has the privileges to edit server settings, configure access control for applications, and configure

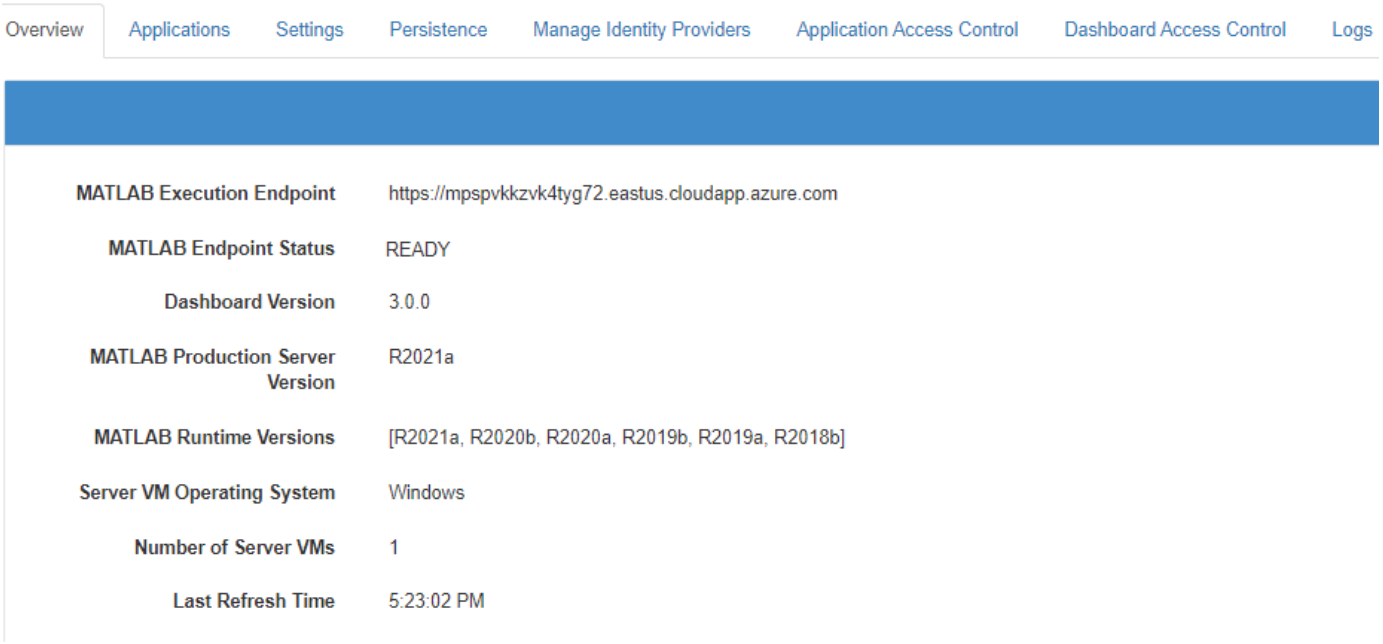
remote persistence services, as well as all the privileges of an application author, including for uploading and deleting applications and viewing logs.

The following table shows the dashboard tabs that users with these roles can access.

Role	Overview	Applications	Settings	Persistence	Manage Identity Providers	Application Access Control	Dashboard Access Control	Logs
Server administrator	✓	✓	✓	✓	✓	✓	✓	✓
Manager	✓	✓	✓	✓	✓	✓		✓
Application author	✓	✓						✓

View Information About Server

To view information about the server instance VMs deployed on Azure, click **Overview** on the dashboard navigation menu.



Upload MATLAB Application

You can run an application on MATLAB Production Server that is created using MATLAB Compiler SDK. Upload and deploy applications using the **Applications** tab in the dashboard.

1. Click **+Upload**.
2. Click **Choose File**, select the file, and click **Deploy**.

For information on how to upload multiple applications, see “Upload MATLAB Applications” on page 9-33.

For information on how to create an application, see “Create Deployable Archive for MATLAB Production Server”.

View MATLAB Execution Endpoint

The Azure application gateway in the deployment provides an HTTPS endpoint URL to make requests to the server. The **MATLAB Execution Endpoint** in the **Overview** tab in the dashboard specifies the HTTPS endpoint. Use this endpoint to execute MATLAB functions deployed to the server. For example, if the MATLAB execution endpoint for your server is `https://mpst4ezclcdtlcay.eastus.cloudapp.azure.com`, to use the MATLAB Production Server RESTful API to execute a MATLAB function `mymagic` located in a deployed application `myapp`, use the URL `https://mpst4ezclcdtlcay.eastus.cloudapp.azure.com/myapp/mymagic`.

For more information on working with the self-signed certificate and managing cookies set by the application gateway, see “Execute MATLAB Functions on MATLAB Production Server (PAYG)” on page 9-59.

Edit Server Configuration

Edit the MATLAB Production Server configuration properties by selecting the **Settings** tab in dashboard. After you update the properties, click **Save** to apply your changes.

Note The server restarts when you click **Save**.

Only the global admin and managers have the privilege to edit the server configuration.

Some examples of server configuration properties follow.

- To allow requests from specific domains, specify parameter values for the **CORS Allowed Origins** property.

  If you want to specify multiple domains, separate them with a comma, for example, `http://www.w3.org, https://www.apache.org`.
- To set the number of MATLAB Production Server workers, specify a parameter value for the **Number of Workers** property.

  When setting the **Number of Workers** property, carefully consider your cluster setup. Each VM in the cluster runs an instance of MATLAB Production Server and each instance runs multiple MATLAB Production Server workers. Using 1 vCPU per MATLAB Production Server worker is recommended. For example, if the server size is set to *Standard_D4s_v3 Server*, which specifies 4 vCPUs, set the number of workers to 4 workers per instance.

Use the Azure Cache for Redis for Data Persistence

MATLAB Production Server uses Redis for data persistence. Persistence allows caching of data between calls to MATLAB code running on the servers. Only the global admin and managers have the privilege to configure remote persistence services.

To view a persistence service that your deployment creates or to create a new remote persistence service, select **Persistence** in the MATLAB Production Server dashboard navigation pane.

Click **+Add** to create a new remote persistence service. Specify the following parameter values, then click **Create**. Creating a persistence service takes 3 minutes on a Windows VM.

Value	Description
Connection Name	Specify a name for the connection to the persistence service. Use this name in MATLAB code to pass data to the cache.
Host and **Port**	Specify the host name and port number for Azure Cache for Redis from the Azure portal. The port number has to be a non-SSL port. To retrieve the host name and port number, follow these steps. • Log in to your Azure portal and select the resource group associated with the deployment. • Select the Azure Cache for Redis resource within the resource group. This resource usually has the name `vmss<uniqueID>redis`. • Select **Overview** and copy the values listed under **Host name** and under **Ports**. You can also use an Azure Cache for Redis that you create outside of this deployment. The steps to retrieve the host name and port number are the same.
Access Key	If you use Azure Cache for Redis for persistence, you must specify an access key. To retrieve the access key, follow these steps. • Log in to your Azure portal and select the resource group associated with this deployment. • Select the Azure Cache for Redis resource within the resource group. This resource usually has the name `vmss<uniqueID>redis`. • Select **Access keys** from the left pane. • In the resulting pane, copy the access key string listed under **Primary**.

For more information on using a data cache, see “Data Caching Basics”.

Set Up Access Control for Applications Using OAuth 2.0 Providers

MATLAB Production Server integrates with OAuth 2.0 providers such as Microsoft Azure Active Directory (Azure AD), Google Identity, and PingFederate, for restricting access to deployed

applications to only certain groups of users. Only the global admin and manager can configure access control for applications.

For more information on how to configure application access control using the **Application Access Control** tab, see “Application Access Control” on page 9-47.

Set Up Access Control for Dashboard Using OAuth 2.0 Providers

MATLAB Production Server integrates with OAuth 2.0 providers such as Microsoft Azure Active Directory (Azure AD), Google Identity, and PingFederate from Ping Identity, and uses JSON Web Tokens (JWTs) to provide role-based access control for accessing the dashboard. You can grant access to certain dashboard areas for specific users or groups of users based on the user role. Only the global admin can configure dashboard access control. For more information on how to configure dashboard access control using the **Dashboard Access Control** tab, see “Dashboard Access Control” on page 9-53.

See Also

More About

- “Dashboard Access Control” on page 9-53
- “Azure Deployment for MATLAB Production Server (PAYG)” on page 9-9
- “Manage Azure Resources for MATLAB Production Server (PAYG)” on page 9-32
- “Application Access Control” on page 9-47

Manage MATLAB Production Server Using the Dashboard

After you deploy the MATLAB Production Server reference architecture in Azure and configure licensing in the cloud, use the MATLAB Production Server dashboard, which is a web-based interface to upload applications, edit server settings, and configure access control for the dashboard and applications.

For information on deploying the reference architecture, see MATLAB Production Server on Microsoft Azure. For information on setting up your MATLAB Production Server license for using in the cloud, see “Configure MATLAB Production Server License for Use on the Cloud”.

Connect to Dashboard

Find the URL to connect to the dashboard in the Azure portal.

Note The Internet Explorer web browser is not supported for interacting with the dashboard.

Complete these steps only after you successfully create your resource group. If your solution uses private IP addresses, you can connect to the dashboard from a VM that belongs to the same virtual network as the VM that hosts the dashboard.

1. In the Azure portal, click **Resource groups**.
2. Select the resource group you created for this deployment.
3. Select **Deployments** from the left pane. In the pane that opens, click **Microsoft.Template**.
4. Select **Outputs** from the left pane.
5. Copy the parameter value for **dashboardURL** and paste it in a browser.

Log In to Dashboard

There are three roles for logging in to the dashboard depending on your role in your organization — administrator, manager, or application author.

Log In to Dashboard as Administrator

If you are accessing the dashboard for the first time, or if you have not configured or not enabled dashboard access control, you must log in using the administrator credentials. These are the credentials that you entered for the dashboard during the deployment process. An administrator has access to all areas of the dashboard. The administrator can log in to server virtual machines (VMs), configure which users or groups of users can access the dashboard and applications, edit server settings, configure remote persistence services, view logs, and upload and delete applications. You cannot change the administrator credentials.

To grant access to only certain dashboard areas for specific users or groups of users, set up dashboard access control after you log in. For more information, see “Dashboard Access Control” on page 9-55.

Log In to Dashboard as Manager or Application Author

If the administrator has configured and enabled dashboard access control, you can log in to the dashboard as a manager or application author depending on your role in your organization. The login process supports single sign-on using OAuth 2.0 providers.

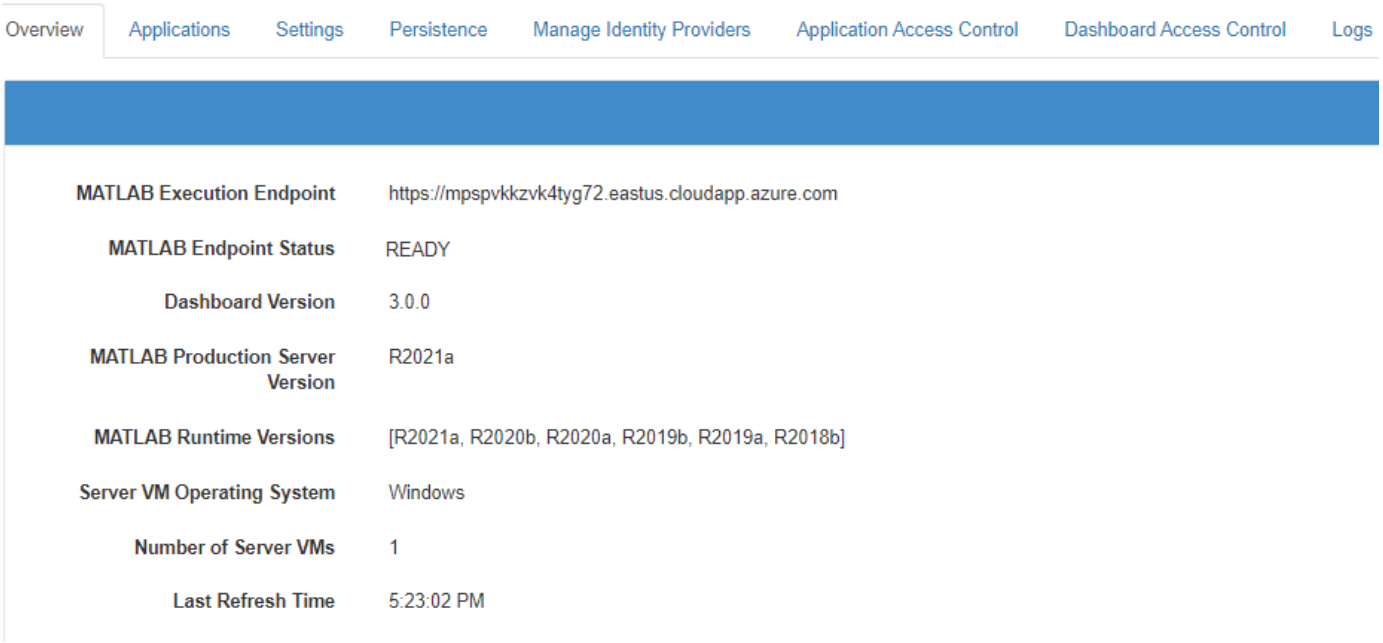
An application author has the privileges to upload and delete applications and view logs. A manager has the privileges to edit server settings, configure access control for applications, and configure remote persistence services, as well as all the privileges of an application author, including for uploading and deleting applications and viewing logs.

The following table shows the dashboard tabs that users with these roles can access.

Role	Overview	Applications	Settings	Persistence	Manage Identity Providers	Application Access Control	Dashboard Access Control	Logs
Server administrator	✓	✓	✓	✓	✓	✓	✓	✓
Manager	✓	✓	✓	✓	✓	✓		✓
Application author	✓	✓						✓

View Information About Server

To view information about the server instance VMs deployed on Azure, click **Overview** on the dashboard navigation menu.



Upload MATLAB Application

You can run an application on MATLAB Production Server that is created using MATLAB Compiler SDK. Upload and deploy applications using the **Applications** tab in the dashboard.

1. Click **+Upload**.
2. Click **Choose File**, select the file, and click **Deploy**.

For information on how to upload multiple applications, see “Upload MATLAB Applications” on page 9-33.

For information on how to create an application, see “Create Deployable Archive for MATLAB Production Server”.

View MATLAB Execution Endpoint

The Azure application gateway in the deployment provides an HTTPS endpoint URL to make requests to the server. The **MATLAB Execution Endpoint** in the **Overview** tab in the dashboard specifies the HTTPS endpoint. Use this endpoint to execute MATLAB functions deployed to the server. For example, if the MATLAB execution endpoint for your server is `https://mpst4ezclcdtlcay.eastus.cloudapp.azure.com`, to use the MATLAB Production Server RESTful API to execute a MATLAB function `mymagic` located in a deployed application `myapp`, use the URL `https://mpst4ezclcdtlcay.eastus.cloudapp.azure.com/myapp/mymagic`.

For more information on working with the self-signed certificate and managing cookies set by the application gateway, see “Execute MATLAB Functions on MATLAB Production Server Reference Architecture” on page 9-61 .

Edit Server Configuration

Edit the MATLAB Production Server configuration properties by selecting the **Settings** tab in dashboard. After you update the properties, click **Save** to apply your changes.

Note The server restarts when you click **Save**.

Only the global admin and managers have the privilege to edit the server configuration.

Some examples of server configuration properties follow.

- To allow requests from specific domains, specify parameter values for the **CORS Allowed Origins** property.

  If you want to specify multiple domains, separate them with a comma, for example, `http://www.w3.org, https://www.apache.org`.
- To set the number of MATLAB Production Server workers per VM, specify a parameter value for the **Number of Workers** property.

  When setting the **Number of Workers** property, consider your cluster setup. Each VM in the cluster runs an instance of MATLAB Production Server and each instance runs multiple MATLAB Production Server workers. Using 1 vCPU per MATLAB Production Server worker is

recommended. For example, if the size of a server VM is *Standard_D4s_v3 Server*, which specifies 4 vCPUs, set the number of workers to no more than 4 workers per instance.

Use the Azure Cache for Redis for Data Persistence

MATLAB Production Server uses Redis for data persistence. Persistence allows caching of data between calls to MATLAB code running on the servers. Only the global admin and managers have the privilege to configure remote persistence services.

To view a persistence service that your deployment creates or to create a new remote persistence service, select **Persistence** in the MATLAB Production Server dashboard navigation pane.

Click **+Add** to create a new remote persistence service. Specify the following parameter values, then click **Create**. Creating a persistence service takes 3 minutes on a Windows VM.

Value	Description
Connection Name	Specify a name for the connection to the persistence service. Use this name in MATLAB code to pass data to the cache.
Host and **Port**	Specify the host name and port number for Azure Cache for Redis from the Azure portal. The port number has to be a non-SSL port. To retrieve the host name and port number, follow these steps. • Log in to your Azure portal and select the resource group associated with the deployment. • Select the Azure Cache for Redis resource within the resource group. This resource usually has the name `vmss<uniqueID>redis`. • Select **Overview** and copy the values listed under **Host name** and under **Ports**. You can also use an Azure Cache for Redis that you create outside of this deployment. The steps to retrieve the host name and port number are the same.

Value	Description
Access Key	If you use Azure Cache for Redis for persistence, you must specify an access key. To retrieve the access key, follow these steps. • Log in to your Azure portal and select the resource group associated with this deployment. • Select the Azure Cache for Redis resource within the resource group. This resource usually has the name `vmss<uniqueID>redis`. • Select **Access keys** from the left pane. • In the resulting pane, copy the access key string listed under **Primary**.

For more information on using a data cache, see “Data Caching Basics”.

Set Up Access Control for Applications Using OAuth 2.0 Providers

MATLAB Production Server integrates with OAuth 2.0 providers such as Microsoft Azure Active Directory (Azure AD), Google Identity, and PingFederate, for restricting access to deployed applications to only certain groups of users. Only the global admin and manager can configure access control for applications.

For more information on how to configure application access control using the **Application Access Control** tab, see “Application Access Control” on page 9-49.

Set Up Access Control for Dashboard Using OAuth 2.0 Providers

MATLAB Production Server integrates with OAuth 2.0 providers such as Microsoft Azure Active Directory (Azure AD), Google Identity, and PingFederate from Ping Identity, and uses JSON Web Tokens (JWTs) to provide role-based access control for accessing the dashboard. You can grant access to certain dashboard areas for specific users or groups of users based on the user role. Only the global admin can configure dashboard access control. For more information on how to configure dashboard access control using the **Dashboard Access Control** tab, see “Dashboard Access Control” on page 9-53.

See Also

More About

- MATLAB Production Server Reference Architecture on Azure
- “Configure MATLAB Production Server License for Use on the Cloud”
- “Dashboard Access Control” on page 9-55
- “Application Access Control” on page 9-49

Manage Azure Resources for MATLAB Production Server (BYOL)

After you deploy the MATLAB Production Server bring your own license (BYOL) environment on Azure, use the Azure portal to manage resources that you provision for in the deployment.

Change the Number of Virtual Machines

Each MATLAB Production Server instance runs on a virtual machine (VM) and each instance can run multiple MATLAB Production Server workers. If you have a standard 24 worker MATLAB Production Server license, and you use 1 worker per VM, you can provision a maximum of 24 VMs. Using 1 vCPU per worker is recommended. For example, if you select the `Standard_D4s_v3 VM` (4 vCPUs) as the **Server VM Size**, you need 6 VMs to fully utilize the workers in your license. You can change the number of VMs after the initial deployment.

1. Log in to the Azure portal and select the resource group associated with the deployment.
2. Select the VM scale set resource (`vmss<uniqueID>`).
3. Select **Scaling** from the left pane to change the number of VMs in the VM scale set.

In the Azure Resource Manager (ARM) template, the `overprovision` property is set to `true` by default. This means that Azure provisions more virtual machines than necessary initially to buffer against any machines that fail. If all VMs are healthy, Azure scales down to the specified the number of VMs.

Change SSL Certificate to Application Gateway

When you deploy MATLAB Production Server, you get an HTTPS endpoint to the Azure application gateway. Use this endpoint to connect to the server instances and the dashboard. To change the SSL certificate to the application gateway, use the Azure portal to update the existing listener.

1. Log in to the Azure portal and select the resource group associated with the deployment.
2. Select the application gateway resource with the name `vmss<uniqueID>-agw`.
3. Select **Listeners** from the left pane and click the listener named `mainentrypoint`.
4. The interface gives you an option to create a new certificate or select an existing certificate.

Upload MATLAB Applications

The Azure file share lets you upload and deploy multiple applications to the server.

1. Log in to the Azure portal and select the resource group associated with the deployment.
2. Select the storage account resource within the resource group. This resource usually has the name `serverlog<uniqueID>`.
3. In the pane that opens, select **File Share** > **deployable-archives** > **auto_deploy**.
4. Select **Upload** to select the applications to upload, then click **Upload**.

View Logs

Application Insights lets you store and view logs related to the deployment.

1. Log in to the Azure portal and select the resource group associated with the deployment.
2. Select the resource `logs-apmservice`.
3. Select **Logs** from the left pane of the resource.
4. Create a new query and click **Run**.

To view logs generated by all the server instances, you can use the following query.

```
traces
| where customDimensions.source == "prodServerInstance"
```

To view the last 200 logs generated by all server instances, you can use the following query.

```
traces
| where customDimensions.source == "prodServerInstance"
| limit 200
```

To view only the error logs generated by all server instances, you can use the following query. For more information on log severity, see `log-severity`.

```
traces
| where customDimensions.source == "prodServerInstance"
| where parse_json(message).severity == "error"
```

To view logs generated by all the server instances within a certain time duration, you can use the following query.

```
traces
| where customDimensions.source == "prodServerInstance"
| where timestamp >= todatetime('2020-02-11T17:21:45.188659Z') and timestamp <= todatetime('2020-02-11T19:22:46.881331Z')
```

Delete Resource Group

A resource group contains all related resources for an Azure solution. You can remove the resource group and all associated cluster resources when you are no longer need them. You cannot undo this.

Note Your license file is tied to your MAC address. If you delete your resource group to delete your cluster, you will need to get a new license file. You can change your MAC address a maximum of 4 times per year.

---

1. Log in to the Azure portal and select the resource group associated with the deployment.
2. Click **Delete resource group** to delete all resources deployed in the group.
3. Enter the name of the resource group to confirm the deletion.

See Also

More About

- “Azure Deployment for MATLAB Production Server (BYOL)” on page 9-2
- “Architecture and Resources on Azure” on page 9-36
- “Manage MATLAB Production Server (BYOL)” on page 9-15

Manage Azure Resources for MATLAB Production Server (PAYG)

After you deploy the MATLAB Production Server pay as you go (PAYG) environment on Azure, use the Azure portal to manage resources that you provision for in the deployment.

Change the Number of Virtual Machines

Each MATLAB Production Server instance runs on a virtual machine (VM) and each instance runs multiple MATLAB Production Server workers. You can change the number of VMs after the initial deployment.

1. Log in to the Azure portal and select the resource group associated with the deployment.
2. Select the VM scale set resource `vmss<uniqueID>`.
3. Select **Scaling** from the left pane to change the number of VMs in the VM scale set.

In the Azure Resource Manager (ARM) template, the `overprovision` property is set to `true` by default. This means that Azure provisions more virtual machines than necessary initially to buffer against any machines that fail. If all VMs are healthy, Azure scales down to the specified the number of VMs.

Change SSL Certificate to Application Gateway

When you deploy MATLAB Production Server, you get an HTTPS endpoint to the Azure application gateway. Use this endpoint to connect to the server instances and the dashboard. To change the SSL certificate to the application gateway, use the Azure portal to update the existing listener.

1. Log in to the Azure portal and select the resource group associated with the deployment.
2. Select the application gateway resource with the name `vmss<uniqueID>-agw`.
3. Select **Listeners** from the left pane and click the listener named `mainentrypoint`.
4. The interface gives you an option to create a new certificate or select an existing certificate.

View Logs

View MATLAB Production Server logs using Azure Application Insights.

1. Log in to the Azure portal and select the resource group associated with the deployment.
2. Select the resource `logs-apmservice`.
3. Select **Logs** from the left pane of the resource.
4. Create a new query and click **Run**.

To view logs generated by all the server instances, you can use the following query.

```
traces
| where customDimensions.source == "prodServerInstance"
```

To view the last 200 logs generated by all server instances, you can use the following query.

```
traces
| where customDimensions.source == "prodServerInstance"
| limit 200
```

To view only the error logs generated by all server instances, you can use the following query. For more information on log severity, see `log-severity`.

```
traces
| where customDimensions.source == "prodServerInstance"
| where parse_json(message).severity == "error"
```

To view logs generated by all the server instances within a certain time duration, you can use the following query.

```
traces
| where customDimensions.source == "prodServerInstance"
| where timestamp >= todatetime('2020-02-11T17:21:45.188659Z')
| and timestamp <= todatetime('2020-02-11T19:22:46.881331Z')
```

Upload MATLAB Applications

The Azure file share lets you upload and deploy multiple applications to the server.

1. Log in to the Azure portal and select the resource group associated with the deployment.
2. Select the storage account resource within the resource group. This resource usually has the name `serverlog<uniqueID>`.
3. In the pane that opens, select **File Share** > **deployable-archives** > **auto_deploy**.
4. Select **Upload** to select the applications to upload, then click **Upload**.

Delete Resource Group

A resource group contains all related resources for an Azure solution. You can remove the resource group and all associated cluster resources when you are no longer need them. You cannot undo this.

1. Log in to the Azure portal and select the resource group associated with the deployment.
2. Click **Delete resource group** to delete all resources deployed in the group.
3. Enter the name of the resource group to confirm the deletion.

See Also

More About

- “Manage MATLAB Production Server (PAYG)” on page 9-20
- “Architecture and Resources on Azure” on page 9-39
- “Azure Deployment for MATLAB Production Server (PAYG)” on page 9-9

Manage Azure Resources for MATLAB Production Server Reference Architecture

After you deploy the MATLAB Production Server reference architecture on Azure, use the Azure portal to manage resources that you provision for in the deployment.

Change the Number of Virtual Machines

Each MATLAB Production Server instance runs on a virtual machine (VM) and each instance runs multiple MATLAB Production Server workers. You can change the number of VMs after the initial deployment.

1. Log in to the Azure portal and select the resource group associated with the deployment.
2. Select the VM scale set resource `vmss<uniqueID>`.
3. Select **Scaling** from the left pane to change the number of VMs in the VM scale set.

In the Azure Resource Manager (ARM) template, the `overprovision` property is set to `true` by default. This means that Azure provisions more virtual machines than necessary initially to buffer against any machines that fail. If all VMs are healthy, Azure scales down to the specified the number of VMs.

Change SSL Certificate to Application Gateway

When you deploy MATLAB Production Server, you get an HTTPS endpoint to the Azure application gateway. Use this endpoint to connect to the server instances and the dashboard. To change the SSL certificate to the application gateway, use the Azure portal to update the existing listener.

1. Log in to the Azure portal and select the resource group associated with the deployment.
2. Select the application gateway resource with the name `vmss<uniqueID>-agw`.
3. Select **Listeners** from the left pane and click the listener named `mainentrypoint`.
4. The interface gives you an option to create a new certificate or select an existing certificate.

View Logs

View MATLAB Production Server logs using Azure Application Insights.

1. Log in to the Azure portal and select the resource group associated with the deployment.
2. Select the resource `logs-apmservice`.
3. Select **Logs** from the left pane of the resource.
4. Create a new query and click **Run**.

To view logs generated by all the server instances, you can use the following query.

```
traces
| where customDimensions.source == "prodServerInstance"
```

To view the last 200 logs generated by all server instances, you can use the following query.

```
traces
| where customDimensions.source == "prodServerInstance"
| limit 200
```

To view only the error logs generated by all server instances, you can use the following query. For more information on log severity, see `log-severity`.

```
traces
| where customDimensions.source == "prodServerInstance"
| where parse_json(message).severity == "error"
```

To view logs generated by all the server instances within a certain time duration, you can use the following query.

```
traces
| where customDimensions.source == "prodServerInstance"
| where timestamp >= todatetime('2020-02-11T17:21:45.188659Z')
| and timestamp <= todatetime('2020-02-11T19:22:46.881331Z')
```

Upload MATLAB Applications

The Azure file share lets you upload and deploy multiple applications to the server.

1. Log in to the Azure portal and select the resource group associated with the deployment.
2. Select the storage account resource within the resource group. This resource usually has the name `serverlog<uniqueID>`.
3. In the pane that opens, select **File Share** > **deployable-archives** > **auto_deploy**.
4. Select **Upload** to select the applications to upload, then click **Upload**.

Delete Resource Group

A resource group contains all related resources for an Azure solution. You can remove the resource group and all associated cluster resources when you are no longer need them. You cannot undo this.

1. Log in to the Azure portal and select the resource group associated with the deployment.
2. Click **Delete resource group** to delete all resources deployed in the group.
3. Enter the name of the resource group to confirm the deletion.

See Also

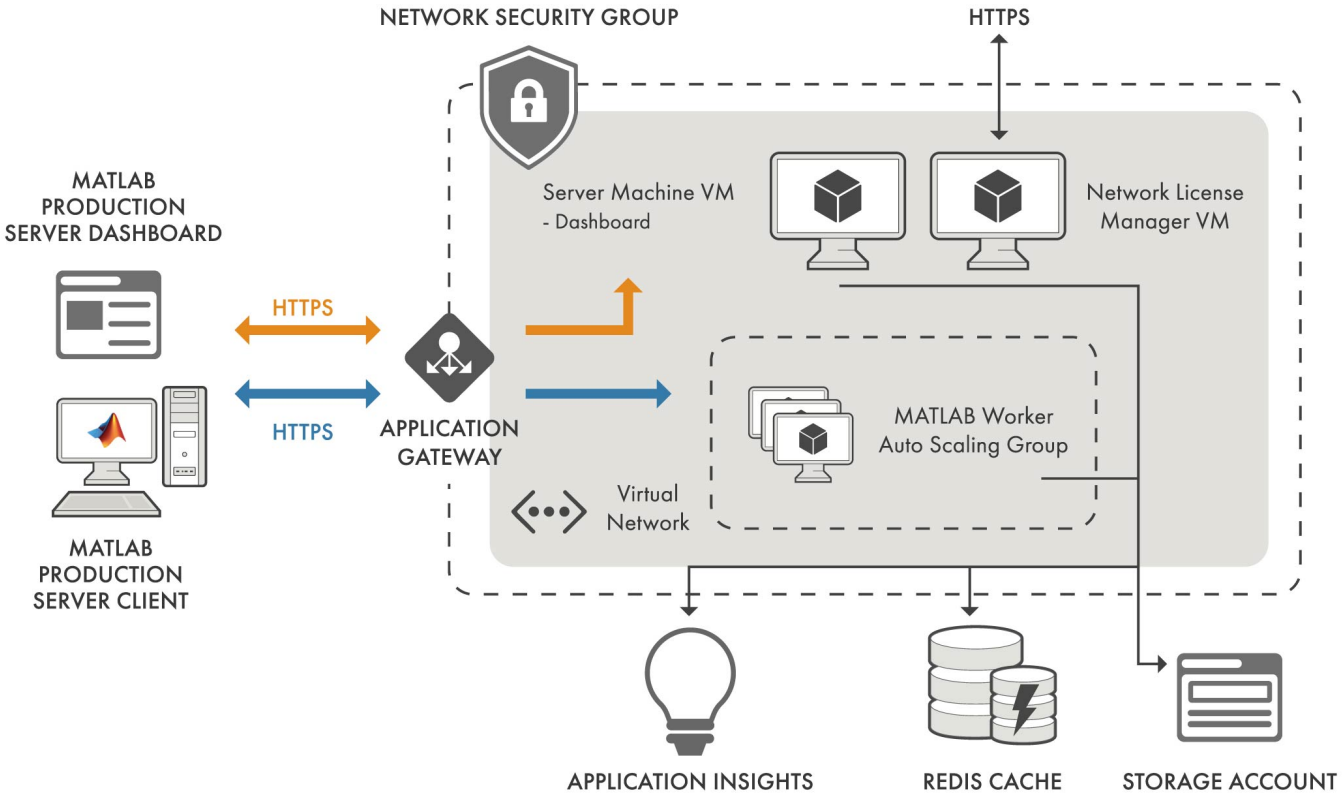
More About

- “Manage MATLAB Production Server Using the Dashboard” on page 9-25
- “Architecture and Resources on Azure” on page 9-42

Architecture and Resources on Azure

Deploying MATLAB Production Server bring your own license (BYOL) on Azure creates several resources in your resource group. The following sections describe the architecture of MATLAB Production Server and the Azure resources that the deployment provisions.

MATLAB Production Server (BYOL) Architecture on Azure



Azure Resources

The MATLAB Production Server deployment in Azure creates the following resources in your resource group.

Resource Name	Resource Name in Azure	Description
MATLAB Production Server dashboard virtual machine	`servermachine`	Virtual machine (VM) that hosts the MATLAB Production Server dashboard. Use the dashboard to: • Get the HTTPS endpoint to make requests. • Upload applications (CTF files) to the server. • Manage server configurations. • Configure access control for the dashboard and applications. For more information about the dashboard, see “Manage MATLAB Production Server (BYOL)” on page 9-15.
MATLAB Production Server dashboard public IP	`servermachine-public-ip`	Public IP address to connect to MATLAB Production Server dashboard. The address provides an HTTPS endpoint to the dashboard for managing server instances. If you choose to not use public IP addresses during deployment, the ARM template does not create this resource.
Virtual machine scale set	`vmss<uniqueID>`	Number of identical VMs to be deployed. Each VM runs an instance of MATLAB Production Server that in turn runs multiple MATLAB Production Server workers. For information on how to change the number of VMs, see “Change the Number of Virtual Machines” on page 9-30.

Resource Name	Resource Name in Azure	Description
Application gateway	`vmss<uniqueID>-agw`	Load balancer for routing traffic to MATLAB Production Server instances. The MATLAB Production Server dashboard retrieves the HTTPS endpoint for making requests to the server from the application gateway resource. Clients use the HTTPS endpoint for making requests to the server.
Storage account	`serverlog<uniqueID>`	Storage account where the deployable archives (CTF files) created by MATLAB Compiler SDK are stored. The deployable archives are stored in an Azure file share.
Virtual network	`vmss<uniqueID>-vnet`	Virtual network that consists of the deployed resources.
Azure Cache for Redis	`vmss<uniqueID>redis`	Redis cache that enables caching of data between calls to MATLAB code running on a server instance. For more information on using a data cache, see “Data Caching Basics”.
Application Insights	`logs-apmservice`	Application performance management service that enables storing and viewing of all logs associated with deployment. For information on how to view the logs, see “View Logs” on page 9-30.

See Also

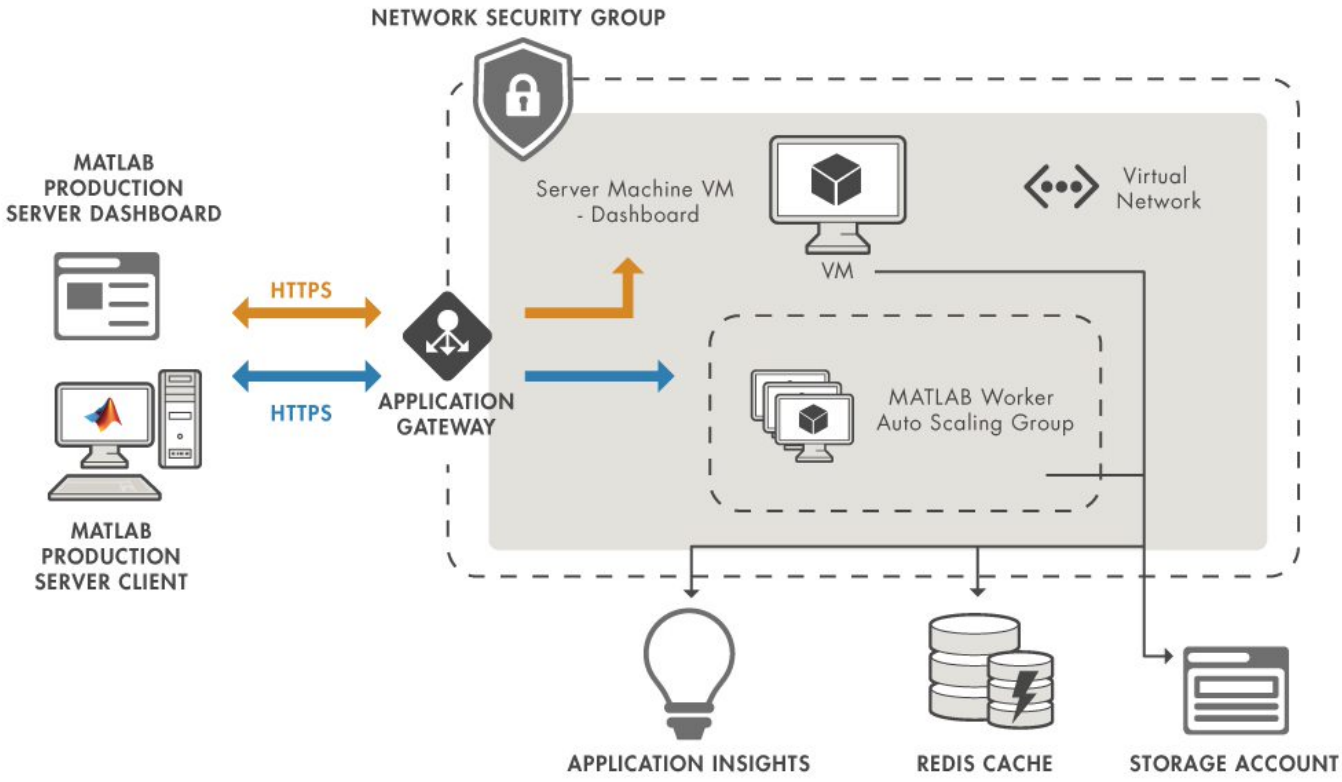
More About

- “Azure Deployment for MATLAB Production Server (BYOL)” on page 9-2
- “Manage MATLAB Production Server (BYOL)” on page 9-15
- “Manage Azure Resources for MATLAB Production Server (BYOL)” on page 9-30

Architecture and Resources on Azure

Deploying MATLAB Production Server pay as you go (PAYG) on Azure creates several resources in your resource group. The following sections describe the architecture of MATLAB Production Server (PAYG) and the Azure resources that the deployment provisions.

MATLAB Production Server (PAYG) Architecture on Azure



Azure Resources

The MATLAB Production Server (PAYG) deployment in Azure creates the following resources in your resource group.

Resource Name	Resource Name in Azure	Description
MATLAB Production Server dashboard virtual machine	`servermachine`	Virtual machine (VM) that hosts the MATLAB Production Server dashboard. Use the dashboard to: • Get the HTTPS endpoint to make requests. • Upload applications (CTF files) to the server. • Manage server configurations. • Configure access control for the dashboard and applications. For more information about the dashboard, see “Manage MATLAB Production Server (PAYG)” on page 9-20.
MATLAB Production Server public IP	`servermachine-public-ip`	Public IP address to connect to MATLAB Production Server dashboard. If you choose to not use public IP addresses during deployment, the ARM template does not create this resource.
Virtual machine scale set	`vmss<uniqueID>`	Number of identical VMs to be deployed. Each VM runs an instance of MATLAB Production Server that in turn runs multiple MATLAB Production Server workers. For information on how to change the number of VMs, see “Change the Number of Virtual Machines” on page 9-32.
Application gateway	`vmss<uniqueID>-agw`	Load balancer for routing traffic to MATLAB Production Server instances. The MATLAB Production Server dashboard retrieves the HTTPS endpoint for making requests to the server from the application gateway resource. Clients use the HTTPS endpoint for making requests to the server.

Resource Name	Resource Name in Azure	Description
Storage account	`serverlog<uniqueID>`	Storage account that stores applications (CTF files) created by MATLAB Compiler SDK. The deployable archives are stored in an Azure file share.
Virtual network	`vmss<uniqueID>-vnet`	Virtual network that consists of the deployed resources.
Azure Cache for Redis	`vmss<uniqueID>redis`	Redis cache that enables caching of data between calls to MATLAB code running on a server instance. For more information on using a data cache, see “Data Caching Basics”.
Application Insights	`logs-apmservice`	Application performance management service that enables storing and viewing of all logs associated with deployment. For information on how to view the logs, see “View Logs” on page 9-32.

See Also

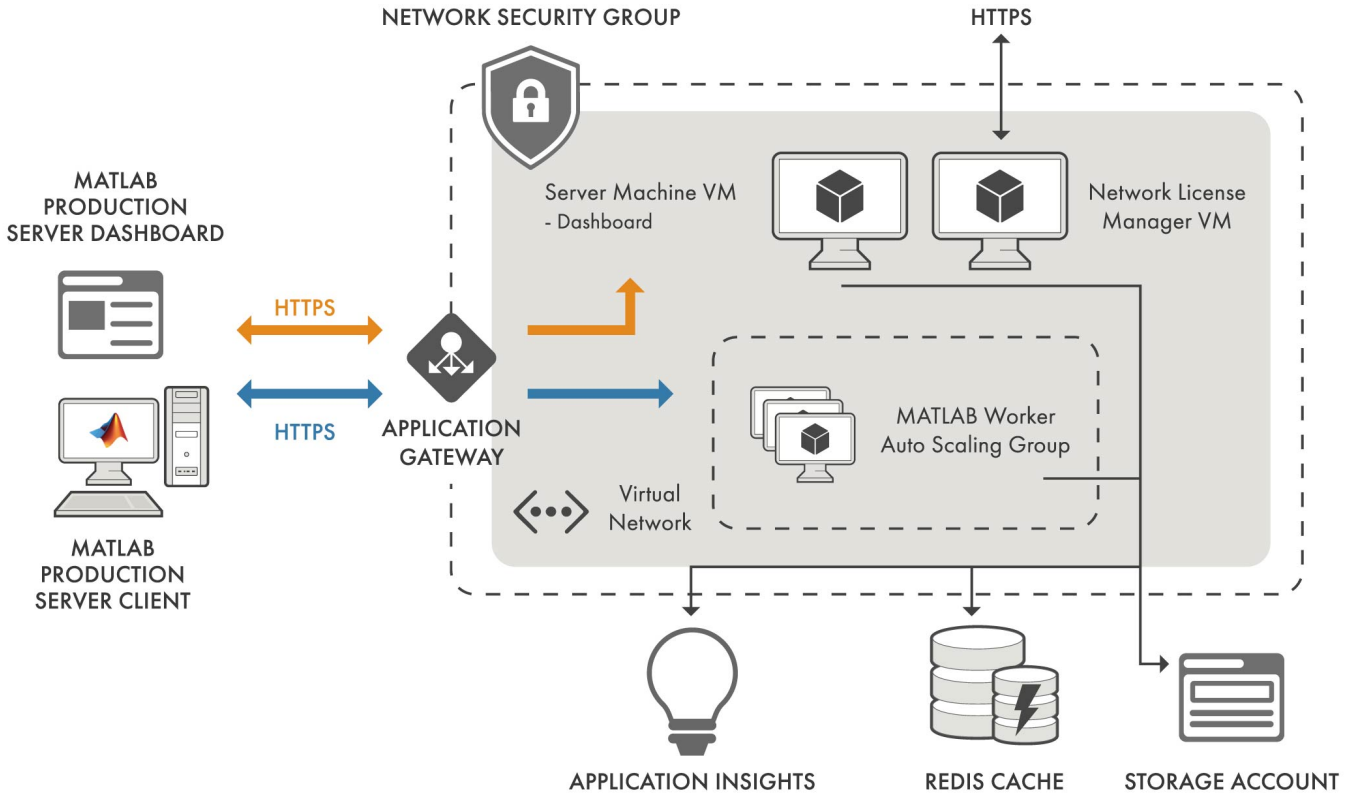
More About

- “Manage Azure Resources for MATLAB Production Server (PAYG)” on page 9-32
- “Manage MATLAB Production Server (PAYG)” on page 9-20
- “Azure Deployment for MATLAB Production Server (PAYG)” on page 9-9

Architecture and Resources on Azure

Deploying the MATLAB Production Server reference architecture on Azure creates several resources in your resource group. The following sections describe the architecture of MATLAB Production Server and the Azure resources that the deployment provisions.

