

MATLAB® Production Server™

Server Management Guide



MATLAB®

R2020a



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 2012–2020 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)

Server Overview	1-2
What Is a Server?	1-2
How Does a Server Manage Work?	1-2
Create a Server	1-4
Prerequisites	1-4
Procedure	1-4
Edit the Configuration File	1-6
About the Server Configuration File	1-6
Common Customizations	1-6
Specify the Default MATLAB Runtime for New Server Instances	1-8
Run mps-setup in Non-Interactive Mode for Silent Install	1-8
Specify the MATLAB Runtime for a Server Instance	1-9
Start a Server Instance	1-10
Prerequisites	1-10
Procedure	1-10
Share the Deployable Archive	1-11
Support Multiple MATLAB Versions	1-12
How the Server Instance Selects the MATLAB Runtime to Use	1-12
Changes to Worker Management	1-13
Control Worker Restarts	1-14
Restart Workers Based on Up Time	1-14
Restart Workers Based on Amount of Memory in Use	1-14
Install a Server Instance as a Windows Service	1-16
Create a New Server Instance as a Windows Service	1-16
Make an Existing Server Instance a Windows Service	1-16
Recovery Options for a Server Instance Running as a Windows Service	1-17

Manage Licenses for MATLAB Production Server

2

Specify or Verify License Server Options in Server Configuration File . . .	2-2
Verify Status of License Server using mps-status	2-3
Force a License Checkout Using mps-license-reset	2-4

Secure a Server

3

Security Overview	3-2
Enable HTTPS	3-3
Configure Client Authentication	3-5
Specify Access to MATLAB Programs	3-6
Adjust Security Protocols	3-7
Improve Startup Time When Security Is Activated	3-8
Access Control	3-9
Access Control Configuration File	3-9
Access Control Policy File	3-9
Use Kerberos and Kerberos Delegation	3-14
Supported Environment	3-14

Troubleshooting

4

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
Setting Log File 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	4-10

Impact of Server Configurations on Processing Asynchronous Requests

5

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

Set Up MATLAB Production Server Dashboard

6

Set Up and Log In to MATLAB Production Server Dashboard	6-2
Set Up the Dashboard	6-2
Log In to the Dashboard	6-4
Reset the Admin Password	6-4
Remove MATLAB Production Server Dashboard	6-5

7

Configuration Properties

8

Cloud Deployment

9

Azure Deployment	9-2
Provision Cloud Resources	9-2
Connect and Log In to Cloud Console	9-7
Upload License File	9-7
Architecture and Resources on Azure	9-8
MATLAB Production Server Architecture on Azure	9-8
Azure Resources	9-8
Use MATLAB Production Server Cloud Console	9-11
View Information About Server Instances	9-11
View HTTPS Endpoint	9-11
Upload a MATLAB Application	9-11
Edit the Server Configuration	9-12
Edit the Azure Cache for Redis Configuration	9-12
Get the License Server MAC Address	9-13
Upload a License File	9-14
Upload an HTTPS Certificate	9-14
Set Up Authentication Using Azure Active Directory	9-14
Configure Azure Resources	9-16
Change the Number of Virtual Machines	9-16
Change Self-Signed Certificate to the Application Gateway	9-16
View Logs	9-17
Delete Resource Group	9-17
Reset Cloud Console Password	9-18
Calculate Number of Workers	9-19

Server Management

- “Server Overview” on page 1-2
- “Create a Server” on page 1-4
- “Edit the Configuration File” on page 1-6
- “Specify the Default MATLAB Runtime for New Server Instances” on page 1-8
- “Specify the MATLAB Runtime for a Server Instance” on page 1-9
- “Start a Server Instance” on page 1-10
- “Share the Deployable Archive” on page 1-11
- “Support Multiple MATLAB Versions” on page 1-12
- “Control Worker Restarts” on page 1-14
- “Install a Server Instance as a Windows Service” on page 1-16
- “Recovery Options for a Server Instance Running as a Windows Service” on page 1-17

Server Overview

In this section...