MATLAB Production Server Reference Architecture on Azure



Azure Resources

The MATLAB Production Server deployment in Azure creates the following resources in your resource group.

Resource Name	Resource Name in Azure	Description
MATLAB Production Server dashboard virtual machine	`servermachine`	Virtual machine (VM) that hosts the MATLAB Production Server dashboard. Use the dashboard to: • Get the HTTPS endpoint to make requests. • Upload applications (CTF files) to the server. • Manage server configurations. • Configure access control for the dashboard and applications. For more information about the dashboard, see “Manage MATLAB Production Server Using the Dashboard” on page 9-25.
MATLAB Production Server public IP	`servermachine-public-ip`	Public IP address to connect to MATLAB Production Server dashboard. If you choose to not use public IP addresses during deployment, the ARM template does not create this resource.
Virtual machine scale set	`vmss<uniqueID>`	Number of identical VMs to be deployed. Each VM runs an instance of MATLAB Production Server that in turn runs multiple MATLAB Production Server workers. For information on how to change the number of VMs, see “Change the Number of Virtual Machines” on page 9-34 .

Resource Name	Resource Name in Azure	Description
Application gateway	`vmss<uniqueID>-agw`	Load balancer for routing traffic to MATLAB Production Server instances. The MATLAB Production Server dashboard retrieves the HTTPS endpoint for making requests to the server from the application gateway resource. Clients use the HTTPS endpoint for making requests to the server.
Storage account	`serverlog<uniqueID>`	Storage account that stores applications (CTF files) created by MATLAB Compiler SDK. The deployable archives are stored in an Azure file share.
Virtual network	`vmss<uniqueID>-vnet`	Virtual network that consists of the deployed resources.
Azure Cache for Redis	`vmss<uniqueID>redis`	Redis cache that enables caching of data between calls to MATLAB code running on a server instance. For more information on using a data cache, see “Data Caching Basics”.
Application Insights	`logs-apmservice`	Application performance management service that enables storing and viewing of all logs associated with deployment. For information on how to view the logs, see “View Logs” on page 9-34.

See Also

More About

- “Manage Azure Resources for MATLAB Production Server Reference Architecture” on page 9-34
- “Manage MATLAB Production Server Using the Dashboard” on page 9-25

Application Access Control

MATLAB Production Server integrates with OAuth 2.0 providers such as Microsoft Azure Active Directory (Azure AD), Google Identity, and PingFederate from Ping Identity, and uses JSON Web Tokens (JWTs) for application access control. Application access control lets server administrators restrict user access to applications or archives deployed to the server.

Note Application access control is only available for clients that make server requests using the MATLAB Production Server RESTful API.

All users can access all applications by default.

To enable access control, configure the identity provider and define access control policy rules in the **Application Access Control** tab of the MATLAB Production Server dashboard. Use the access control policy rules to specify which users and groups of users have permission to execute deployed applications. After you enable access control, clients can generate a bearer access token that they must send with every server request. The server uses the bearer token to verify the identify of a client.

You must log in to the dashboard as an administrator or manager to configure application access control. For more information about the dashboard user roles, see “Dashboard Access Control” on page 9-51.

Configure Identity Provider and Specify Access Control Policy Rules

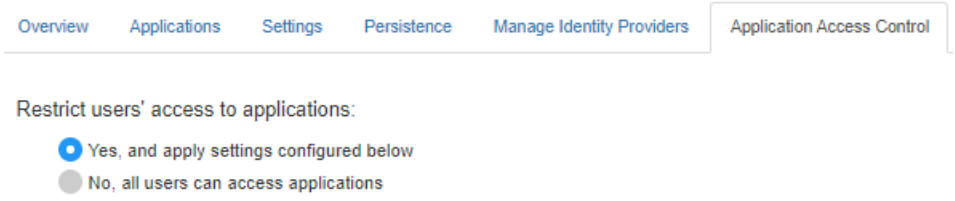
To configure an identity provider, register an application with the identity provider. Then, specify application-specific values and access control policy rules in the dashboard. The fields required to configure an identity provider vary based on the identity provider that you use.

For information about configuring specific identity providers and rules, see:

- “Configure Application Access Control Using Azure AD” on page 9-99
- “Configure Application Access Control Using Google Identity” on page 9-103
- “Configure Application Access Control Using PingFederate” on page 9-105
- “Configure Application Access Control Using Other Identity Providers” on page 9-107

Enable Application Access Control

After you configure the identity provider and specify access control policy rules, you must enable application access control from the dashboard.



Generate Access Token

After you enable application access control, clients can generate a bearer token. Client programs can use third-party libraries for token generation. For a list of OAuth libraries, see OAuth libraries. Client programs use this bearer token in the HTTP authorization header when making a request to the server using the MATLAB Production Server RESTful API. The format for this header is `Authorization:Bearer <access token>`.

See Also

More About

- “Manage Azure Resources for MATLAB Production Server (BYOL)” on page 9-30
- “Manage MATLAB Production Server (BYOL)” on page 9-15
- “Dashboard Access Control” on page 9-51
- “RESTful API for MATLAB Function Execution”

Application Access Control

MATLAB Production Server integrates with OAuth 2.0 providers such as Microsoft Azure Active Directory (Azure AD), Google Identity, and PingFederate from Ping Identity, and uses JSON Web Tokens (JWTs) for application access control. Application access control lets server administrators restrict user access to applications or archives deployed to the server.

Note Application access control is only available for clients that make server requests using the MATLAB Production Server RESTful API.

All users can access all applications by default.

To enable access control, configure the identity provider and define access control policy rules in the **Application Access Control** tab of the MATLAB Production Server dashboard. Use the access control policy rules to specify which users and groups of users have permission to execute deployed applications. After you enable access control, clients can generate a bearer access token that they must send with every server request. The server uses the bearer token to verify the identify of a client.

You must log in to the dashboard as an administrator or manager to configure application access control. For more information about the dashboard user roles, see “Dashboard Access Control” on page 9-55.

Configure Identity Provider and Specify Access Control Policy Rules

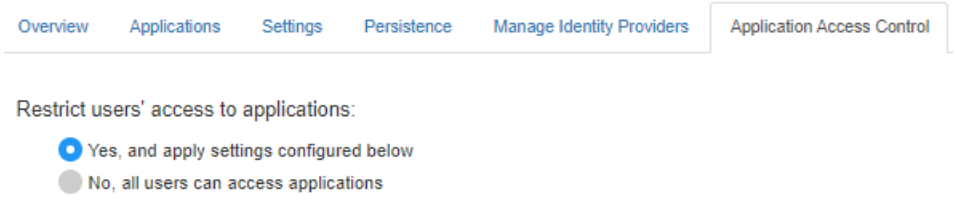
To configure an identity provider, register an application with the identity provider. Then, specify application-specific values and access control policy rules in the dashboard. The fields required to configure an identity provider vary based on the identity provider that you use.

For information about configuring specific identity providers and rules, see:

- “Configure Application Access Control Using Azure AD” on page 9-119
- “Configure Application Access Control Using Google Identity” on page 9-123
- “Configure Application Access Control Using PingFederate” on page 9-125
- “Configure Application Access Control Using Other Identity Providers” on page 9-127

Enable Application Access Control

After you configure the identity provider and specify access control policy rules, you must enable application access control from the dashboard.



Generate Access Token

After you enable application access control, clients can generate a bearer token. Client programs can use third-party libraries for token generation. For a list of OAuth libraries, see OAuth libraries. Client programs use this bearer token in the HTTP authorization header when making a request to the server using the MATLAB Production Server RESTful API. The format for this header is `Authorization:Bearer <access token>`.

See Also

More About

- “Manage MATLAB Production Server (PAYG)” on page 9-20
- “Dashboard Access Control” on page 9-53
- “RESTful API for MATLAB Function Execution”

Application Access Control

MATLAB Production Server integrates with OAuth 2.0 providers such as Microsoft Azure Active Directory (Azure AD), Google Identity, and PingFederate from Ping Identity, and uses JSON Web Tokens (JWTs) for application access control. Application access control lets server administrators restrict user access to applications or archives deployed to the server.

Note Application access control is only available for clients that make server requests using the MATLAB Production Server RESTful API.

All users can access all applications by default.

To enable access control, configure the identity provider and define access control policy rules in the **Application Access Control** tab of the MATLAB Production Server dashboard. Use the access control policy rules to specify which users and groups of users have permission to execute deployed applications. After you enable access control, clients can generate a bearer access token that they must send with every server request. The server uses the bearer token to verify the identify of a client.

You must log in to the dashboard as an administrator or manager to configure application access control. For more information about the dashboard user roles, see “Dashboard Access Control” on page 9-55.

Configure Identity Provider and Specify Access Control Policy Rules

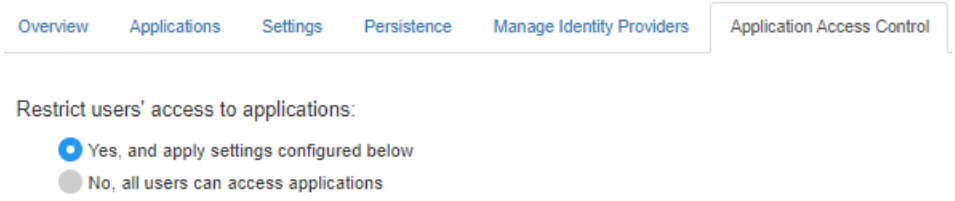
To configure an identity provider, register an application with the identity provider. Then, specify application-specific values and access control policy rules in the dashboard. The fields required to configure an identity provider vary based on the identity provider that you use.

For information about configuring specific identity providers and rules, see:

- “Configure Application Access Control Using Azure AD” on page 9-109
- “Configure Application Access Control Using Google Identity” on page 9-103
- “Configure Application Access Control Using PingFederate” on page 9-115
- “Configure Application Access Control Using Other Identity Providers” on page 9-117

Enable Application Access Control

After you configure the identity provider and specify access control policy rules, you must enable application access control from the dashboard.



Generate Access Token

After you enable application access control, clients can generate a bearer token. Client programs can use third-party libraries for token generation. For a list of OAuth libraries, see OAuth libraries. Client programs use this bearer token in the HTTP authorization header when making a request to the server using the MATLAB Production Server RESTful API. The format for this header is `Authorization:Bearer <access token>`.

See Also

More About

- MATLAB Production Server Reference Architecture on Azure
- “Manage MATLAB Production Server Using the Dashboard” on page 9-25
- “Dashboard Access Control” on page 9-55
- “RESTful API for MATLAB Function Execution”

Dashboard Access Control

MATLAB Production Server on Microsoft Azure lets server administrators use OpenID Connect (OIDC) identity providers such as Microsoft Azure Active Directory (Azure AD), Google identity, PingFederate from Ping Identity, and others to configure role-based access control for the dashboard. Role-based access control allows administrators to grant a dashboard user the privileges to perform tasks on the dashboard based on their role.

Dashboard User Roles

The dashboard access control feature supports the following roles.

- Application author — Application authors can upload and delete applications (deployable archives) and view logs.
- Manager — Managers can edit server settings, configure access control for applications, manage persistence services, and have all the privileges of an application author, which include uploading and deleting applications and viewing logs.
- Server administrator — Administrators can log in to server virtual machines and configure which users or groups of users can access the dashboard, and have all the privileges of a manager, which include editing server logs, configuring access control for applications, uploading and deleting applications, and viewing logs.

The following table shows the dashboard tabs that users with these roles can access.

Role	Overview	Applications	Settings	Persistence	Manage Identity Providers	Application Access Control	Dashboard Access Control	Logs
Server administrator	✓	✓	✓	✓	✓	✓	✓	✓
Manager	✓	✓	✓	✓	✓	✓		✓
Application author	✓	✓						✓

Note Only the server administrator can log in to the dashboard when dashboard access control is disabled or not configured. Only the server administrator has privileges to configure dashboard access control.

Configure Identity Provider and Specify Access Control Policies

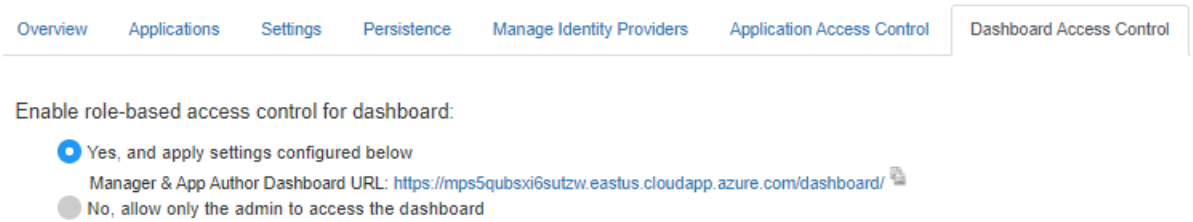
To enable dashboard access control for MATLAB Production Server, you must configure an identity provider and specify access control policies. The fields required to configure an identity provider vary based on the identity provider that you use. The access control policies define areas of the dashboard that users or groups of users can access and tasks that they can perform in these areas.

For information about configuring specific identity providers and policies, see:

- “Configure Dashboard Access Control Using Azure AD” on page 9-63
- “Configure Dashboard Access Control Using Google Identity” on page 9-66
- “Configure Dashboard Access Control Using PingFederate Identity Provider” on page 9-68
- “Configure Dashboard Access Control Using Other OpenID Connect Providers” on page 9-71

Enable Access Control

After you configure the identity provider and specify access control policies, you must enable dashboard access control. After enabling dashboard access control, a dashboard login URL that supports single sign-on (SSO) becomes available. Share this URL with managers and application authors.



See Also

More About

- “Manage MATLAB Production Server (BYOL)” on page 9-15
- “Application Access Control” on page 9-45

Dashboard Access Control

MATLAB Production Server on Microsoft Azure lets server administrators use OpenID Connect (OIDC) identity providers such as Microsoft Azure Active Directory (Azure AD), Google identity, PingFederate from Ping Identity, and others to configure role-based access control for the dashboard. Role-based access control allows administrators to grant a dashboard user the privileges to perform tasks on the dashboard based on their role.

Dashboard User Roles

The dashboard access control feature supports the following roles.

- Application author — Application authors can upload and delete applications (deployable archives) and view logs.
- Manager — Managers can edit server settings, configure access control for applications, manage persistence services, and have all the privileges of an application author, which include uploading and deleting applications and viewing logs.
- Server administrator — Administrators can log in to server virtual machines and configure which users or groups of users can access the dashboard, and have all the privileges of a manager, which include editing server logs, configuring access control for applications, uploading and deleting applications, and viewing logs.

The following table shows the dashboard tabs that users with these roles can access.

Role	Overview	Applications	Settings	Persistence	Manage Identity Providers	Application Access Control	Dashboard Access Control	Logs
Server administrator	✓	✓	✓	✓	✓	✓	✓	✓
Manager	✓	✓	✓	✓	✓	✓		✓
Application author	✓	✓						✓

Note Only the server administrator can log in to the dashboard when dashboard access control is disabled or not configured. Only the server administrator has privileges to configure dashboard access control.

Configure Identity Provider and Specify Access Control Policies

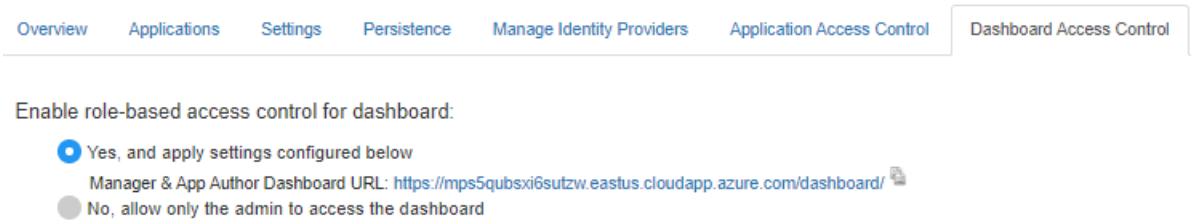
To enable dashboard access control for MATLAB Production Server, you must configure an identity provider and specify access control policies. The fields required to configure an identity provider vary based on the identity provider that you use. The access control policies define areas of the dashboard that users or groups of users can access and tasks that they can perform in these areas.

For information about configuring specific identity providers and policies, see:

- “Configure Dashboard Access Control Using Azure AD” on page 9-87
- “Configure Dashboard Access Control Using Google Identity” on page 9-90
- “Configure Dashboard Access Control Using PingFederate Identity Provider” on page 9-92
- “Configure Dashboard Access Control Using Other OpenID Connect Providers” on page 9-95

Enable Access Control

After you configure the identity provider and specify access control policies, you must enable dashboard access control. After enabling dashboard access control, a dashboard login URL that supports single sign-on (SSO) becomes available. Share this URL with managers and application authors.



See Also

More About

- “Manage MATLAB Production Server (PAYG)” on page 9-20
- “Application Access Control” on page 9-47

Dashboard Access Control

MATLAB Production Server on Microsoft Azure lets server administrators use OpenID Connect (OIDC) identity providers such as Microsoft Azure Active Directory (Azure AD), Google identity, PingFederate from Ping Identity, and others to configure role-based access control for the dashboard. Role-based access control allows administrators to grant a dashboard user the privileges to perform tasks on the dashboard based on their role.

Dashboard User Roles

The dashboard access control feature supports the following roles.

- Application author — Application authors can upload and delete applications (deployable archives) and view logs.
- Manager — Managers can edit server settings, configure access control for applications, manage persistence services, and have all the privileges of an application author, which include uploading and deleting applications and viewing logs.
- Server administrator — Administrators can log in to server virtual machines and configure which users or groups of users can access the dashboard, and have all the privileges of a manager, which include editing server logs, configuring access control for applications, uploading and deleting applications, and viewing logs.

The following table shows the dashboard tabs that users with these roles can access.

Role	Overview	Applications	Settings	Persistence	Manage Identity Providers	Application Access Control	Dashboard Access Control	Logs
Server administrator	✓	✓	✓	✓	✓	✓	✓	✓
Manager	✓	✓	✓	✓	✓	✓		✓
Application author	✓	✓						✓

Note Only the server administrator can log in to the dashboard when dashboard access control is disabled or not configured. Only the server administrator has privileges to configure dashboard access control.

Configure Identity Provider and Specify Access Control Policies

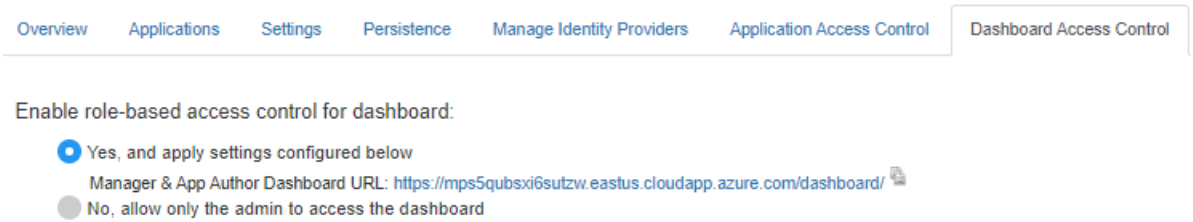
To enable dashboard access control for MATLAB Production Server, you must configure an identity provider and specify access control policies. The fields required to configure an identity provider vary based on the identity provider that you use. The access control policies define areas of the dashboard that users or groups of users can access and tasks that they can perform in these areas.

For information about configuring specific identity providers and policies, see:

- “Configure Dashboard Access Control Using Azure AD” on page 9-75
- “Configure Dashboard Access Control Using Google Identity” on page 9-78
- “Configure Dashboard Access Control Using PingFederate Identity Provider” on page 9-80
- “Configure Dashboard Access Control Using Other OpenID Connect Providers” on page 9-83

Enable Access Control

After you configure the identity provider and specify access control policies, you must enable dashboard access control. After enabling dashboard access control, a dashboard login URL that supports single sign-on (SSO) becomes available. Share this URL with managers and application authors.



See Also

More About

- MATLAB Production Server Reference Architecture on Azure
- “Manage MATLAB Production Server Using the Dashboard” on page 9-25
- “Application Access Control” on page 9-49

Execute MATLAB Functions on MATLAB Production Server (BYOL)

Client applications for MATLAB Production Server bring your own license (BYOL) are different from those for on-premises server instances in several ways. To execute MATLAB functions deployed on MATLAB Production Server (BYOL), you must use the MATLAB execution endpoint URL specified in the cloud console. Depending on the implementation of your client program, you might have to update your code to use the Azure application gateway self-signed SSL certificate and cookie-based session affinity.

Similar to client applications for on-premise server installations, you must use the MATLAB Production Server client libraries for client applications that you write using Java®, .NET, C, and Python®.

Use MATLAB Execution Endpoint URL

After your MATLAB Production Server (BYOL) deployment to Azure is complete, log in to the dashboard to retrieve the MATLAB execution endpoint. The **Overview** tab in the dashboard specifies the **MATLAB Execution Endpoint**. For information on accessing the dashboard, see “Connect to Dashboard” on page 9-15.

This endpoint is an HTTPS URL that client programs use to make requests to the server and execute MATLAB functions deployed to the server. For example, if the MATLAB execution endpoint for your server is `https://mpst4ezclcdtlcay.eastus.cloudapp.azure.com`, to use the MATLAB Production Server RESTful API to execute a MATLAB function `mymagic` located in a deployed application `myapp`, specify the URL `https://mpst4ezclcdtlcay.eastus.cloudapp.azure.com/myapp/mymagic`.

Download Client Libraries

If you want to write client programs in Java, .NET, C, and Python for invoking MATLAB functions deployed on the server, you must use the MATLAB Production Server client libraries. Download the client libraries from MATLAB Production Server Client Libraries.

Work with Self-Signed Certificate

The Azure application gateway in the deployment requires an SSL certificate. The application gateway provides the MATLAB execution endpoint, which is an HTTPS URL that client programs use to send requests to the server. IT is recommended that the application gateway uses an SSL certificate signed by a certificate authority. However, if your application gateway uses a self-signed certificate, your client programs might require some modifications.

Client programs might require disabling host name verification to avoid encountering an exception caused by a failure in host name verification. The verification can fail due to a mismatch between the host names in the HTTPS URL for MATLAB function execution and the common name (CN) of the self-signed certificate. For example, the host name for the MATLAB execution endpoint can have the value `<uniqueID>.<location>.cloudapp.azure.com`, but the CN of your self-signed certificate can have the value `azure.com`.

Depending on the implementation of your client program, you might also have to retrieve the self-signed certificate that the application gateway uses and add the certificate to your local truststore.

For more information on configuring the client environment, see “Handle Exceptions” for a Java client and “Handle Exceptions” for a .NET client.

Manage HTTP Cookie

The Azure application gateway provides cookie-based session affinity, where it uses cookies to keep a user session on the same server. On receiving a request from a client program, the application gateway sets the `Set-Cookie` HTTP response header with information about the server virtual machine (VM) that processes the request.

Asynchronous Request Execution

A client program that uses asynchronous requests to execute a MATLAB function deployed to the server must set the `Cookie` HTTP request header with the value of the `Set-Cookie` header for all subsequent requests. This ensures that same server VM that processes the first request processes all subsequent requests for that session.

Synchronous Request Execution

A client program that uses synchronous requests to execute a MATLAB function deployed to the server must not set the `Cookie` HTTP request header with the value of the `Set-Cookie` header, and must clear the value of the `Cookie` header if it has been previously set. This ensures that the synchronous requests are load balanced and the same server VM does not process them.

The default property in a Java client that uses `MWHttpClient` sets HTTP cookies. For information about disabling cookies, see “Configure Client-Server Connection”. The Java client API that uses protobuf and the .NET client API do not set HTTP cookies by default.

See Also

More About

- “Manage MATLAB Production Server (BYOL)” on page 9-15
- “Manage Azure Resources for MATLAB Production Server (BYOL)” on page 9-30
- “RESTful API for MATLAB Function Execution”

Execute MATLAB Functions on MATLAB Production Server (PAYG)

Client applications for MATLAB Production Server pay as you go (PAYG) are different from those for on-premises server instances in several ways. To execute MATLAB functions deployed on MATLAB Production Server (PAYG), you must use the MATLAB execution endpoint URL specified in the dashboard. Depending on the implementation of your client program, you might have to update your code to use the Azure application gateway self-signed SSL certificate and cookie-based session affinity.

Similar to client applications for on-premise server installations, you must use the MATLAB Production Server client libraries for client applications that you write using Java, .NET, C, and Python.

Use MATLAB Execution Endpoint URL

After your MATLAB Production Server (PAYG) deployment to Azure is complete, log in to the dashboard to retrieve the MATLAB execution endpoint. The **Overview** tab in the dashboard specifies the **MATLAB Execution Endpoint**. For information on accessing the dashboard, see “Connect to Dashboard” on page 9-20.

This endpoint is an HTTPS URL that client programs use to make requests to the server and execute MATLAB functions deployed to the server. For example, if the MATLAB execution endpoint for your server is `https://mpst4ezclcdtlcay.eastus.cloudapp.azure.com`, to use the MATLAB Production Server RESTful API to execute a MATLAB function `mymagic` located in a deployed application `myapp`, specify the URL `https://mpst4ezclcdtlcay.eastus.cloudapp.azure.com/myapp/mymagic`.

Download Client Libraries

If you want to write client programs in Java, .NET, C, and Python for invoking MATLAB functions deployed on the server, you must use the MATLAB Production Server client libraries. Download the client libraries from `https://www.mathworks.com/products/matlab-production-server/client-libraries.html`.

Work with Self-Signed Certificate

The Azure application gateway in the deployment requires an SSL certificate. The application gateway provides the MATLAB execution endpoint, which is an HTTPS URL that client programs use to send requests to the server. IT is recommended that the application gateway uses an SSL certificate signed by a certificate authority. However, if your application gateway uses a self-signed certificate, your client programs might require some modifications.

Client programs might require disabling host name verification to avoid encountering an exception caused by a failure in host name verification. The verification can fail due to a mismatch between the host names in the HTTPS URL for MATLAB function execution and the common name (CN) of the self-signed certificate. For example, the host name for the MATLAB execution endpoint can have the value `<uniqueID>.<location>.cloudapp.azure.com`, but the CN of your self-signed certificate can have the value `azure.com`.

Depending on the implementation of your client program, you might also have to retrieve the self-signed certificate that the application gateway uses and add the certificate to your local truststore.

For more information on configuring the client environment, see “Handle Exceptions” for a Java client and “Handle Exceptions” for a .NET client.

Manage HTTP Cookie

The Azure application gateway provides cookie-based session affinity, where it uses cookies to keep a user session on the same server. On receiving a request from a client program, the application gateway sets the `Set-Cookie` HTTP response header with information about the server virtual machine (VM) that processes the request.

Asynchronous Request Execution

A client program that uses asynchronous requests to execute a MATLAB function deployed to the server must set the `Cookie` HTTP request header with the value of the `Set-Cookie` header for all subsequent requests. This ensures that same server VM that processes the first request processes all subsequent requests for that session.

Synchronous Request Execution

A client program that uses synchronous requests to execute a MATLAB function deployed to the server must not set the `Cookie` HTTP request header with the value of the `Set-Cookie` header, and must clear the value of the `Cookie` header if it has been previously set. This ensures that the synchronous requests are load balanced and the same server VM does not process them.

Synchronous Request Execution

A client program that uses synchronous requests to execute a MATLAB function deployed to the server must not set the `Cookie` HTTP request header with the value of the `Set-Cookie` header, and must clear the value of the `Cookie` header if it has been previously set. This ensures that the synchronous requests are load balanced and the same server VM does not process them.

The default property in a Java client that uses `MWHttpClient` sets HTTP cookies. For information about disabling cookies, see “Configure Client-Server Connection”. The Java client API that uses protobuf and the .NET client API do not set HTTP cookies by default.

See Also

More About

- “Manage MATLAB Production Server (PAYG)” on page 9-20
- “Manage Azure Resources for MATLAB Production Server (PAYG)” on page 9-32
- “RESTful API for MATLAB Function Execution”

Execute MATLAB Functions on MATLAB Production Server Reference Architecture

Client applications for MATLAB Production Server on the cloud are different from those for on-premises server instances in several ways. To execute MATLAB functions deployed on MATLAB Production Server on the cloud, you must use the MATLAB execution endpoint URL specified in the dashboard. Depending on the implementation of your client program, you might have to update your code to use the Azure application gateway self-signed SSL certificate and cookie-based session affinity.

Similar to client applications for on-premises server installations, you must use the MATLAB Production Server client libraries for client applications that you write using Java, .NET, C, and Python.

Use MATLAB Execution Endpoint URL

After your MATLAB Production Server deployment to Azure is complete, log in to the dashboard to retrieve the MATLAB execution endpoint. The **Overview** tab in the dashboard specifies the **MATLAB Execution Endpoint**. For information on accessing the dashboard, see “Connect to Dashboard” on page 9-25.

This endpoint is an HTTPS URL that client programs use to make requests to the server and execute MATLAB functions deployed to the server. For example, if the MATLAB execution endpoint for your server is `https://mpst4ezclcdtlcay.eastus.cloudapp.azure.com`, to use the MATLAB Production Server RESTful API to execute a MATLAB function `mymagic` located in a deployed application `myapp`, specify the URL `https://mpst4ezclcdtlcay.eastus.cloudapp.azure.com/myapp/mymagic`.

Download Client Libraries

If you want to write client programs in Java, .NET, C, and Python for invoking MATLAB functions deployed on the server, you must use the MATLAB Production Server client libraries. To download the client libraries, see MATLAB Production Server Client Libraries.

Work with Self-Signed Certificate

The Azure application gateway in the deployment requires an SSL certificate. The application gateway provides the MATLAB execution endpoint, which is an HTTPS URL that client programs use to send requests to the server. IT is recommended that the application gateway uses an SSL certificate signed by a certificate authority. However, if your application gateway uses a self-signed certificate, your client programs might require some modifications.

Client programs might require disabling host name verification to avoid encountering an exception caused by a failure in host name verification. The verification can fail due to a mismatch between the host names in the HTTPS URL for MATLAB function execution and the common name (CN) of the self-signed certificate. For example, the host name for the MATLAB execution endpoint can have the value `<uniqueID>.<location>.cloudapp.azure.com`, but the CN of your self-signed certificate can have the value `azure.com`.

Depending on the implementation of your client program, you might also have to retrieve the self-signed certificate that the application gateway uses and add the certificate to your local truststore.

For more information on configuring the client environment, see “Handle Exceptions” for a Java client and “Handle Exceptions” for a .NET client.

Manage HTTP Cookie

The Azure application gateway provides cookie-based session affinity, where it uses cookies to keep a user session on the same server. On receiving a request from a client program, the application gateway sets the `Set-Cookie` HTTP response header with information about the server virtual machine (VM) that processes the request.

Asynchronous Request Execution

A client program that uses asynchronous requests to execute a MATLAB function deployed to the server must set the `Cookie` HTTP request header with the value of the `Set-Cookie` header for all subsequent requests. This ensures that same server VM that processes the first request processes all subsequent requests for that session.

Synchronous Request Execution

A client program that uses synchronous requests to execute a MATLAB function deployed to the server must not set the `Cookie` HTTP request header with the value of the `Set-Cookie` header, and must clear the value of the `Cookie` header if it has been previously set. This ensures that the synchronous requests are load balanced and the same server VM does not process them.

The default property in a Java client that uses `MWHttpClient` sets HTTP cookies. For information about disabling cookies, see “Configure Client-Server Connection”. The Java client API that uses protobuf and the .NET client API do not set HTTP cookies by default.

See Also

More About

- MATLAB Production Server on Microsoft Azure
- “Manage MATLAB Production Server Using the Dashboard” on page 9-25
- “RESTful API for MATLAB Function Execution”
- “Synchronous RESTful Requests Using Protocol Buffers in the Java Client”
- “Synchronous RESTful Requests Using Protocol Buffers in .NET Client”

Configure Dashboard Access Control Using Azure AD

MATLAB Production Server administrators can use Microsoft Azure AD to configure role-based access control for the MATLAB Production Server Dashboard. Role-based access control allows administrators to grant access to specific areas of the dashboard to certain users or groups of users. For more information about the roles that the dashboard supports, see “Dashboard Access Control” on page 9-51.

To enable dashboard access control, configure Azure AD and specify access control policies, in consultation with the Azure AD administrator.

Configure Identity Provider

To configure Azure AD:

1. Log in to the dashboard to retrieve the redirect URI of the dashboard.
2. In the Azure portal, use the redirect URI to register the dashboard as a client application with the provider.
3. In the dashboard, enter values specific to the registered application and Azure AD.

Retrieve Redirect URI from Dashboard

To retrieve the redirect URI, start creating a configuration for Azure AD in the dashboard:

1. Navigate to either the **Dashboard Access Control** tab or the **Manage Identity Providers** tab.
2. Click **Create** and select **Azure AD**.
3. In **Create Identity Provider for Dashboard Access Control**, note the redirect URI of the dashboard.

   Later, you return to this view to specify the values required to configure your identify provider in the dashboard.

Register Application in Azure Portal

Use the Azure portal to register a web client application for dashboard access control. When registering the application, use the redirect URI from the MATLAB Production Server dashboard. Typically, the Azure AD administrator registers the application.

1. Sign in to the Azure portal.
2. From **Azure Active Directory**, select **App registrations** and click **New registration**.
3. In the resulting pane, enter the name of the application (for example, `MATLAB Production Server Dashboard App`).
4. For the **Redirect URI**, select **Web**. In the corresponding value field, enter the redirect URI of the dashboard and click **Register**. A web page displays the details of your registered application.
5. Click **Manifest** in the left navigation pane. In the JSON that is displayed in the resulting pane, set the value for `groupMembershipClaims` to `"SecurityGroup"`. Click **Save**.

For more information on how to register an application, see the Microsoft Azure documentation.

Specify Values in Dashboard

In the Azure portal, find the values of the client application that you registered and enter them into the dashboard.

1. Sign in to the Azure portal.
2. From **Azure Active Directory**, select **App registrations** and then select the application that you registered for the dashboard. Copy the value from the **Application (client) ID** and paste it into the **Client ID** field in the dashboard.
3. From **App registrations**, select **Certificates & secrets**. Under **Certificates & secrets**, create a new client secret or use an existing one. Copy the value for the client secret and paste it into the **Client Secret** field in the dashboard.
4. From **Azure Active Directory**, select **Properties**. Copy the value from **Directory (tenant) ID** and paste it into the **Tenant ID** in the dashboard.
5. On the dashboard, click **Create**.

Specify Dashboard Access Control Policy

Before you can specify dashboard access control policies, you must have users and groups set up in Azure AD. Consult the Azure AD administrator for this setup.

The access control policies define areas of the dashboard that users or groups of users can access and tasks that they can perform in these areas. Use the policies to assign the manager and application author roles to users or groups of users in your organization by entering their Azure user names and group IDs into the dashboard.

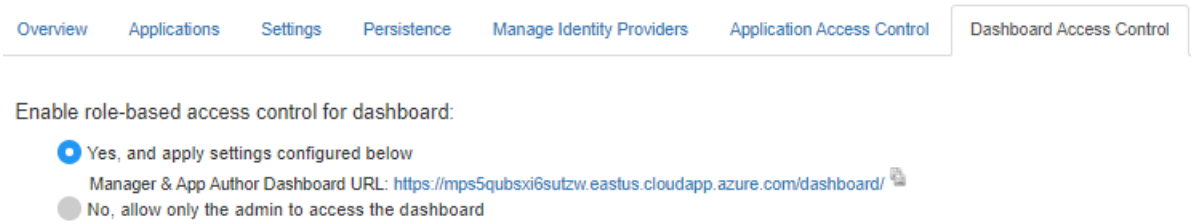
Configure Users and Groups in Dashboard

In the Azure portal, find user names and group IDs and enter them into the dashboard.

1. Sign in to the Azure portal.
2. From **Azure Active Directory** and then **Users**. Copy the values for the user names and paste them into the **Users** field in the dashboard. Use a comma to separate multiple user names.
3. From **Azure Active Directory** and then **Groups**. Copy the values for the object IDs and paste them into the **Groups** field in the dashboard. Use a comma to separate multiple object IDs names.
4. On the dashboard, click **Save**.

Enable Dashboard Access Control

After you configure Azure AD and specify access control policies, you must enable dashboard access control by selecting the **Yes** option. After enabling dashboard access control, a dashboard login URL that supports single sign-on (SSO) becomes available. Share this URL with managers and application authors.



See Also

Related Examples

- “Dashboard Access Control” on page 9-51
- “Configure Dashboard Access Control Using Google Identity” on page 9-66
- “Configure Dashboard Access Control Using PingFederate Identity Provider” on page 9-68
- “Configure Dashboard Access Control Using Other OpenID Connect Providers” on page 9-71

Configure Dashboard Access Control Using Google Identity

MATLAB Production Server administrators can use Google identity provider to configure role-based access control for the MATLAB Production Server Dashboard. Role-based access control allows administrators to grant access to specific areas of the dashboard to certain users or groups of users. For more information about the roles that the dashboard supports, see “Dashboard Access Control” on page 9-51.

To enable dashboard access control, configure the identity provider and specify access control policies, in consultation with the Google Identity administrator.

Configure Google Identity

To configure Google Identity:

1. Log in to the dashboard to retrieve the redirect URI of the dashboard.
2. On the Google Cloud Platform Console, use the redirect URI to register the dashboard as a client application with the provider.
3. In the dashboard, enter values specific to the registered application and Google Identity.

Retrieve Redirect URI from Dashboard

To retrieve the redirect URI, start creating a configuration for Google Identity in the dashboard:

1. Navigate to either the **Dashboard Access Control** tab or the **Manage Identity Providers** tab.
2. Click **Create** and select **Google**.
3. In **Create Identity Provider for Dashboard Access Control**, note the redirect URI of the dashboard.

   Later, you return to this view to specify the values required to configure your identify provider in the dashboard.

Register Application in Google Cloud Console

Use the Google Cloud Console to register a web client application for dashboard access control. Use the redirect URI from the MATLAB Production Server Dashboard when registering the application.

1. Sign in to the Google Cloud Platform Console and navigate to the **Credentials** page.
2. On the **Credentials** page, click **Create credentials** and select **OAuth client ID**.
3. From **Application type** drop down, select **Web Application**.
4. Enter the name of you client application (for example, `MATLAB Production Server Dashboard App`).
5. Under **Authorized redirect URIs**, click **Add URI**.
6. Copy the redirect URI from MATLAB Production Server Dashboard and paste it into the **URIs** field in the Google Cloud Console.
7. Click **Create**.
8. The Google identity provider creates an application with a client ID and client secret. Note the values of the client ID and client secret. You enter these values next in the dashboard.

Specify Client ID and Client Secret in Dashboard

1. Enter the noted client ID and client secret values from the previous section in the **Client ID** and **Client Secret** fields respectively in MATLAB Production Server dashboard.
2. Click **Create** to complete the configuration of the identity provider.

Specify Dashboard Access Control Policy

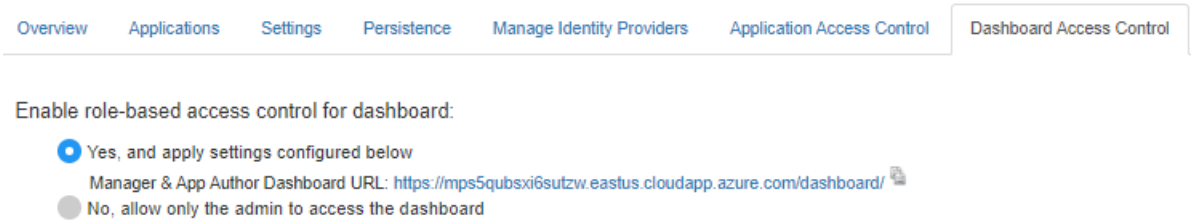
Before you can specify dashboard access control policies, you must have users, and groups, if applicable, set up in Google. Consult the Google identity provider administrator for this setup.

The access control policies define areas of the dashboard that users can access and tasks that they can perform in these areas. Use the policies to assign the manager and application author roles to users in your organization by entering their Google user names.

1. On the **Dashboard Access Control** tab of the dashboard, select Google as the identity provider.
2. In the **Dashboard Access Control Policy** section, enter Google user names to assign manager and application author roles to users in your organization. Click **Save** after you enter the values.

Enable Dashboard Access Control

After you configure Google Identity and specify access control policies, you must enable dashboard access control by selecting the **Yes** option. After enabling dashboard access control, a dashboard login URL that supports single sign-on (SSO) becomes available. Share this URL with managers and application authors.



See Also

Related Examples

- “Dashboard Access Control” on page 9-51
- “Configure Dashboard Access Control Using Azure AD” on page 9-63
- “Configure Dashboard Access Control Using PingFederate Identity Provider” on page 9-68
- “Configure Dashboard Access Control Using Other OpenID Connect Providers” on page 9-71

Configure Dashboard Access Control Using PingFederate Identity Provider

MATLAB Production Server administrators can use PingFederate from Ping Identity to configure role-based access control for the MATLAB Production Server Dashboard. Role-based access control allows administrators to grant access to specific areas of the dashboard to certain users or groups of users. For more information about the roles that the dashboard supports, see “Dashboard Access Control” on page 9-51.

To enable dashboard access control for MATLAB Production Server, configure PingFederate and specify access control policies, in consultation with the PingFederate administrator.

Prerequisites

Refer to the PingFederate documentation to configure OAuth use cases, clients, and endpoints to configure OpenID® provider information:

- Configuring OAuth Use Cases
- Configuring OAuth Clients
- OAuth 2.0 Endpoints
- Configuring OpenID Provider Information

Configure PingFederate Identity Provider

To configure PingFederate:

1. Log in to the dashboard to retrieve the Redirect URI of the dashboard.
2. Use the Redirect URI to register a client application in PingFederate.
3. In the dashboard, enter values specific to the registered application and PingFederate.

Retrieve Redirect URI from Dashboard

To retrieve the redirect URI, start creating a configuration for your identity provider in the dashboard:

1. Navigate to either the **Dashboard Access Control** tab or the **Manage Identity Providers** tab.
2. Click **Create** and select **PingFederate**.
3. In **Create Identity Provider for Dashboard Access Control**, note the redirect URI of the dashboard.

   Later, you return to this view to specify the values required to configure your identify provider in the dashboard.

Register Application in PingFederate

Register an application in PingFederate for MATLAB Production Server Dashboard, if you do not already have one. Consult the PingFederate administrator to register the application. Provide the **Redirect URI** of the MATLAB Production Server Dashboard when registering the application.

Specify Values in Dashboard

After you register the application with PingFederate, you receive application specific values such as the **Client ID** and **Client Secret**. Enter the values specific to the application and values specific to PingFederate in the dashboard under **Create Identity Provider for Dashboard Access Control**.

The following table describes the values that you must enter. Click **Create** after you enter the values.

Field	Description
Client ID	Application ID of the registered client application.
Client Secret	Client secret of the registered client application.
OIDC Issuer	Discovery endpoint URI of the OIDC provider.
JWT Issuer	JWT issuer metadata of the OIDC provider.
JWKS URI	URI to retrieve the JSON Web Key Set (JWKS).

Specify Dashboard Access Control Policy

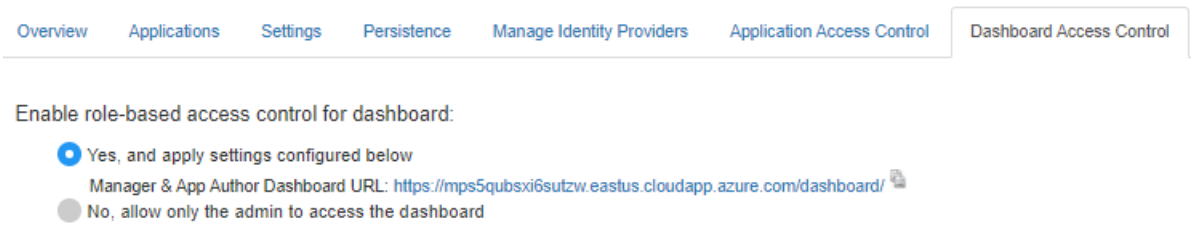
Before you can specify dashboard access control policies, you must have users, and groups, if applicable, set up in PingFederate. Consult the PingFederate administrator for this setup.

The access control policies define areas of the dashboard that users or groups of users can access and tasks that they can perform in these areas. Use the policies to assign the manager and application author roles to users or groups of users in your organization by entering their user names and group IDs. Click **Save** after you enter the values.

1. In the **Dashboard Access Control** tab of the dashboard, select PingFederate as the identity provider.
2. In the **Dashboard Access Control Policy** section, enter identity provider specific user names and group IDs to assign manager and application author roles to users or groups of users in your organization. Use a comma to separate multiple user names and group IDs. Click **Save** after you enter the values.

Enable Dashboard Access Control

After you configure PingFederate and specify access control policies, you must enable dashboard access control by selecting the **Yes** option. After enabling dashboard access control, a dashboard login URL that supports single sign-on (SSO) becomes available. Share this URL with managers and application authors.



See Also

Related Examples

- “Dashboard Access Control” on page 9-51
- “Configure Dashboard Access Control Using Other OpenID Connect Providers” on page 9-71
- “Configure Dashboard Access Control Using Azure AD” on page 9-63
- “Configure Dashboard Access Control Using Google Identity” on page 9-66

Configure Dashboard Access Control Using Other OpenID Connect Providers

MATLAB Production Server Dashboard supports the use of any OpenID Connect (OIDC) identity provider for role-based access control for the dashboard. Role-based access control allows server administrators to grant access to specific areas of the dashboard to certain users or groups of users. For more information about the roles that the dashboard supports, see “Dashboard Access Control” on page 9-51.

To enable dashboard access control, configure the OIDC provider and specify dashboard access control policies, in consultation with the OIDC provider administrator.

Configure Identity Provider

To configure an identity provider:

1. Log in to the dashboard to retrieve the redirect URI of the dashboard.
2. At the identity provider's website, use the redirect URI to register the dashboard as a client application with the provider.
3. In the dashboard, enter values specific to the registered application and identity provider.

Retrieve Redirect URI from Dashboard

To retrieve the redirect URI, start creating a configuration for your identity provider in the dashboard:

1. Navigate to either the **Dashboard Access Control** tab or the **Manage Identity Providers** tab.
2. Click **Create** and select **Other**.
3. In **Create Identity Provider for Dashboard Access Control**, note the redirect URI of the dashboard.

   Later, you return to this view to specify the values required to configure your identify provider in the dashboard.

Register Application with Identity Provider

Register an application with the OIDC provider for MATLAB Production Server Dashboard. Consult the OIDC provider administrator to register the application. When registering the application, provide the redirect URI of the MATLAB Production Server Dashboard.

Specify Values in Dashboard

After you register the application with the identity provider, you receive application-specific values such as the client ID and client secret. Enter the values in the dashboard under **Create Identity Provider for Dashboard Access Control**.

The following table describes the values that you must enter. Click **Create** after you enter the values.

Field	Description
Client ID	Application ID of the registered client application

Field	Description
Client Secret	Client secret of the registered client application
OIDC Issuer	Discovery endpoint URI of the OIDC provider
JWT Issuer	JWT issuer metadata of the OIDC provider
JWKS URI	URI to retrieve the JSON Web Key Set (JWKS)

Under **Create Identity Provider for Dashboard Access Control**, you have the option to provide values other than the defaults for **UserAttribute ID** and **GroupAttribute ID**. **UserAttribute ID** is the JWT claim name that uniquely identifies a user. **GroupAttribute ID** is the JWT claim name that lists the groups that a user belongs to. Depending on the identity provider you use, you might have to change the defaults.

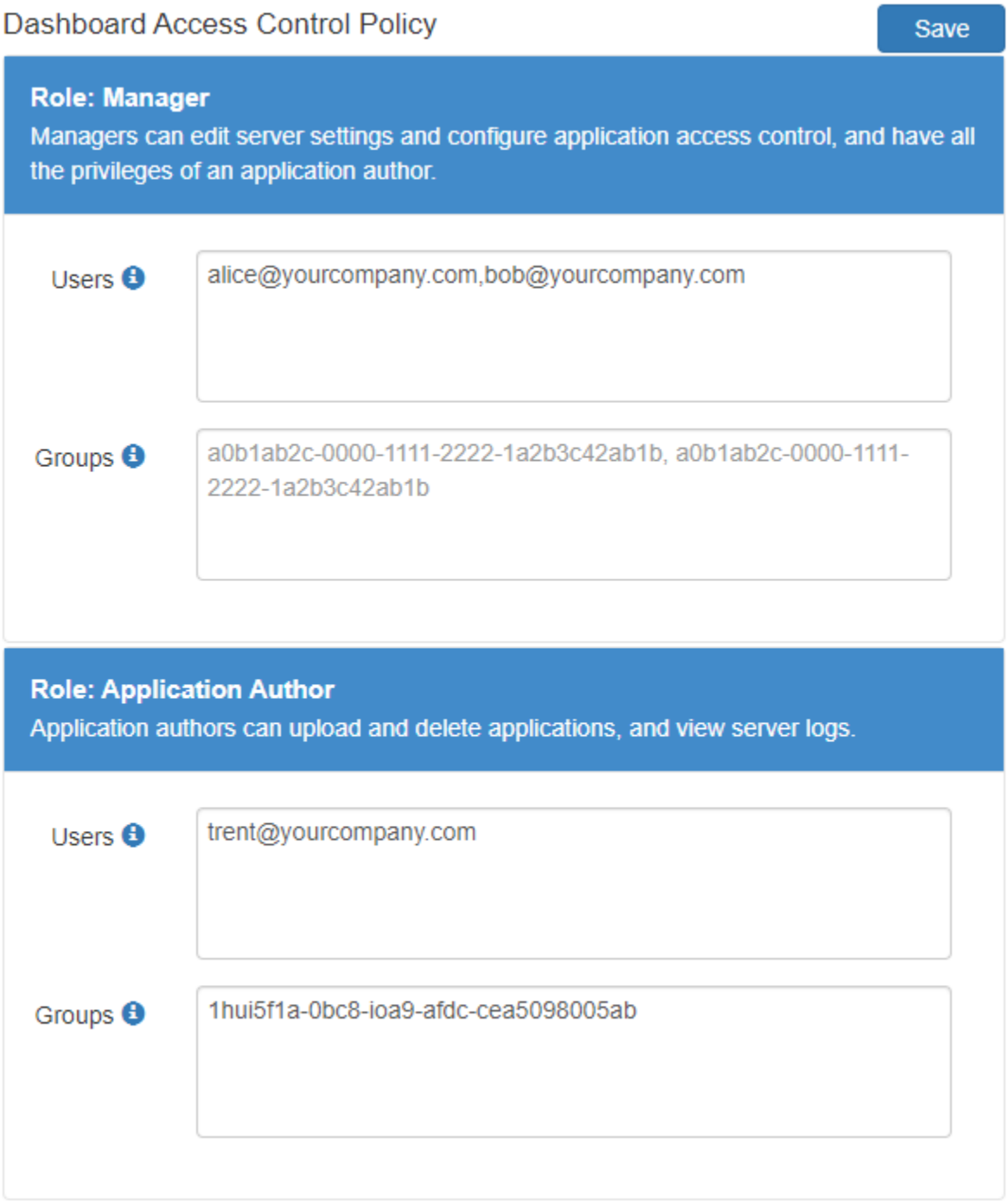
Specify Dashboard Access Control Policy

Before you can specify dashboard access control policies, you must have users, and if applicable, groups, set up in the identity provider. Consult the OIDC provider administrator for this setup.

The access control policies define areas of the dashboard that users or groups of users can access and tasks that they can perform in these areas.

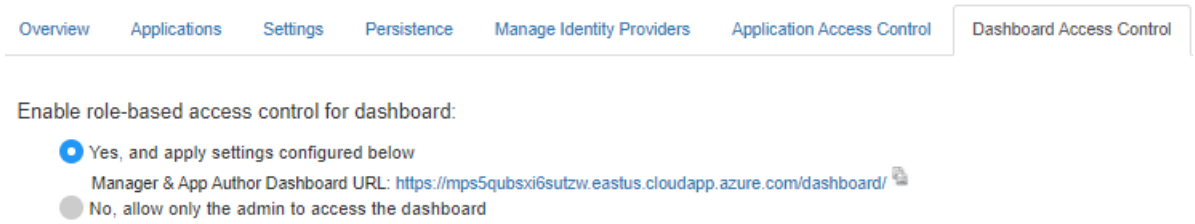
1. On the **Dashboard Access Control** tab of the dashboard, select the identity provider that you want to use.
2. In the **Dashboard Access Control Policy** section, enter identity provider specific user names and group IDs to assign manager and application author roles to users or groups of users in your organization. Use a comma to separate multiple user names or group IDs. Click **Save** after you enter the values.

   For example, in the following illustration, the users *alice@yourcompany.com* and *bob@yourcompany.com* have the manager role. The user *trent@yourcompany.com* and all users that belong to group ID *1hui5f1a-0bc8-ioa9-afdc-cea5098005ab* have the application author role.



Enable Dashboard Access Control

After you configure the identity provider and specify access control policies, you must enable dashboard access control by selecting the **Yes** option. After enabling dashboard access control, a dashboard login URL that supports single sign-on (SSO) becomes available. Share this URL with managers and application authors.



See Also

Related Examples

- “Dashboard Access Control” on page 9-51
- “Configure Dashboard Access Control Using Azure AD” on page 9-63
- “Configure Dashboard Access Control Using Google Identity” on page 9-66
- “Configure Dashboard Access Control Using PingFederate Identity Provider” on page 9-68

Configure Dashboard Access Control Using Azure AD

MATLAB Production Server administrators can use Microsoft Azure AD to configure role-based access control for the MATLAB Production Server Dashboard. Role-based access control allows administrators to grant access to specific areas of the dashboard to certain users or groups of users. For more information about the roles that the dashboard supports, see “Dashboard Access Control” on page 9-51.

To enable dashboard access control, configure Azure AD and specify access control policies, in consultation with the Azure AD administrator.

Configure Identity Provider

To configure Azure AD:

1. Log in to the dashboard to retrieve the redirect URI of the dashboard.
2. In the Azure portal, use the redirect URI to register the dashboard as a client application with the provider.
3. In the dashboard, enter values specific to the registered application and Azure AD.

Retrieve Redirect URI from Dashboard

To retrieve the redirect URI, start creating a configuration for Azure AD in the dashboard:

1. Navigate to either the **Dashboard Access Control** tab or the **Manage Identity Providers** tab.
2. Click **Create** and select **Azure AD**.
3. In **Create Identity Provider for Dashboard Access Control**, note the redirect URI of the dashboard.

   Later, you return to this view to specify the values required to configure your identify provider in the dashboard.

Register Application in Azure Portal

Use the Azure portal to register a web client application for dashboard access control. When registering the application, use the redirect URI from the MATLAB Production Server dashboard. Typically, the Azure AD administrator registers the application.

1. Sign in to the Azure portal.
2. From **Azure Active Directory**, select **App registrations** and click **New registration**.
3. In the resulting pane, enter the name of the application (for example, `MATLAB Production Server Dashboard App`).
4. For the **Redirect URI**, select **Web**. In the corresponding value field, enter the redirect URI of the dashboard and click **Register**. A web page displays the details of your registered application.
5. Click **Manifest** in the left navigation pane. In the JSON that is displayed in the resulting pane, set the value for `groupMembershipClaims` to `"SecurityGroup"`. Click **Save**.

For more information on how to register an application, see the Microsoft Azure documentation.

Specify Values in Dashboard

In the Azure portal, find the values of the client application that you registered and enter them into the dashboard.

1. Sign in to the Azure portal.
2. From **Azure Active Directory**, select **App registrations** and then select the application that you registered for the dashboard. Copy the value from the **Application (client) ID** and paste it into the **Client ID** field in the dashboard.
3. From **App registrations**, select **Certificates & secrets**. Under **Certificates & secrets**, create a new client secret or use an existing one. Copy the value for the client secret and paste it into the **Client Secret** field in the dashboard.
4. From **Azure Active Directory**, select **Properties**. Copy the value from **Directory (tenant) ID** and paste it into the **Tenant ID** in the dashboard.
5. On the dashboard, click **Create**.

Specify Dashboard Access Control Policy

Before you can specify dashboard access control policies, you must have users and groups set up in Azure AD. Consult the Azure AD administrator for this setup.

The access control policies define areas of the dashboard that users or groups of users can access and tasks that they can perform in these areas. Use the policies to assign the manager and application author roles to users or groups of users in your organization by entering their Azure user names and group IDs into the dashboard.

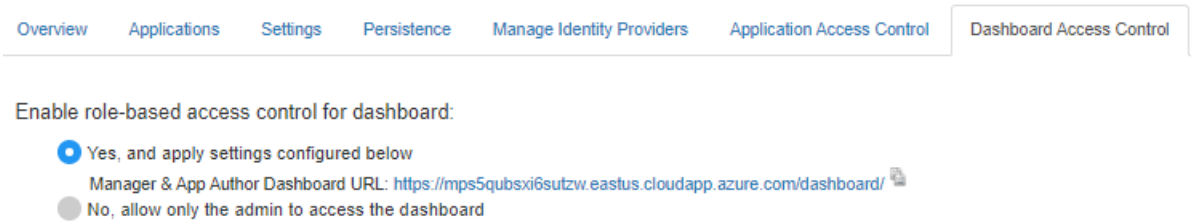
Configure Users and Groups in Dashboard

In the Azure portal, find user names and group IDs and enter them into the dashboard.

1. Sign in to the Azure portal.
2. From **Azure Active Directory** and then **Users**. Copy the values for the user names and paste them into the **Users** field in the dashboard. Use a comma to separate multiple user names.
3. From **Azure Active Directory** and then **Groups**. Copy the values for the object IDs and paste them into the **Groups** field in the dashboard. Use a comma to separate multiple object IDs names.
4. On the dashboard, click **Save**.

Enable Dashboard Access Control

After you configure Azure AD and specify access control policies, you must enable dashboard access control by selecting the **Yes** option. After enabling dashboard access control, a dashboard login URL that supports single sign-on (SSO) becomes available. Share this URL with managers and application authors.



See Also

Related Examples

- “Dashboard Access Control” on page 9-55
- “Configure Dashboard Access Control Using Google Identity” on page 9-78
- “Configure Dashboard Access Control Using PingFederate Identity Provider” on page 9-80
- “Configure Dashboard Access Control Using Other OpenID Connect Providers” on page 9-83

Configure Dashboard Access Control Using Google Identity

MATLAB Production Server administrators can use Google identity provider to configure role-based access control for the MATLAB Production Server Dashboard. Role-based access control allows administrators to grant access to specific areas of the dashboard to certain users or groups of users. For more information about the roles that the dashboard supports, see “Dashboard Access Control” on page 9-51.

To enable dashboard access control, configure the identity provider and specify access control policies, in consultation with the Google Identity administrator.

Configure Google Identity

To configure Google Identity:

1. Log in to the dashboard to retrieve the redirect URI of the dashboard.
2. On the Google Cloud Platform Console, use the redirect URI to register the dashboard as a client application with the provider.
3. In the dashboard, enter values specific to the registered application and Google Identity.

Retrieve Redirect URI from Dashboard

To retrieve the redirect URI, start creating a configuration for Google Identity in the dashboard:

1. Navigate to either the **Dashboard Access Control** tab or the **Manage Identity Providers** tab.
2. Click **Create** and select **Google**.
3. In **Create Identity Provider for Dashboard Access Control**, note the redirect URI of the dashboard.

   Later, you return to this view to specify the values required to configure your identify provider in the dashboard.

Register Application in Google Cloud Console

Use the Google Cloud Console to register a web client application for dashboard access control. Use the redirect URI from the MATLAB Production Server Dashboard when registering the application.

1. Sign in to the Google Cloud Platform Console and navigate to the **Credentials** page.
2. On the **Credentials** page, click **Create credentials** and select **OAuth client ID**.
3. From **Application type** drop down, select **Web Application**.
4. Enter the name of you client application (for example, `MATLAB Production Server Dashboard App`).
5. Under **Authorized redirect URIs**, click **Add URI**.
6. Copy the redirect URI from MATLAB Production Server Dashboard and paste it into the **URIs** field in the Google Cloud Console.
7. Click **Create**.
8. The Google identity provider creates an application with a client ID and client secret. Note the values of the client ID and client secret. You enter these values next in the dashboard.

Specify Client ID and Client Secret in Dashboard

1. Enter the noted client ID and client secret values from the previous section in the **Client ID** and **Client Secret** fields respectively in MATLAB Production Server dashboard.
2. Click **Create** to complete the configuration of the identity provider.

Specify Dashboard Access Control Policy

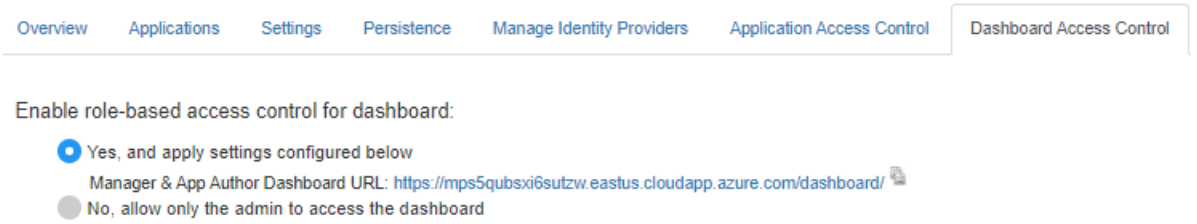
Before you can specify dashboard access control policies, you must have users, and groups, if applicable, set up in Google. Consult the Google identity provider administrator for this setup.

The access control policies define areas of the dashboard that users can access and tasks that they can perform in these areas. Use the policies to assign the manager and application author roles to users in your organization by entering their Google user names.

1. On the **Dashboard Access Control** tab of the dashboard, select Google as the identity provider.
2. In the **Dashboard Access Control Policy** section, enter Google user names to assign manager and application author roles to users in your organization. Click **Save** after you enter the values.

Enable Dashboard Access Control

After you configure Google Identity and specify access control policies, you must enable dashboard access control by selecting the **Yes** option. After enabling dashboard access control, a dashboard login URL that supports single sign-on (SSO) becomes available. Share this URL with managers and application authors.



See Also

Related Examples

- “Dashboard Access Control” on page 9-55
- “Configure Dashboard Access Control Using Azure AD” on page 9-75
- “Configure Dashboard Access Control Using PingFederate Identity Provider” on page 9-80
- “Configure Dashboard Access Control Using Other OpenID Connect Providers” on page 9-83

Configure Dashboard Access Control Using PingFederate Identity Provider

MATLAB Production Server administrators can use PingFederate from Ping Identity to configure role-based access control for the MATLAB Production Server Dashboard. Role-based access control allows administrators to grant access to specific areas of the dashboard to certain users or groups of users. For more information about the roles that the dashboard supports, see “Dashboard Access Control” on page 9-51.

To enable dashboard access control for MATLAB Production Server, configure PingFederate and specify access control policies, in consultation with the PingFederate administrator.

Prerequisites

Refer to the PingFederate documentation to configure OAuth use cases, clients, and endpoints to configure OpenID provider information:

- Configuring OAuth Use Cases
- Configuring OAuth Clients
- OAuth 2.0 Endpoints
- Configuring OpenID Provider Information

Configure PingFederate Identity Provider

To configure PingFederate:

1. Log in to the dashboard to retrieve the Redirect URI of the dashboard.
2. Use the Redirect URI to register a client application in PingFederate.
3. In the dashboard, enter values specific to the registered application and PingFederate.

Retrieve Redirect URI from Dashboard

To retrieve the redirect URI, start creating a configuration for your identity provider in the dashboard:

1. Navigate to either the **Dashboard Access Control** tab or the **Manage Identity Providers** tab.
2. Click **Create** and select **PingFederate**.
3. In **Create Identity Provider for Dashboard Access Control**, note the redirect URI of the dashboard.

   Later, you return to this view to specify the values required to configure your identify provider in the dashboard.

Register Application in PingFederate

Register an application in PingFederate for MATLAB Production Server Dashboard, if you do not already have one. Consult the PingFederate administrator to register the application. Provide the **Redirect URI** of the MATLAB Production Server Dashboard when registering the application.

Specify Values in Dashboard

After you register the application with PingFederate, you receive application specific values such as the **Client ID** and **Client Secret**. Enter the values specific to the application and values specific to PingFederate in the dashboard under **Create Identity Provider for Dashboard Access Control**.

The following table describes the values that you must enter. Click **Create** after you enter the values.

Field	Description
Client ID	Application ID of the registered client application.
Client Secret	Client secret of the registered client application.
OIDC Issuer	Discovery endpoint URI of the OIDC provider.
JWT Issuer	JWT issuer metadata of the OIDC provider.
JWKS URI	URI to retrieve the JSON Web Key Set (JWKS).

Specify Dashboard Access Control Policy

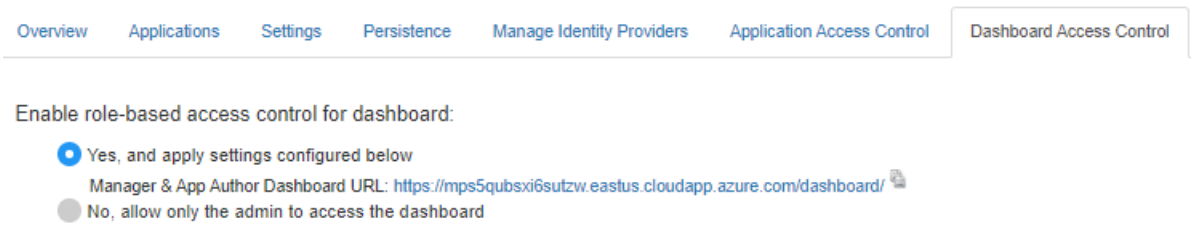
Before you can specify dashboard access control policies, you must have users, and groups, if applicable, set up in PingFederate. Consult the PingFederate administrator for this setup.

The access control policies define areas of the dashboard that users or groups of users can access and tasks that they can perform in these areas. Use the policies to assign the manager and application author roles to users or groups of users in your organization by entering their user names and group IDs. Click **Save** after you enter the values.

1. In the **Dashboard Access Control** tab of the dashboard, select PingFederate as the identity provider.
2. In the **Dashboard Access Control Policy** section, enter identity provider specific user names and group IDs to assign manager and application author roles to users or groups of users in your organization. Use a comma to separate multiple user names and group IDs. Click **Save** after you enter the values.

Enable Dashboard Access Control

After you configure PingFederate and specify access control policies, you must enable dashboard access control by selecting the **Yes** option. After enabling dashboard access control, a dashboard login URL that supports single sign-on (SSO) becomes available. Share this URL with managers and application authors.



See Also

Related Examples

- “Dashboard Access Control” on page 9-55
- “Configure Dashboard Access Control Using Azure AD” on page 9-75
- “Configure Dashboard Access Control Using Google Identity” on page 9-78
- “Configure Dashboard Access Control Using Other OpenID Connect Providers” on page 9-83

Configure Dashboard Access Control Using Other OpenID Connect Providers

MATLAB Production Server Dashboard supports the use of any OpenID Connect (OIDC) identity provider for role-based access control for the dashboard. Role-based access control allows server administrators to grant access to specific areas of the dashboard to certain users or groups of users. For more information about the roles that the dashboard supports, see “Dashboard Access Control” on page 9-51.

To enable dashboard access control, configure the OIDC provider and specify dashboard access control policies, in consultation with the OIDC provider administrator.

Configure Identity Provider

To configure an identity provider:

1. Log in to the dashboard to retrieve the redirect URI of the dashboard.
2. At the identity provider's website, use the redirect URI to register the dashboard as a client application with the provider.
3. In the dashboard, enter values specific to the registered application and identity provider.

Retrieve Redirect URI from Dashboard

To retrieve the redirect URI, start creating a configuration for your identity provider in the dashboard:

1. Navigate to either the **Dashboard Access Control** tab or the **Manage Identity Providers** tab.
2. Click **Create** and select **Other**.
3. In **Create Identity Provider for Dashboard Access Control**, note the redirect URI of the dashboard.

   Later, you return to this view to specify the values required to configure your identify provider in the dashboard.

Register Application with Identity Provider

Register an application with the OIDC provider for MATLAB Production Server Dashboard. Consult the OIDC provider administrator to register the application. When registering the application, provide the redirect URI of the MATLAB Production Server Dashboard.

Specify Values in Dashboard

After you register the application with the identity provider, you receive application-specific values such as the client ID and client secret. Enter the values in the dashboard under **Create Identity Provider for Dashboard Access Control**.

The following table describes the values that you must enter. Click **Create** after you enter the values.

Field	Description
Client ID	Application ID of the registered client application

Field	Description
Client Secret	Client secret of the registered client application
OIDC Issuer	Discovery endpoint URI of the OIDC provider
JWT Issuer	JWT issuer metadata of the OIDC provider
JWKS URI	URI to retrieve the JSON Web Key Set (JWKS)

Under **Create Identity Provider for Dashboard Access Control**, you have the option to provide values other than the defaults for **UserAttribute ID** and **GroupAttribute ID**. **UserAttribute ID** is the JWT claim name that uniquely identifies a user. **GroupAttribute ID** is the JWT claim name that lists the groups that a user belongs to. Depending on the identity provider you use, you might have to change the defaults.

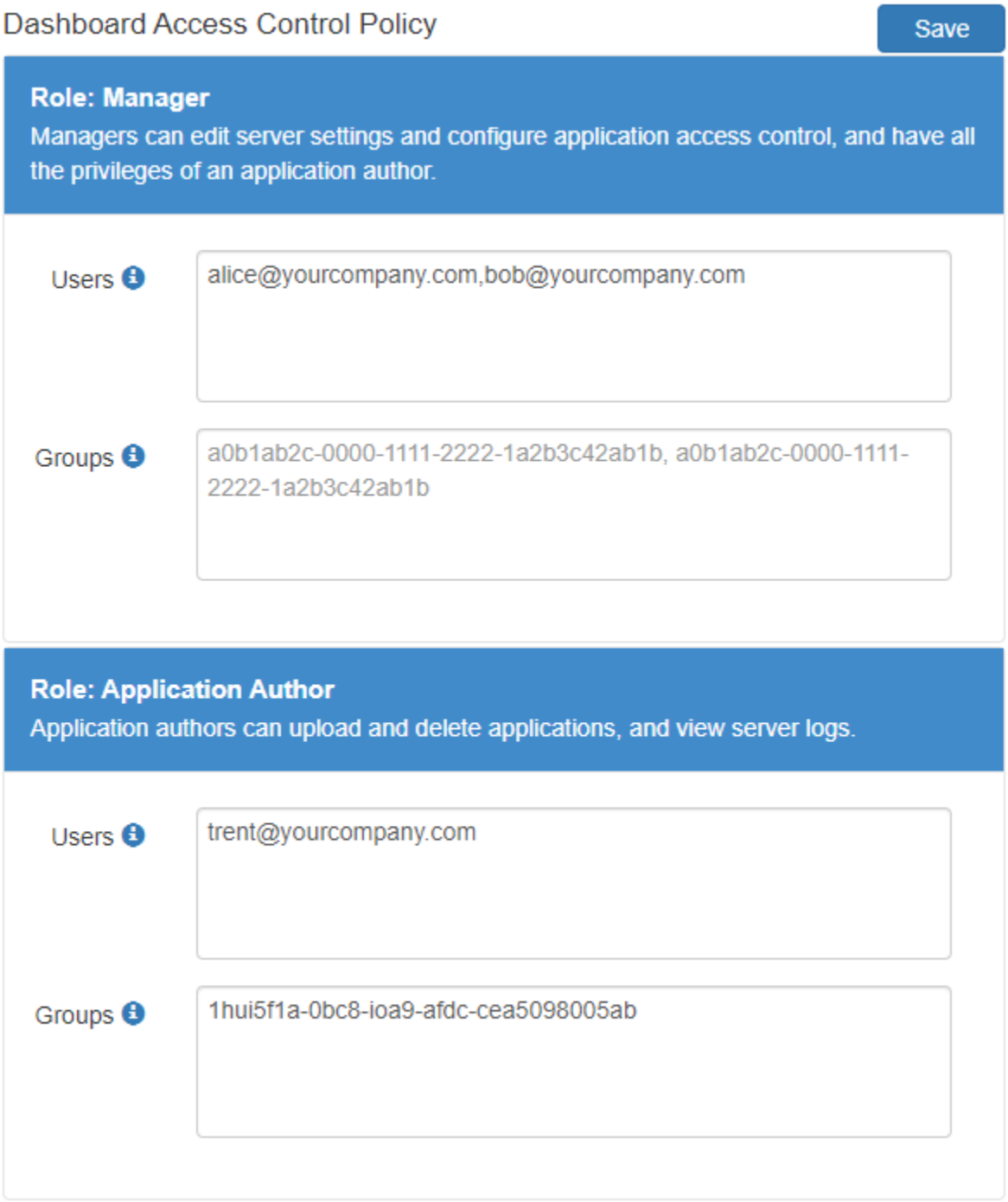
Specify Dashboard Access Control Policy

Before you can specify dashboard access control policies, you must have users, and if applicable, groups, set up in the identity provider. Consult the OIDC provider administrator for this setup.

The access control policies define areas of the dashboard that users or groups of users can access and tasks that they can perform in these areas.

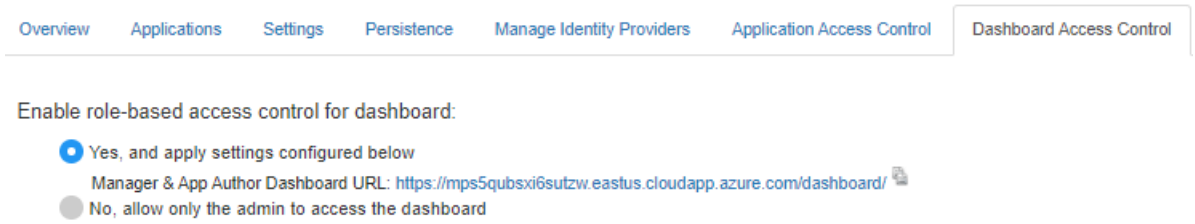
1. On the **Dashboard Access Control** tab of the dashboard, select the identity provider that you want to use.
2. In the **Dashboard Access Control Policy** section, enter identity provider specific user names and group IDs to assign manager and application author roles to users or groups of users in your organization. Use a comma to separate multiple user names or group IDs. Click **Save** after you enter the values.

   For example, in the following illustration, the users *alice@yourcompany.com* and *bob@yourcompany.com* have the manager role. The user *trent@yourcompany.com* and all users that belong to group ID *1hui5f1a-0bc8-ioa9-afdc-cea5098005ab* have the application author role.



Enable Dashboard Access Control

After you configure the identity provider and specify access control policies, you must enable dashboard access control by selecting the **Yes** option. After enabling dashboard access control, a dashboard login URL that supports single sign-on (SSO) becomes available. Share this URL with managers and application authors.



See Also

Related Examples

- “Dashboard Access Control” on page 9-55
- “Configure Dashboard Access Control Using Azure AD” on page 9-75
- “Configure Dashboard Access Control Using Google Identity” on page 9-78
- “Configure Dashboard Access Control Using PingFederate Identity Provider” on page 9-80

Configure Dashboard Access Control Using Azure AD

MATLAB Production Server administrators can use Microsoft Azure AD to configure role-based access control for the MATLAB Production Server Dashboard. Role-based access control allows administrators to grant access to specific areas of the dashboard to certain users or groups of users. For more information about the roles that the dashboard supports, see “Dashboard Access Control” on page 9-51.

To enable dashboard access control, configure Azure AD and specify access control policies, in consultation with the Azure AD administrator.

Configure Identity Provider

To configure Azure AD:

1. Log in to the dashboard to retrieve the redirect URI of the dashboard.
2. In the Azure portal, use the redirect URI to register the dashboard as a client application with the provider.
3. In the dashboard, enter values specific to the registered application and Azure AD.

Retrieve Redirect URI from Dashboard

To retrieve the redirect URI, start creating a configuration for Azure AD in the dashboard:

1. Navigate to either the **Dashboard Access Control** tab or the **Manage Identity Providers** tab.
2. Click **Create** and select **Azure AD**.
3. In **Create Identity Provider for Dashboard Access Control**, note the redirect URI of the dashboard.

   Later, you return to this view to specify the values required to configure your identify provider in the dashboard.

Register Application in Azure Portal

Use the Azure portal to register a web client application for dashboard access control. When registering the application, use the redirect URI from the MATLAB Production Server dashboard. Typically, the Azure AD administrator registers the application.

1. Sign in to the Azure portal.
2. From **Azure Active Directory**, select **App registrations** and click **New registration**.
3. In the resulting pane, enter the name of the application (for example, `MATLAB Production Server Dashboard App`).
4. For the **Redirect URI**, select **Web**. In the corresponding value field, enter the redirect URI of the dashboard and click **Register**. A web page displays the details of your registered application.
5. Click **Manifest** in the left navigation pane. In the JSON that is displayed in the resulting pane, set the value for `groupMembershipClaims` to `"SecurityGroup"`. Click **Save**.

For more information on how to register an application, see the Microsoft Azure documentation.

Specify Values in Dashboard

In the Azure portal, find the values of the client application that you registered and enter them into the dashboard.

1. Sign in to the Azure portal.
2. From **Azure Active Directory**, select **App registrations** and then select the application that you registered for the dashboard. Copy the value from the **Application (client) ID** and paste it into the **Client ID** field in the dashboard.
3. From **App registrations**, select **Certificates & secrets**. Under **Certificates & secrets**, create a new client secret or use an existing one. Copy the value for the client secret and paste it into the **Client Secret** field in the dashboard.
4. From **Azure Active Directory**, select **Properties**. Copy the value from **Directory (tenant) ID** and paste it into the **Tenant ID** in the dashboard.
5. On the dashboard, click **Create**.

Specify Dashboard Access Control Policy

Before you can specify dashboard access control policies, you must have users and groups set up in Azure AD. Consult the Azure AD administrator for this setup.

The access control policies define areas of the dashboard that users or groups of users can access and tasks that they can perform in these areas. Use the policies to assign the manager and application author roles to users or groups of users in your organization by entering their Azure user names and group IDs into the dashboard.

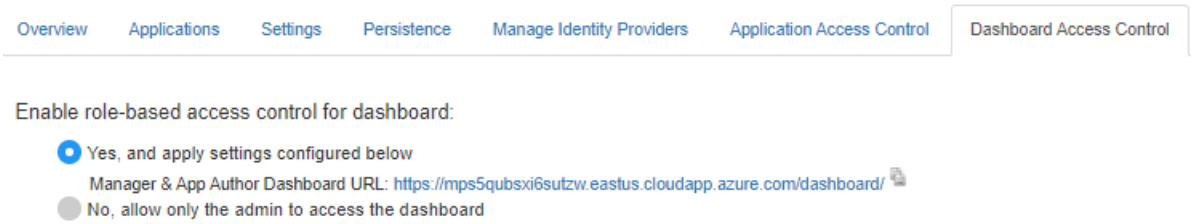
Configure Users and Groups in Dashboard

In the Azure portal, find user names and group IDs and enter them into the dashboard.

1. Sign in to the Azure portal.
2. From **Azure Active Directory** and then **Users**. Copy the values for the user names and paste them into the **Users** field in the dashboard. Use a comma to separate multiple user names.
3. From **Azure Active Directory** and then **Groups**. Copy the values for the object IDs and paste them into the **Groups** field in the dashboard. Use a comma to separate multiple object IDs names.
4. On the dashboard, click **Save**.

Enable Dashboard Access Control

After you configure Azure AD and specify access control policies, you must enable dashboard access control by selecting the **Yes** option. After enabling dashboard access control, a dashboard login URL that supports single sign-on (SSO) becomes available. Share this URL with managers and application authors.



See Also

Related Examples

- “Dashboard Access Control” on page 9-53
- “Configure Dashboard Access Control Using Google Identity” on page 9-90
- “Configure Dashboard Access Control Using PingFederate Identity Provider” on page 9-92
- “Configure Dashboard Access Control Using Other OpenID Connect Providers” on page 9-95

Configure Dashboard Access Control Using Google Identity

MATLAB Production Server administrators can use Google identity provider to configure role-based access control for the MATLAB Production Server Dashboard. Role-based access control allows administrators to grant access to specific areas of the dashboard to certain users or groups of users. For more information about the roles that the dashboard supports, see “Dashboard Access Control” on page 9-51.

To enable dashboard access control, configure the identity provider and specify access control policies, in consultation with the Google Identity administrator.

Configure Google Identity

To configure Google Identity:

1. Log in to the dashboard to retrieve the redirect URI of the dashboard.
2. On the Google Cloud Platform Console, use the redirect URI to register the dashboard as a client application with the provider.
3. In the dashboard, enter values specific to the registered application and Google Identity.

Retrieve Redirect URI from Dashboard

To retrieve the redirect URI, start creating a configuration for Google Identity in the dashboard:

1. Navigate to either the **Dashboard Access Control** tab or the **Manage Identity Providers** tab.
2. Click **Create** and select **Google**.
3. In **Create Identity Provider for Dashboard Access Control**, note the redirect URI of the dashboard.

   Later, you return to this view to specify the values required to configure your identify provider in the dashboard.

Register Application in Google Cloud Console

Use the Google Cloud Console to register a web client application for dashboard access control. Use the redirect URI from the MATLAB Production Server Dashboard when registering the application.

1. Sign in to the Google Cloud Platform Console and navigate to the **Credentials** page.
2. On the **Credentials** page, click **Create credentials** and select **OAuth client ID**.
3. From **Application type** drop down, select **Web Application**.
4. Enter the name of you client application (for example, `MATLAB Production Server Dashboard App`).
5. Under **Authorized redirect URIs**, click **Add URI**.
6. Copy the redirect URI from MATLAB Production Server Dashboard and paste it into the **URIs** field in the Google Cloud Console.
7. Click **Create**.
8. The Google identity provider creates an application with a client ID and client secret. Note the values of the client ID and client secret. You enter these values next in the dashboard.

Specify Client ID and Client Secret in Dashboard

1. Enter the noted client ID and client secret values from the previous section in the **Client ID** and **Client Secret** fields respectively in MATLAB Production Server dashboard.
2. Click **Create** to complete the configuration of the identity provider.

Specify Dashboard Access Control Policy

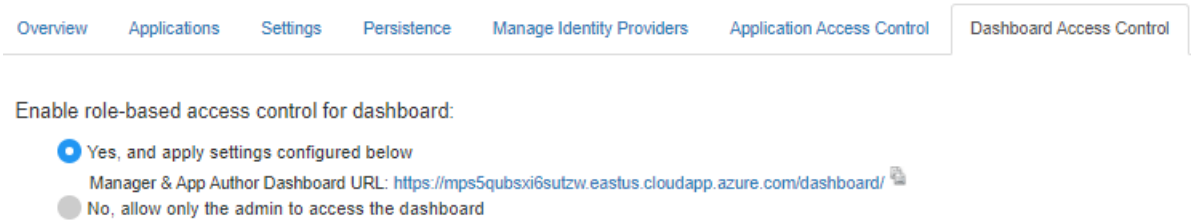
Before you can specify dashboard access control policies, you must have users, and groups, if applicable, set up in Google. Consult the Google identity provider administrator for this setup.

The access control policies define areas of the dashboard that users can access and tasks that they can perform in these areas. Use the policies to assign the manager and application author roles to users in your organization by entering their Google user names.

1. On the **Dashboard Access Control** tab of the dashboard, select Google as the identity provider.
2. In the **Dashboard Access Control Policy** section, enter Google user names to assign manager and application author roles to users in your organization. Click **Save** after you enter the values.

Enable Dashboard Access Control

After you configure Google Identity and specify access control policies, you must enable dashboard access control by selecting the **Yes** option. After enabling dashboard access control, a dashboard login URL that supports single sign-on (SSO) becomes available. Share this URL with managers and application authors.



See Also

Related Examples

- “Dashboard Access Control” on page 9-53
- “Configure Dashboard Access Control Using Azure AD” on page 9-87
- “Configure Dashboard Access Control Using PingFederate Identity Provider” on page 9-92
- “Configure Dashboard Access Control Using Other OpenID Connect Providers” on page 9-95

Configure Dashboard Access Control Using PingFederate Identity Provider

MATLAB Production Server administrators can use PingFederate from Ping Identity to configure role-based access control for the MATLAB Production Server Dashboard. Role-based access control allows administrators to grant access to specific areas of the dashboard to certain users or groups of users. For more information about the roles that the dashboard supports, see “Dashboard Access Control” on page 9-51.

To enable dashboard access control for MATLAB Production Server, configure PingFederate and specify access control policies, in consultation with the PingFederate administrator.

Prerequisites

Refer to the PingFederate documentation to configure OAuth use cases, clients, and endpoints to configure OpenID provider information:

- Configuring OAuth Use Cases
- Configuring OAuth Clients
- OAuth 2.0 Endpoints
- Configuring OpenID Provider Information

Configure PingFederate Identity Provider

To configure PingFederate:

1. Log in to the dashboard to retrieve the Redirect URI of the dashboard.
2. Use the Redirect URI to register a client application in PingFederate.
3. In the dashboard, enter values specific to the registered application and PingFederate.