“What Is a Server?” on page 1-2

“How Does a Server Manage Work?” on page 1-2
--

What Is a Server?

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.

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

In addition, each server has its own `auto_deploy` folder, which contains the deployable archives you want the server to host for clients.

The server also manages the MATLAB Runtime (MATLAB Compiler), which enables 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:

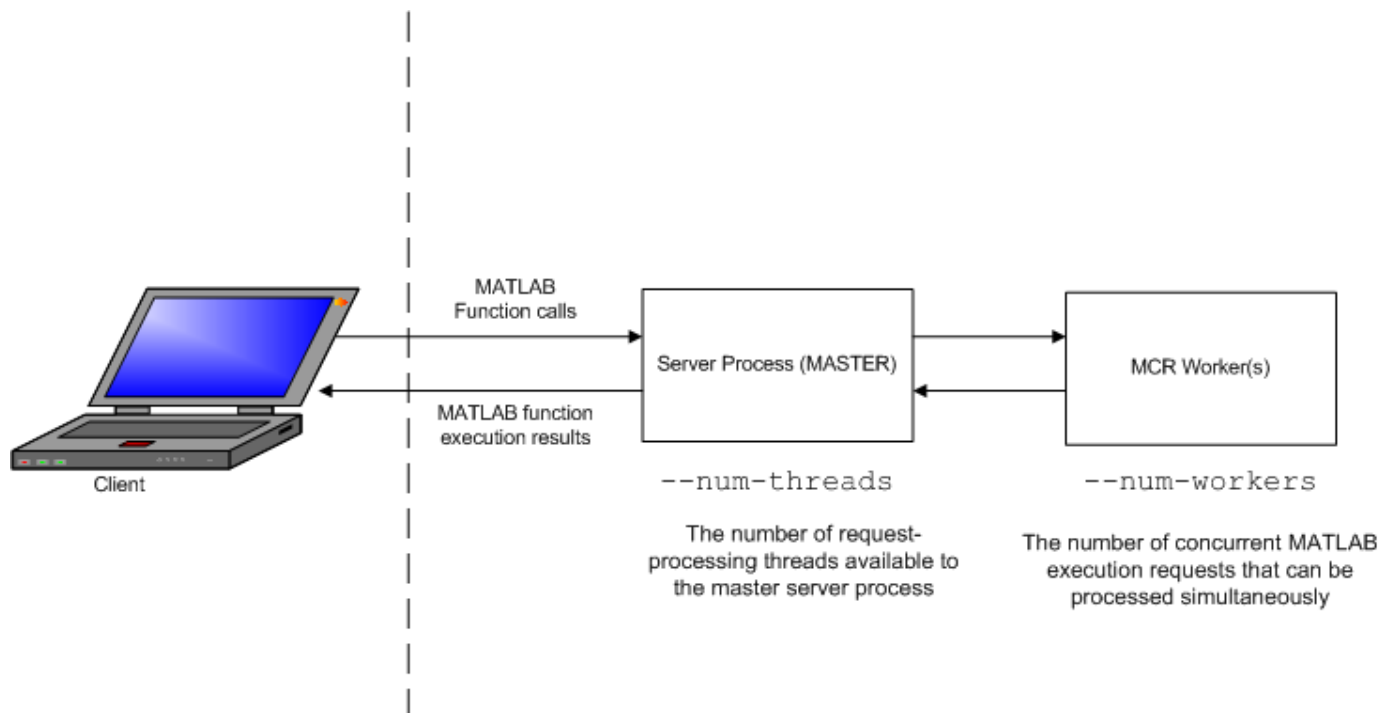
- 1 The client sends MATLAB function calls to the master server process (the main process on the server).
- 2 MATLAB function calls are passed to one or more MATLAB Runtime workers.
- 3 MATLAB functions are executed by the MATLAB Runtime worker.
- 4 Results of MATLAB function execution are passed back to the master server process.
- 5 Results of MATLAB function execution are passed back for processing by the client.

The server is the middleman 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 master 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 master 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 the client's behalf.

See Also

`mcr-root` | `mps-setup`

More About

- "Create a Server" on page 1-4
- "Support Multiple MATLAB Versions" on page 1-12

Create a Server

Before you can deploy your MATLAB code with MATLAB Production Server, you must create a server 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, ensure you have:

- installed MATLAB Production Server on your machine. For more information, see “Install MATLAB Production Server”.
- 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 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 configuration or instance from the command-line in an on-premise installation, 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 are creating 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 using 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/auto_deploy...ok
prod_server_1/x509...ok
prod_server_1/endpoint...ok
prod_server_1/log...ok
prod_server_1/old_logs...ok
prod_server_1/.mps_socket...ok
```

```
prod_server_1/pid...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.

See Also

[mps-new](#) | [mps-service](#)

More About

- “Install a Server Instance as a Windows Service” on page 1-16
- “Start a Server Instance”

Edit the Configuration File

In this section...

“About the Server Configuration File” on page 1-6

“Common Customizations” on page 1-6

About the Server Configuration File

To change server properties for an on-premise installation of MATLAB Production Server, edit the `main_config` configuration file that corresponds to your specific server instance. The server configuration file is located at

`server_name/config/main_config`

For a server deployment on the cloud, see “Edit the Configuration File” on page 1-6.

When editing the server configuration, remember these coding considerations:

- Each server has its own server configuration file.
- You enter only one configuration property and related options per line. Each configuration property entry starts with two dashes (- -).
- Any line beginning with a pound sign (#) is ignored as a comment.
- Lines of white space are ignored.

Common Customizations

- “Setting Default Port Number for Client Requests” on page 1-6
- “Setting Number of Available Workers” on page 1-6
- “Setting Number of Available Threads” on page 1-6

Setting Default Port Number for Client Requests

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

Setting Number of Available Workers

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

Setting Number of Available Threads

Use the `num-threads` property to set the number of request-processing threads available to the master 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, code the following:

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

See Also

[https](#)

More About

- “Create a Server” on page 1-4
- “Control Worker Restarts” on page 1-14

Specify the 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-premise 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 “Download and Install the MATLAB Runtime”.

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 the MATLAB Runtime for a Server Instance” on page 1-9
- “Support Multiple MATLAB Versions” on page 1-12

Specify the MATLAB Runtime for a Server Instance

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

- 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 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 version number of the MATLAB Runtime (`vnnn`). The MATLAB Runtime versions that you specify must be compatible with MATLAB Production Server.

- 5 Restart the server instance.

See Also

`mps-start`

More About

- “Specify the Default MATLAB Runtime for New Server Instances” on page 1-8
- “Edit the Configuration File” on page 1-6
- “Support Multiple MATLAB Versions” on page 1-12

Start a Server Instance

In this section...
“Prerequisites” on page 1-10
“Procedure” on page 1-10

Follow the procedure below to start a server instance in an on-premise installation of MATLAB Production Server.

Prerequisites

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

- Installed the MATLAB Runtime
- Created a server instance on page 1-4
- Specified the default MATLAB Runtime for the instance on page 1-8

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 you want to create. *path* should end with the server name.
- `server_name` — Name of the server instance you want to start or stop.
- `-f` — Forces command to succeed, regardless of whether the server is already started or stopped.

Note If needed, use the `mps -status` command to verify the server is running.

See Also

`mps-new` | `mps-service`

More About

- “Install a Server Instance as a Windows Service” on page 1-16
- “Share the Deployable Archive” on page 1-11

Share the Deployable Archive

After you create the deployable archive, share it with clients of MATLAB Production Server by copying it to your server for hosting. For information on how to create a deployable archive, see “Package Deployable Archives with Production Server Compiler App” and “Package Deployable Archives from Command Line”.