Retrieve Redirect URI from Dashboard

To retrieve the redirect URI, start creating a configuration for your identity provider in the dashboard:

1. Navigate to either the **Dashboard Access Control** tab or the **Manage Identity Providers** tab.
2. Click **Create** and select **PingFederate**.
3. In **Create Identity Provider for Dashboard Access Control**, note the redirect URI of the dashboard.

   Later, you return to this view to specify the values required to configure your identify provider in the dashboard.

Register Application in PingFederate

Register an application in PingFederate for MATLAB Production Server Dashboard, if you do not already have one. Consult the PingFederate administrator to register the application. Provide the **Redirect URI** of the MATLAB Production Server Dashboard when registering the application.

Specify Values in Dashboard

After you register the application with PingFederate, you receive application specific values such as the **Client ID** and **Client Secret**. Enter the values specific to the application and values specific to PingFederate in the dashboard under **Create Identity Provider for Dashboard Access Control**.

The following table describes the values that you must enter. Click **Create** after you enter the values.

Field	Description
Client ID	Application ID of the registered client application.
Client Secret	Client secret of the registered client application.
OIDC Issuer	Discovery endpoint URI of the OIDC provider.
JWT Issuer	JWT issuer metadata of the OIDC provider.
JWKS URI	URI to retrieve the JSON Web Key Set (JWKS).

Specify Dashboard Access Control Policy

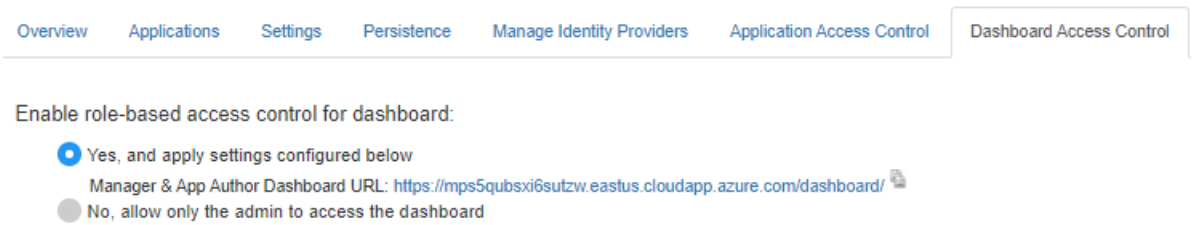
Before you can specify dashboard access control policies, you must have users, and groups, if applicable, set up in PingFederate. Consult the PingFederate administrator for this setup.

The access control policies define areas of the dashboard that users or groups of users can access and tasks that they can perform in these areas. Use the policies to assign the manager and application author roles to users or groups of users in your organization by entering their user names and group IDs. Click **Save** after you enter the values.

1. In the **Dashboard Access Control** tab of the dashboard, select PingFederate as the identity provider.
2. In the **Dashboard Access Control Policy** section, enter identity provider specific user names and group IDs to assign manager and application author roles to users or groups of users in your organization. Use a comma to separate multiple user names and group IDs. Click **Save** after you enter the values.

Enable Dashboard Access Control

After you configure PingFederate and specify access control policies, you must enable dashboard access control by selecting the **Yes** option. After enabling dashboard access control, a dashboard login URL that supports single sign-on (SSO) becomes available. Share this URL with managers and application authors.



See Also

Related Examples

- “Dashboard Access Control” on page 9-53
- “Configure Dashboard Access Control Using Azure AD” on page 9-87
- “Configure Dashboard Access Control Using Google Identity” on page 9-90
- “Configure Dashboard Access Control Using Other OpenID Connect Providers” on page 9-95

Configure Dashboard Access Control Using Other OpenID Connect Providers

MATLAB Production Server Dashboard supports the use of any OpenID Connect (OIDC) identity provider for role-based access control for the dashboard. Role-based access control allows server administrators to grant access to specific areas of the dashboard to certain users or groups of users. For more information about the roles that the dashboard supports, see “Dashboard Access Control” on page 9-51.

To enable dashboard access control, configure the OIDC provider and specify dashboard access control policies, in consultation with the OIDC provider administrator.

Configure Identity Provider

To configure an identity provider:

1. Log in to the dashboard to retrieve the redirect URI of the dashboard.
2. At the identity provider's website, use the redirect URI to register the dashboard as a client application with the provider.
3. In the dashboard, enter values specific to the registered application and identity provider.

Retrieve Redirect URI from Dashboard

To retrieve the redirect URI, start creating a configuration for your identity provider in the dashboard:

1. Navigate to either the **Dashboard Access Control** tab or the **Manage Identity Providers** tab.
2. Click **Create** and select **Other**.
3. In **Create Identity Provider for Dashboard Access Control**, note the redirect URI of the dashboard.

   Later, you return to this view to specify the values required to configure your identify provider in the dashboard.

Register Application with Identity Provider

Register an application with the OIDC provider for MATLAB Production Server Dashboard. Consult the OIDC provider administrator to register the application. When registering the application, provide the redirect URI of the MATLAB Production Server Dashboard.

Specify Values in Dashboard

After you register the application with the identity provider, you receive application-specific values such as the client ID and client secret. Enter the values in the dashboard under **Create Identity Provider for Dashboard Access Control**.

The following table describes the values that you must enter. Click **Create** after you enter the values.

Field	Description
Client ID	Application ID of the registered client application

Field	Description
Client Secret	Client secret of the registered client application
OIDC Issuer	Discovery endpoint URI of the OIDC provider
JWT Issuer	JWT issuer metadata of the OIDC provider
JWKS URI	URI to retrieve the JSON Web Key Set (JWKS)

Under **Create Identity Provider for Dashboard Access Control**, you have the option to provide values other than the defaults for **UserAttribute ID** and **GroupAttribute ID**. **UserAttribute ID** is the JWT claim name that uniquely identifies a user. **GroupAttribute ID** is the JWT claim name that lists the groups that a user belongs to. Depending on the identity provider you use, you might have to change the defaults.

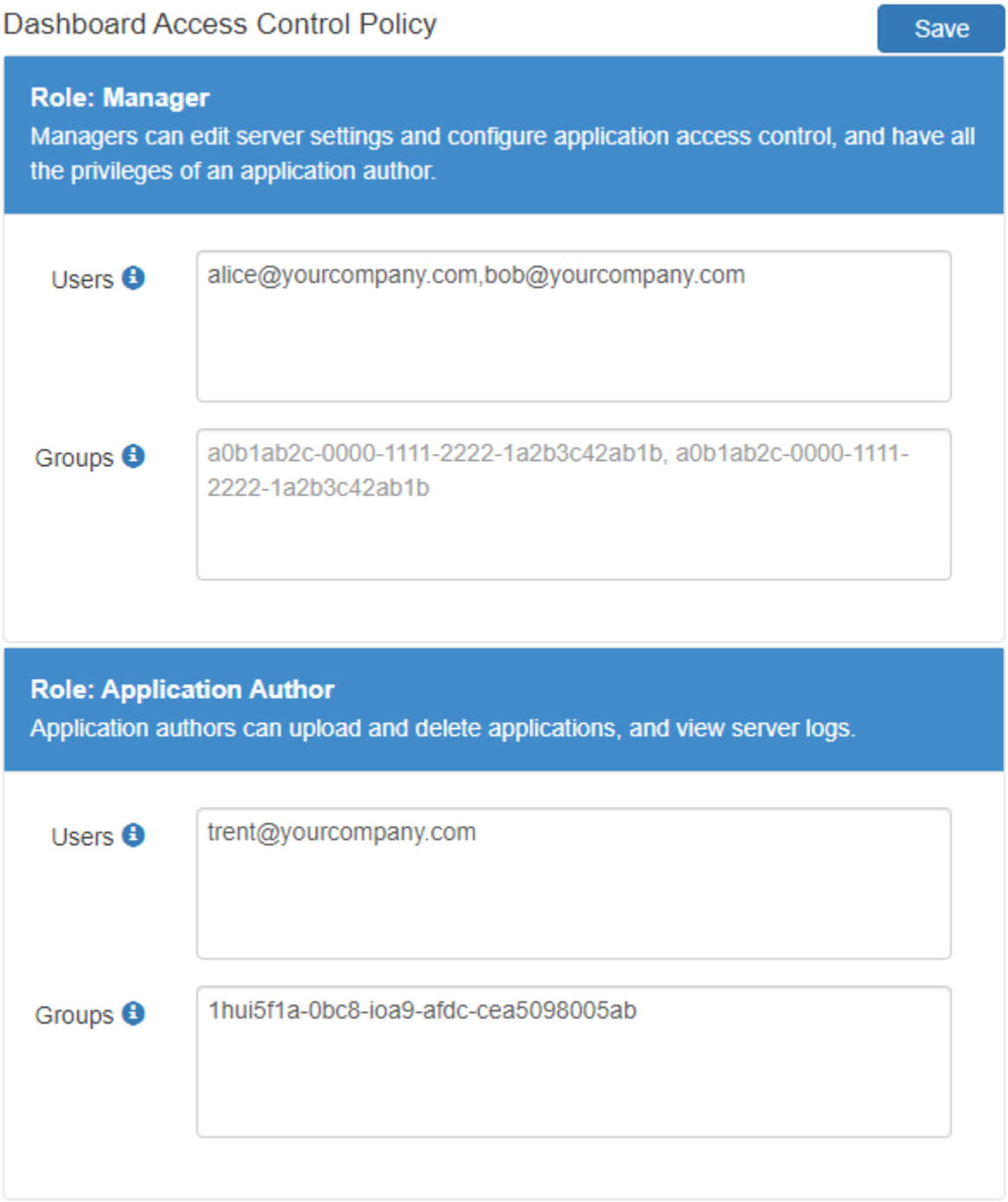
Specify Dashboard Access Control Policy

Before you can specify dashboard access control policies, you must have users, and if applicable, groups, set up in the identity provider. Consult the OIDC provider administrator for this setup.

The access control policies define areas of the dashboard that users or groups of users can access and tasks that they can perform in these areas.

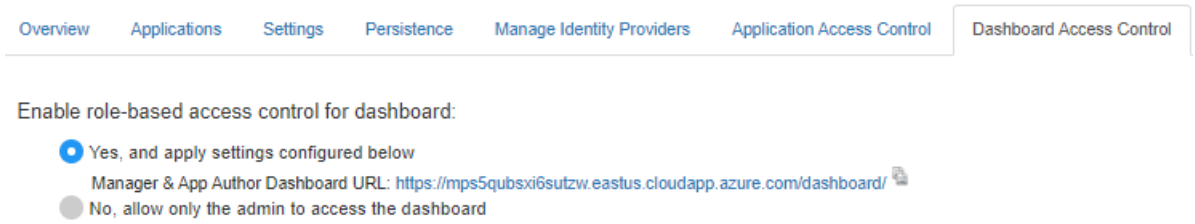
1. On the **Dashboard Access Control** tab of the dashboard, select the identity provider that you want to use.
2. In the **Dashboard Access Control Policy** section, enter identity provider specific user names and group IDs to assign manager and application author roles to users or groups of users in your organization. Use a comma to separate multiple user names or group IDs. Click **Save** after you enter the values.

   For example, in the following illustration, the users *alice@yourcompany.com* and *bob@yourcompany.com* have the manager role. The user *trent@yourcompany.com* and all users that belong to group ID *1hui5f1a-0bc8-ioa9-afdc-cea5098005ab* have the application author role.



Enable Dashboard Access Control

After you configure the identity provider and specify access control policies, you must enable dashboard access control by selecting the **Yes** option. After enabling dashboard access control, a dashboard login URL that supports single sign-on (SSO) becomes available. Share this URL with managers and application authors.



See Also

Related Examples

- “Dashboard Access Control” on page 9-53
- “Configure Dashboard Access Control Using Azure AD” on page 9-87
- “Configure Dashboard Access Control Using Google Identity” on page 9-90
- “Configure Dashboard Access Control Using PingFederate Identity Provider” on page 9-92

Configure Application Access Control Using Azure AD

MATLAB Production Server administrators can use Microsoft Azure AD to restrict access to deployed applications to only certain users or groups of users. To enable application access control, configure Azure AD and specify access control policies, in consultation with the Azure AD administrator.

Register Application in Azure Portal

To use Azure AD for application access control, register a server application and a client application in the Azure portal. These applications are different from the application that you might have registered for dashboard access control. These applications are not related to the applications deployed to MATLAB Production Server or client applications written using the MATLAB Production Server client libraries.

Note The application registration process is determined by Azure and is subject to change.

Register Server Application in Azure

1. Sign in to the Azure portal.
2. From **Azure Active Directory**, select **App registrations** and click **New registration**.
3. In the resulting pane, enter the name of the application (for example, `MATLAB Production Server App`) then select **Register**.
4. In the application that you registered, select **Expose an API**.
5. Click **Add a scope**, and enter the scope information for your application. Click **Add Scope**. For more information on adding a scope, see the Microsoft Azure documentation. The following table lists the fields and values that you enter to add a scope.

Field	Value
Scope name	Enter a name, for example, `user_impersonation`.
Who can consent	Select `Admin and users`.
Admin consent display name	Enter a name, for example, `Access MATLAB Production Server App`.
Admin consent description	Enter a description, for example, `Allow the application to access MATLAB Production Server App on behalf of the signed-in user`.
User consent display name	Enter a name, for example, `Access MATLAB Production Server App`.
User consent description	Enter a description, for example, `Allow the application to access MATLAB Production Server App on behalf of the signed-in user`.
State	Select `Enabled`.

6. Click **Manifest** in the left navigation pane. In the JSON that is displayed, set the value for `groupMembershipClaims` to `"SecurityGroup"`. Click **Save**.

Register Client Application in Azure

In the Azure portal, register a client application. The client application helps clients that send requests to the server to generate an access token. You can register the client application as either a native app or a web app. If you register the client application as a native app, users have to log in using a user name and password to generate the access token. If you register the client application as a web app, users have to log in using the browser with single sign-on to generate the access token.

Registering client applications can require higher privileges in Azure based on your organization setup.

Register Client Application as Native Client

1. Sign in to the Azure portal.
2. From **Azure Active Directory**, select **App registrations** and click **New registration**.
3. In the pane that opens, enter the following registration information for your application, then click **Register**.

Field	Value
Name	Enter a name, for example, `MATLAB Production Server Native Client`.
Redirect URI	Select `Public client/native (mobile & desktop)`.

4. Click **Manifest** in the left navigation pane. In the JSON, set the value for `allowPublicClient` to `true`. Click **Save**.
5. Click **API permissions** and click **Add a permission**.
6. In the pane that opens, click **APIs my organization uses**.
7. Search for the `MATLAB Production Server App` server application that you registered earlier. In the pane that opens, select the scope name (for example, `user_impersonation`) and click **Add permissions.**

Register Client Application as Web Client

1. Sign in to the Azure portal.
2. From **Azure Active Directory**, select **App registrations** and click **New registration**.
3. In the pane that opens, enter the following registration information for your application, then click **Register**.

Field	Value
Name	Enter a name, for example, `MATLAB Production Server Web Client`.
Redirect URI	Select `Web`. Enter a valid redirect URI that will be used by your client application

4. Select **Certificates & secrets** in the left navigation pane. Under **Client secrets**, create a new client secret, and save the value of the secret.
5. Click **API permissions**, then click **Add a permission** and select **APIs my organization uses**.
6. Search for the `MATLAB Production Server App` server application that you registered earlier. In the pane that opens, select the scope name, for example, `user_impersonation`, then click **Add permissions.**

Configure Identity Provider

After you register the server application and client application in the Azure portal, create a configuration for Azure AD in the **Application Access Control** tab of the dashboard. Click **Create** and select **Azure AD**.

In the Azure portal, find the tenant ID for your organization, and the application ID for the server application that you registered earlier. Enter the tenant ID and application ID in the dashboard under **Create Identity Provider for Application Access Control**.

1. Sign in to the Azure portal.
2. From **Azure Active Directory**, select **Properties**. Copy the value from **Directory (tenant) ID** and paste it into **Tenant ID** field in the dashboard.
3. From **Azure Active Directory**, select **App registrations**. Select the application used for MATLAB Production Server, for example, `MATLAB Production Server App`. Copy the value from **Application (client) ID** and paste it into the **Server App ID** field in the dashboard.
4. In the dashboard, click **Create**. If the server is running on a Windows virtual machine, saving the values can take up to 30 seconds.

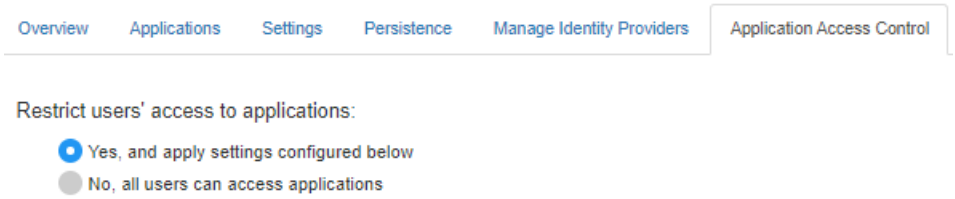
Specify Access Control Policy Rules

Specify the applications that certain user groups can access by defining access control policy rules. To define the rule, click **Add Rule** under **Access Control Policy** in the **Application Access Control** tab of the dashboard. Specify the following values.

Field	Value
Rule ID	Name for the rule
Description	Description for your rule
Users	User names set up in Azure AD that are allowed access to deployed applications
Groups	Object IDs of the groups set up in Azure AD groups that are allowed access to deployed applications
Applications	Applications that the specified users and groups can access. To select all applications, select `Apply this rule to all applications`.

Enable Application Access Control

After you configure the identity provider and specify access control policy rules, you must enable dashboard access control by selecting the **Yes** option from the dashboard.



Generate Access Token

After application access control is enabled, users that are specified in the access control policy rules can generate a bearer access token. If the registered client application is a native app, log in using a user name and password, or integrated Windows authentication to generate the access token. If the registered client application is a web app, log in using the browser with single sign-on to generate the access token. You can use the Microsoft identity platform authentication libraries (Microsoft-supported client libraries or compatible client libraries in different programming languages) to generate the access token. For more information, see Microsoft documentation. Use this access token in the HTTP authorization header when you make a request to the server using the MATLAB Production Server RESTful API. The format for this header is `Authorization:Bearer <access token>`.

See Also

Related Examples

- “Application Access Control” on page 9-45
- “Configure Application Access Control Using Google Identity” on page 9-103
- “Configure Application Access Control Using PingFederate” on page 9-105
- “Configure Application Access Control Using Other Identity Providers” on page 9-107

Configure Application Access Control Using Google Identity

MATLAB Production Server administrators can use Google Identity to restrict access to deployed applications to only certain users or groups of users. To enable application access control, configure Google Identity and specify access control policies, in consultation with the Google Identity administrator.

Register Application in Google Cloud Platform Console

To use Google Identity for application access control, register an application in the Google Cloud Platform Console. For more information about registering an application, see Google Identity documentation.

Configure Identity Provider in Dashboard

After you register the application in Google, create a configuration for Google Identity in the **Application Access Control** tab of the dashboard. Click **Create** and select **Google**. In **Create Identity Provider for Application Access Control**, enter application-specific and identity provider-specific values. Click **Create**. If the server is running on a Windows virtual machine, saving the values can take up to 30 seconds.

The following table describes the values that you must enter.

Field	Value
Name	Name for your Google identity provider configuration
App ID	Client ID of the application registered in Google for application access control

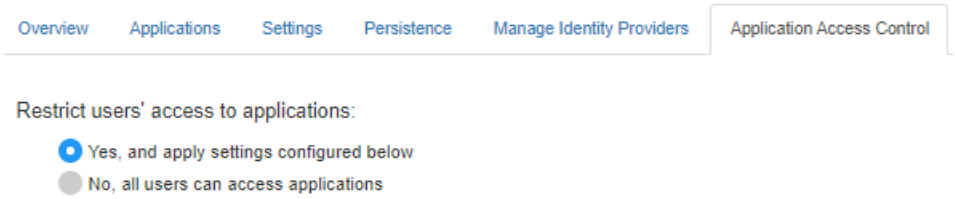
Specify Access Control Policy Rules

Specify the applications that certain users or user groups can access by defining access control policy rules. To define a rule, click **Add Rule** under **Access Control Policy** in the **Application Access Control** tab of the dashboard. Specify the following values.

Field	Value
Rule ID	Name for the rule
Description	Description for the rule
Users	Google user names that are allowed access to deployed applications
Groups	Google group IDs, if applicable, that are allowed access to deployed applications
Applications	Applications that the specified users and groups have permission to access Select `Apply this rule to all applications` to select all applications.

Enable Application Access Control

After you configure the identity provider and specify access control policy rules, you must enable dashboard access control by selecting the **Yes** option from the dashboard.



Generate Access Token

After application access control is enabled, users that are specified in the access control policy rules can generate a bearer access token. For more information about generating an access token, see the Google documentation for Using OAuth 2.0 for Web Server Applications.

Client programs use this access token in the HTTP authorization header when making a request to the server using the MATLAB Production Server RESTful API. The format for this header is `Authorization:Bearer <access token>`.

See Also

Related Examples

- “Application Access Control” on page 9-45
- “Configure Application Access Control Using Azure AD” on page 9-99
- “Configure Application Access Control Using PingFederate” on page 9-105
- “Configure Application Access Control Using Other Identity Providers” on page 9-107

Configure Application Access Control Using PingFederate

MATLAB Production Server administrators can use PingFederate from Ping Identity to restrict access to deployed applications to only certain users or groups of users. To enable application access control, configure PingFederate and specify access control policy rules, in consultation with the PingFederate administrator.

Prerequisites

Refer to the PingFederate documentation to configure OAuth use cases, clients and endpoints, and to configure OpenID provider information:

- Configuring OAuth Use Cases
- Configuring OAuth Clients
- OAuth 2.0 Endpoints
- Configuring OpenID Provider Information

Configure PingFederate in Dashboard

1. After you register the application with PingFederate, create a configuration for PingFederate in the **Application Access Control** tab of the dashboard. Click **Create** and select **PingFederate**.
2. In **Create Identity Provider for Application Access Control**, enter application-specific and identity provider-specific values. Click **Create**. If the server is running on a Windows virtual machine, saving the values can take up to 30 seconds.

   The following table describes the values that you must enter.

Field	Value
Name	Name for your PingFederate configuration.
App ID	Intended recipient of the JWT. The recipient helps in validating the *aud* claim in the JWT.
JWT Issuer	JWT issuer metadata of the identity provider. The metadata string must match the *iss* claim in the JWT.
JWKS URI	URI to retrieve the JSON Web Key Set (JWKS).

Specify Access Control Policy Rules

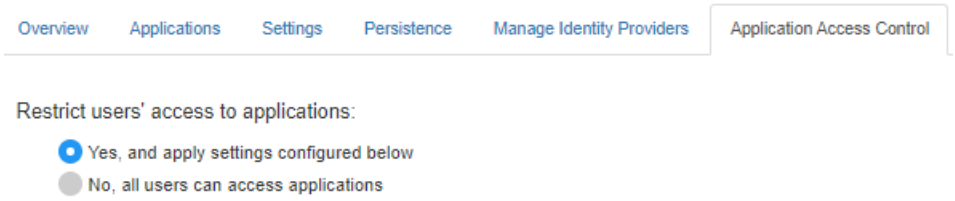
Specify the applications that certain users or user groups can access by defining access control policy rules. To define a rule, click **Add Rule** under **Access Control Policy** in the **Application Access Control** tab of the dashboard. Then, specify the following information.

Field	Value
Rule ID	Name for the rule
Description	Description for your rule

Field	Value
Users	User names that are allowed access to deployed applications
Groups	Group IDs, if applicable, that are allowed access to deployed applications
Applications	Applications that you want to allow the specified groups of users to access. To select all applications, select `Apply this rule to all applications`.

Enable Application Access Control

After you configure the identity provider and specify access control policy rules, you must enable dashboard access control by selecting the **Yes** option from the dashboard.



Generate Access Token

After application access control is enabled, users that are specified in the access control policy rules can generate a bearer access token. For more information about generating an access token, see the PingFederate OAuth 2.0 Developer Guide.

Clients programs use this access token in the HTTP authorization header when making a request to the server using the MATLAB Production Server RESTful API. The format for this header is `Authorization:Bearer <access token>`.

See Also

Related Examples

- “Application Access Control” on page 9-45
- “Configure Application Access Control Using Azure AD” on page 9-99
- “Configure Application Access Control Using Google Identity” on page 9-103
- “Configure Application Access Control Using Other Identity Providers” on page 9-107

Configure Application Access Control Using Other Identity Providers

MATLAB Production Server integrates with several OAuth 2.0 providers for application access control. Application access control lets server administrators restrict access to deployed applications to only certain users or groups of users. To enable application access control, configure an identity provider and specify access control policy rules, in consultation with the OAuth 2.0 provider administrator.

Register Application With Identity Provider

To use an identity provider for application access control, register an application with the identity provider. Consult the identity provider administrator to register the application.

Configure Identity Provider in Dashboard

After you register the application with the identity provider, create a configuration for the identity provider in the **Application Access Control** tab of the dashboard. Click **Create** and select **Other**. In **Create Identity Provider for Application Access Control**, enter application-specific and identity provider-specific values. Click **Create**. If the server is running on a Windows virtual machine, saving the values can take up to 30 seconds.

The following table describes the values that you must enter.

Field	Value
Name	Name for your identity provider.
App ID	Intended recipient of the JWT. The recipient helps in validating the *aud* claim in the JWT.
JWT Issuer	JWT issuer metadata of the identity provider. The metadata string must match the *iss* claim in the JWT.
JWKS URI	URI to retrieve the JSON Web Key Set (JWKS).

Under **Create Identity Provider for Application Access Control**, you have the option to provide values other than the defaults for **UserAttribute ID** and **GroupAttribute ID**. **UserAttribute ID** is the JWT claim name that uniquely identifies a user. **GroupAttribute ID** is the JWT claim name that lists the groups that a user belongs to. Depending on the identity provider you use, you might have to change the defaults.

Specify Access Control Policy Rules

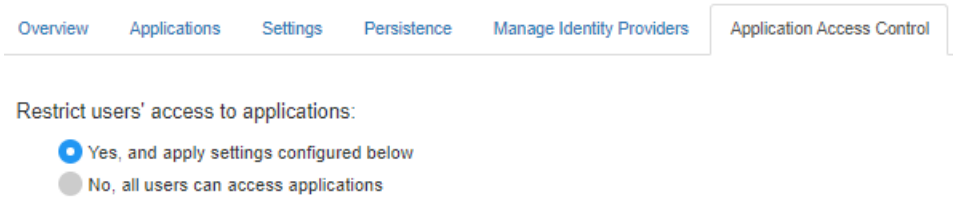
Specify the applications that certain users or user groups can access by defining access control policy rules. To define a rule, click **Add Rule** under **Access Control Policy** in the **Application Access Control** tab of the dashboard. Then, specify the following information.

Field	Value
Rule ID	Name for the rule.

Field	Value
Description	Description for your rule.
Users	User names set up in the identity provider that are allowed access to deployed applications.
Groups	Group IDs set up in the identity provider that are allowed access to deployed applications.
Applications	Applications that the specified users and groups can access. To select all applications, select `Apply this rule to all applications`.

Enable Application Access Control

After you configure the identity provider and specify access control policy rules, you must enable dashboard access control by selecting the **Yes** option from the dashboard.



Generate Access Token

After you enable application access control, clients can generate a bearer token. Client programs can use third-part libraries for token generation. For a list of OAuth libraries, see OAuth libraries. Client programs use this bearer token in the HTTP authorization header when making a request to the server using the MATLAB Production Server RESTful API. The format for this header is `Authorization:Bearer <access token>`.

See Also

Related Examples

- “Application Access Control” on page 9-45
- “Configure Application Access Control Using Azure AD” on page 9-99
- “Configure Application Access Control Using Google Identity” on page 9-103
- “Configure Application Access Control Using PingFederate” on page 9-105

Configure Application Access Control Using Azure AD

MATLAB Production Server administrators can use Microsoft Azure AD to restrict access to deployed applications to only certain users or groups of users. To enable application access control, configure Azure AD and specify access control policies, in consultation with the Azure AD administrator.

Register Application in Azure Portal

To use Azure AD for application access control, register a server application and a client application in the Azure portal. These applications are different from the application that you might have registered for dashboard access control. These applications are not related to the applications deployed to MATLAB Production Server or client applications written using the MATLAB Production Server client libraries.

Note The application registration process is determined by Azure and is subject to change.

Register Server Application in Azure

1. Sign in to the Azure portal.
2. From **Azure Active Directory**, select **App registrations** and click **New registration**.
3. In the resulting pane, enter the name of the application (for example, `MATLAB Production Server App`) then select **Register**.
4. In the application that you registered, select **Expose an API**.
5. Click **Add a scope**, and enter the scope information for your application. Click **Add Scope**. For more information on adding a scope, see the Microsoft Azure documentation. The following table lists the fields and values that you enter to add a scope.

Field	Value
Scope name	Enter a name, for example, `user_impersonation`.
Who can consent	Select `Admin and users`.
Admin consent display name	Enter a name, for example, `Access MATLAB Production Server App`.
Admin consent description	Enter a description, for example, `Allow the application to access MATLAB Production Server App on behalf of the signed-in user`.
User consent display name	Enter a name, for example, `Access MATLAB Production Server App`.
User consent description	Enter a description, for example, `Allow the application to access MATLAB Production Server App on behalf of the signed-in user`.
State	Select `Enabled`.

6. Click **Manifest** in the left navigation pane. In the JSON that is displayed, set the value for `groupMembershipClaims` to `"SecurityGroup"`. Click **Save**.

Register Client Application in Azure

In the Azure portal, register a client application. The client application helps clients that send requests to the server to generate an access token. You can register the client application as either a native app or a web app. If you register the client application as a native app, users have to log in using a user name and password to generate the access token. If you register the client application as a web app, users have to log in using the browser with single sign-on to generate the access token.

Registering client applications can require higher privileges in Azure based on your organization setup.

Register Client Application as Native Client

1. Sign in to the Azure portal.
2. From **Azure Active Directory**, select **App registrations** and click **New registration**.
3. In the pane that opens, enter the following registration information for your application, then click **Register**.

Field	Value
Name	Enter a name, for example, `MATLAB Production Server Native Client`.
Redirect URI	Select `Public client/native (mobile & desktop)`.

4. Click **Manifest** in the left navigation pane. In the JSON, set the value for `allowPublicClient` to `true`. Click **Save**.
5. Click **API permissions** and click **Add a permission**.
6. In the pane that opens, click **APIs my organization uses**.
7. Search for the `MATLAB Production Server App` server application that you registered earlier. In the pane that opens, select the scope name (for example, `user_impersonation`) and click **Add permissions.**

Register Client Application as Web Client

1. Sign in to the Azure portal.
2. From **Azure Active Directory**, select **App registrations** and click **New registration**.
3. In the pane that opens, enter the following registration information for your application, then click **Register**.

Field	Value
Name	Enter a name, for example, `MATLAB Production Server Web Client`.
Redirect URI	Select `Web`. Enter a valid redirect URI that will be used by your client application

4. Select **Certificates & secrets** in the left navigation pane. Under **Client secrets**, create a new client secret, and save the value of the secret.
5. Click **API permissions**, then click **Add a permission** and select **APIs my organization uses**.
6. Search for the `MATLAB Production Server App` server application that you registered earlier. In the pane that opens, select the scope name, for example, `user_impersonation`, then click **Add permissions.**

Configure Identity Provider

After you register the server application and client application in the Azure portal, create a configuration for Azure AD in the **Application Access Control** tab of the dashboard. Click **Create** and select **Azure AD**.

In the Azure portal, find the tenant ID for your organization, and the application ID for the server application that you registered earlier. Enter the tenant ID and application ID in the dashboard under **Create Identity Provider for Application Access Control**.

1. Sign in to the Azure portal.
2. From **Azure Active Directory**, select **Properties**. Copy the value from **Directory (tenant) ID** and paste it into **Tenant ID** field in the dashboard.
3. From **Azure Active Directory**, select **App registrations**. Select the application used for MATLAB Production Server, for example, `MATLAB Production Server App`. Copy the value from **Application (client) ID** and paste it into the **Server App ID** field in the dashboard.
4. In the dashboard, click **Create**. If the server is running on a Windows virtual machine, saving the values can take up to 30 seconds.

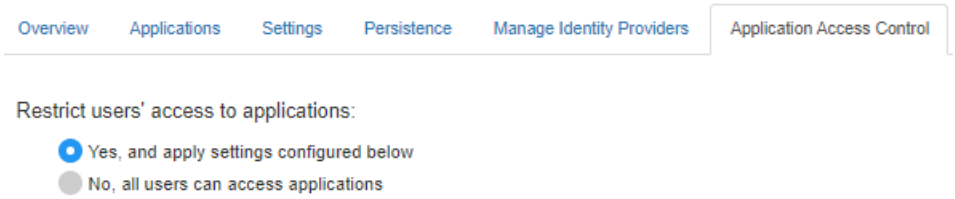
Specify Access Control Policy Rules

Specify the applications that certain user groups can access by defining access control policy rules. To define the rule, click **Add Rule** under **Access Control Policy** in the **Application Access Control** tab of the dashboard. Specify the following values.

Field	Value
Rule ID	Name for the rule
Description	Description for your rule
Users	User names set up in Azure AD that are allowed access to deployed applications
Groups	Object IDs of the groups set up in Azure AD groups that are allowed access to deployed applications
Applications	Applications that the specified users and groups can access. To select all applications, select `Apply this rule to all applications`.

Enable Application Access Control

After you configure the identity provider and specify access control policy rules, you must enable dashboard access control by selecting the **Yes** option from the dashboard.



Generate Access Token

After application access control is enabled, users that are specified in the access control policy rules can generate a bearer access token. If the registered client application is a native app, log in using a user name and password, or integrated Windows authentication to generate the access token. If the registered client application is a web app, log in using the browser with single sign-on to generate the access token. You can use the Microsoft identity platform authentication libraries (Microsoft-supported client libraries or compatible client libraries in different programming languages) to generate the access token. For more information, see Microsoft documentation. Use this access token in the HTTP authorization header when you make a request to the server using the MATLAB Production Server RESTful API. The format for this header is `Authorization:Bearer <access token>`.

See Also

Related Examples

- “Application Access Control” on page 9-49
- “Configure Application Access Control Using Google Identity” on page 9-113
- “Configure Application Access Control Using PingFederate” on page 9-115
- “Configure Application Access Control Using Other Identity Providers” on page 9-117

Configure Application Access Control Using Google Identity

MATLAB Production Server administrators can use Google Identity to restrict access to deployed applications to only certain users or groups of users. To enable application access control, configure Google Identity and specify access control policies, in consultation with the Google Identity administrator.

Register Application in Google Cloud Platform Console

To use Google Identity for application access control, register an application in the Google Cloud Platform Console. For more information about registering an application, see Google Identity documentation.

Configure Identity Provider in Dashboard

After you register the application in Google, create a configuration for Google Identity in the **Application Access Control** tab of the dashboard. Click **Create** and select **Google**. In **Create Identity Provider for Application Access Control**, enter application-specific and identity provider-specific values. Click **Create**. If the server is running on a Windows virtual machine, saving the values can take up to 30 seconds.

The following table describes the values that you must enter.

Field	Value
Name	Name for your Google identity provider configuration
App ID	Client ID of the application registered in Google for application access control

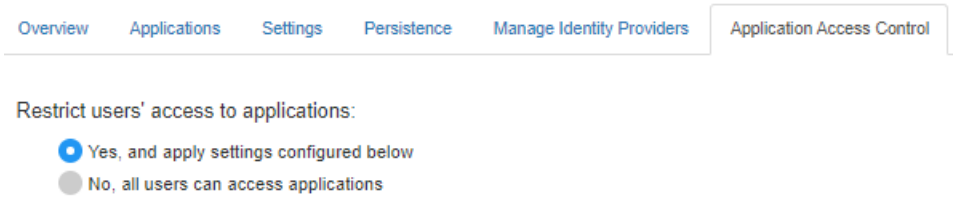
Specify Access Control Policy Rules

Specify the applications that certain users or user groups can access by defining access control policy rules. To define a rule, click **Add Rule** under **Access Control Policy** in the **Application Access Control** tab of the dashboard. Specify the following values.

Field	Value
Rule ID	Name for the rule
Description	Description for the rule
Users	Google user names that are allowed access to deployed applications
Groups	Google group IDs, if applicable, that are allowed access to deployed applications
Applications	Applications that the specified users and groups have permission to access Select `Apply this rule to all applications` to select all applications.

Enable Application Access Control

After you configure the identity provider and specify access control policy rules, you must enable dashboard access control by selecting the **Yes** option from the dashboard.



Generate Access Token

After application access control is enabled, users that are specified in the access control policy rules can generate a bearer access token. For more information about generating an access token, see the Google documentation for Using OAuth 2.0 for Web Server Applications.

Client programs use this access token in the HTTP authorization header when making a request to the server using the MATLAB Production Server RESTful API. The format for this header is `Authorization:Bearer <access token>`.

See Also

Related Examples

- “Application Access Control” on page 9-49
- “Configure Application Access Control Using Azure AD” on page 9-109
- “Configure Application Access Control Using PingFederate” on page 9-115
- “Configure Application Access Control Using Other Identity Providers” on page 9-117

Configure Application Access Control Using PingFederate

MATLAB Production Server administrators can use PingFederate from Ping Identity to restrict access to deployed applications to only certain users or groups of users. To enable application access control, configure PingFederate and specify access control policy rules, in consultation with the PingFederate administrator.

Prerequisites

Refer to the PingFederate documentation to configure OAuth use cases, clients and endpoints, and to configure OpenID provider information:

- Configuring OAuth Use Cases
- Configuring OAuth Clients
- OAuth 2.0 Endpoints
- Configuring OpenID Provider Information

Configure PingFederate in Dashboard

1. After you register the application with PingFederate, create a configuration for PingFederate in the **Application Access Control** tab of the dashboard. Click **Create** and select **PingFederate**.
2. In **Create Identity Provider for Application Access Control**, enter application-specific and identity provider-specific values. Click **Create**. If the server is running on a Windows virtual machine, saving the values can take up to 30 seconds.

   The following table describes the values that you must enter.

Field	Value
Name	Name for your PingFederate configuration.
App ID	Intended recipient of the JWT. The recipient helps in validating the *aud* claim in the JWT.
JWT Issuer	JWT issuer metadata of the identity provider. The metadata string must match the *iss* claim in the JWT.
JWKS URI	URI to retrieve the JSON Web Key Set (JWKS).

Specify Access Control Policy Rules

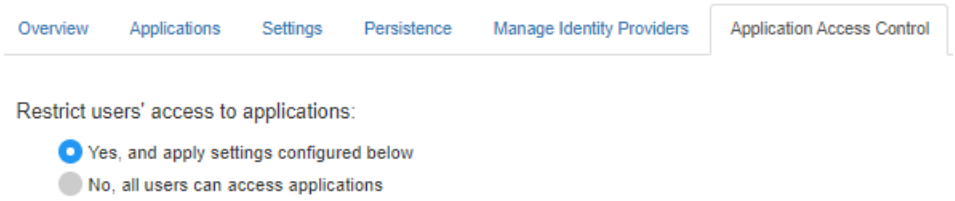
Specify the applications that certain users or user groups can access by defining access control policy rules. To define a rule, click **Add Rule** under **Access Control Policy** in the **Application Access Control** tab of the dashboard. Then, specify the following information.

Field	Value
Rule ID	Name for the rule
Description	Description for your rule

Field	Value
Users	User names that are allowed access to deployed applications
Groups	Group IDs, if applicable, that are allowed access to deployed applications
Applications	Applications that you want to allow the specified groups of users to access. To select all applications, select `Apply this rule to all applications`.

Enable Application Access Control

After you configure the identity provider and specify access control policy rules, you must enable dashboard access control by selecting the **Yes** option from the dashboard.



Generate Access Token

After application access control is enabled, users that are specified in the access control policy rules can generate a bearer access token. For more information about generating an access token, see the PingFederate OAuth 2.0 Developer Guide.

Clients programs use this access token in the HTTP authorization header when making a request to the server using the MATLAB Production Server RESTful API. The format for this header is `Authorization:Bearer <access token>`.

See Also

Related Examples

- “Application Access Control” on page 9-49
- “Configure Application Access Control Using Azure AD” on page 9-109
- “Configure Application Access Control Using Google Identity” on page 9-103
- “Configure Application Access Control Using Other Identity Providers” on page 9-117

Configure Application Access Control Using Other Identity Providers

MATLAB Production Server integrates with several OAuth 2.0 providers for application access control. Application access control lets server administrators restrict access to deployed applications to only certain users or groups of users. To enable application access control, configure an identity provider and specify access control policy rules, in consultation with the OAuth 2.0 provider administrator.

Register Application With Identity Provider

To use an identity provider for application access control, register an application with the identity provider. Consult the identity provider administrator to register the application.

Configure Identity Provider in Dashboard

After you register the application with the identity provider, create a configuration for the identity provider in the **Application Access Control** tab of the dashboard. Click **Create** and select **Other**. In **Create Identity Provider for Application Access Control**, enter application-specific and identity provider-specific values. Click **Create**. If the server is running on a Windows virtual machine, saving the values can take up to 30 seconds.

The following table describes the values that you must enter.

Field	Value
Name	Name for your identity provider.
App ID	Intended recipient of the JWT. The recipient helps in validating the *aud* claim in the JWT.
JWT Issuer	JWT issuer metadata of the identity provider. The metadata string must match the *iss* claim in the JWT.
JWKS URI	URI to retrieve the JSON Web Key Set (JWKS).

Under **Create Identity Provider for Application Access Control**, you have the option to provide values other than the defaults for **UserAttribute ID** and **GroupAttribute ID**. **UserAttribute ID** is the JWT claim name that uniquely identifies a user. **GroupAttribute ID** is the JWT claim name that lists the groups that a user belongs to. Depending on the identity provider you use, you might have to change the defaults.

Specify Access Control Policy Rules

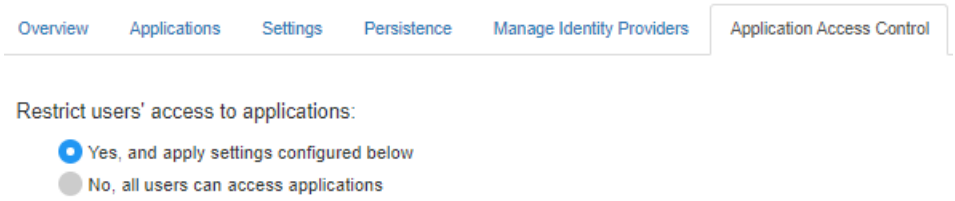
Specify the applications that certain users or user groups can access by defining access control policy rules. To define a rule, click **Add Rule** under **Access Control Policy** in the **Application Access Control** tab of the dashboard. Then, specify the following information.

Field	Value
Rule ID	Name for the rule.

Field	Value
Description	Description for your rule.
Users	User names set up in the identity provider that are allowed access to deployed applications.
Groups	Group IDs set up in the identity provider that are allowed access to deployed applications.
Applications	Applications that the specified users and groups can access. To select all applications, select `Apply this rule to all applications`.

Enable Application Access Control

After you configure the identity provider and specify access control policy rules, you must enable dashboard access control by selecting the **Yes** option from the dashboard.



Generate Access Token

After you enable application access control, clients can generate a bearer token. Client programs can use third-part libraries for token generation. For a list of OAuth libraries, see OAuth libraries. Client programs use this bearer token in the HTTP authorization header when making a request to the server using the MATLAB Production Server RESTful API. The format for this header is `Authorization:Bearer <access token>`.

See Also

Related Examples

- “Application Access Control” on page 9-49
- “Configure Application Access Control Using Azure AD” on page 9-99
- “Configure Application Access Control Using Google Identity” on page 9-103
- “Configure Application Access Control Using PingFederate” on page 9-105

Configure Application Access Control Using Azure AD

MATLAB Production Server administrators can use Microsoft Azure AD to restrict access to deployed applications to only certain users or groups of users. To enable application access control, configure Azure AD and specify access control policies, in consultation with the Azure AD administrator.

Register Application in Azure Portal

To use Azure AD for application access control, register a server application and a client application in the Azure portal. These applications are different from the application that you might have registered for dashboard access control. These applications are not related to the applications deployed to MATLAB Production Server or client applications written using the MATLAB Production Server client libraries.

Note The application registration process is determined by Azure and is subject to change.

Register Server Application in Azure

1. Sign in to the Azure portal.
2. From **Azure Active Directory**, select **App registrations** and click **New registration**.
3. In the resulting pane, enter the name of the application (for example, `MATLAB Production Server App`) then select **Register**.
4. In the application that you registered, select **Expose an API**.
5. Click **Add a scope**, and enter the scope information for your application. Click **Add Scope**. For more information on adding a scope, see the Microsoft Azure documentation. The following table lists the fields and values that you enter to add a scope.

Field	Value
Scope name	Enter a name, for example, `user_impersonation`.
Who can consent	Select `Admin and users`.
Admin consent display name	Enter a name, for example, `Access MATLAB Production Server App`.
Admin consent description	Enter a description, for example, `Allow the application to access MATLAB Production Server App on behalf of the signed-in user`.
User consent display name	Enter a name, for example, `Access MATLAB Production Server App`.
User consent description	Enter a description, for example, `Allow the application to access MATLAB Production Server App on behalf of the signed-in user`.
State	Select `Enabled`.

6. Click **Manifest** in the left navigation pane. In the JSON that is displayed, set the value for `groupMembershipClaims` to `"SecurityGroup"`. Click **Save**.

Register Client Application in Azure

In the Azure portal, register a client application. The client application helps clients that send requests to the server to generate an access token. You can register the client application as either a native app or a web app. If you register the client application as a native app, users have to log in using a user name and password to generate the access token. If you register the client application as a web app, users have to log in using the browser with single sign-on to generate the access token.

Registering client applications can require higher privileges in Azure based on your organization setup.

Register Client Application as Native Client

1. Sign in to the Azure portal.
2. From **Azure Active Directory**, select **App registrations** and click **New registration**.
3. In the pane that opens, enter the following registration information for your application, then click **Register**.

Field	Value
Name	Enter a name, for example, `MATLAB Production Server Native Client`.
Redirect URI	Select `Public client/native (mobile & desktop)`.

4. Click **Manifest** in the left navigation pane. In the JSON, set the value for `allowPublicClient` to `true`. Click **Save**.
5. Click **API permissions** and click **Add a permission**.
6. In the pane that opens, click **APIs my organization uses**.
7. Search for the `MATLAB Production Server App` server application that you registered earlier. In the pane that opens, select the scope name (for example, `user_impersonation`) and click **Add permissions.**

Register Client Application as Web Client

1. Sign in to the Azure portal.
2. From **Azure Active Directory**, select **App registrations** and click **New registration**.
3. In the pane that opens, enter the following registration information for your application, then click **Register**.

Field	Value
Name	Enter a name, for example, `MATLAB Production Server Web Client`.
Redirect URI	Select `Web`. Enter a valid redirect URI that will be used by your client application

4. Select **Certificates & secrets** in the left navigation pane. Under **Client secrets**, create a new client secret, and save the value of the secret.
5. Click **API permissions**, then click **Add a permission** and select **APIs my organization uses**.
6. Search for the `MATLAB Production Server App` server application that you registered earlier. In the pane that opens, select the scope name, for example, `user_impersonation`, then click **Add permissions.**

Configure Identity Provider

After you register the server application and client application in the Azure portal, create a configuration for Azure AD in the **Application Access Control** tab of the dashboard. Click **Create** and select **Azure AD**.

In the Azure portal, find the tenant ID for your organization, and the application ID for the server application that you registered earlier. Enter the tenant ID and application ID in the dashboard under **Create Identity Provider for Application Access Control**.

1. Sign in to the Azure portal.
2. From **Azure Active Directory**, select **Properties**. Copy the value from **Directory (tenant) ID** and paste it into **Tenant ID** field in the dashboard.
3. From **Azure Active Directory**, select **App registrations**. Select the application used for MATLAB Production Server, for example, `MATLAB Production Server App`. Copy the value from **Application (client) ID** and paste it into the **Server App ID** field in the dashboard.
4. In the dashboard, click **Create**. If the server is running on a Windows virtual machine, saving the values can take up to 30 seconds.

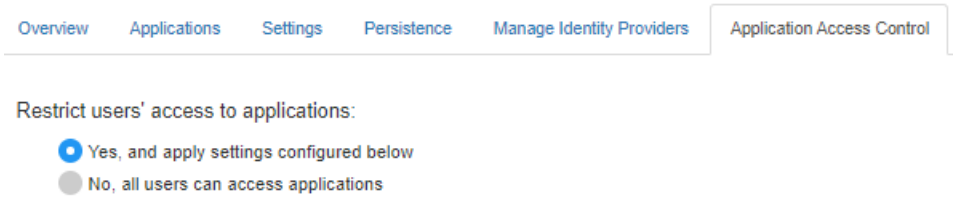
Specify Access Control Policy Rules

Specify the applications that certain user groups can access by defining access control policy rules. To define the rule, click **Add Rule** under **Access Control Policy** in the **Application Access Control** tab of the dashboard. Specify the following values.

Field	Value
Rule ID	Name for the rule
Description	Description for your rule
Users	User names set up in Azure AD that are allowed access to deployed applications
Groups	Object IDs of the groups set up in Azure AD groups that are allowed access to deployed applications
Applications	Applications that the specified users and groups can access. To select all applications, select `Apply this rule to all applications`.

Enable Application Access Control

After you configure the identity provider and specify access control policy rules, you must enable dashboard access control by selecting the **Yes** option from the dashboard.



Generate Access Token

After application access control is enabled, users that are specified in the access control policy rules can generate a bearer access token. If the registered client application is a native app, log in using a user name and password, or integrated Windows authentication to generate the access token. If the registered client application is a web app, log in using the browser with single sign-on to generate the access token. You can use the Microsoft identity platform authentication libraries (Microsoft-supported client libraries or compatible client libraries in different programming languages) to generate the access token. For more information, see Microsoft documentation. Use this access token in the HTTP authorization header when you make a request to the server using the MATLAB Production Server RESTful API. The format for this header is `Authorization:Bearer <access token>`.

See Also

Related Examples

- “Application Access Control” on page 9-47
- “Configure Application Access Control Using Google Identity” on page 9-103
- “Configure Application Access Control Using PingFederate” on page 9-105
- “Configure Application Access Control Using Other Identity Providers” on page 9-107

Configure Application Access Control Using Google Identity

MATLAB Production Server administrators can use Google Identity to restrict access to deployed applications to only certain users or groups of users. To enable application access control, configure Google Identity and specify access control policies, in consultation with the Google Identity administrator.

Register Application in Google Cloud Platform Console

To use Google Identity for application access control, register an application in the Google Cloud Platform Console. For more information about registering an application, see Google Identity documentation.

Configure Identity Provider in Dashboard

After you register the application in Google, create a configuration for Google Identity in the **Application Access Control** tab of the dashboard. Click **Create** and select **Google**. In **Create Identity Provider for Application Access Control**, enter application-specific and identity provider-specific values. Click **Create**. If the server is running on a Windows virtual machine, saving the values can take up to 30 seconds.

The following table describes the values that you must enter.

Field	Value
Name	Name for your Google identity provider configuration
App ID	Client ID of the application registered in Google for application access control

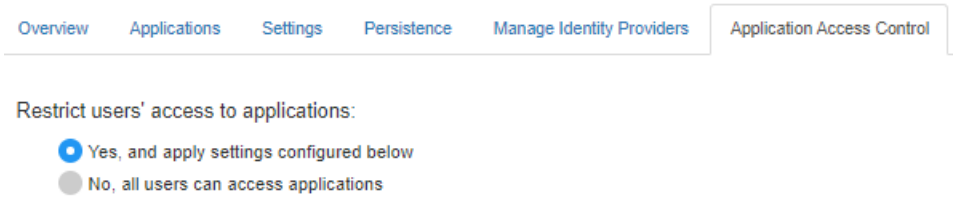
Specify Access Control Policy Rules

Specify the applications that certain users or user groups can access by defining access control policy rules. To define a rule, click **Add Rule** under **Access Control Policy** in the **Application Access Control** tab of the dashboard. Specify the following values.

Field	Value
Rule ID	Name for the rule
Description	Description for the rule
Users	Google user names that are allowed access to deployed applications
Groups	Google group IDs, if applicable, that are allowed access to deployed applications
Applications	Applications that the specified users and groups have permission to access Select `Apply this rule to all applications` to select all applications.

Enable Application Access Control

After you configure the identity provider and specify access control policy rules, you must enable dashboard access control by selecting the **Yes** option from the dashboard.



Generate Access Token

After application access control is enabled, users that are specified in the access control policy rules can generate a bearer access token. For more information about generating an access token, see the Google documentation for Using OAuth 2.0 for Web Server Applications.

Client programs use this access token in the HTTP authorization header when making a request to the server using the MATLAB Production Server RESTful API. The format for this header is `Authorization:Bearer <access token>`.

See Also

Related Examples

- “Application Access Control” on page 9-47
- “Configure Application Access Control Using Azure AD” on page 9-99
- “Configure Application Access Control Using PingFederate” on page 9-105
- “Configure Application Access Control Using Other Identity Providers” on page 9-107

Configure Application Access Control Using PingFederate

MATLAB Production Server administrators can use PingFederate from Ping Identity to restrict access to deployed applications to only certain users or groups of users. To enable application access control, configure PingFederate and specify access control policy rules, in consultation with the PingFederate administrator.

Prerequisites

Refer to the PingFederate documentation to configure OAuth use cases, clients and endpoints, and to configure OpenID provider information:

- Configuring OAuth Use Cases
- Configuring OAuth Clients
- OAuth 2.0 Endpoints
- Configuring OpenID Provider Information

Configure PingFederate in Dashboard

1. After you register the application with PingFederate, create a configuration for PingFederate in the **Application Access Control** tab of the dashboard. Click **Create** and select **PingFederate**.
2. In **Create Identity Provider for Application Access Control**, enter application-specific and identity provider-specific values. Click **Create**. If the server is running on a Windows virtual machine, saving the values can take up to 30 seconds.

   The following table describes the values that you must enter.

Field	Value
Name	Name for your PingFederate configuration.
App ID	Intended recipient of the JWT. The recipient helps in validating the *aud* claim in the JWT.
JWT Issuer	JWT issuer metadata of the identity provider. The metadata string must match the *iss* claim in the JWT.
JWKS URI	URI to retrieve the JSON Web Key Set (JWKS).

Specify Access Control Policy Rules

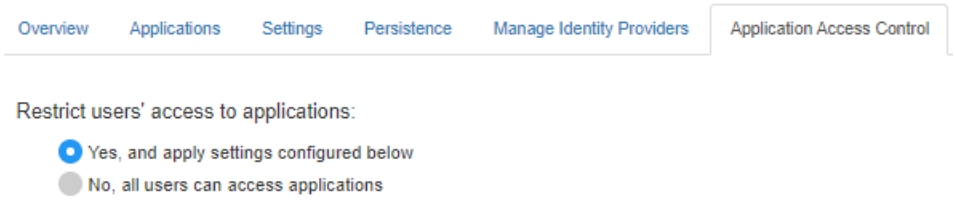
Specify the applications that certain users or user groups can access by defining access control policy rules. To define a rule, click **Add Rule** under **Access Control Policy** in the **Application Access Control** tab of the dashboard. Then, specify the following information.

Field	Value
Rule ID	Name for the rule
Description	Description for your rule

Field	Value
Users	User names that are allowed access to deployed applications
Groups	Group IDs, if applicable, that are allowed access to deployed applications
Applications	Applications that you want to allow the specified groups of users to access. To select all applications, select `Apply this rule to all applications`.

Enable Application Access Control

After you configure the identity provider and specify access control policy rules, you must enable dashboard access control by selecting the **Yes** option from the dashboard.



Generate Access Token

After application access control is enabled, users that are specified in the access control policy rules can generate a bearer access token. For more information about generating an access token, see the PingFederate OAuth 2.0 Developer Guide.

Clients programs use this access token in the HTTP authorization header when making a request to the server using the MATLAB Production Server RESTful API. The format for this header is `Authorization:Bearer <access token>`.

See Also

Related Examples

- “Application Access Control” on page 9-47
- “Configure Application Access Control Using Azure AD” on page 9-99
- “Configure Application Access Control Using Google Identity” on page 9-103
- “Configure Application Access Control Using Other Identity Providers” on page 9-107

Configure Application Access Control Using Other Identity Providers

MATLAB Production Server integrates with several OAuth 2.0 providers for application access control. Application access control lets server administrators restrict access to deployed applications to only certain users or groups of users. To enable application access control, configure an identity provider and specify access control policy rules, in consultation with the OAuth 2.0 provider administrator.

Register Application With Identity Provider

To use an identity provider for application access control, register an application with the identity provider. Consult the identity provider administrator to register the application.

Configure Identity Provider in Dashboard

After you register the application with the identity provider, create a configuration for the identity provider in the **Application Access Control** tab of the dashboard. Click **Create** and select **Other**. In **Create Identity Provider for Application Access Control**, enter application-specific and identity provider-specific values. Click **Create**. If the server is running on a Windows virtual machine, saving the values can take up to 30 seconds.

The following table describes the values that you must enter.

Field	Value
Name	Name for your identity provider.
App ID	Intended recipient of the JWT. The recipient helps in validating the *aud* claim in the JWT.
JWT Issuer	JWT issuer metadata of the identity provider. The metadata string must match the *iss* claim in the JWT.
JWKS URI	URI to retrieve the JSON Web Key Set (JWKS).

Under **Create Identity Provider for Application Access Control**, you have the option to provide values other than the defaults for **UserAttribute ID** and **GroupAttribute ID**. **UserAttribute ID** is the JWT claim name that uniquely identifies a user. **GroupAttribute ID** is the JWT claim name that lists the groups that a user belongs to. Depending on the identity provider you use, you might have to change the defaults.

Specify Access Control Policy Rules

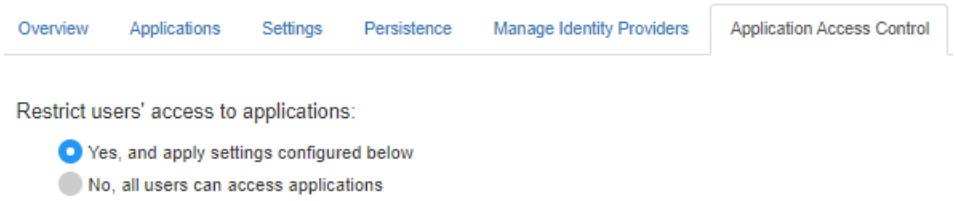
Specify the applications that certain users or user groups can access by defining access control policy rules. To define a rule, click **Add Rule** under **Access Control Policy** in the **Application Access Control** tab of the dashboard. Then, specify the following information.

Field	Value
Rule ID	Name for the rule.

Field	Value
Description	Description for your rule.
Users	User names set up in the identity provider that are allowed access to deployed applications.
Groups	Group IDs set up in the identity provider that are allowed access to deployed applications.
Applications	Applications that the specified users and groups can access. To select all applications, select `Apply this rule to all applications`.

Enable Application Access Control

After you configure the identity provider and specify access control policy rules, you must enable dashboard access control by selecting the **Yes** option from the dashboard.



Generate Access Token

After you enable application access control, clients can generate a bearer token. Client programs can use third-part libraries for token generation. For a list of OAuth libraries, see OAuth libraries. Client programs use this bearer token in the HTTP authorization header when making a request to the server using the MATLAB Production Server RESTful API. The format for this header is `Authorization:Bearer <access token>`.

See Also

Related Examples

- “Application Access Control” on page 9-47
- “Configure Application Access Control Using Azure AD” on page 9-99
- “Configure Application Access Control Using Google Identity” on page 9-103
- “Configure Application Access Control Using PingFederate” on page 9-105

Run MATLAB Production Server on Azure Using Reference Architecture

Deploy MATLAB Production Server on Microsoft Azure using a customizable reference architecture. Use this reference architecture when you want to launch MATLAB Production Server in a specific region, combine MATLAB Production Server with your existing cloud resources, or automate deployment.

Requirements

- A MATLAB Production Server license that meets the following conditions:
  - Linked to a MathWorks Account.
  - Concurrent license type. To check your license type, view your MathWorks Account.
  - Configured to use a network license manager on the virtual network. By default, the deployment of MATLAB Production Server includes a network license manager, but you can also use an existing license manager. In either case, activate or move the license after deployment. For details, see “Configure MATLAB Production Server License for Use on the Cloud”.
- An Azure account. You are responsible for the costs of all Azure services.

Run from GitHub

To launch MATLAB Production Server in Azure, use the reference architecture templates provided in the following GitHub repository:

- MATLAB Production Server on Microsoft Azure

You can run the template directly from the link in the GitHub repository. Choose your MATLAB Production Server release, and then click the **Deploy to Azure** icon to deploy your resources.

See Also

Related Examples

- “Run MATLAB Production Server on Amazon Web Services Using Reference Architecture” on page 10-48

Run MATLAB Parallel Server and MATLAB Production Server on Azure

Running MATLAB Production Server and MATLAB Parallel Server™ on Azure lets you deploy MATLAB functions that use parallel language commands to MATLAB Production Server on the cloud. To run MATLAB Production Server and MATLAB Parallel Server on Azure, you can deploy them using reference architectures. The deployments must use the Network License Manager for MATLAB (License Manager) to manage license files.

In the following instructions, you deploy MATLAB Production Server and the License Manager first, followed by MATLAB Parallel Server. You can choose a different sequence for your deployments. The sequence in which you deploy the products does not matter, as long as you deploy MATLAB Production Server, MATLAB Parallel Server, and the License Manager in the same virtual network.

Deploy MATLAB Production Server

Launch the MATLAB Production Server deployment template from GitHub. For details, see Launch Template. For information on providing values for the parameters in the template, see Configure Cloud Services (GitHub). If you use a value for the **Subnet1** parameter that is different than the default, record the value. You will need it later when you deploy MATLAB Parallel Server.

Deploy MATLAB Parallel Server

Launch the MATLAB Parallel Server deployment template from GitHub. For details, see Launch Template. Select a release, then choose the **Use Existing Virtual Network** template to deploy MATLAB Parallel Server in the same virtual network as MATLAB Production Server. For information on providing values for the parameters in the template, see Configure Cloud Resources (GitHub).

The values for the following parameters depend on the parameters in the MATLAB Production Server deployment:

- **Resource Group** — Enter the name of the resource group in which MATLAB Production Server and the License Manager are deployed.
- **Virtual Network Resource ID** — Enter the resource ID of the existing virtual network in which MATLAB Production Server and the License Manager are deployed. The default name for the virtual network in which MATLAB Production Server and the License Manager are deployed is `mps-network`.
- **Subnet** — Enter the name of **Subnet1** from the MATLAB Production Server deployment.
- **License Server** — Enter the port number and private IP address of the license server in the format `port@hostname`. The default port number is 27000. The hostname for the license server is the private IP address of the `netlm-server-nic` resource in your resource group.

Upload License File

After you deploy MATLAB Production Server, MATLAB Parallel Server, and License Manager, log in to the License Manager dashboard to upload the license file. For information on accessing the License Manager dashboard, see “Upload License File” on page 9-7.

Use a text editor to combine the MATLAB Production Server and MATLAB Parallel Server license files and upload them through the License Manager dashboard. If you require assistance, contact MathWorks Technical Support.

Connect to Cluster from MATLAB

After you upload the license file, you can connect to your cluster on Azure from the MATLAB desktop. For more information, see Connect to Your Cluster from MATLAB (GitHub). After setting the cloud cluster as default, the next time you run a parallel language command such as `parfor`, `spmd`, `parfeval`, or `batch`, MATLAB connects to the cluster.

Package MATLAB Application into Deployable Archive

You can package MATLAB functions that use parallel language commands into deployable archives (CTF files) and deploy them to MATLAB Production Server. For information on creating deployable archives, see “Create Deployable Archive for MATLAB Production Server”. For information on deploying the archives to MATLAB Production Server, see “Upload MATLAB Application” on page 9-27.

See Also

Related Examples

- “Use Parallel Computing Resources in Deployable Archives”

External Websites

- MATLAB Production Server (BYOL) on Azure Marketplace
- MATLAB Parallel Server (BYOL) on Azure Marketplace

AWS Deployment

- “AWS Deployment for MATLAB Production Server (PAYG)” on page 10-2
- “Manage MATLAB Production Server (PAYG)” on page 10-9
- “Manage AWS Resources for MATLAB Production Server (PAYG)” on page 10-14
- “Architecture and Resources” on page 10-17
- “Application Access Control” on page 10-20
- “Configure Application Access Control Using Azure AD” on page 10-22
- “Configure Application Access Control Using Google Identity” on page 10-26
- “Configure Application Access Control Using PingFederate” on page 10-28
- “Configure Application Access Control Using Other Identity Providers” on page 10-30
- “Dashboard Access Control” on page 10-32
- “Configure Dashboard Access Control Using Azure AD” on page 10-34
- “Configure Dashboard Access Control Using Google Identity” on page 10-37
- “Configure Dashboard Access Control Using PingFederate Identity Provider” on page 10-39
- “Configure Dashboard Access Control Using Other OpenID Connect Providers” on page 10-42
- “Execute MATLAB Functions on MATLAB Production Server (PAYG)” on page 10-46
- “Run MATLAB Production Server on Amazon Web Services Using Reference Architecture” on page 10-48
- “Manage MATLAB Production Server Using the Dashboard” on page 10-49
- “Manage AWS Resources for MATLAB Production Server” on page 10-54
- “Dashboard Access Control” on page 10-57
- “Configure Dashboard Access Control Using Azure AD” on page 10-59
- “Configure Dashboard Access Control Using Google Identity” on page 10-62
- “Configure Dashboard Access Control Using PingFederate Identity Provider” on page 10-64
- “Configure Dashboard Access Control Using Other OpenID Connect Providers” on page 10-67
- “Application Access Control” on page 10-71
- “Configure Application Access Control Using Azure AD” on page 10-73
- “Configure Application Access Control Using Google Identity” on page 10-77
- “Configure Application Access Control Using PingFederate” on page 10-79
- “Configure Application Access Control Using Other Identity Providers” on page 10-81
- “Architecture and Resources” on page 10-83
- “Execute MATLAB Functions on MATLAB Production Server” on page 10-86

AWS Deployment for MATLAB Production Server (PAYG)

The deployment process for using the MATLAB Production Server pay-as-you-go (PAYG) offering on AWS consists of launching the software from the AWS marketplace, viewing price and license agreements, choosing whether you want to deploy in a new or existing virtual private cloud (VPC), and then launching the CloudFormation template to specify cloud resources for your server deployment. After deployment to AWS is complete, you configure and manage MATLAB Production Server by logging in to the MATLAB Production Server (PAYG) dashboard.

If you already have a MATLAB Production Server license and want to deploy MATLAB Production Server on AWS, you can deploy the reference architecture from GitHub. See MATLAB Production Server Reference Architecture on AWS.

To run an application on MATLAB Production Server, you need to create the application using MATLAB Compiler SDK. For more information, see “Create Deployable Archive for MATLAB Production Server”.

Software Costs

Charges for the use of MATLAB Production Server software are on a per vCPU/per hour basis. The total amount is based on the number of vCPUs across all the Amazon EC2® instances in your auto scaling group. After deploying MATLAB Production Server on AWS, you must change the number of MATLAB Production Server workers from the MATLAB Production Server dashboard to match the total number of vCPUs of the EC2 instances in your auto scaling group.

The default EC2 instance that the deployment uses is `m5.xlarge`, which has 4 vCPUs and runs 4 MATLAB Production Server workers. If you select a different EC2 instance during deployment, you must change the number of MATLAB Production Server workers from the MATLAB Production Server dashboard to match the number of vCPUs in the instance that you selected. For example, if you select `m5.2xlarge`, which has 8 vCPUs, you must change the number of workers in the dashboard to 8 to match the number of vCPUs. The total charge for the MATLAB Production Server software is based on the number of vCPUs across all your VMs. For example, if you select an 8 vCPU `m5.2xlarge` instance and choose to run 4 instances in your auto scaling group, you are charged for 8 vCPUs x 4 instances = 32 vCPUs of usage per hour.

Prepare AWS Account

Before you can start the deployment, you must have the following:

- An AWS account to deploy resources on AWS and configure your server environment.
- A key pair for your AWS account. For more information, see Amazon EC2 Key Pairs.
- An X.509 certificate. For information about creating a certificate, see Create and Sign an X509 Certificate.
- If necessary, request a service limit increase for the Amazon EC2 instance type or VPCs. You might need to do this if you already have existing deployments that use that instance type or you think you might exceed the default limit. To request a service limit increase, see AWS Support. For more information about service quotas, see Amazon EC2 Service Quotas.

You are responsible for the cost of the AWS services and resources that the deployment uses.

Subscribe to MATLAB Production Server (PAYG) Software

Go to the MATLAB Production Server (PAYG) offering page on AWS marketplace. To subscribe, click **Continue to Subscribe**.

Doing so takes you to a page where you can view the terms and pricing details. Click `Continue to Configuration` to continue the software configuration process.

Deploy to Existing or New VPC

You can deploy MATLAB Production Server to an existing VPC if you already have an infrastructure on AWS. Otherwise, you can deploy to a new VPC.

1. Select the delivery method — if you want to deploy a new VPC or use an existing VPC for your MATLAB Production Server deployment.
2. Select the MATLAB Production Server version to deploy.
3. Select the region where you want to deploy.
4. Click `Continue to Launch` to review your configuration.

Launch CloudFormation Console

To launch the CloudFormation console, click `Launch`. From the console, you create the resources stack for the deployment and specify stack details for server.

Create Stack

Do not change the default selections on this page. The default selections use the CloudFormation template for deploying MATLAB Production Server (PAYG). Click `Next`.

Specify Stack Details

Configure cloud resources for your server environment in this step.

Stack name

First, enter a name for your stack that will hold the AWS resources that you provision, for example, `Boston`.

Server

Next, you configure the Amazon EC2 instances and data persistence. Each MATLAB Production Server instance runs on an EC2 instance and each MATLAB Production Server instance runs multiple MATLAB Production Server workers. To deploy a server instance, you must specify parameters for the EC2 instance or virtual machine (VM), such as the size and number of VMs. All the VMs in the deployment use Linux.

Parameter Name	Value
Number of Server VMs	Specify the number of VMs to run MATLAB Production Server instances, for example, `6`. The deployment template sets the default to 2 VMs for load balancing. You can change the number of VMs after the initial deployment. For more information, see “Change Number of Virtual Machines” on page 10-14.
Server VM Type	Choose the Amazon EC2 instance type to use for the server instances, for example, `m5.xlarge`. These Amazon EC2 instance types are supported: • `m5.xlarge`, `m5.2xlarge`, `m5.4xlarge`, `m5.8xlarge`, `m5.12xlarge`, `m5.16xlarge`, `m5.24xlarge`, `m5.metal` • `c5.xlarge`, `c5.2xlarge`, `c5.4xlarge`, `c5.8xlarge`, `c5.12xlarge`, `c5.16xlarge`, `c5.24xlarge`, `c5.metal` • `r5.xlarge`, `r5.2xlarge`, `r5.4xlarge`, `r5.8xlarge`, `r5.12xlarge`, `r5.16xlarge`, `r5.24xlarge`, `r5.metal` For more information, see Amazon EC2 Instance Types.
Create Elasticache® for Redis	Choose whether you want to create an Elasticache for Redis. Creating this service allows you to use the persistence functionality of the server. Persistence provides a mechanism to cache data between calls to MATLAB code running on a server instance. For more information, see “Data Caching Basics”. You can provision a Redis cache after the initial deployment. For information on connecting to the deployed Redis cache, see .

Dashboard Login

After you deploy the server VMs, you can manage the server using the MATLAB Production Server dashboard. The dashboard provides a web-based interface to configure and manage MATLAB Production Server in the cloud. Specify the login credentials for the dashboard.

Parameter Name	Value
Username for Dashboard	Specify the administrator user name to log in to the MATLAB Production Server dashboard.
Password for Dashboard	Specify the administrator password to log in to the MATLAB Production Server dashboard.
Confirm Password for Dashboard	Reenter the administrator password to log in to the MATLAB Production Server dashboard.

Network

You can specify which key pair can access the VMs in the deployment, which IP addresses can access the dashboard, whether your solution should use a public IP address, and enable HTTPS communication for the server.

Parameter Name	Value
Name of Existing Key Pair	Choose the name of an existing Amazon EC2 key pair to allow access to all the VMs in the stack, for example: `boston-keypair`. For information about creating an Amazon EC2 key pair, see Amazon EC2 Key Pairs.
Allow Connections from IP Address	Specify the range of IP addresses that is permitted to connect to the dashboard that manages the server, for example, `10.0.0.1/24`. Specify the range of IP addresses in CIDR notation, which provides the IP address before the slash and the subnet mask after the slash. The mask determines the number of IP addresses to include. You can use a CIDR calculator to determine the CIDR notation for a range of IP addresses.
Make Solution Available over Internet	Make your solution available over the Internet by setting this parameter to `Yes`. The solution will use public IP addresses. If you set this parameter to `No`, the CloudFormation template does not assign a public IP address for the VM that hosts the dashboard. To access the dashboard, you can use a different VM located in the same virtual network as the VM that hosts the dashboard.

Parameter Name	Value
ARN of SSL Certificate	Provide the Amazon® Resource Name (ARN) of an SSL certificate to enable secure HTTPS communication using the HTTPS server endpoint, for example, `123456789012`. Find the ARN in the AWS Certificate Manager. For information on creating and uploading a self-signed certificate, see Create and Sign an X509 Certificate and “Import SSL Certificate” on page 10-14.

Existing VPC

If you want to deploy resources for MATLAB Production Server in an existing VPC, you must specify the following parameters, in addition to the parameters specified earlier. If you chose to deploy in a new VPC, the template does not provide you with an option to specify the following parameters.

Parameter Name	Value
Existing VPC ID	Find the ID of your existing VPC in the VPC dashboard and enter it here. **1** Navigate to the VPC dashboard and locate your VPC. **2** Use the value under **VPC ID**.
IP address range of your existing VPC	Find the IP address range of your existing VPC in the VPC dashboard and enter it here. **1** Navigate to the VPC dashboard and select your VPC. **2** Use the value under **IPv4 CIDR**.
Subnet 1 ID	Specify the ID of the existing subnet that will host the dashboard and other resources.
Subnet 2 ID	Specify the ID of the existing subnet that will host the server VMs.

If you use an existing VPC, you must open the following ports. When deploying in a new VPC, the deployment opens these ports by default.

- `443` – Required for communicating with the dashboard and the MATLAB execution endpoint.
- `8000`, `8002` – Required for communication between the dashboard, MATLAB Production Server workers, and various microservices within the virtual network. These ports do not need to be accessible over the Internet.
- `9910` – Required for the elastic load balancer health check to work. This port needs to be accessible over the Internet.
- `22`, `3389` – Required for remote login functionality for troubleshooting and debugging. You can close these ports after the deployment is complete.

Configure Stack Options

Configuring stack options is optional. You can leave all fields blank or enter values based on your requirement. Click `Next` to review your stack details and stack options.

Review

Review or edit your stack details and any stack options that you set. You must select the acknowledgement to create IAM resources. Otherwise, the deployment produces a `Requires capabilities : [CAPABILITY_IAM]` error and fails to create resources.

When you are satisfied with your stack configuration, click `Create Stack` to start the creation of resources in AWS for your server environment. Resource creation can take up to 20 minutes. After resource creation, it can take up to 15 minutes for the resources to be active.

Connect and Log In to Dashboard

After the deployment successfully creates all resources, connect and log in to the MATLAB Production Server dashboard to configure and manage your server.

Note The Internet Explorer web browser is not supported for interacting with the dashboard.

If your solution uses private IP addresses, you can connect to the dashboard from a VM located in the same virtual network as the VM that hosts the dashboard.

1. In the AWS management console, select the stack that you deployed. You can find the stack under the CloudFormation service.
2. In the stack detail pane, expand the **Outputs** section.
3. Look for the key named `MATLABProductionServerDashboardURL` and click the corresponding URL listed under **Value**. This URL is the HTTPS endpoint to the MATLAB Production Server dashboard.
4. Use the administrator user name and password that you specified in the dashboard login step of the deployment process to log in.

Note For password reset requests, contact MathWorks Support.

After you log in, configure role-based access control for the dashboard, which lets you grant users the privileges to perform tasks on the dashboard and server, according to their role. For more information on how to configure role-based access control, see “Dashboard Access Control” on page 10-32.

See Also

More About

- “Architecture and Resources” on page 10-17

- “Manage MATLAB Production Server (PAYG)” on page 10-9

Manage MATLAB Production Server (PAYG)

After you deploy the MATLAB Production Server pay-as-you-go (PAYG) environment in AWS, use the MATLAB Production Server dashboard, which is a web-based interface to upload applications, edit server settings, and configure access control for the dashboard and applications.

Connect to Dashboard

Find the URL to connect to the dashboard in the AWS management console.

Note The Internet Explorer web browser is not supported for interacting with the dashboard.

Complete these steps only after you successfully create your stack. If your solution uses private IP addresses, you can connect to the dashboard from a virtual machine (VM) that belongs to the same virtual network as the VM that hosts the dashboard.

1. In the AWS management console, select the stack that you deployed. You can find the stack under the CloudFormation service.
2. In the stack detail pane, expand the **Outputs** section
3. Look for the key named `MATLABProductionServerDashboardURL` and click the corresponding URL listed under value. This is the HTTPS endpoint to the MATLAB Production Server cloud dashboard.

Log In to Dashboard

There are three roles for logging in to the dashboard depending on your role in your organization — global admin, manager, or application author.

Log In to Dashboard as Global Admin

If you are accessing the dashboard for the first time, or if you have not configured or not enabled dashboard access control, you must log in using the global admin credentials. These are the credentials that you entered during the “Dashboard Login” on page 9-11 step of the deployment process. A global admin has access to all areas of the dashboard. A global admin can log in to server virtual machines (VMs), configure which users or groups of users can access the dashboard and applications, edit server settings, configure remote persistence services, view logs, and upload and delete applications. You cannot change the global admin credentials.

To grant access to only certain dashboard areas for specific users or groups of users, set up dashboard access control after you log in. For more information, see “Dashboard Access Control” on page 9-55.

Log In to Dashboard as Manager or Application Author

If the global admin has configured and enabled dashboard access control, you can log in to the dashboard as a manager or application author depending on your role in your organization. The login process supports single sign-on using Azure AD.

An application author has the privileges to upload and delete applications and view logs. A manager has the privileges to edit server settings, configure access control for applications, and configure

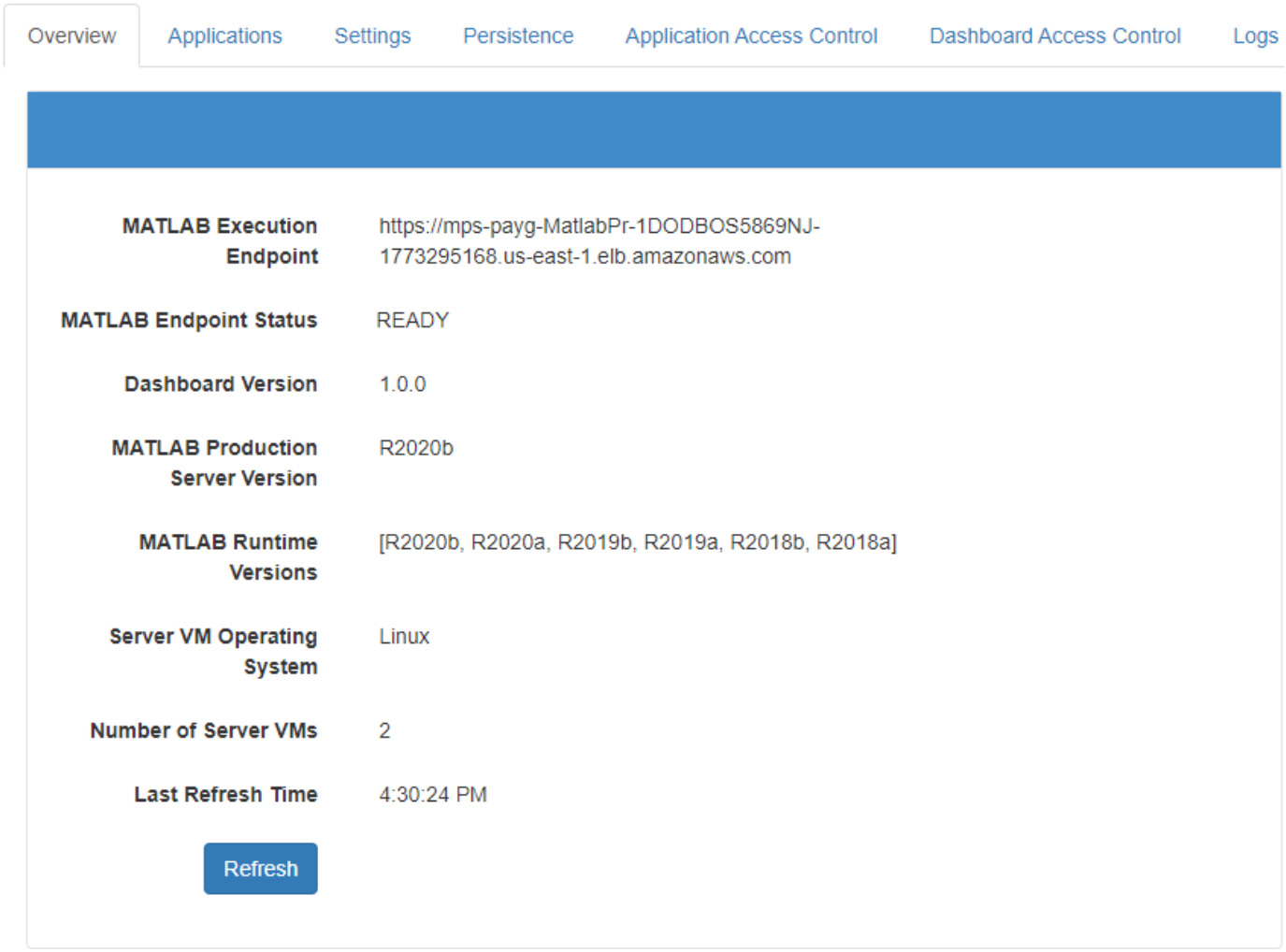
remote persistence services, as well as all the privileges of an application author, including for uploading and deleting applications and viewing logs.

The following table shows the dashboard tabs that users with these roles can access.

Role	Overview	Applications	Settings	Persistence	Manage Identity Providers	Application Access Control	Dashboard Access Control	Logs
Server administrator	✓	✓	✓	✓	✓	✓	✓	✓
Manager	✓	✓	✓	✓	✓	✓		✓
Application author	✓	✓						✓

View Information About Server

To view information about the server instance VMs deployed on AWS, click **Overview** on the dashboard navigation menu.



Upload MATLAB Application

You can run an application on MATLAB Production Server that is created using MATLAB Compiler SDK. Upload and deploy applications using the **Applications** tab in the dashboard.

1. Click **+Upload**.
2. Click **Choose File**, select the file, and click **Deploy**.

For information on how to create an application, see “Create Deployable Archive for MATLAB Production Server”.

View MATLAB Execution Endpoint

The deployment provides an HTTPS endpoint URL to make requests to the server. The **MATLAB Execution Endpoint** in the **Overview** tab in the dashboard specifies the HTTPS endpoint. Use this URL to execute MATLAB functions deployed to the server. For example, if the MATLAB execution endpoint for your server is `https://refarch-mps-1n8k9DNJ8NJAH-1476423457.us-east-1.elb.amazonaws.com`, to use the MATLAB Production Server RESTful API to execute a

MATLAB function `mymagic` located in a deployed application `myapp`, use the URL `https://payg-mps-1n8k9DNJ8NJAH-1476423457.us-east-1.elb.amazonaws.com/myapp/mymagic`.

For information on working with self-signed SSL certificates and managing cookies set by the load balancer, see “Execute MATLAB Functions on MATLAB Production Server (PAYG)” on page 10-46.

Edit Server Configuration

Edit the MATLAB Production Server configuration properties by selecting the **Settings** tab in dashboard. After you update the properties, click **Save** to apply your changes.

Note The server restarts when you click **Save**.

Only the global admin and managers have the privilege to edit the server configuration.

Some examples of server configuration properties follow.

- To allow requests from specific domains, specify parameter values for the **CORS Allowed Origins** property.

  If you want to specify multiple domains, separate them with a comma, for example, `http://www.w3.org, https://www.apache.org`.
- To set the number of MATLAB Production Server workers, specify a parameter value for the **Number of Workers** property.

  When setting the **Number of Workers** property, carefully consider your cluster setup. Each VM in the cluster runs an instance of MATLAB Production Server and each instance runs multiple MATLAB Production Server workers. Using 1 vCPU per MATLAB Production Server worker is recommended. For example, if the server size is set to `m5.xlarge`, which specifies 4 vCPUs, set the number of workers to 4 workers per instance.

Use Amazon ElastiCache for Redis for Data Persistence

MATLAB Production Server uses Redis for data persistence. Persistence allows caching of data between calls to MATLAB code running on the servers. Only the global admin and managers have the privilege to configure remote persistence services.

To view a persistence service that your deployment creates or to create a new remote persistence service, select **Persistence** in the MATLAB Production Server dashboard navigation pane.

Click **+Add** to create a new remote persistence service. Specify the following parameter values, then click **Create**.

Value	Description
Connection Name	Specify a name for the connection to the persistence service. Use this name in MATLAB code to pass data to the cache.

Value	Description
Host and **Port**	Specify the host name and port number for your Redis cache. The port number has to be a non-SSL port. If you are using Amazon ElastiCache® for Redis, use the following procedure to retrieve the host name and port number. 1 Log in to your AWS management console and select the stack associated with the deployment. You can access the stack from the CloudFormation service. 2 In the stack detail pane, expand the **Outputs** section. 3 From the **Outputs** section, copy the value corresponding to the key named `MATLABProductionServerCloudStackCacheClusterAddress` and paste it in the **Host** field in the dashboard. 4 From the **Outputs** section, copy the value corresponding to the key named `MATLABProductionServerCloudStackCacheClusterPort` and paste it in the **Port** field in the dashboard. You can also use a Redis cache that you create outside of this deployment.
Access Key	Leave this field blank.

For more information on using a data cache, see “Data Caching Basics”.

Set Up Access Control for Applications Using Azure Active Directory

MATLAB Production Server uses Azure AD for restricting access to deployed applications to only certain groups of users. Only the global admin and manager can configure access control for applications.

For more information on how to configure application access control using the **Application Access Control** tab, see “Application Access Control” on page 10-20.

Set Up Access Control for Dashboard Using Azure Active Directory

MATLAB Production Server uses Azure AD to provide role-based access control for accessing the dashboard. You can grant access to certain dashboard areas for specific users or groups of users based on the user role. Only the global admin can configure dashboard access control. For more information on how to configure dashboard access control using the **Dashboard Access Control** tab, see “Dashboard Access Control” on page 10-32.

Manage AWS Resources for MATLAB Production Server (PAYG)

After you deploy the MATLAB Production Server pay as you go (PAYG) environment on AWS, use the AWS management console to manage resources that you provision for in the deployment.

Change Number of Virtual Machines

Each MATLAB Production Server instance runs on a virtual machine (VM) and each instance runs multiple MATLAB Production Server workers. You can change the number of VMs after the initial deployment.

1. Log in to the AWS management console and select the CloudFormation stack associated with the deployment.
2. Select the **Outputs** tab from the top pane.
3. Click the link corresponding to **MATLABProductionServerAutoScalingGroup**. Doing so opens a new window that lets you manage auto scaling groups.
4. You can change the number of server VMs in this window. For information on how to do so, see Manual scaling for Amazon EC2 Auto Scaling.

Import SSL Certificate

When you deploy MATLAB Production Server (PAYG), you get two HTTPS endpoint URLs. One endpoint lets you connect to the server instances and the other to the dashboard. To use HTTPS, you must upload an SSL certificate to the load balancer from the AWS certificate manager. For information on creating a self-signed SSL certificate, see Create and sign an X509 certificate.

1. Open the AWS certificate manager.
2. Click **Import a Certificate**.
3. Copy the contents of the CRT file containing the certificate into the field labeled **Certificate body**.
4. Copy the contents of the PEM file containing the private key into the field labeled **Certificate private key**.
5. Leave the filed labeled **Certificate chain** blank and click **Next**.
6. Click **Review and Import**.
7. Review the values and click **Import**.
8. Copy the value of the ARN field from the **Details** section of the certificate and paste it into the **ARN of SSL Certificate** parameter during deployment.

Change SSL Certificate

When you deploy MATLAB Production Server (PAYG), you get two HTTPS endpoint URLs. One endpoint lets you connect to the server instances and the other to the dashboard. You can change the X.509 certificates for these resources.

1. Log in to the AWS management console and select the CloudFormation stack associated with the deployment.
2. Select the **Outputs** tab from the top pane.

3. To change the X.509 certificate for the server instances, click the link corresponding to **MATLABProductionServerLoadBalancer** key.
4. To change the X.509 certificate for the dashboard, click the link corresponding to **MATLABProductionServerDashboardLoadBalancer** key.
5. Select the **Listeners** tab, the click **Change** under **SSL Certificate**.

For more information, see Replace the SSL certificate for your Classic Load Balancer.

View Logs

View MATLAB Production Server logs using AWS CloudWatch .

1. Log in to the AWS management console and select the CloudFormation stack associated with the deployment.
2. Select the **Outputs** tab from the top pane.
3. Click the link corresponding to **MATLABProductionServerWorkerVMLogGroup**. Doing so opens a new CloudWatch window that displays details for the `MATLABProductionServerWorkerVMLogGroup`.
4. Select **View in Logs Insights**. Doing so opens a new Logs Insights console with an existing query. Ignore this default query.

To view logs generated by all the server instances, you can use the following query.

```
fields @timestamp, @message
| filter @logStream like 'prodServerInstance'
| limit 200
```

To view the last 200 logs generated by all server instances, you can use the following query.

```
fields @timestamp, @message
| filter @logStream like 'prodServerInstance'
| limit 200
```

To view only the error logs generated by all server instances, you can use the following query. For more information on log severity, see `log-severity`.

```
fields @timestamp, @message
| filter @logStream like 'prodServerInstance'
| filter severity like 'error'
```

To view logs generated by all the server instances within a certain time duration, you can use the custom range selector.

Handle Timeouts

If the deployed MATLAB function takes more than 120 seconds to finish execution, the client application receives a `504 GATEWAY_TIMEOUT` error. This is because the load balancer configuration closes a connection that is idle for more than 120 seconds. To avoid the time out error, you can increase the load balancer timeout.

1. In the AWS management console, select the stack that you deployed. You can find the stack under the CloudFormation service.

2. In the stack detail pane, expand the **Outputs** section.
3. Look for the key named `MATLABProductionServerLoadBalancer` and click the corresponding URL listed under value to configure the load balancer.
4. Click **Edit idle timeout** under the **Attributes** section.
5. Set the timeout value based on the time that your deployed functions require for execution.

Delete Stack

A stack contains all the related AWS resources for a solution. You can remove the stack and all the associated cluster resources when you no longer need them. You cannot undo this.

1. Log in to the AWS management console and select the stack associated with the deployment. You can find the stack in the CloudFormation service.
2. Click **Delete**.

If you do not want to delete the entire deployment but want to minimize the cost you can bring the number of instances in the Auto Scaling Group down to 0 and then scale it back up when the need arises.

See Also

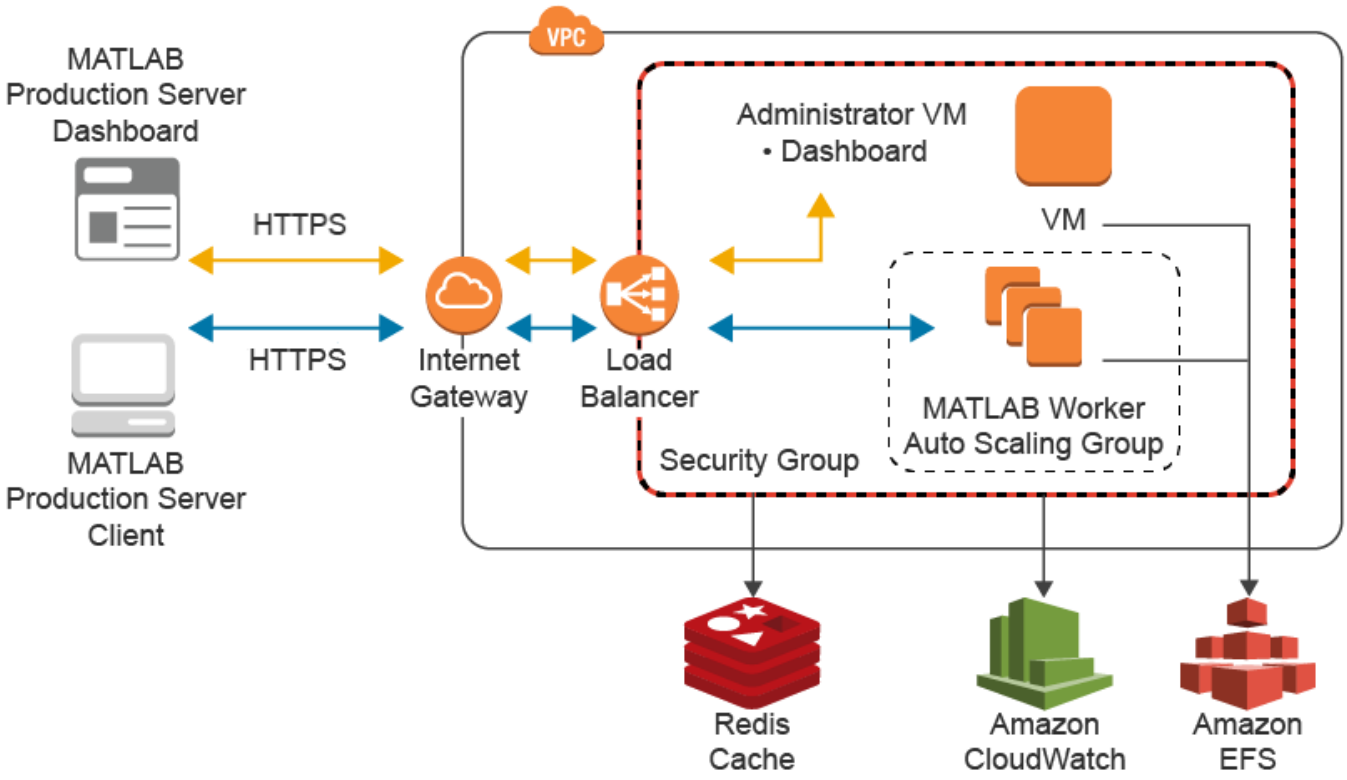
More About

- “Manage MATLAB Production Server (PAYG)” on page 10-9
- “AWS Deployment for MATLAB Production Server (PAYG)” on page 10-2
- “Architecture and Resources” on page 10-17

Architecture and Resources

Deploying MATLAB Production Server pay-as-you-go (PAYG) on AWS creates several resources in your resource group. The following sections describe the architecture of MATLAB Production Server (PAYG) and the AWS resources that the deployment provisions.

MATLAB Production Server (PAYG) Architecture on AWS



AWS Resources

The MATLAB Production Server (PAYG) deployment in AWS creates the following resources in your resource group.

Resource Type	Description
Amazon EC2 Instance	Virtual machine (VM) that hosts the MATLAB Production Server dashboard. Use the dashboard to: • Get the HTTPS endpoint to make requests to the server. • Upload applications (CTF files) to the server. • Manage server configurations. • Configure access control for the dashboard and applications. For more information about the dashboard, see “Manage MATLAB Production Server (PAYG)” on page 10-9.
Auto Scaling Group	Manages the number of identical VMs for hosting MATLAB Production Server instances. Each VM runs an instance of MATLAB Production Server that in turn runs multiple MATLAB Production Server workers. For information on how to change the number of VMs, see “Change Number of Virtual Machines” on page 10-14.
Load balancer	The deployment uses two classic load balancers: **1** Load balancer for routing traffic to MATLAB Production Server instances. Clients use this endpoint for making requests to the server. **2** Load balancer for routing traffic to MATLAB Production Server dashboard. The dashboard retrieves the HTTPS endpoint for making requests to the server from the load balancer resource.
Amazon EFS	Storage account that stores applications (CTF files) created by MATLAB Compiler SDK.
Amazon Virtual Private Cloud™	Virtual network that consists of the deployed resources.

Resource Type	Description
Amazon ElastiCache for Redis	Amazon ElastiCache for Redis enables caching of data between calls to MATLAB code running on a server instance. For information on connecting to the deployed Redis cache, see “Use Amazon ElastiCache for Redis for Data Persistence” on page 10-12. For information on using a data cache, see “Data Caching Basics”.
Amazon CloudWatch	Enables viewing of logs. For information on how to view the logs, see “View Logs” on page 10-15.

See Also

More About

- “Manage AWS Resources for MATLAB Production Server (PAYG)” on page 10-14
- “Manage MATLAB Production Server (PAYG)” on page 10-9
- “AWS Deployment for MATLAB Production Server (PAYG)” on page 10-2

Application Access Control

MATLAB Production Server integrates with OAuth 2.0 providers such as Microsoft Azure Active Directory (Azure AD), Google Identity, and PingFederate from Ping Identity, and uses JSON Web Tokens (JWTs) for application access control. Application access control lets server administrators restrict user access to applications or archives deployed to the server.

Note Application access control is only available for clients that make server requests using the MATLAB Production Server RESTful API.

All users can access all applications by default.

To enable access control, configure the identity provider and define access control policy rules in the **Application Access Control** tab of the MATLAB Production Server dashboard. Use the access control policy rules to specify which users and groups of users have permission to execute deployed applications. After you enable access control, clients can generate a bearer access token that they must send with every server request. The server uses the bearer token to verify the identify of a client.

You must log in to the dashboard as an administrator or manager to configure application access control. For more information about the dashboard user roles, see “Dashboard Access Control” on page 10-32.

Configure Identity Provider and Specify Access Control Policy Rules

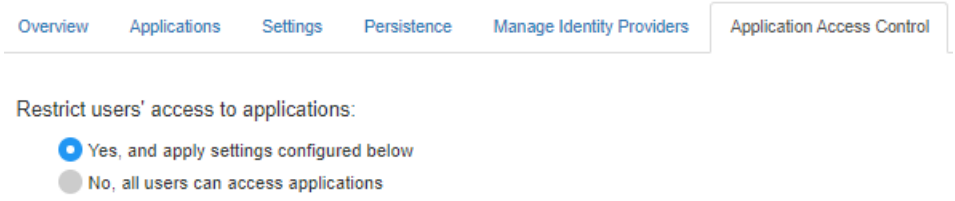
To configure an identity provider, register an application with the identity provider. Then, specify application-specific values and access control policy rules in the dashboard. The fields required to configure an identity provider vary based on the identity provider that you use.

For information about configuring specific identity providers and rules, see:

- “Configure Application Access Control Using Azure AD” on page 10-22
- “Configure Application Access Control Using Google Identity” on page 10-26
- “Configure Application Access Control Using PingFederate” on page 10-28
- “Configure Application Access Control Using Other Identity Providers” on page 10-30

Enable Application Access Control

After you configure the identity provider and specify access control policy rules, you must enable application access control from the dashboard.



Generate Access Token

After you enable application access control, clients can generate a bearer token. Client programs can use third-party libraries for token generation. For a list of OAuth libraries, see OAuth libraries. Client programs use this bearer token in the HTTP authorization header when making a request to the server using the MATLAB Production Server RESTful API. The format for this header is `Authorization:Bearer <access token>`.

See Also

More About

- “Manage MATLAB Production Server (PAYG)” on page 10-9
- “Dashboard Access Control” on page 10-32
- “RESTful API for MATLAB Function Execution”

Configure Application Access Control Using Azure AD

MATLAB Production Server administrators can use Microsoft Azure AD to restrict access to deployed applications to only certain users or groups of users. To enable application access control, configure Azure AD and specify access control policies, in consultation with the Azure AD administrator.

Register Application in Azure Portal

To use Azure AD for application access control, register a server application and a client application in the Azure portal. These applications are different from the application that you might have registered for dashboard access control. These applications are not related to the applications deployed to MATLAB Production Server or client applications written using the MATLAB Production Server client libraries.

Note The application registration process is determined by Azure and is subject to change.

Register Server Application in Azure

1. Sign in to the Azure portal.
2. From **Azure Active Directory**, select **App registrations** and click **New registration**.
3. In the resulting pane, enter the name of the application (for example, `MATLAB Production Server App`) then select **Register**.
4. In the application that you registered, select **Expose an API**.
5. Click **Add a scope**, and enter the scope information for your application. Click **Add Scope**. For more information on adding a scope, see the Microsoft Azure documentation. The following table lists the fields and values that you enter to add a scope.

Field	Value
Scope name	Enter a name, for example, `user_impersonation`.
Who can consent	Select `Admin and users`.
Admin consent display name	Enter a name, for example, `Access MATLAB Production Server App`.
Admin consent description	Enter a description, for example, `Allow the application to access MATLAB Production Server App on behalf of the signed-in user`.
User consent display name	Enter a name, for example, `Access MATLAB Production Server App`.
User consent description	Enter a description, for example, `Allow the application to access MATLAB Production Server App on behalf of the signed-in user`.
State	Select `Enabled`.

6. Click **Manifest** in the left navigation pane. In the JSON that is displayed, set the value for `groupMembershipClaims` to `"SecurityGroup"`. Click **Save**.

Register Client Application in Azure

In the Azure portal, register a client application. The client application helps clients that send requests to the server to generate an access token. You can register the client application as either a native app or a web app. If you register the client application as a native app, users have to log in using a user name and password to generate the access token. If you register the client application as a web app, users have to log in using the browser with single sign-on to generate the access token.

Registering client applications can require higher privileges in Azure based on your organization setup.

Register Client Application as Native Client

1. Sign in to the Azure portal.
2. From **Azure Active Directory**, select **App registrations** and click **New registration**.
3. In the pane that opens, enter the following registration information for your application, then click **Register**.

Field	Value
Name	Enter a name, for example, `MATLAB Production Server Native Client`.
Redirect URI	Select `Public client/native (mobile & desktop)`.

4. Click **Manifest** in the left navigation pane. In the JSON, set the value for `allowPublicClient` to `true`. Click **Save**.
5. Click **API permissions** and click **Add a permission**.
6. In the pane that opens, click **APIs my organization uses**.
7. Search for the `MATLAB Production Server App` server application that you registered earlier. In the pane that opens, select the scope name (for example, `user_impersonation`) and click **Add permissions.**

Register Client Application as Web Client

1. Sign in to the Azure portal.
2. From **Azure Active Directory**, select **App registrations** and click **New registration**.
3. In the pane that opens, enter the following registration information for your application, then click **Register**.

Field	Value
Name	Enter a name, for example, `MATLAB Production Server Web Client`.
Redirect URI	Select `Web`. Enter a valid redirect URI that will be used by your client application

4. Select **Certificates & secrets** in the left navigation pane. Under **Client secrets**, create a new client secret, and save the value of the secret.
5. Click **API permissions**, then click **Add a permission** and select **APIs my organization uses**.
6. Search for the `MATLAB Production Server App` server application that you registered earlier. In the pane that opens, select the scope name, for example, `user_impersonation`, then click **Add permissions.**

Configure Identity Provider

After you register the server application and client application in the Azure portal, create a configuration for Azure AD in the **Application Access Control** tab of the dashboard. Click **Create** and select **Azure AD**.

In the Azure portal, find the tenant ID for your organization, and the application ID for the server application that you registered earlier. Enter the tenant ID and application ID in the dashboard under **Create Identity Provider for Application Access Control**.

1. Sign in to the Azure portal.
2. From **Azure Active Directory**, select **Properties**. Copy the value from **Directory (tenant) ID** and paste it into **Tenant ID** field in the dashboard.
3. From **Azure Active Directory**, select **App registrations**. Select the application used for MATLAB Production Server, for example, `MATLAB Production Server App`. Copy the value from **Application (client) ID** and paste it into the **Server App ID** field in the dashboard.
4. In the dashboard, click **Create**. If the server is running on a Windows virtual machine, saving the values can take up to 30 seconds.

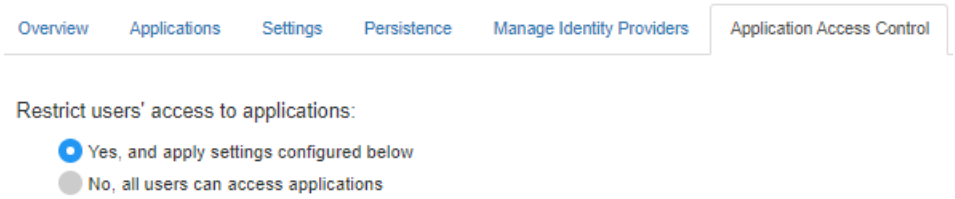
Specify Access Control Policy Rules

Specify the applications that certain user groups can access by defining access control policy rules. To define the rule, click **Add Rule** under **Access Control Policy** in the **Application Access Control** tab of the dashboard. Specify the following values.

Field	Value
Rule ID	Name for the rule
Description	Description for your rule
Users	User names set up in Azure AD that are allowed access to deployed applications
Groups	Object IDs of the groups set up in Azure AD groups that are allowed access to deployed applications
Applications	Applications that the specified users and groups can access. To select all applications, select `Apply this rule to all applications`.

Enable Application Access Control

After you configure the identity provider and specify access control policy rules, you must enable dashboard access control by selecting the **Yes** option from the dashboard.



Generate Access Token

After application access control is enabled, users that are specified in the access control policy rules can generate a bearer access token. If the registered client application is a native app, log in using a user name and password, or integrated Windows authentication to generate the access token. If the registered client application is a web app, log in using the browser with single sign-on to generate the access token. You can use the Microsoft identity platform authentication libraries (Microsoft-supported client libraries or compatible client libraries in different programming languages) to generate the access token. For more information, see Microsoft documentation. Use this access token in the HTTP authorization header when you make a request to the server using the MATLAB Production Server RESTful API. The format for this header is `Authorization:Bearer <access token>`.

See Also

Related Examples

- “Application Access Control” on page 10-20
- “Configure Application Access Control Using Google Identity” on page 10-26
- “Configure Application Access Control Using PingFederate” on page 10-28
- “Configure Application Access Control Using Other Identity Providers” on page 10-30

Configure Application Access Control Using Google Identity

MATLAB Production Server administrators can use Google Identity to restrict access to deployed applications to only certain users or groups of users. To enable application access control, configure Google Identity and specify access control policies, in consultation with the Google Identity administrator.

Register Application in Google Cloud Platform Console

To use Google Identity for application access control, register an application in the Google Cloud Platform Console. For more information about registering an application, see Google Identity documentation.

Configure Identity Provider in Dashboard

After you register the application in Google, create a configuration for Google Identity in the **Application Access Control** tab of the dashboard. Click **Create** and select **Google**. In **Create Identity Provider for Application Access Control**, enter application-specific and identity provider-specific values. Click **Create**. If the server is running on a Windows virtual machine, saving the values can take up to 30 seconds.

The following table describes the values that you must enter.

Field	Value
Name	Name for your Google identity provider configuration
App ID	Client ID of the application registered in Google for application access control

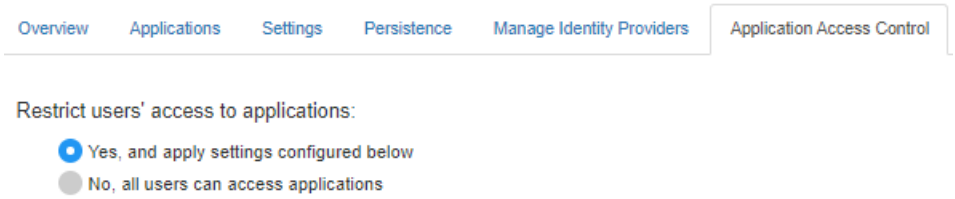
Specify Access Control Policy Rules

Specify the applications that certain users or user groups can access by defining access control policy rules. To define a rule, click **Add Rule** under **Access Control Policy** in the **Application Access Control** tab of the dashboard. Specify the following values.

Field	Value
Rule ID	Name for the rule
Description	Description for the rule
Users	Google user names that are allowed access to deployed applications
Groups	Google group IDs, if applicable, that are allowed access to deployed applications
Applications	Applications that the specified users and groups have permission to access Select `Apply this rule to all applications` to select all applications.

Enable Application Access Control

After you configure the identity provider and specify access control policy rules, you must enable dashboard access control by selecting the **Yes** option from the dashboard.



Generate Access Token

After application access control is enabled, users that are specified in the access control policy rules can generate a bearer access token. For more information about generating an access token, see the Google documentation for Using OAuth 2.0 for Web Server Applications.

Client programs use this access token in the HTTP authorization header when making a request to the server using the MATLAB Production Server RESTful API. The format for this header is `Authorization:Bearer <access token>`.

See Also

Related Examples

- “Application Access Control” on page 10-20
- “Configure Application Access Control Using Azure AD” on page 10-22
- “Configure Application Access Control Using PingFederate” on page 10-28
- “Configure Application Access Control Using Other Identity Providers” on page 10-30

Configure Application Access Control Using PingFederate

MATLAB Production Server administrators can use PingFederate from Ping Identity to restrict access to deployed applications to only certain users or groups of users. To enable application access control, configure PingFederate and specify access control policy rules, in consultation with the PingFederate administrator.

Prerequisites

Refer to the PingFederate documentation to configure OAuth use cases, clients and endpoints, and to configure OpenID provider information:

- Configuring OAuth Use Cases
- Configuring OAuth Clients
- OAuth 2.0 Endpoints
- Configuring OpenID Provider Information

Configure PingFederate in Dashboard

1. After you register the application with PingFederate, create a configuration for PingFederate in the **Application Access Control** tab of the dashboard. Click **Create** and select **PingFederate**.
2. In **Create Identity Provider for Application Access Control**, enter application-specific and identity provider-specific values. Click **Create**. If the server is running on a Windows virtual machine, saving the values can take up to 30 seconds.

   The following table describes the values that you must enter.

Field	Value
Name	Name for your PingFederate configuration.
App ID	Intended recipient of the JWT. The recipient helps in validating the *aud* claim in the JWT.
JWT Issuer	JWT issuer metadata of the identity provider. The metadata string must match the *iss* claim in the JWT.
JWKS URI	URI to retrieve the JSON Web Key Set (JWKS).

Specify Access Control Policy Rules

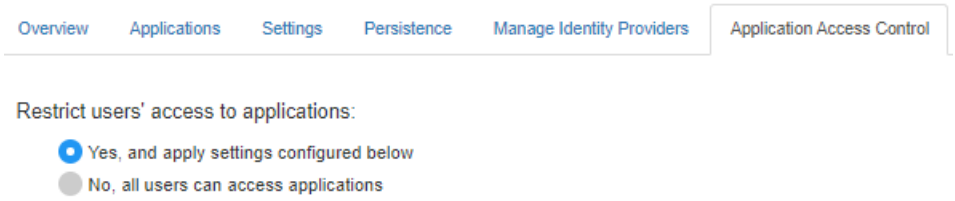
Specify the applications that certain users or user groups can access by defining access control policy rules. To define a rule, click **Add Rule** under **Access Control Policy** in the **Application Access Control** tab of the dashboard. Then, specify the following information.

Field	Value
Rule ID	Name for the rule
Description	Description for your rule

Field	Value
Users	User names that are allowed access to deployed applications
Groups	Group IDs, if applicable, that are allowed access to deployed applications
Applications	Applications that you want to allow the specified groups of users to access. To select all applications, select `Apply this rule to all applications`.

Enable Application Access Control

After you configure the identity provider and specify access control policy rules, you must enable dashboard access control by selecting the **Yes** option from the dashboard.



Generate Access Token

After application access control is enabled, users that are specified in the access control policy rules can generate a bearer access token. For more information about generating an access token, see the PingFederate OAuth 2.0 Developer Guide.

Clients programs use this access token in the HTTP authorization header when making a request to the server using the MATLAB Production Server RESTful API. The format for this header is `Authorization:Bearer <access token>`.

See Also

Related Examples

- “Application Access Control” on page 10-20
- “Configure Application Access Control Using Azure AD” on page 10-22
- “Configure Application Access Control Using Google Identity” on page 10-26
- “Configure Application Access Control Using Other Identity Providers” on page 10-30

Configure Application Access Control Using Other Identity Providers

Register Application With Identity Provider

To use an identity provider for application access control, register an application with the identity provider. Consult the identity provider administrator to register the application.

Configure Identity Provider in Dashboard

After you register the application with the identity provider, create a configuration for the identity provider in the **Application Access Control** tab of the dashboard. Click **Create** and select **Other**. In **Create Identity Provider for Application Access Control**, enter application-specific and identity provider-specific values. Click **Create**. If the server is running on a Windows virtual machine, saving the values can take up to 30 seconds.

The following table describes the values that you must enter.

Field	Value
Name	Name for your identity provider.
App ID	Intended recipient of the JWT. The recipient helps in validating the *aud* claim in the JWT.
JWT Issuer	JWT issuer metadata of the identity provider. The metadata string must match the *iss* claim in the JWT.
JWKS URI	URI to retrieve the JSON Web Key Set (JWKS).

Under **Create Identity Provider for Application Access Control**, you have the option to provide values other than the defaults for **UserAttribute ID** and **GroupAttribute ID**. **UserAttribute ID** is the JWT claim name that uniquely identifies a user. **GroupAttribute ID** is the JWT claim name that lists the groups that a user belongs to. Depending on the identity provider you use, you might have to change the defaults.

Specify Access Control Policy Rules

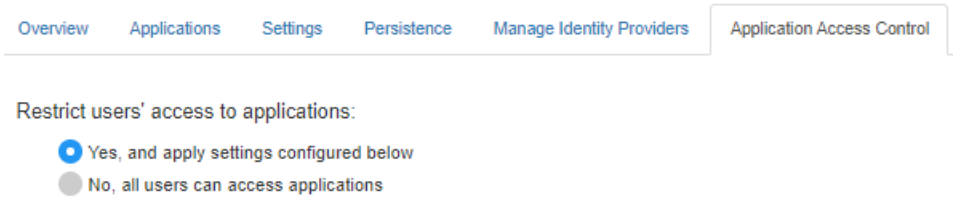
Specify the applications that certain users or user groups can access by defining access control policy rules. To define a rule, click **Add Rule** under **Access Control Policy** in the **Application Access Control** tab of the dashboard. Then, specify the following information.

Field	Value
Rule ID	Name for the rule.
Description	Description for your rule.
Users	User names set up in the identity provider that are allowed access to deployed applications.
Groups	Group IDs set up in the identity provider that are allowed access to deployed applications.

Field	Value
Applications	Applications that the specified users and groups can access. To select all applications, select `Apply this rule to all applications`.

Enable Application Access Control

After you configure the identity provider and specify access control policy rules, you must enable dashboard access control by selecting the **Yes** option from the dashboard.



Generate Access Token

After you enable application access control, clients can generate a bearer token. Client programs can use third-part libraries for token generation. For a list of OAuth libraries, see OAuth libraries. Client programs use this bearer token in the HTTP authorization header when making a request to the server using the MATLAB Production Server RESTful API. The format for this header is `Authorization:Bearer <access token>`.

See Also

Related Examples

- “Application Access Control” on page 10-20
- “Configure Application Access Control Using Azure AD” on page 10-22
- “Configure Application Access Control Using Google Identity” on page 10-26
- “Configure Application Access Control Using PingFederate” on page 10-28

Dashboard Access Control

MATLAB Production Server lets server administrators use OpenID Connect (OIDC) identity providers such as Microsoft Azure Active Directory (Azure AD), Google Identity, PingFederate from Ping Identity, and others to configure role-based access control for the dashboard. Role-based access control allows administrators to grant a dashboard user the privileges to perform tasks on the dashboard based on their role.

Note To use dashboard access control, your MATLAB Production Server deployment must use HTTPS.

Dashboard User Roles

The dashboard access control feature supports the following roles.

- Application author — Application authors can upload and delete applications (deployable archives) and view logs.
- Manager — Managers can edit server settings, configure access control for applications, manage persistence services, and have all the privileges of an application author, which include uploading and deleting applications and viewing logs.
- Server administrator — Administrators can log in to server virtual machines and configure which users or groups of users can access the dashboard, and have all the privileges of a manager, which include editing server logs, configuring access control for applications, uploading and deleting applications, and viewing logs.

The following table shows the dashboard tabs that users with these roles can access.

Role	Overview	Applications	Settings	Persistence	Manage Identity Providers	Application Access Control	Dashboard Access Control	Logs
Server administrator	✓	✓	✓	✓	✓	✓	✓	✓
Manager	✓	✓	✓	✓	✓	✓		✓
Application author	✓	✓						✓

Note Only the server administrator can log in to the dashboard when dashboard access control is disabled or not configured. Only the server administrator has privileges to configure dashboard access control.

Configure Identity Provider and Specify Access Control Policies

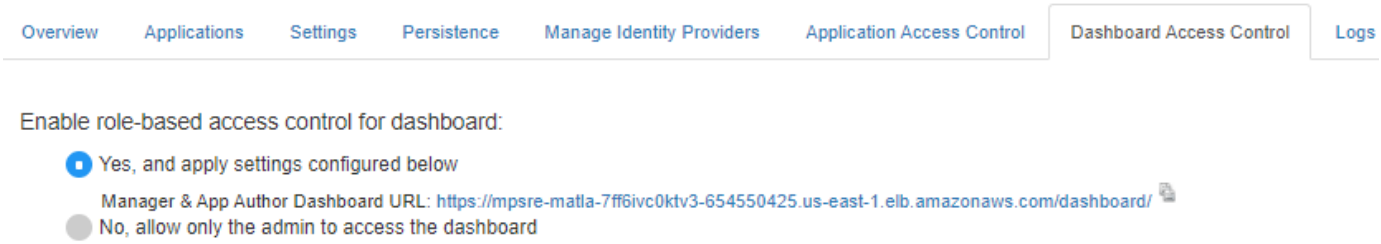
To enable dashboard access control for MATLAB Production Server, you must configure an identity provider and specify access control policies. The fields required to configure an identity provider vary based on the identity provider that you use. The access control policies define areas of the dashboard that users or groups of users can access and tasks that they can perform in these areas.

For information about configuring specific identity providers and policies, see:

- “Configure Dashboard Access Control Using Azure AD” on page 10-34
- “Configure Dashboard Access Control Using Google Identity” on page 10-37
- “Configure Dashboard Access Control Using PingFederate Identity Provider” on page 10-39
- “Configure Dashboard Access Control Using Other OpenID Connect Providers” on page 10-42

Enable Access Control

After you configure the identity provider and specify access control policies, you must enable dashboard access control. After enabling dashboard access control, a dashboard login URL that supports single sign-on (SSO) becomes available. Share this URL with managers and application authors.



See Also

More About

- “Manage MATLAB Production Server (PAYG)” on page 10-9
- “Application Access Control” on page 10-20

Configure Dashboard Access Control Using Azure AD

MATLAB Production Server administrators can use Microsoft Azure AD to configure role-based access control for the MATLAB Production Server Dashboard. Role-based access control allows administrators to grant access to specific areas of the dashboard to certain users or groups of users. For more information about the roles that the dashboard supports, see “Dashboard Access Control” on page 10-32.

To enable dashboard access control, configure Azure AD and specify access control policies, in consultation with the Azure AD administrator.

Configure Identity Provider

To configure Azure AD:

1. Log in to the dashboard to retrieve the redirect URI of the dashboard.
2. In the Azure portal, use the redirect URI to register the dashboard as a client application with the provider.
3. In the dashboard, enter values specific to the registered application and Azure AD.

Retrieve Redirect URI from Dashboard

To retrieve the redirect URI, start creating a configuration for Azure AD in the dashboard:

1. Navigate to either the **Dashboard Access Control** tab or the **Manage Identity Providers** tab.
2. Click **Create** and select **Azure AD**.
3. In **Create Identity Provider for Dashboard Access Control**, note the redirect URI of the dashboard.

   Later, you return to this view to specify the values required to configure your identify provider in the dashboard.

Register Application in Azure Portal

Use the Azure portal to register a web client application for dashboard access control. When registering the application, use the redirect URI from the MATLAB Production Server dashboard. Typically, the Azure AD administrator registers the application.

1. Sign in to the Azure portal.
2. From **Azure Active Directory**, select **App registrations** and click **New registration**.
3. In the resulting pane, enter the name of the application (for example, `MATLAB Production Server Dashboard App`).
4. For the **Redirect URI**, select **Web**. In the corresponding value field, enter the redirect URI of the dashboard and click **Register**. A web page displays the details of your registered application.
5. Click **Manifest** in the left navigation pane. In the JSON that is displayed in the resulting pane, set the value for `groupMembershipClaims` to `"SecurityGroup"`. Click **Save**.

For more information on how to register an application, see the Microsoft Azure documentation.

Specify Values in Dashboard

In the Azure portal, find the values of the client application that you registered and enter them into the dashboard.

1. Sign in to the Azure portal.
2. From **Azure Active Directory**, select **App registrations** and then select the application that you registered for the dashboard. Copy the value from the **Application (client) ID** and paste it into the **Client ID** field in the dashboard.
3. From **App registrations**, select **Certificates & secrets**. Under **Certificates & secrets**, create a new client secret or use an existing one. Copy the value for the client secret and paste it into the **Client Secret** field in the dashboard.
4. From **Azure Active Directory**, select **Properties**. Copy the value from **Directory (tenant) ID** and paste it into the **Tenant ID** in the dashboard.
5. On the dashboard, click **Create**.

Specify Dashboard Access Control Policy

Before you can specify dashboard access control policies, you must have users and groups set up in Azure AD. Consult the Azure AD administrator for this setup.

The access control policies define areas of the dashboard that users or groups of users can access and tasks that they can perform in these areas. Use the policies to assign the manager and application author roles to users or groups of users in your organization by entering their Azure user names and group IDs into the dashboard.

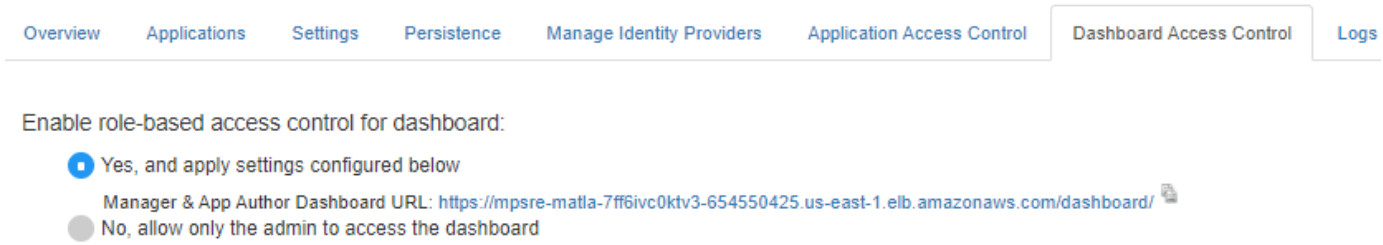
Configure Users and Groups in Dashboard

In the Azure portal, find user names and group IDs and enter them into the dashboard.

1. Sign in to the Azure portal.
2. From **Azure Active Directory** and then **Users**. Copy the values for the user names and paste them into the **Users** field in the dashboard. Use a comma to separate multiple user names.
3. From **Azure Active Directory** and then **Groups**. Copy the values for the object IDs and paste them into the **Groups** field in the dashboard. Use a comma to separate multiple object IDs names.
4. On the dashboard, click **Save**.

Enable Dashboard Access Control

After you configure Azure AD and specify access control policies, you must enable dashboard access control by selecting the **Yes** option. After enabling dashboard access control, a dashboard login URL that supports single sign-on (SSO) becomes available. Share this URL with managers and application authors.



See Also

Related Examples

- “Dashboard Access Control” on page 10-32
- “Configure Dashboard Access Control Using Google Identity” on page 10-37
- “Configure Dashboard Access Control Using PingFederate Identity Provider” on page 10-39
- “Configure Dashboard Access Control Using Other OpenID Connect Providers” on page 10-42

Configure Dashboard Access Control Using Google Identity

MATLAB Production Server administrators can use Google identity provider to configure role-based access control for the MATLAB Production Server Dashboard. Role-based access control allows administrators to grant access to specific areas of the dashboard to certain users or groups of users. For more information about the roles that the dashboard supports, see “Dashboard Access Control” on page 10-32.

To enable dashboard access control, configure the identity provider and specify access control policies, in consultation with the Google Identity administrator.

Configure Google Identity

To configure Google Identity:

1. Log in to the dashboard to retrieve the redirect URI of the dashboard.
2. On the Google Cloud Platform Console, use the redirect URI to register the dashboard as a client application with the provider.
3. In the dashboard, enter values specific to the registered application and Google Identity.

Retrieve Redirect URI from Dashboard

To retrieve the redirect URI, start creating a configuration for Google Identity in the dashboard:

1. Navigate to either the **Dashboard Access Control** tab or the **Manage Identity Providers** tab.
2. Click **Create** and select **Google**.
3. In **Create Identity Provider for Dashboard Access Control**, note the redirect URI of the dashboard.

   Later, you return to this view to specify the values required to configure your identify provider in the dashboard.

Register Application in Google Cloud Console

Use the Google Cloud Console to register a web client application for dashboard access control. Use the redirect URI from the MATLAB Production Server Dashboard when registering the application.

1. Sign in to the Google Cloud Platform Console and navigate to the **Credentials** page.
2. On the **Credentials** page, click **Create credentials** and select **OAuth client ID**.
3. From **Application type** drop down, select **Web Application**.
4. Enter the name of you client application (for example, `MATLAB Production Server Dashboard App`).
5. Under **Authorized redirect URIs**, click **Add URI**.
6. Copy the redirect URI from MATLAB Production Server Dashboard and paste it into the **URIs** field in the Google Cloud Console.
7. Click **Create**.
8. The Google identity provider creates an application with a client ID and client secret. Note the values of the client ID and client secret. You enter these values next in the dashboard.

Specify Client ID and Client Secret in Dashboard

1. Enter the noted client ID and client secret values from the previous section in the **Client ID** and **Client Secret** fields respectively in MATLAB Production Server dashboard.
2. Click **Create** to complete the configuration of the identity provider.

Specify Dashboard Access Control Policy

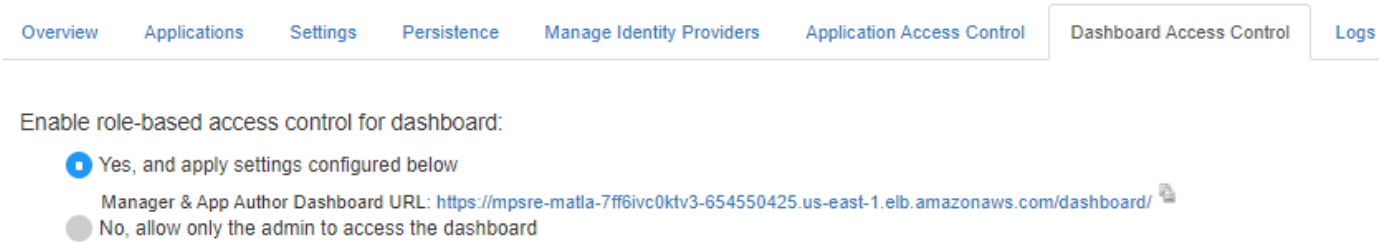
Before you can specify dashboard access control policies, you must have users, and groups, if applicable, set up in Google. Consult the Google identity provider administrator for this setup.

The access control policies define areas of the dashboard that users can access and tasks that they can perform in these areas. Use the policies to assign the manager and application author roles to users in your organization by entering their Google user names.

1. On the **Dashboard Access Control** tab of the dashboard, select Google as the identity provider.
2. In the **Dashboard Access Control Policy** section, enter Google user names to assign manager and application author roles to users in your organization. Click **Save** after you enter the values.

Enable Dashboard Access Control

After you configure Google Identity and specify access control policies, you must enable dashboard access control by selecting the **Yes** option. After enabling dashboard access control, a dashboard login URL that supports single sign-on (SSO) becomes available. Share this URL with managers and application authors.



See Also

Related Examples

- “Dashboard Access Control” on page 10-32
- “Configure Dashboard Access Control Using Azure AD” on page 10-34
- “Configure Dashboard Access Control Using PingFederate Identity Provider” on page 10-39
- “Configure Dashboard Access Control Using Other OpenID Connect Providers” on page 10-42

Configure Dashboard Access Control Using PingFederate Identity Provider

MATLAB Production Server administrators can use PingFederate from Ping Identity to configure role-based access control for the MATLAB Production Server Dashboard. Role-based access control allows administrators to grant access to specific areas of the dashboard to certain users or groups of users. For more information about the roles that the dashboard supports, see “Dashboard Access Control” on page 10-32.

To enable dashboard access control for MATLAB Production Server, configure PingFederate and specify access control policies, in consultation with the PingFederate administrator.

Prerequisites

Refer to the PingFederate documentation to configure OAuth use cases, clients, and endpoints to configure OpenID provider information:

- Configuring OAuth Use Cases
- Configuring OAuth Clients
- OAuth 2.0 Endpoints
- Configuring OpenID Provider Information

Configure PingFederate Identity Provider

To configure PingFederate:

1. Log in to the dashboard to retrieve the Redirect URI of the dashboard.
2. Use the Redirect URI to register a client application in PingFederate.
3. In the dashboard, enter values specific to the registered application and PingFederate.

Retrieve Redirect URI from Dashboard

To retrieve the redirect URI, start creating a configuration for your identity provider in the dashboard:

1. Navigate to either the **Dashboard Access Control** tab or the **Manage Identity Providers** tab.
2. Click **Create** and select **PingFederate**.
3. In **Create Identity Provider for Dashboard Access Control**, note the redirect URI of the dashboard.

   Later, you return to this view to specify the values required to configure your identify provider in the dashboard.

Register Application in PingFederate

Register an application in PingFederate for MATLAB Production Server Dashboard, if you do not already have one. Consult the PingFederate administrator to register the application. Provide the **Redirect URI** of the MATLAB Production Server Dashboard when registering the application.

Specify Values in Dashboard

After you register the application with PingFederate, you receive application specific values such as the **Client ID** and **Client Secret**. Enter the values specific to the application and values specific to PingFederate in the dashboard under **Create Identity Provider for Dashboard Access Control**.

The following table describes the values that you must enter. Click **Create** after you enter the values.

Field	Description
Client ID	Application ID of the registered client application.
Client Secret	Client secret of the registered client application.
OIDC Issuer	Discovery endpoint URI of the OIDC provider.
JWT Issuer	JWT issuer metadata of the OIDC provider.
JWKS URI	URI to retrieve the JSON Web Key Set (JWKS).

Specify Dashboard Access Control Policy

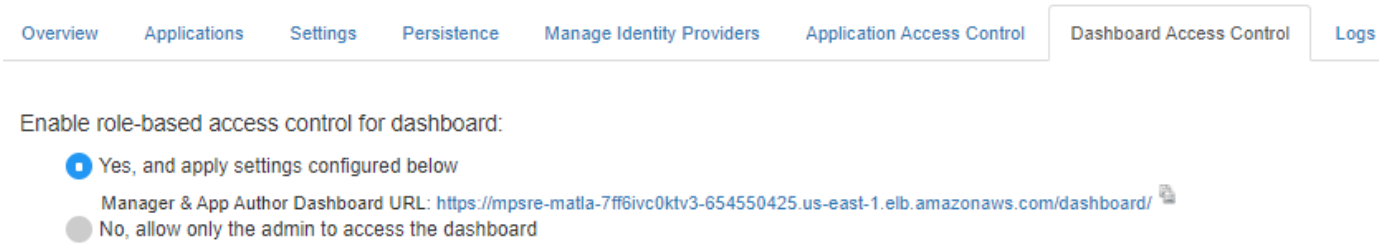
Before you can specify dashboard access control policies, you must have users, and groups, if applicable, set up in PingFederate. Consult the PingFederate administrator for this setup.

The access control policies define areas of the dashboard that users or groups of users can access and tasks that they can perform in these areas. Use the policies to assign the manager and application author roles to users or groups of users in your organization by entering their user names and group IDs. Click **Save** after you enter the values.

1. In the **Dashboard Access Control** tab of the dashboard, select PingFederate as the identity provider.
2. In the **Dashboard Access Control Policy** section, enter identity provider specific user names and group IDs to assign manager and application author roles to users or groups of users in your organization. Use a comma to separate multiple user names and group IDs. Click **Save** after you enter the values.

Enable Dashboard Access Control

After you configure PingFederate and specify access control policies, you must enable dashboard access control by selecting the **Yes** option. After enabling dashboard access control, a dashboard login URL that supports single sign-on (SSO) becomes available. Share this URL with managers and application authors.



See Also

Related Examples

- “Dashboard Access Control” on page 10-32
- “Configure Dashboard Access Control Using Azure AD” on page 10-34
- “Configure Dashboard Access Control Using Google Identity” on page 10-37
- “Configure Dashboard Access Control Using Other OpenID Connect Providers” on page 10-42

Configure Dashboard Access Control Using Other OpenID Connect Providers

MATLAB Production Server Dashboard supports the use of any OpenID Connect (OIDC) identity provider for role-based access control for the dashboard. Role-based access control allows server administrators to grant access to specific areas of the dashboard to certain users or groups of users. For more information about the roles that the dashboard supports, see “Dashboard Access Control” on page 10-32.

To enable dashboard access control, configure the OIDC provider and specify dashboard access control policies, in consultation with the OIDC provider administrator.

Configure Identity Provider

To configure an identity provider:

1. Log in to the dashboard to retrieve the redirect URI of the dashboard.
2. At the identity provider's website, use the redirect URI to register the dashboard as a client application with the provider.
3. In the dashboard, enter values specific to the registered application and identity provider.

Retrieve Redirect URI from Dashboard

To retrieve the redirect URI, start creating a configuration for your identity provider in the dashboard:

1. Navigate to either the **Dashboard Access Control** tab or the **Manage Identity Providers** tab.
2. Click **Create** and select **Other**.
3. In **Create Identity Provider for Dashboard Access Control**, note the redirect URI of the dashboard.

   Later, you return to this view to specify the values required to configure your identify provider in the dashboard.

Register Application with Identity Provider

Register an application with the OIDC provider for MATLAB Production Server Dashboard. Consult the OIDC provider administrator to register the application. When registering the application, provide the redirect URI of the MATLAB Production Server Dashboard.

Specify Values in Dashboard

After you register the application with the identity provider, you receive application-specific values such as the client ID and client secret. Enter the values in the dashboard under **Create Identity Provider for Dashboard Access Control**.

The following table describes the values that you must enter. Click **Create** after you enter the values.

Field	Description
Client ID	Application ID of the registered client application

Field	Description
Client Secret	Client secret of the registered client application
OIDC Issuer	Discovery endpoint URI of the OIDC provider
JWT Issuer	JWT issuer metadata of the OIDC provider
JWKS URI	URI to retrieve the JSON Web Key Set (JWKS)

Under **Create Identity Provider for Dashboard Access Control**, you have the option to provide values other than the defaults for **UserAttribute ID** and **GroupAttribute ID**. **UserAttribute ID** is the JWT claim name that uniquely identifies a user. **GroupAttribute ID** is the JWT claim name that lists the groups that a user belongs to. Depending on the identity provider you use, you might have to change the defaults.

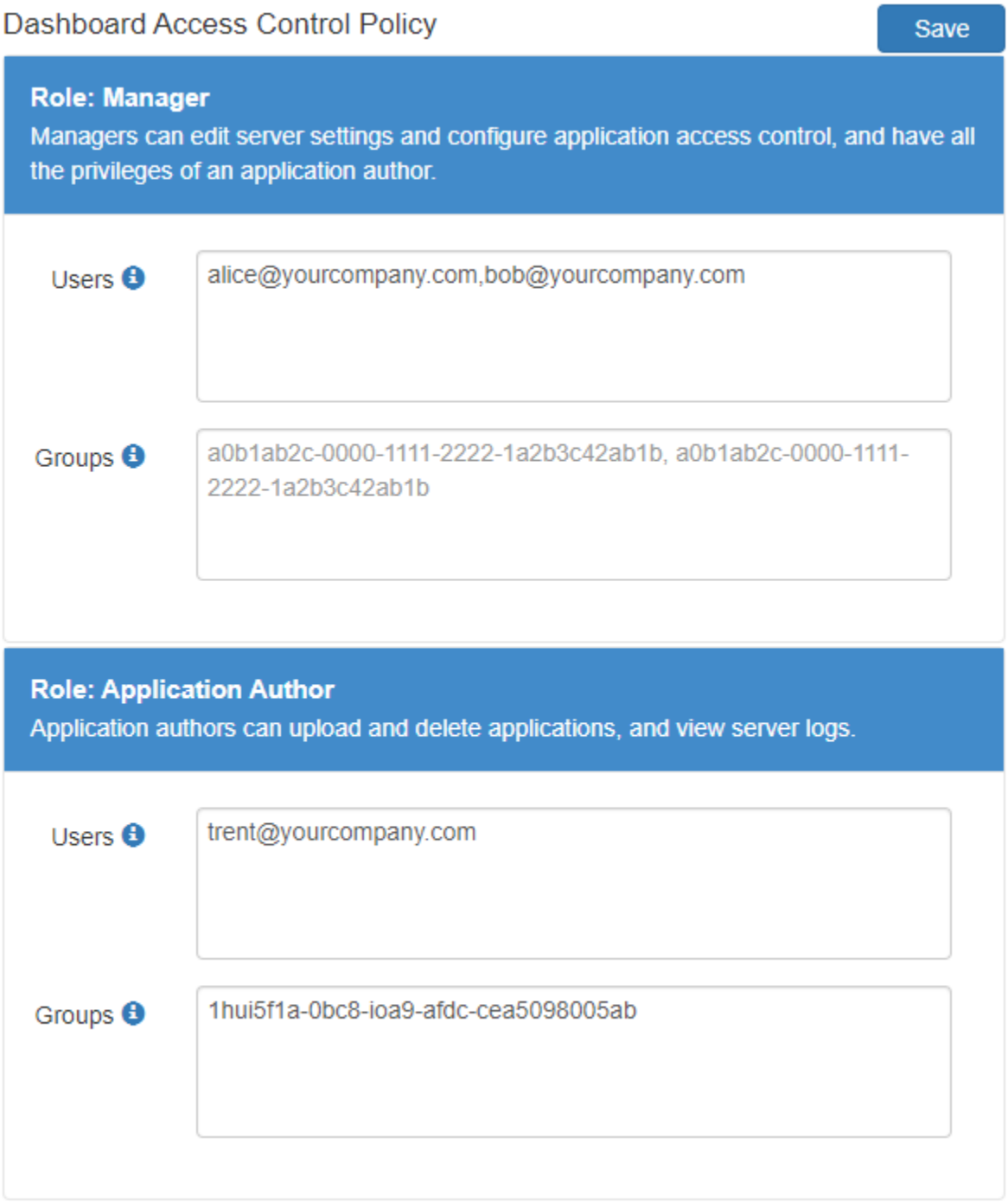
Specify Dashboard Access Control Policy

Before you can specify dashboard access control policies, you must have users, and if applicable, groups, set up in the identity provider. Consult the OIDC provider administrator for this setup.

The access control policies define areas of the dashboard that users or groups of users can access and tasks that they can perform in these areas.

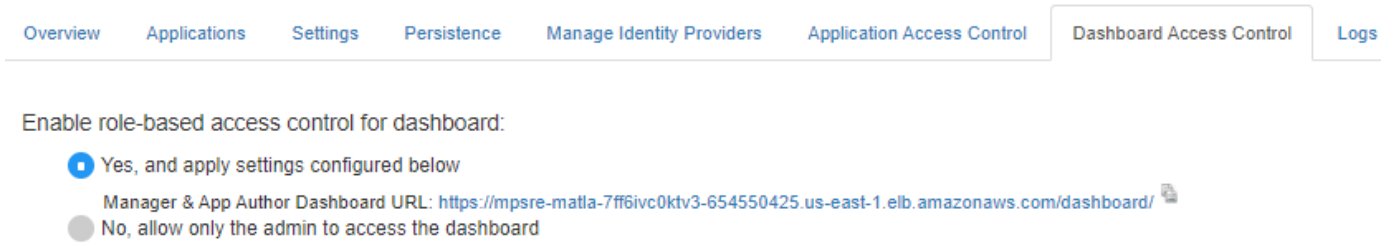
1. On the **Dashboard Access Control** tab of the dashboard, select the identity provider that you want to use.
2. In the **Dashboard Access Control Policy** section, enter identity provider specific user names and group IDs to assign manager and application author roles to users or groups of users in your organization. Use a comma to separate multiple user names or group IDs. Click **Save** after you enter the values.

   For example, in the following illustration, the users *alice@yourcompany.com* and *bob@yourcompany.com* have the manager role. The user *trent@yourcompany.com* and all users that belong to group ID *1hui5f1a-0bc8-ioa9-afdc-cea5098005ab* have the application author role.



Enable Dashboard Access Control

After you configure the identity provider and specify access control policies, you must enable dashboard access control by selecting the **Yes** option. After enabling dashboard access control, a dashboard login URL that supports single sign-on (SSO) becomes available. Share this URL with managers and application authors.



See Also

Related Examples

- “Dashboard Access Control” on page 10-32
- “Configure Dashboard Access Control Using Azure AD” on page 10-34
- “Configure Dashboard Access Control Using Google Identity” on page 10-37
- “Configure Dashboard Access Control Using PingFederate Identity Provider” on page 10-39

Execute MATLAB Functions on MATLAB Production Server (PAYG)

Client applications for MATLAB Production Server pay as you go (PAYG) are different from those for on-premises server instances in several ways. To execute MATLAB functions deployed on MATLAB Production Server (PAYG), you must use the MATLAB execution endpoint URL specified in the dashboard. Depending on the implementation of your client program, you might have to update your code to use the sticky session or session affinity feature of the elastic load balancer.

Similar to client applications for on-premise server installations, you must use the MATLAB Production Server client libraries for client applications that you write using Java, .NET, C, and Python.

Use MATLAB Execution Endpoint URL

After your MATLAB Production Server (PAYG) deployment to AWS is complete, log in to the dashboard to retrieve the MATLAB execution endpoint. The **Overview** tab in the dashboard specifies the **MATLAB Execution Endpoint**. For information on accessing the dashboard, see .

This endpoint is an HTTPS URL that client programs use to make requests to the server and execute MATLAB functions deployed to the server. For example, if the MATLAB execution endpoint for your server is `https://payg-mps-1n8k9DNJ8NJAH-1476423457.us-east-1.elb.amazonaws.com`, to use the MATLAB Production Server RESTful API to execute a MATLAB function `mymagic` located in a deployed application `myapp`, specify the URL `https://payg-mps-1n8k9DNJ8NJAH-1476423457.us-east-1.elb.amazonaws.com/myapp/mymagic`.

Download Client Libraries

If you want to write client programs in Java, .NET, C, and Python for invoking MATLAB functions deployed on the server, you must use the MATLAB Production Server client libraries. Download the client libraries from MATLAB Production Server Client Libraries.

Work with Self-Signed SSL Certificate

If your deployment uses a self-signed SSL certificate, you might have to manually override the default security behavior of the client application or add a new HTTPS endpoint to the application gateway.

You can update the client application to disable hostname verification when sending HTTPS requests to the server to avoid encountering an exception caused by a failure in host name verification. Depending on the implementation of your client program, you might also have to retrieve the self-signed certificate that the load balancer uses and add the certificate to your local truststore. For more information on configuring the client environment, see “Handle Exceptions” for a Java client and “Handle Exceptions” for a .NET client.

Manage HTTP Cookie

The elastic load balancer uses sticky sessions, where it uses cookies to keep a user session on the same server. On receiving a request from a client program, the application gateway sets the `Set-Cookie` HTTP response header with information about the server virtual machine (VM) that processes the request.

Asynchronous Request Execution

A client program that uses asynchronous requests to execute a MATLAB function deployed to the server must set the `Cookie` HTTP request header with the value of the `Set-Cookie` header for all subsequent requests. This ensures that same server VM that processes the first request processes all subsequent requests for that session.

Synchronous Request Execution

A client program that uses synchronous requests to execute a MATLAB function deployed to the server must not set the `Cookie` HTTP request header with the value of the `Set-Cookie` header, and must clear the value of the `Cookie` header if it has been previously set. This ensures that the synchronous requests are load balanced and the same server VM does not process them.

The default property in a Java client that uses `MWHttpClient` sets the HTTP cookie. For information about disabling cookies, see “Configure Client-Server Connection”. The Java client API that uses protobuf and the .NET client API do not set the HTTP cookie by default.

See Also

More About

- “Manage MATLAB Production Server (PAYG)” on page 10-9
- “Manage AWS Resources for MATLAB Production Server (PAYG)” on page 10-14
- “RESTful API for MATLAB Function Execution”

Run MATLAB Production Server on Amazon Web Services Using Reference Architecture

Deploy MATLAB Production Server in Amazon Web Services (AWS) using a customizable reference architecture. Use this reference architecture when you want to launch MATLAB Production Server in a specific region, combine MATLAB Production Server with your existing cloud resources, or automate deployment.

Requirements

- A MATLAB Production Server license that meets the following conditions:
  - Linked to a MathWorks Account.
  - Concurrent license type. To check your license type, view your MathWorks Account.
  - Configured to use a network license manager on the virtual network. By default, the deployment of MATLAB Production Server includes a network license manager, but you can also use an existing license manager. In either case, activate or move the license after deployment. For details, see “Configure MATLAB Production Server License for Use on the Cloud”.
- An AWS account. You are responsible for the costs of all AWS services.
- An SSH Key Pair for your AWS account in the appropriate region. For more information, see Amazon EC2 Key Pairs.

Run from GitHub

To launch MATLAB Production Server in AWS, use the reference architecture templates provided in the following GitHub repository:

- MATLAB Production Server on Amazon Web Services

You can run the template directly from the link in the GitHub repository. Choose your MATLAB Production Server release, and then click the **Launch Stack** icon to deploy your resources.

See Also

Related Examples

- “Run MATLAB Production Server on Azure Using Reference Architecture” on page 9-129
- “Configure MATLAB Production Server License for Use on the Cloud”

Manage MATLAB Production Server Using the Dashboard

After you deploy the MATLAB Production Server reference architecture in Amazon Web Services, and configure licensing in the cloud, use the MATLAB Production Server dashboard, which is a web-based interface to upload applications, edit server settings, and configure access control for the dashboard and applications.

For information on deploying the reference architecture, see MATLAB Production Server on AWS. For information on setting up your MATLAB Production Server license for using in the cloud, see "Configure MATLAB Production Server License for Use on the Cloud".

Connect to Dashboard

Find the URL to connect to the dashboard in the AWS management console.

Note The Internet Explorer web browser is not supported for interacting with the dashboard.

Complete these steps only after you successfully create your stack. If your solution uses private IP addresses, you can connect to the dashboard from a virtual machine (VM) that belongs to the same virtual network as the VM that hosts the dashboard.

1. In the AWS management console, select the stack that you deployed. You can find the stack under the CloudFormation service.
2. In the stack detail pane, expand the **Outputs** section
3. Look for the key named `MATLABProductionServerDashboardURL` and click the corresponding URL listed under value. This is the HTTPS endpoint to the MATLAB Production Server cloud dashboard.

Log In to Dashboard

There are three roles for logging in to the dashboard depending on your role in your organization — global admin, manager, or application author.

Log In to Dashboard as Administrator

If you are accessing the dashboard for the first time, or if you have not configured or not enabled dashboard access control, you must log in using the administrator credentials. These are the credentials that you entered during the dashboard login step of the deployment process. An administrator has access to all areas of the dashboard. An administrator can log in to server virtual machines (VMs), configure which users or groups of users can access the dashboard and applications, edit server settings, configure remote persistence services, view logs, and upload and delete applications. You cannot change the administrator credentials.

To grant access to only certain dashboard areas for specific users or groups of users, set up dashboard access control after you log in. For more information, see "Dashboard Access Control" on page 10-57.

Log In to Dashboard as Manager or Application Author

If the administrator has configured and enabled dashboard access control, you can log in to the dashboard as a manager or application author depending on your role in your organization. The login process supports single sign-on using OpenID Connect (OIDC) identity providers.

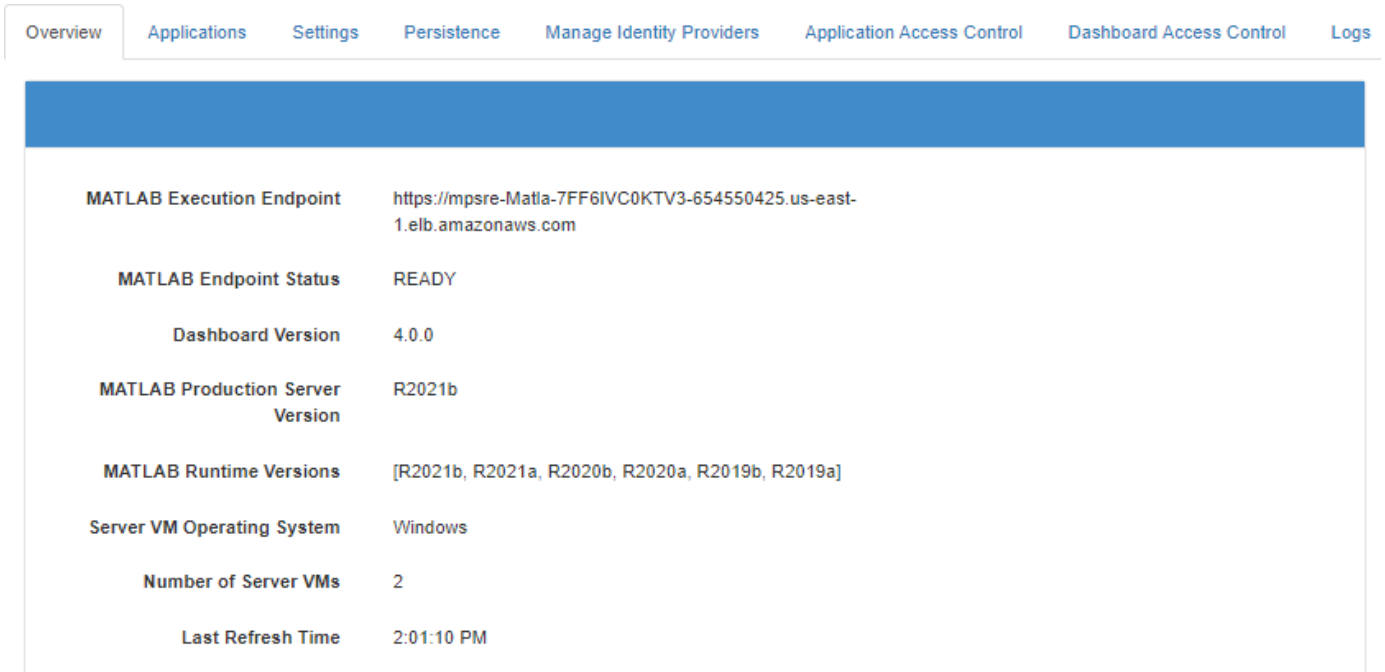
An application author has the privileges to upload and delete applications and view logs. A manager has the privileges to edit server settings, configure access control for applications, and configure remote persistence services, as well as all the privileges of an application author, including for uploading and deleting applications and viewing logs.

The following table shows the dashboard tabs that users with these roles can access.

Role	Overview	Applications	Settings	Persistence	Manage Identity Providers	Application Access Control	Dashboard Access Control	Logs
Server administrator	✓	✓	✓	✓	✓	✓	✓	✓
Manager	✓	✓	✓	✓	✓	✓		✓
Application author	✓	✓						✓

View Information About Server

To view information about the server instance VMs deployed on Azure, click **Overview** on the dashboard navigation menu.



Upload MATLAB Application

You can run an application on MATLAB Production Server that is created using MATLAB Compiler SDK. Upload and deploy applications using the **Applications** tab in the dashboard.

1. Click **+Upload**.
2. Click **Choose File**, select the file, and click **Deploy**.

For information on how to upload multiple applications, see “Upload MATLAB Applications” on page 10-55.

For information on how to create an application, see “Create Deployable Archive for MATLAB Production Server”.

View MATLAB Execution Endpoint

The deployment provides an HTTPS endpoint URL to make requests to the server. The **MATLAB Execution Endpoint** in the **Overview** tab in the dashboard specifies the HTTPS endpoint. Use this URL to execute MATLAB functions deployed to the server. For example, if the MATLAB execution endpoint for your server is `https://refarch-mps-1n8k9DNJ8NJAH-1476423457.us-east-1.elb.amazonaws.com`, to use the MATLAB Production Server RESTful API to execute a MATLAB function `mymagic` located in a deployed application `myapp`, use the URL `https://payg-mps-1n8k9DNJ8NJAH-1476423457.us-east-1.elb.amazonaws.com/myapp/mymagic`.

For information on working with self-signed SSL certificates and managing cookies set by the load balancer, see “Execute MATLAB Functions on MATLAB Production Server” on page 10-86.

Edit Server Configuration

Edit the MATLAB Production Server configuration properties by selecting the **Settings** tab in dashboard. After you update the properties, click **Save** to apply your changes.

Note The server restarts when you click **Save**.

Only the administrator and managers have the privilege to edit the server configuration.

Some examples of server configuration properties follow.

- To allow requests from specific domains, specify parameter values for the **CORS Allowed Origins** property.

  If you want to specify multiple domains, separate them with a comma, for example, `http://www.w3.org, https://www.apache.org`.
- To set the number of MATLAB Production Server workers, specify a parameter value for the **Number of Workers** property.

  When setting the **Number of Workers** property, carefully consider your cluster setup. Each VM in the cluster runs an instance of MATLAB Production Server and each instance runs multiple MATLAB Production Server workers. Using 1 vCPU per MATLAB Production Server worker is recommended. For example, if the server size is set to *Standard_D4s_v3 Server*, which specifies 4 vCPUs, set the number of workers to no more than 4 workers per instance.

Use Amazon ElastiCache for Redis for Data Persistence

MATLAB Production Server uses Redis for data persistence. Persistence allows caching of data between calls to MATLAB code running on the servers. Only the administrator and managers have the privilege to configure remote persistence services.

To view a persistence service that your deployment creates or to create a new remote persistence service, select **Persistence** in the MATLAB Production Server dashboard navigation pane.

Click **+Add** to create a new remote persistence service. Specify the following parameter values, then click **Create**.

Value	Description
Connection Name	Specify a name for the connection to the persistence service. Use this name in MATLAB code to pass data to the cache.
Host and **Port**	Specify the host name and port number for your Redis cache. The port number has to be a non-SSL port. If you are using Amazon ElastiCache for Redis, use the following procedure to retrieve the host name and port number. 1 Log in to your AWS management console and select the stack associated with the deployment. You can access the stack from the CloudFormation service. 2 In the stack detail pane, expand the **Outputs** section. 3 From the **Outputs** section, copy the value corresponding to the key named `MATLABProductionServerCloudStackCacheClusterAddress` and paste it in the **Host** field in the dashboard. 4 From the **Outputs** section, copy the value corresponding to the key named `MATLABProductionServerCloudStackCacheClusterPort` and paste it in the **Port** field in the dashboard. You can also use a Redis cache that you create outside of this deployment.
Access Key	Leave this field blank.

For more information on using a data cache, see “Data Caching Basics”.

Set Up Access Control for Applications Using OAuth 2.0 Providers

MATLAB Production Server integrates with OAuth 2.0 providers such as Microsoft Azure Active Directory (Azure AD), Google Identity, and PingFederate, for restricting access to deployed applications to only certain groups of users. Only the administrator and manager can configure access control for applications.

For more information on how to configure application access control using the **Application Access Control** tab, see “Application Access Control” on page 10-71.

Set Up Access Control for Dashboard Using OAuth 2.0 Providers

MATLAB Production Server integrates with OAuth 2.0 providers such as Microsoft Azure Active Directory (Azure AD), Google Identity, and PingFederate from Ping Identity, and uses JSON Web Tokens (JWTs) to provide role-based access control for accessing the dashboard. You can grant access to certain dashboard areas for specific users or groups of users based on the user role. Only the administrator can configure dashboard access control. For more information on how to configure dashboard access control using the **Dashboard Access Control** tab, see “Dashboard Access Control” on page 10-57.

See Also

More About

- MATLAB Production Server Reference Architecture on Azure
- “Configure MATLAB Production Server License for Use on the Cloud”
- “Dashboard Access Control” on page 10-57
- “Application Access Control” on page 10-71

Manage AWS Resources for MATLAB Production Server

After you deploy the MATLAB Production Server environment on AWS, use the AWS management console to manage resources that you provision for in the deployment.

Change Number of Virtual Machines

Each MATLAB Production Server instance runs on a virtual machine (VM) and each instance runs multiple MATLAB Production Server workers. You can change the number of VMs after the initial deployment.

1. Log in to the AWS management console and select the CloudFormation stack associated with the deployment.
2. Select the **Outputs** tab from the top pane.
3. Click the link corresponding to **MATLABProductionServerAutoScalingGroup**. Doing so opens a new window that lets you manage auto scaling groups.
4. You can change the number of server VMs in this window. For information on how to do so, see Manual scaling for Amazon EC2 Auto Scaling.

Import SSL Certificate

The MATLAB Production Server deployment requires an SSL certificate present in your AWS account. When you deploy MATLAB Production Server, you get two HTTPS endpoint URLs. One endpoint lets you connect to the server instances and the other to the dashboard. For information on creating a self-signed SSL certificate, see Create and sign an X509 certificate.

1. Open the AWS certificate manager.
2. Click **Import a Certificate**.
3. Copy the contents of the CRT file containing the certificate into the field labeled **Certificate body**.
4. Copy the contents of the PEM file containing the private key into the field labeled **Certificate private key**.
5. Leave the filed labeled **Certificate chain** blank and click **Next**.
6. Click **Review and Import**.
7. Review the values and click **Import**.
8. Copy the value of the ARN field from the **Details** section of the certificate and paste it into the **ARN of SSL Certificate** parameter during deployment.

Change SSL Certificate

When you deploy MATLAB Production Server, you get two HTTPS endpoint URLs. One endpoint lets you connect to the server instances and the other to the dashboard. You can change the X.509 certificates for these resources.

1. Log in to the AWS management console and select the CloudFormation stack associated with the deployment.
2. Select the **Outputs** tab from the top pane.

3. To change the X.509 certificate for the server instances, click the link corresponding to **MATLABProductionServerLoadBalancer** key.
4. To change the X.509 certificate for the dashboard, click the link corresponding to **MATLABProductionServerDashboardLoadBalancer** key.
5. Select the **Listeners** tab, the click **Change** under **SSL Certificate**.

For more information, see Replace the SSL certificate for your Classic Load Balancer.

Upload MATLAB Applications

The Amazon S3™ bucket lets you upload and deploy multiple applications to the server.

1. Log in to the AWS management console and select the CloudFormation stack associated with the deployment.
2. Select the **Outputs** tab from the top pane.
3. Click the link corresponding to **MATLABProductionServerApplicationsBucket**. Doing so opens a new window that displays details for the Amazon S3 bucket.
4. In the pane that opens, click the **auto_deploy** folder.
5. Select **Upload** to select the applications to upload, then click **Upload**.

View Logs

View MATLAB Production Server logs using AWS CloudWatch .

1. Log in to the AWS management console and select the CloudFormation stack associated with the deployment.
2. Select the **Outputs** tab from the top pane.
3. Click the link corresponding to **MATLABProductionServerWorkerVMLogGroup**. Doing so opens a new CloudWatch window that displays details for the `MATLABProductionServerWorkerVMLogGroup`.
4. Select **View in Logs Insights**. Doing so opens a new Logs Insights console with an existing query. Ignore this default query.

To view logs generated by all the server instances, you can use the following query.

```
fields @timestamp, @message
| filter @logStream like 'prodServerInstance'
| limit 200
```

To view the last 200 logs generated by all server instances, you can use the following query.

```
fields @timestamp, @message
| filter @logStream like 'prodServerInstance'
| limit 200
```

To view only the error logs generated by all server instances, you can use the following query. For more information on log severity, see `log-severity`.

```
fields @timestamp, @message
| filter @logStream like 'prodServerInstance'
| filter severity like 'error'
```

To view logs generated by all the server instances within a certain time duration, you can use the custom range selector.

Handle Timeouts

If the deployed MATLAB function takes more than 120 seconds to finish execution, the client application receives a `504 GATEWAY_TIMEOUT` error. This is because the load balancer configuration closes a connection that is idle for more than 120 seconds. To avoid the time out error, you can increase the load balancer timeout.

1. In the AWS management console, select the stack that you deployed. You can find the stack under the CloudFormation service.
2. In the stack detail pane, expand the **Outputs** section.
3. Look for the key named `MATLABProductionServerLoadBalancer` and click the corresponding URL listed under value to configure the load balancer.
4. Click **Edit idle timeout** under the **Attributes** section.
5. Set the timeout value based on the time that your deployed functions require for execution.

Delete Stack

A stack contains all the related AWS resources for a solution. You can remove the stack and all the associated cluster resources when you no longer need them. You cannot undo this.

1. Log in to the AWS management console and select the stack associated with the deployment. You can find the stack in the CloudFormation service.
2. Click **Delete**.

If you do not want to delete the entire deployment but want to minimize the cost you can bring the number of instances in the Auto Scaling Group down to 0 and then scale it back up when the need arises.

See Also

More About

- “Manage MATLAB Production Server Using the Dashboard” on page 10-49
- “Architecture and Resources” on page 10-83

Dashboard Access Control

MATLAB Production Server lets server administrators use OpenID Connect (OIDC) identity providers such as Microsoft Azure Active Directory (Azure AD), Google Identity, PingFederate from Ping Identity, and others to configure role-based access control for the dashboard. Role-based access control allows administrators to grant a dashboard user the privileges to perform tasks on the dashboard based on their role.

Dashboard User Roles

The dashboard access control feature supports the following roles.

- Application author — Application authors can upload and delete applications (deployable archives) and view logs.
- Manager — Managers can edit server settings, configure access control for applications, manage persistence services, and have all the privileges of an application author, which include uploading and deleting applications and viewing logs.
- Server administrator — Administrators can log in to server virtual machines and configure which users or groups of users can access the dashboard, and have all the privileges of a manager, which include editing server logs, configuring access control for applications, uploading and deleting applications, and viewing logs.

The following table shows the dashboard tabs that users with these roles can access.

Role	Overview	Applications	Settings	Persistence	Manage Identity Providers	Application Access Control	Dashboard Access Control	Logs
Server administrator	✓	✓	✓	✓	✓	✓	✓	✓
Manager	✓	✓	✓	✓	✓	✓		✓
Application author	✓	✓						✓

Note Only the server administrator can log in to the dashboard when dashboard access control is disabled or not configured. Only the server administrator has privileges to configure dashboard access control.

Configure Identity Provider and Specify Access Control Policies

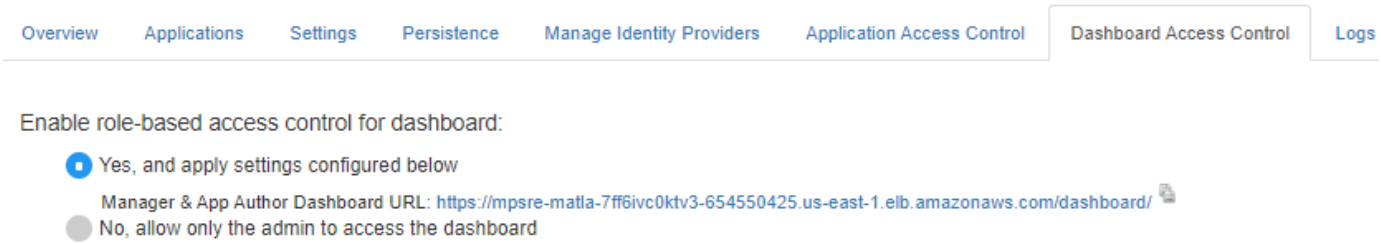
To enable dashboard access control for MATLAB Production Server, you must configure an identity provider and specify access control policies. The fields required to configure an identity provider vary based on the identity provider that you use. The access control policies define areas of the dashboard that users or groups of users can access and tasks that they can perform in these areas.

For information about configuring specific identity providers and policies, see:

- “Configure Dashboard Access Control Using Azure AD” on page 10-59
- “Configure Dashboard Access Control Using Google Identity” on page 10-62
- “Configure Dashboard Access Control Using PingFederate Identity Provider” on page 10-64
- “Configure Dashboard Access Control Using Other OpenID Connect Providers” on page 10-67

Enable Access Control

After you configure the identity provider and specify access control policies, you must enable dashboard access control. After enabling dashboard access control, a dashboard login URL that supports single sign-on (SSO) becomes available. Share this URL with managers and application authors.



See Also

More About

- “Manage MATLAB Production Server Using the Dashboard” on page 10-49
- “Application Access Control” on page 10-71

Configure Dashboard Access Control Using Azure AD

MATLAB Production Server administrators can use Microsoft Azure AD to configure role-based access control for the MATLAB Production Server Dashboard. Role-based access control allows administrators to grant access to specific areas of the dashboard to certain users or groups of users. For more information about the roles that the dashboard supports, see “Dashboard Access Control” on page 10-57.

To enable dashboard access control, configure Azure AD and specify access control policies, in consultation with the Azure AD administrator.

Configure Identity Provider

To configure Azure AD:

1. Log in to the dashboard to retrieve the redirect URI of the dashboard.
2. In the Azure portal, use the redirect URI to register the dashboard as a client application with the provider.
3. In the dashboard, enter values specific to the registered application and Azure AD.

Retrieve Redirect URI from Dashboard

To retrieve the redirect URI, start creating a configuration for Azure AD in the dashboard:

1. Navigate to either the **Dashboard Access Control** tab or the **Manage Identity Providers** tab.
2. Click **Create** and select **Azure AD**.
3. In **Create Identity Provider for Dashboard Access Control**, note the redirect URI of the dashboard.

   Later, you return to this view to specify the values required to configure your identify provider in the dashboard.

Register Application in Azure Portal

Use the Azure portal to register a web client application for dashboard access control. When registering the application, use the redirect URI from the MATLAB Production Server dashboard. Typically, the Azure AD administrator registers the application.

1. Sign in to the Azure portal.
2. From **Azure Active Directory**, select **App registrations** and click **New registration**.
3. In the resulting pane, enter the name of the application (for example, `MATLAB Production Server Dashboard App`).
4. For the **Redirect URI**, select **Web**. In the corresponding value field, enter the redirect URI of the dashboard and click **Register**. A web page displays the details of your registered application.
5. Click **Manifest** in the left navigation pane. In the JSON that is displayed in the resulting pane, set the value for `groupMembershipClaims` to `"SecurityGroup"`. Click **Save**.

For more information on how to register an application, see the Microsoft Azure documentation.

Specify Values in Dashboard

In the Azure portal, find the values of the client application that you registered and enter them into the dashboard.

1. Sign in to the Azure portal.
2. From **Azure Active Directory**, select **App registrations** and then select the application that you registered for the dashboard. Copy the value from the **Application (client) ID** and paste it into the **Client ID** field in the dashboard.
3. From **App registrations**, select **Certificates & secrets**. Under **Certificates & secrets**, create a new client secret or use an existing one. Copy the value for the client secret and paste it into the **Client Secret** field in the dashboard.
4. From **Azure Active Directory**, select **Properties**. Copy the value from **Directory (tenant) ID** and paste it into the **Tenant ID** in the dashboard.
5. On the dashboard, click **Create**.

Specify Dashboard Access Control Policy

Before you can specify dashboard access control policies, you must have users and groups set up in Azure AD. Consult the Azure AD administrator for this setup.

The access control policies define areas of the dashboard that users or groups of users can access and tasks that they can perform in these areas. Use the policies to assign the manager and application author roles to users or groups of users in your organization by entering their Azure user names and group IDs into the dashboard.

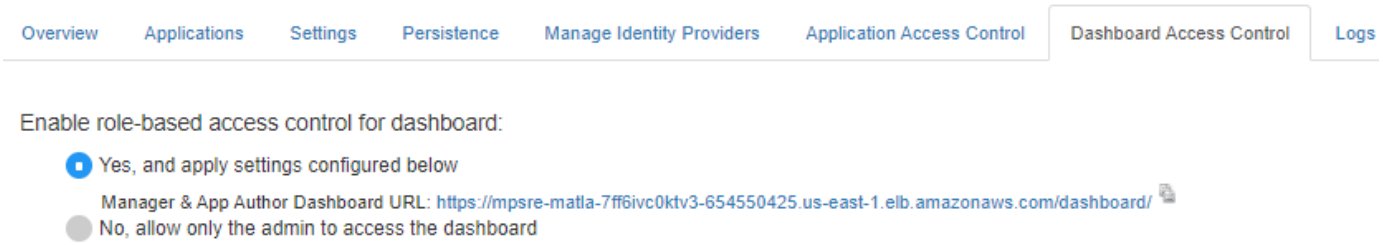
Configure Users and Groups in Dashboard

In the Azure portal, find user names and group IDs and enter them into the dashboard.

1. Sign in to the Azure portal.
2. From **Azure Active Directory** and then **Users**. Copy the values for the user names and paste them into the **Users** field in the dashboard. Use a comma to separate multiple user names.
3. From **Azure Active Directory** and then **Groups**. Copy the values for the object IDs and paste them into the **Groups** field in the dashboard. Use a comma to separate multiple object IDs names.
4. On the dashboard, click **Save**.

Enable Dashboard Access Control

After you configure Azure AD and specify access control policies, you must enable dashboard access control by selecting the **Yes** option. After enabling dashboard access control, a dashboard login URL that supports single sign-on (SSO) becomes available. Share this URL with managers and application authors.



See Also

Related Examples

- “Dashboard Access Control” on page 10-57
- “Configure Dashboard Access Control Using Google Identity” on page 9-66
- “Configure Dashboard Access Control Using PingFederate Identity Provider” on page 9-68
- “Configure Dashboard Access Control Using Other OpenID Connect Providers” on page 9-71

Configure Dashboard Access Control Using Google Identity

MATLAB Production Server administrators can use Google identity provider to configure role-based access control for the MATLAB Production Server Dashboard. Role-based access control allows administrators to grant access to specific areas of the dashboard to certain users or groups of users. For more information about the roles that the dashboard supports, see “Dashboard Access Control” on page 10-57.

To enable dashboard access control, configure the identity provider and specify access control policies, in consultation with the Google Identity administrator.

Configure Google Identity

To configure Google Identity:

1. Log in to the dashboard to retrieve the redirect URI of the dashboard.
2. On the Google Cloud Platform Console, use the redirect URI to register the dashboard as a client application with the provider.
3. In the dashboard, enter values specific to the registered application and Google Identity.

Retrieve Redirect URI from Dashboard

To retrieve the redirect URI, start creating a configuration for Google Identity in the dashboard:

1. Navigate to either the **Dashboard Access Control** tab or the **Manage Identity Providers** tab.
2. Click **Create** and select **Google**.
3. In **Create Identity Provider for Dashboard Access Control**, note the redirect URI of the dashboard.

   Later, you return to this view to specify the values required to configure your identify provider in the dashboard.

Register Application in Google Cloud Console

Use the Google Cloud Console to register a web client application for dashboard access control. Use the redirect URI from the MATLAB Production Server Dashboard when registering the application.

1. Sign in to the Google Cloud Platform Console and navigate to the **Credentials** page.
2. On the **Credentials** page, click **Create credentials** and select **OAuth client ID**.
3. From **Application type** drop down, select **Web Application**.
4. Enter the name of you client application (for example, `MATLAB Production Server Dashboard App`).
5. Under **Authorized redirect URIs**, click **Add URI**.
6. Copy the redirect URI from MATLAB Production Server Dashboard and paste it into the **URIs** field in the Google Cloud Console.
7. Click **Create**.
8. The Google identity provider creates an application with a client ID and client secret. Note the values of the client ID and client secret. You enter these values next in the dashboard.

Specify Client ID and Client Secret in Dashboard

1. Enter the noted client ID and client secret values from the previous section in the **Client ID** and **Client Secret** fields respectively in MATLAB Production Server dashboard.
2. Click **Create** to complete the configuration of the identity provider.

Specify Dashboard Access Control Policy

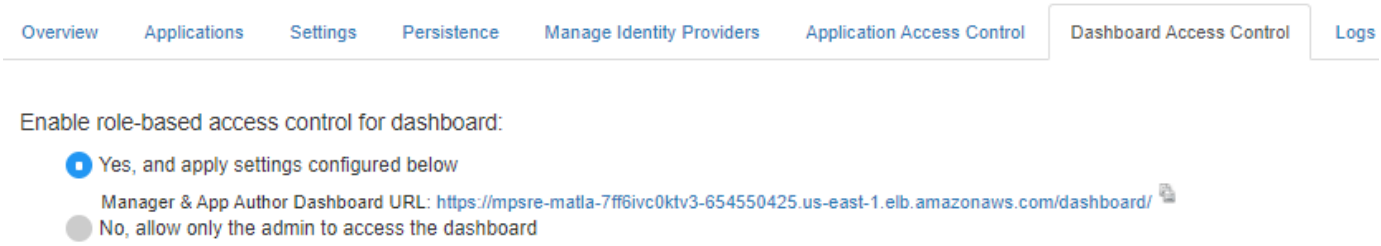
Before you can specify dashboard access control policies, you must have users, and groups, if applicable, set up in Google. Consult the Google identity provider administrator for this setup.

The access control policies define areas of the dashboard that users can access and tasks that they can perform in these areas. Use the policies to assign the manager and application author roles to users in your organization by entering their Google user names.

1. On the **Dashboard Access Control** tab of the dashboard, select Google as the identity provider.
2. In the **Dashboard Access Control Policy** section, enter Google user names to assign manager and application author roles to users in your organization. Click **Save** after you enter the values.

Enable Dashboard Access Control

After you configure Google Identity and specify access control policies, you must enable dashboard access control by selecting the **Yes** option. After enabling dashboard access control, a dashboard login URL that supports single sign-on (SSO) becomes available. Share this URL with managers and application authors.



See Also

Related Examples

- “Dashboard Access Control” on page 10-57
- “Configure Dashboard Access Control Using Azure AD” on page 9-63
- “Configure Dashboard Access Control Using PingFederate Identity Provider” on page 9-68
- “Configure Dashboard Access Control Using Other OpenID Connect Providers” on page 9-71

Configure Dashboard Access Control Using PingFederate Identity Provider

MATLAB Production Server administrators can use PingFederate from Ping Identity to configure role-based access control for the MATLAB Production Server Dashboard. Role-based access control allows administrators to grant access to specific areas of the dashboard to certain users or groups of users. For more information about the roles that the dashboard supports, see “Dashboard Access Control” on page 10-57.

To enable dashboard access control for MATLAB Production Server, configure PingFederate and specify access control policies, in consultation with the PingFederate administrator.

Prerequisites

Refer to the PingFederate documentation to configure OAuth use cases, clients, and endpoints to configure OpenID provider information:

- Configuring OAuth Use Cases
- Configuring OAuth Clients
- OAuth 2.0 Endpoints
- Configuring OpenID Provider Information

Configure PingFederate Identity Provider

To configure PingFederate:

1. Log in to the dashboard to retrieve the Redirect URI of the dashboard.
2. Use the Redirect URI to register a client application in PingFederate.
3. In the dashboard, enter values specific to the registered application and PingFederate.

Retrieve Redirect URI from Dashboard

To retrieve the redirect URI, start creating a configuration for your identity provider in the dashboard:

1. Navigate to either the **Dashboard Access Control** tab or the **Manage Identity Providers** tab.
2. Click **Create** and select **PingFederate**.
3. In **Create Identity Provider for Dashboard Access Control**, note the redirect URI of the dashboard.

   Later, you return to this view to specify the values required to configure your identify provider in the dashboard.

Register Application in PingFederate

Register an application in PingFederate for MATLAB Production Server Dashboard, if you do not already have one. Consult the PingFederate administrator to register the application. Provide the **Redirect URI** of the MATLAB Production Server Dashboard when registering the application.

Specify Values in Dashboard

After you register the application with PingFederate, you receive application specific values such as the **Client ID** and **Client Secret**. Enter the values specific to the application and values specific to PingFederate in the dashboard under **Create Identity Provider for Dashboard Access Control**.

The following table describes the values that you must enter. Click **Create** after you enter the values.

Field	Description
Client ID	Application ID of the registered client application.
Client Secret	Client secret of the registered client application.
OIDC Issuer	Discovery endpoint URI of the OIDC provider.
JWT Issuer	JWT issuer metadata of the OIDC provider.
JWKS URI	URI to retrieve the JSON Web Key Set (JWKS).

Specify Dashboard Access Control Policy

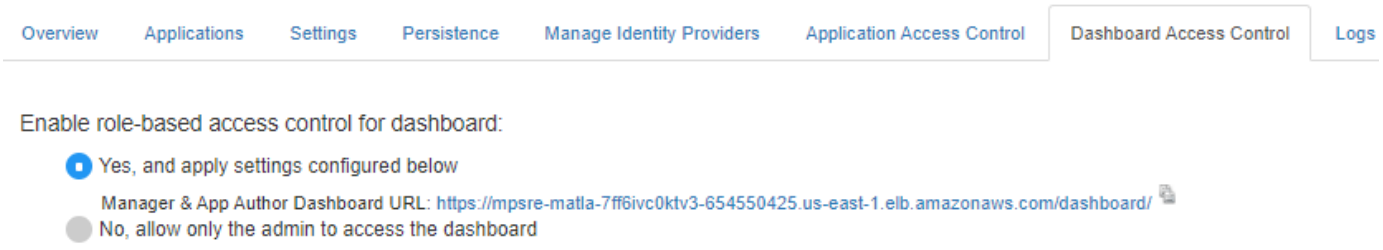
Before you can specify dashboard access control policies, you must have users, and groups, if applicable, set up in PingFederate. Consult the PingFederate administrator for this setup.

The access control policies define areas of the dashboard that users or groups of users can access and tasks that they can perform in these areas. Use the policies to assign the manager and application author roles to users or groups of users in your organization by entering their user names and group IDs. Click **Save** after you enter the values.

1. In the **Dashboard Access Control** tab of the dashboard, select PingFederate as the identity provider.
2. In the **Dashboard Access Control Policy** section, enter identity provider specific user names and group IDs to assign manager and application author roles to users or groups of users in your organization. Use a comma to separate multiple user names and group IDs. Click **Save** after you enter the values.

Enable Dashboard Access Control

After you configure PingFederate and specify access control policies, you must enable dashboard access control by selecting the **Yes** option. After enabling dashboard access control, a dashboard login URL that supports single sign-on (SSO) becomes available. Share this URL with managers and application authors.



See Also

Related Examples

- “Dashboard Access Control” on page 10-57
- “Configure Dashboard Access Control Using Other OpenID Connect Providers” on page 9-71
- “Configure Dashboard Access Control Using Azure AD” on page 9-63
- “Configure Dashboard Access Control Using Google Identity” on page 9-66

Configure Dashboard Access Control Using Other OpenID Connect Providers

MATLAB Production Server Dashboard supports the use of any OpenID Connect (OIDC) identity provider for role-based access control for the dashboard. Role-based access control allows server administrators to grant access to specific areas of the dashboard to certain users or groups of users. For more information about the roles that the dashboard supports, see “Dashboard Access Control” on page 10-57.

To enable dashboard access control, configure the OIDC provider and specify dashboard access control policies, in consultation with the OIDC provider administrator.

Configure Identity Provider

To configure an identity provider:

1. Log in to the dashboard to retrieve the redirect URI of the dashboard.
2. At the identity provider's website, use the redirect URI to register the dashboard as a client application with the provider.
3. In the dashboard, enter values specific to the registered application and identity provider.

Retrieve Redirect URI from Dashboard

To retrieve the redirect URI, start creating a configuration for your identity provider in the dashboard:

1. Navigate to either the **Dashboard Access Control** tab or the **Manage Identity Providers** tab.
2. Click **Create** and select **Other**.
3. In **Create Identity Provider for Dashboard Access Control**, note the redirect URI of the dashboard.

   Later, you return to this view to specify the values required to configure your identify provider in the dashboard.

Register Application with Identity Provider

Register an application with the OIDC provider for MATLAB Production Server Dashboard. Consult the OIDC provider administrator to register the application. When registering the application, provide the redirect URI of the MATLAB Production Server Dashboard.

Specify Values in Dashboard

After you register the application with the identity provider, you receive application-specific values such as the client ID and client secret. Enter the values in the dashboard under **Create Identity Provider for Dashboard Access Control**.

The following table describes the values that you must enter. Click **Create** after you enter the values.

Field	Description
Client ID	Application ID of the registered client application

Field	Description
Client Secret	Client secret of the registered client application
OIDC Issuer	Discovery endpoint URI of the OIDC provider
JWT Issuer	JWT issuer metadata of the OIDC provider
JWKS URI	URI to retrieve the JSON Web Key Set (JWKS)

Under **Create Identity Provider for Dashboard Access Control**, you have the option to provide values other than the defaults for **UserAttribute ID** and **GroupAttribute ID**. **UserAttribute ID** is the JWT claim name that uniquely identifies a user. **GroupAttribute ID** is the JWT claim name that lists the groups that a user belongs to. Depending on the identity provider you use, you might have to change the defaults.

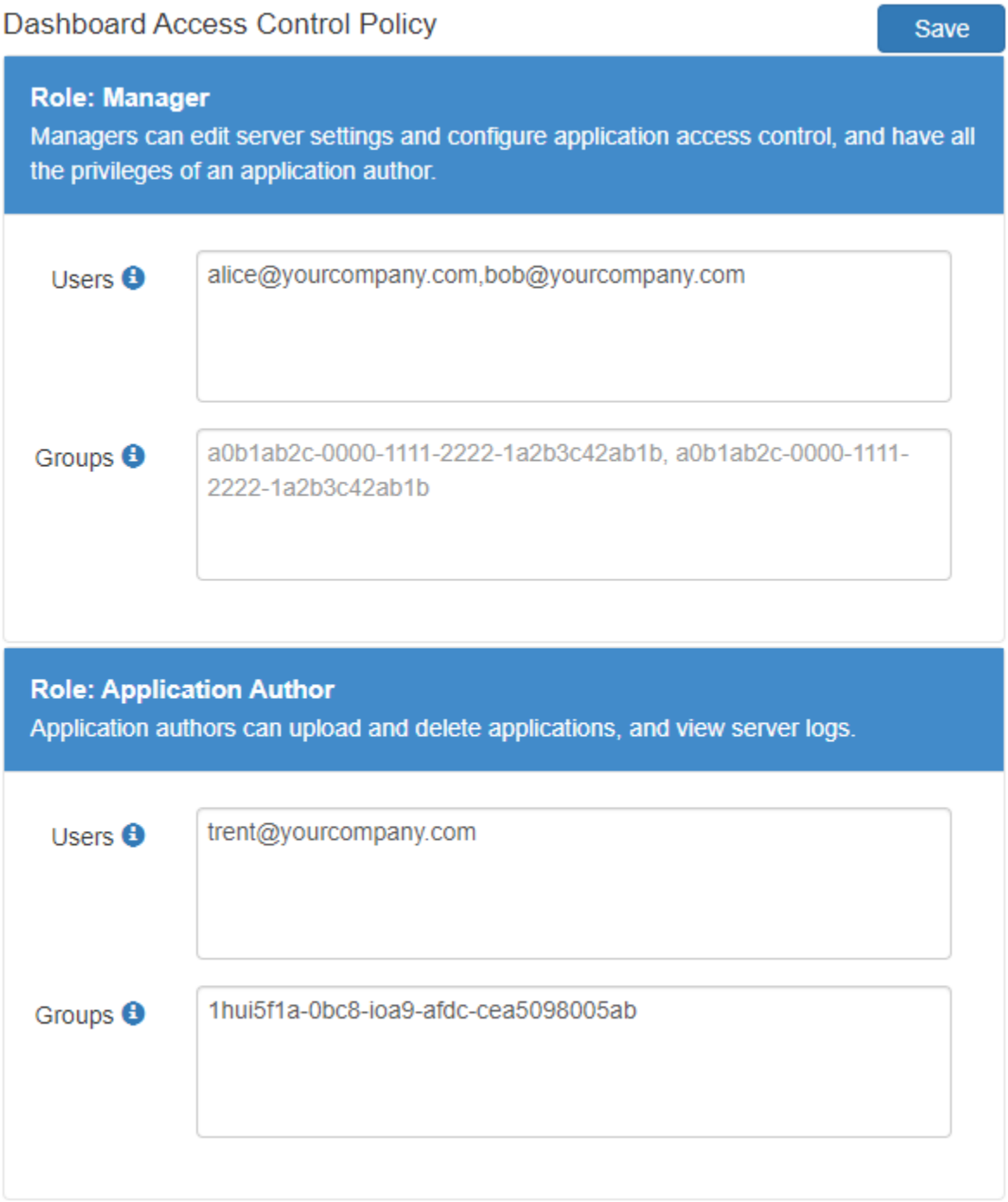
Specify Dashboard Access Control Policy

Before you can specify dashboard access control policies, you must have users, and if applicable, groups, set up in the identity provider. Consult the OIDC provider administrator for this setup.

The access control policies define areas of the dashboard that users or groups of users can access and tasks that they can perform in these areas.

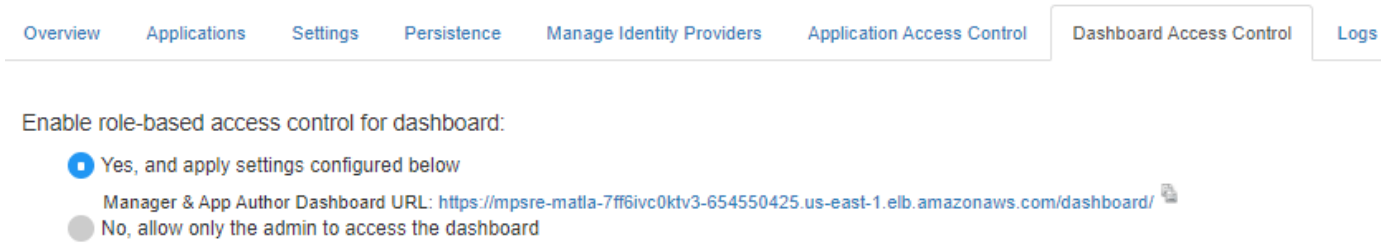
1. On the **Dashboard Access Control** tab of the dashboard, select the identity provider that you want to use.
2. In the **Dashboard Access Control Policy** section, enter identity provider specific user names and group IDs to assign manager and application author roles to users or groups of users in your organization. Use a comma to separate multiple user names or group IDs. Click **Save** after you enter the values.

   For example, in the following illustration, the users *alice@yourcompany.com* and *bob@yourcompany.com* have the manager role. The user *trent@yourcompany.com* and all users that belong to group ID *1hui5f1a-0bc8-ioa9-afdc-cea5098005ab* have the application author role.



Enable Dashboard Access Control

After you configure the identity provider and specify access control policies, you must enable dashboard access control by selecting the **Yes** option. After enabling dashboard access control, a dashboard login URL that supports single sign-on (SSO) becomes available. Share this URL with managers and application authors.



See Also

Related Examples

- “Dashboard Access Control” on page 10-57
- “Configure Dashboard Access Control Using Azure AD” on page 9-63
- “Configure Dashboard Access Control Using Google Identity” on page 9-66
- “Configure Dashboard Access Control Using PingFederate Identity Provider” on page 9-68

Application Access Control

MATLAB Production Server integrates with OAuth 2.0 providers such as Microsoft Azure Active Directory (Azure AD), Google Identity, and PingFederate from Ping Identity, and uses JSON Web Tokens (JWTs) for application access control. Application access control lets server administrators restrict user access to applications or archives deployed to the server.

Note Application access control is only available for clients that make server requests using the MATLAB Production Server RESTful API.

All users can access all applications by default.

To enable access control, configure the identity provider and define access control policy rules in the **Application Access Control** tab of the MATLAB Production Server dashboard. Use the access control policy rules to specify which users and groups of users have permission to execute deployed applications. After you enable access control, clients can generate a bearer access token that they must send with every server request. The server uses the bearer token to verify the identify of a client.

You must log in to the dashboard as an administrator or manager to configure application access control. For more information about the dashboard user roles, see “Dashboard Access Control” on page 10-57.

Configure Identity Provider and Specify Access Control Policy Rules

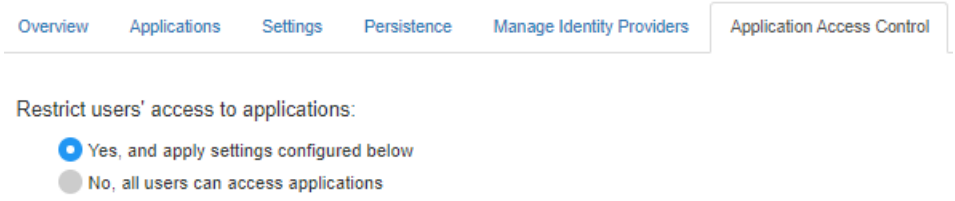
To configure an identity provider, register an application with the identity provider. Then, specify application-specific values and access control policy rules in the dashboard. The fields required to configure an identity provider vary based on the identity provider that you use.

For information about configuring specific identity providers and rules, see:

- “Configure Application Access Control Using Azure AD” on page 10-73
- “Configure Application Access Control Using Google Identity” on page 10-77
- “Configure Application Access Control Using PingFederate” on page 10-79
- “Configure Application Access Control Using Other Identity Providers” on page 10-81

Enable Application Access Control

After you configure the identity provider and specify access control policy rules, you must enable application access control from the dashboard.



Generate Access Token

After you enable application access control, clients can generate a bearer token. Client programs can use third-party libraries for token generation. For a list of OAuth libraries, see OAuth libraries. Client programs use this bearer token in the HTTP authorization header when making a request to the server using the MATLAB Production Server RESTful API. The format for this header is `Authorization:Bearer <access token>`.

See Also

More About

- “Manage AWS Resources for MATLAB Production Server” on page 10-54
- “Manage MATLAB Production Server Using the Dashboard” on page 10-49
- “Dashboard Access Control” on page 10-57
- “RESTful API for MATLAB Function Execution”

Configure Application Access Control Using Azure AD

MATLAB Production Server administrators can use Microsoft Azure AD to restrict access to deployed applications to only certain users or groups of users. To enable application access control, configure Azure AD and specify access control policies, in consultation with the Azure AD administrator.

Register Application in Azure Portal

To use Azure AD for application access control, register a server application and a client application in the Azure portal. These applications are different from the application that you might have registered for dashboard access control. These applications are not related to the applications deployed to MATLAB Production Server or client applications written using the MATLAB Production Server client libraries.

Note The application registration process is determined by Azure and is subject to change.

Register Server Application in Azure

1. Sign in to the Azure portal.
2. From **Azure Active Directory**, select **App registrations** and click **New registration**.
3. In the resulting pane, enter the name of the application (for example, `MATLAB Production Server App`) then select **Register**.
4. In the application that you registered, select **Expose an API**.
5. Click **Add a scope**, and enter the scope information for your application. Click **Add Scope**. For more information on adding a scope, see the Microsoft Azure documentation. The following table lists the fields and values that you enter to add a scope.

Field	Value
Scope name	Enter a name, for example, `user_impersonation`.
Who can consent	Select `Admin and users`.
Admin consent display name	Enter a name, for example, `Access MATLAB Production Server App`.
Admin consent description	Enter a description, for example, `Allow the application to access MATLAB Production Server App on behalf of the signed-in user`.
User consent display name	Enter a name, for example, `Access MATLAB Production Server App`.
User consent description	Enter a description, for example, `Allow the application to access MATLAB Production Server App on behalf of the signed-in user`.
State	Select `Enabled`.

6. Click **Manifest** in the left navigation pane. In the JSON that is displayed, set the value for `groupMembershipClaims` to `"SecurityGroup"`. Click **Save**.

Register Client Application in Azure

In the Azure portal, register a client application. The client application helps clients that send requests to the server to generate an access token. You can register the client application as either a native app or a web app. If you register the client application as a native app, users have to log in using a user name and password to generate the access token. If you register the client application as a web app, users have to log in using the browser with single sign-on to generate the access token.

Registering client applications can require higher privileges in Azure based on your organization setup.

Register Client Application as Native Client

1. Sign in to the Azure portal.
2. From **Azure Active Directory**, select **App registrations** and click **New registration**.
3. In the pane that opens, enter the following registration information for your application, then click **Register**.

Field	Value
Name	Enter a name, for example, `MATLAB Production Server Native Client`.
Redirect URI	Select `Public client/native (mobile & desktop)`.

4. Click **Manifest** in the left navigation pane. In the JSON, set the value for `allowPublicClient` to `true`. Click **Save**.
5. Click **API permissions** and click **Add a permission**.
6. In the pane that opens, click **APIs my organization uses**.
7. Search for the `MATLAB Production Server App` server application that you registered earlier. In the pane that opens, select the scope name (for example, `user_impersonation`) and click **Add permissions.**

Register Client Application as Web Client

1. Sign in to the Azure portal.
2. From **Azure Active Directory**, select **App registrations** and click **New registration**.
3. In the pane that opens, enter the following registration information for your application, then click **Register**.

Field	Value
Name	Enter a name, for example, `MATLAB Production Server Web Client`.
Redirect URI	Select `Web`. Enter a valid redirect URI that will be used by your client application

4. Select **Certificates & secrets** in the left navigation pane. Under **Client secrets**, create a new client secret, and save the value of the secret.
5. Click **API permissions**, then click **Add a permission** and select **APIs my organization uses**.
6. Search for the `MATLAB Production Server App` server application that you registered earlier. In the pane that opens, select the scope name, for example, `user_impersonation`, then click **Add permissions.**

Configure Identity Provider

After you register the server application and client application in the Azure portal, create a configuration for Azure AD in the **Application Access Control** tab of the dashboard. Click **Create** and select **Azure AD**.

In the Azure portal, find the tenant ID for your organization, and the application ID for the server application that you registered earlier. Enter the tenant ID and application ID in the dashboard under **Create Identity Provider for Application Access Control**.

1. Sign in to the Azure portal.
2. From **Azure Active Directory**, select **Properties**. Copy the value from **Directory (tenant) ID** and paste it into **Tenant ID** field in the dashboard.
3. From **Azure Active Directory**, select **App registrations**. Select the application used for MATLAB Production Server, for example, `MATLAB Production Server App`. Copy the value from **Application (client) ID** and paste it into the **Server App ID** field in the dashboard.
4. In the dashboard, click **Create**. If the server is running on a Windows virtual machine, saving the values can take up to 30 seconds.

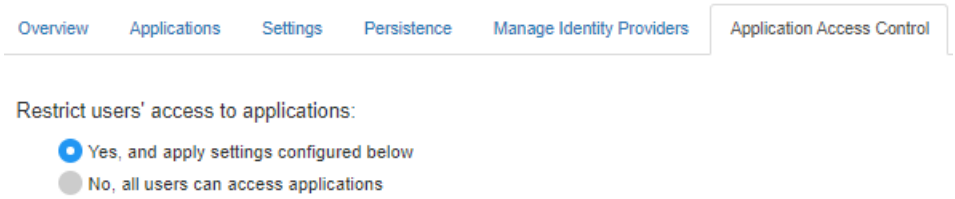
Specify Access Control Policy Rules

Specify the applications that certain user groups can access by defining access control policy rules. To define the rule, click **Add Rule** under **Access Control Policy** in the **Application Access Control** tab of the dashboard. Specify the following values.

Field	Value
Rule ID	Name for the rule
Description	Description for your rule
Users	User names set up in Azure AD that are allowed access to deployed applications
Groups	Object IDs of the groups set up in Azure AD groups that are allowed access to deployed applications
Applications	Applications that the specified users and groups can access. To select all applications, select `Apply this rule to all applications`.

Enable Application Access Control

After you configure the identity provider and specify access control policy rules, you must enable dashboard access control by selecting the **Yes** option from the dashboard.



Generate Access Token

After application access control is enabled, users that are specified in the access control policy rules can generate a bearer access token. If the registered client application is a native app, log in using a user name and password, or integrated Windows authentication to generate the access token. If the registered client application is a web app, log in using the browser with single sign-on to generate the access token. You can use the Microsoft identity platform authentication libraries (Microsoft-supported client libraries or compatible client libraries in different programming languages) to generate the access token. For more information, see Microsoft documentation. Use this access token in the HTTP authorization header when you make a request to the server using the MATLAB Production Server RESTful API. The format for this header is `Authorization:Bearer <access token>`.

See Also

Related Examples

- “Application Access Control” on page 10-71
- “Configure Application Access Control Using Google Identity” on page 10-77
- “Configure Application Access Control Using PingFederate” on page 10-79
- “Configure Application Access Control Using Other Identity Providers” on page 10-81

Configure Application Access Control Using Google Identity

MATLAB Production Server administrators can use Google Identity to restrict access to deployed applications to only certain users or groups of users. To enable application access control, configure Google Identity and specify access control policies, in consultation with the Google Identity administrator.

Register Application in Google Cloud Platform Console

To use Google Identity for application access control, register an application in the Google Cloud Platform Console. For more information about registering an application, see Google Identity documentation.

Configure Identity Provider in Dashboard

After you register the application in Google, create a configuration for Google Identity in the **Application Access Control** tab of the dashboard. Click **Create** and select **Google**. In **Create Identity Provider for Application Access Control**, enter application-specific and identity provider-specific values. Click **Create**. If the server is running on a Windows virtual machine, saving the values can take up to 30 seconds.

The following table describes the values that you must enter.

Field	Value
Name	Name for your Google identity provider configuration
App ID	Client ID of the application registered in Google for application access control

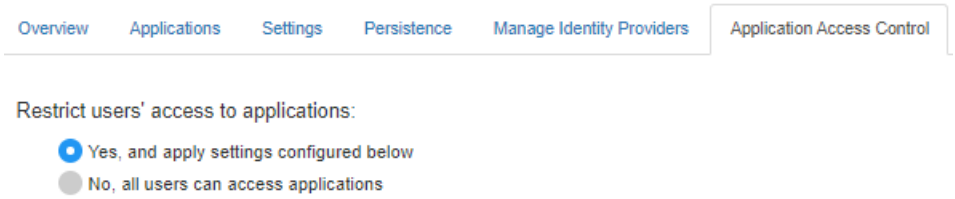
Specify Access Control Policy Rules

Specify the applications that certain users or user groups can access by defining access control policy rules. To define a rule, click **Add Rule** under **Access Control Policy** in the **Application Access Control** tab of the dashboard. Specify the following values.

Field	Value
Rule ID	Name for the rule
Description	Description for the rule
Users	Google user names that are allowed access to deployed applications
Groups	Google group IDs, if applicable, that are allowed access to deployed applications
Applications	Applications that the specified users and groups have permission to access Select `Apply this rule to all applications` to select all applications.

Enable Application Access Control

After you configure the identity provider and specify access control policy rules, you must enable dashboard access control by selecting the **Yes** option from the dashboard.



Generate Access Token

After application access control is enabled, users that are specified in the access control policy rules can generate a bearer access token. For more information about generating an access token, see the Google documentation for Using OAuth 2.0 for Web Server Applications.

Client programs use this access token in the HTTP authorization header when making a request to the server using the MATLAB Production Server RESTful API. The format for this header is `Authorization:Bearer <access token>`.

See Also

Related Examples

- “Application Access Control” on page 10-71
- “Configure Application Access Control Using Azure AD” on page 10-73
- “Configure Application Access Control Using PingFederate” on page 10-79
- “Configure Application Access Control Using Other Identity Providers” on page 10-81

Configure Application Access Control Using PingFederate

MATLAB Production Server administrators can use PingFederate from Ping Identity to restrict access to deployed applications to only certain users or groups of users. To enable application access control, configure PingFederate and specify access control policy rules, in consultation with the PingFederate administrator.

Prerequisites

Refer to the PingFederate documentation to configure OAuth use cases, clients and endpoints, and to configure OpenID provider information:

- Configuring OAuth Use Cases
- Configuring OAuth Clients
- OAuth 2.0 Endpoints
- Configuring OpenID Provider Information

Configure PingFederate in Dashboard

1. After you register the application with PingFederate, create a configuration for PingFederate in the **Application Access Control** tab of the dashboard. Click **Create** and select **PingFederate**.
2. In **Create Identity Provider for Application Access Control**, enter application-specific and identity provider-specific values. Click **Create**. If the server is running on a Windows virtual machine, saving the values can take up to 30 seconds.

   The following table describes the values that you must enter.

Field	Value
Name	Name for your PingFederate configuration.
App ID	Intended recipient of the JWT. The recipient helps in validating the *aud* claim in the JWT.
JWT Issuer	JWT issuer metadata of the identity provider. The metadata string must match the *iss* claim in the JWT.
JWKS URI	URI to retrieve the JSON Web Key Set (JWKS).

Specify Access Control Policy Rules

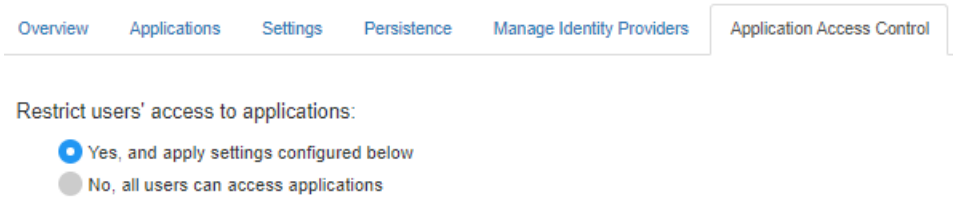
Specify the applications that certain users or user groups can access by defining access control policy rules. To define a rule, click **Add Rule** under **Access Control Policy** in the **Application Access Control** tab of the dashboard. Then, specify the following information.

Field	Value
Rule ID	Name for the rule
Description	Description for your rule

Field	Value
Users	User names that are allowed access to deployed applications
Groups	Group IDs, if applicable, that are allowed access to deployed applications
Applications	Applications that you want to allow the specified groups of users to access. To select all applications, select `Apply this rule to all applications`.

Enable Application Access Control

After you configure the identity provider and specify access control policy rules, you must enable dashboard access control by selecting the **Yes** option from the dashboard.



Generate Access Token

After application access control is enabled, users that are specified in the access control policy rules can generate a bearer access token. For more information about generating an access token, see the PingFederate OAuth 2.0 Developer Guide.

Clients programs use this access token in the HTTP authorization header when making a request to the server using the MATLAB Production Server RESTful API. The format for this header is `Authorization:Bearer <access token>`.

See Also

Related Examples

- “Application Access Control” on page 10-71
- “Configure Application Access Control Using Azure AD” on page 10-73
- “Configure Application Access Control Using Google Identity” on page 10-77
- “Configure Application Access Control Using Other Identity Providers” on page 10-81

Configure Application Access Control Using Other Identity Providers

MATLAB Production Server integrates with several OAuth 2.0 providers for application access control. Application access control lets server administrators restrict access to deployed applications to only certain users or groups of users. To enable application access control, configure an identity provider and specify access control policy rules, in consultation with the OAuth 2.0 provider administrator.

Register Application With Identity Provider

To use an identity provider for application access control, register an application with the identity provider. Consult the identity provider administrator to register the application.

Configure Identity Provider in Dashboard

After you register the application with the identity provider, create a configuration for the identity provider in the **Application Access Control** tab of the dashboard. Click **Create** and select **Other**. In **Create Identity Provider for Application Access Control**, enter application-specific and identity provider-specific values. Click **Create**. If the server is running on a Windows virtual machine, saving the values can take up to 30 seconds.

The following table describes the values that you must enter.

Field	Value
Name	Name for your identity provider.
App ID	Intended recipient of the JWT. The recipient helps in validating the *aud* claim in the JWT.
JWT Issuer	JWT issuer metadata of the identity provider. The metadata string must match the *iss* claim in the JWT.
JWKS URI	URI to retrieve the JSON Web Key Set (JWKS).

Under **Create Identity Provider for Application Access Control**, you have the option to provide values other than the defaults for **UserAttribute ID** and **GroupAttribute ID**. **UserAttribute ID** is the JWT claim name that uniquely identifies a user. **GroupAttribute ID** is the JWT claim name that lists the groups that a user belongs to. Depending on the identity provider you use, you might have to change the defaults.

Specify Access Control Policy Rules

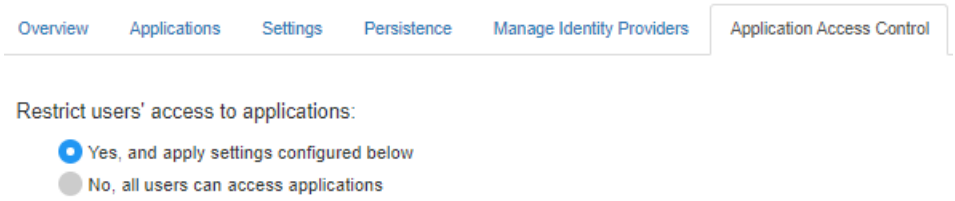
Specify the applications that certain users or user groups can access by defining access control policy rules. To define a rule, click **Add Rule** under **Access Control Policy** in the **Application Access Control** tab of the dashboard. Then, specify the following information.

Field	Value
Rule ID	Name for the rule.

Field	Value
Description	Description for your rule.
Users	User names set up in the identity provider that are allowed access to deployed applications.
Groups	Group IDs set up in the identity provider that are allowed access to deployed applications.
Applications	Applications that the specified users and groups can access. To select all applications, select `Apply this rule to all applications`.

Enable Application Access Control

After you configure the identity provider and specify access control policy rules, you must enable dashboard access control by selecting the **Yes** option from the dashboard.



Generate Access Token

After you enable application access control, clients can generate a bearer token. Client programs can use third-part libraries for token generation. For a list of OAuth libraries, see OAuth libraries. Client programs use this bearer token in the HTTP authorization header when making a request to the server using the MATLAB Production Server RESTful API. The format for this header is `Authorization:Bearer <access token>`.

See Also

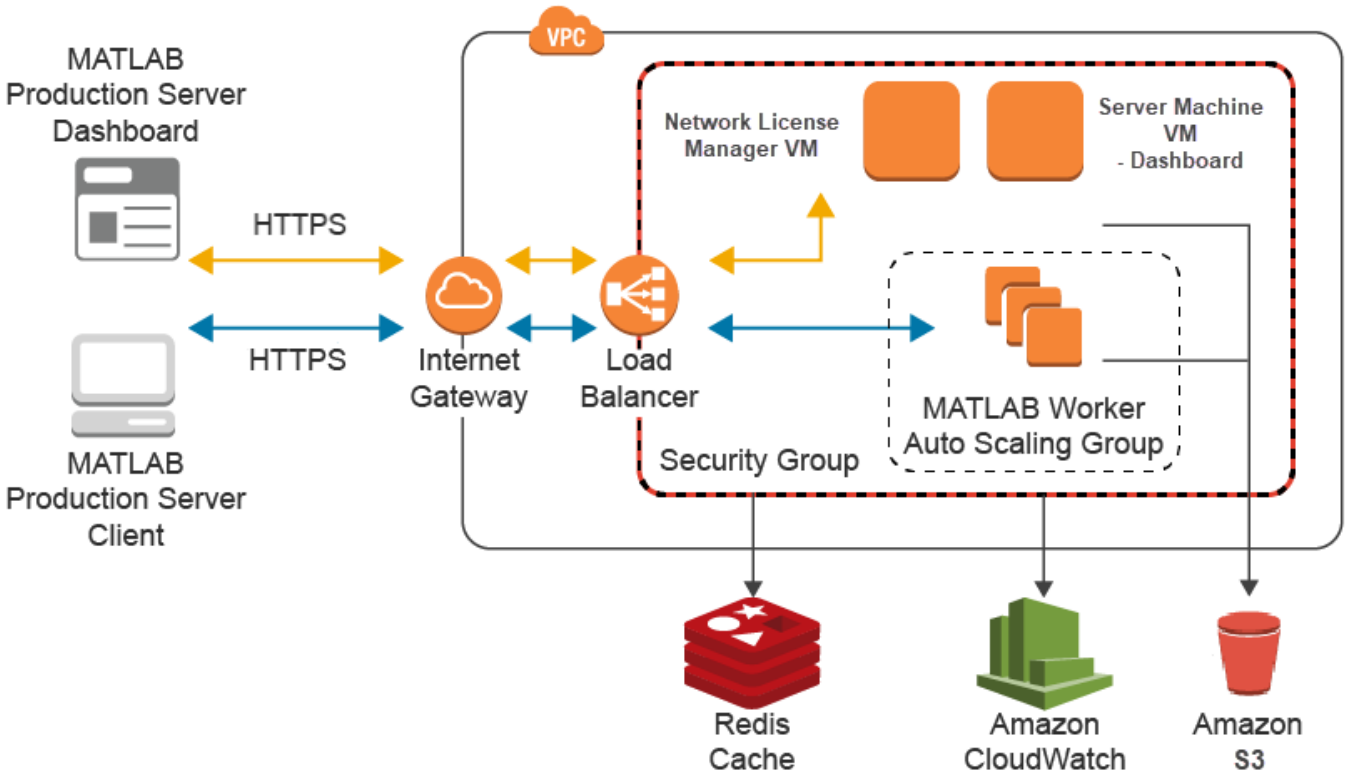
Related Examples

- “Application Access Control” on page 10-71
- “Configure Application Access Control Using Azure AD” on page 10-73
- “Configure Application Access Control Using Google Identity” on page 10-77
- “Configure Application Access Control Using PingFederate” on page 10-79

Architecture and Resources

Deploying MATLAB Production Server on AWS creates several resources in your resource group. The following sections describe the architecture of MATLAB Production Server and the AWS resources that the deployment provisions.

MATLAB Production Server Architecture on AWS



AWS Resources

The MATLAB Production Server deployment in AWS creates the following resources in your resource group.

Resource Type	Description
Amazon EC2 Instance	Virtual machine (VM) that hosts the MATLAB Production Server dashboard. Use the dashboard to: • Get the HTTPS endpoint to make requests to the server. • Upload applications (CTF files) to the server. • Manage server configurations. • Configure access control for the dashboard and applications. For more information about the dashboard, see “Manage MATLAB Production Server Using the Dashboard” on page 10-49.
Auto Scaling Group	Manages the number of identical VMs for hosting MATLAB Production Server instances. Each VM runs an instance of MATLAB Production Server that in turn runs multiple MATLAB Production Server workers. For information on how to change the number of VMs, see “Change Number of Virtual Machines” on page 10-54.
Load balancer	The deployment uses two elastic load balancers: **1** Load balancer for routing traffic to MATLAB Production Server instances. Clients use this endpoint for making requests to the server. **2** Load balancer for routing traffic to MATLAB Production Server dashboard. The dashboard retrieves the HTTPS endpoint for making requests to the server from the load balancer resource.
Amazon S3	Storage account that stores applications (CTF files) created by MATLAB Compiler SDK.
Amazon Virtual Private Cloud	Virtual network that consists of the deployed resources.

Resource Type	Description
Amazon ElastiCache for Redis	Amazon ElastiCache for Redis enables caching of data between calls to MATLAB code running on a server instance. For information on connecting to the deployed Redis cache, see “Use Amazon ElastiCache for Redis for Data Persistence” on page 10-52. For information on using a data cache, see “Data Caching Basics”.
Amazon CloudWatch	Enables viewing of logs. For information on how to view the logs, see “View Logs” on page 10-55.

See Also

More About

- “Manage AWS Resources for MATLAB Production Server” on page 10-54
- “Manage MATLAB Production Server Using the Dashboard” on page 10-49

Execute MATLAB Functions on MATLAB Production Server

Client applications for MATLAB Production Server are different from those for on-premises server instances in several ways. To execute MATLAB functions deployed on MATLAB Production Server, you must use the MATLAB execution endpoint URL specified in the dashboard. Depending on the implementation of your client program, you might have to update your code to use the sticky session or session affinity feature of the elastic load balancer.

Similar to client applications for on-premise server installations, you must use the MATLAB Production Server client libraries for client applications that you write using Java, .NET, C, and Python.

Use MATLAB Execution Endpoint URL

After your MATLAB Production Server deployment to AWS is complete, log in to the dashboard to retrieve the MATLAB execution endpoint. The **Overview** tab in the dashboard specifies the **MATLAB Execution Endpoint**. For information on accessing the dashboard, see “Manage MATLAB Production Server Using the Dashboard” on page 10-49.

This endpoint is an HTTPS URL that client programs use to make requests to the server and execute MATLAB functions deployed to the server. For example, if the MATLAB execution endpoint for your server is `https://payg-mps-1n8k9DNJ8NJAH-1476423457.us-east-1.elb.amazonaws.com`, to use the MATLAB Production Server RESTful API to execute a MATLAB function `mymagic` located in a deployed application `myapp`, specify the URL `https://payg-mps-1n8k9DNJ8NJAH-1476423457.us-east-1.elb.amazonaws.com/myapp/mymagic`.

Download Client Libraries

If you want to write client programs in Java, .NET, C, and Python for invoking MATLAB functions deployed on the server, you must use the MATLAB Production Server client libraries. Download the client libraries from MATLAB Production Server Client Libraries.

Work with Self-Signed SSL Certificate

If your deployment uses a self-signed SSL certificate, you might have to manually override the default security behavior of the client application or add a new HTTPS endpoint to the application gateway.

You can update the client application to disable hostname verification when sending HTTPS requests to the server to avoid encountering an exception caused by a failure in host name verification. Depending on the implementation of your client program, you might also have to retrieve the self-signed certificate that the load balancer uses and add the certificate to your local truststore. For more information on configuring the client environment, see “Handle Exceptions” for a Java client and “Handle Exceptions” for a .NET client.

Manage HTTP Cookie

The elastic load balancer uses sticky sessions, where it uses cookies to keep a user session on the same server. On receiving a request from a client program, the application gateway sets the `Set-Cookie` HTTP response header with information about the server virtual machine (VM) that processes the request.

Asynchronous Request Execution

A client program that uses asynchronous requests to execute a MATLAB function deployed to the server must set the `Cookie` HTTP request header with the value of the `Set-Cookie` header for all subsequent requests. This ensures that same server VM that processes the first request processes all subsequent requests for that session.

Synchronous Request Execution

A client program that uses synchronous requests to execute a MATLAB function deployed to the server must not set the `Cookie` HTTP request header with the value of the `Set-Cookie` header, and must clear the value of the `Cookie` header if it has been previously set. This ensures that the synchronous requests are load balanced and the same server VM does not process them.

The default property in a Java client that uses `MWHttpClient` sets the HTTP cookie. For information about disabling cookies, see “Configure Client-Server Connection”. The Java client API that uses protobuf and the .NET client API do not set the HTTP cookie by default.

See Also

More About

- “Manage MATLAB Production Server Using the Dashboard” on page 10-49
- “Manage AWS Resources for MATLAB Production Server” on page 10-54
- “RESTful API for MATLAB Function Execution”