In order to share the deployable archive, you must have a server created and running.

- 1 Locate your deployable archive in the `for_redistribution_files_only` folder of your compiler project folder.

The deployable archive has the name `project_name.ctf`.

- 2 Copy `project_name.ctf` to the `\server_name\auto_deploy` folder in your server instance.

For example, if your server is named `prod_server_1` and located in `C:\tmp`, copy `project_name.ctf` to `C:\tmp\prod_server_1\auto_deploy`.

See Also

auto-deploy-root

More About

- “Modifying Deployed Functions”
- “Create a Server” on page 1-4

Support Multiple MATLAB Versions

In this section...

“How the Server Instance Selects the MATLAB Runtime to Use” on page 1-12
--

“Changes to Worker Management” on page 1-13

MATLAB Production Server instances can host deployable archives compiled using multiple versions of MATLAB Compiler SDK™. To do so, add multiple `mcr-root` properties to the configuration file for the instance.

- 1 Install the required versions of the MATLAB Runtime.

Note

- A server instance should only be configured to use MATLAB Runtime roots on a local file system. Otherwise, a network partition may cause worker processes to fail.
- All values for `mcr-root` must be for the same OS/hardware combination.

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

The configuration file is at `instanceRoot/config/main_config`.

- 4 Locate the entry for the `mcr-root` property.

```
--mcr-root mCRuNsEtOKEN
```

- 5 For each version of MATLAB Runtime that the instance supports, add an instance of the `mcr_root` property. Order the versions from the latest to the oldest.

For example, to configure the instance to use the v81 and v82 versions of the MATLAB Runtime, specify the following.

```
--mcr-root C:\Program Files\MATLAB\MATLAB Runtime\v82  
--mcr-root C:\Program Files\MATLAB\MATLAB Compiler Runtime\v81
```

- 6 Restart the server instance.

If you have a server environment deployed on the cloud, the deployment sets the `mcr_root` property to support multiple MATLAB Runtime versions.

How the Server Instance Selects the MATLAB Runtime to Use

Once the server instance is configured to use multiple versions of MATLAB Runtime, it scans the list of provided MATLAB Runtime installations 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 also changes to how the server instance manages the workers used to process requests.

When using a single MATLAB Runtime installation, the server instance starts workers as needed until `num-workers` workers are running. Once running, workers may be restarted in response to 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 worker is returned to the pool, it can be allocated to process new requests using any of the configured MATLAB Runtime versions.

See Also

More About

- “Specify the Default MATLAB Runtime for New Server Instances” on page 1-8
- “Specify the MATLAB Runtime for a Server Instance” on page 1-9
- “Edit the Configuration File” on page 1-6
- “Start a Server Instance” on page 1-10

Control Worker Restarts

In this section...

“Restart Workers Based on Up Time” on page 1-14

“Restart Workers Based on Amount of Memory in Use” on page 1-14

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

- “Edit the Configuration File” on page 1-6
- “Support Multiple MATLAB Versions” on page 1-12

Install a Server Instance as a Windows Service

In this section...

“Create a New Server Instance as a Windows Service” on page 1-16
--

“Make an Existing Server Instance a Windows Service” on page 1-16

Create a New Server Instance as a Windows Service

To create a new on-premise MATLAB Production Server instance and register it as a Windows service, use the `mps-new` command with the `--service` option.

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

You can change the name, description, and user for the Windows service from the defaults using optional flags to the `mps-new` command.

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

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.

This feature is available only for an on-premise installation of MATLAB Production Server.

Make an Existing Server Instance a Windows Service

To create a new Windows service for an existing on-premise MATLAB Production Server instance, use the `mps-service` command with the `create` option.

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

You can change the name, description, and user for the Windows service from the defaults using optional flags to the `mps-service` command.

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.

See Also

More About

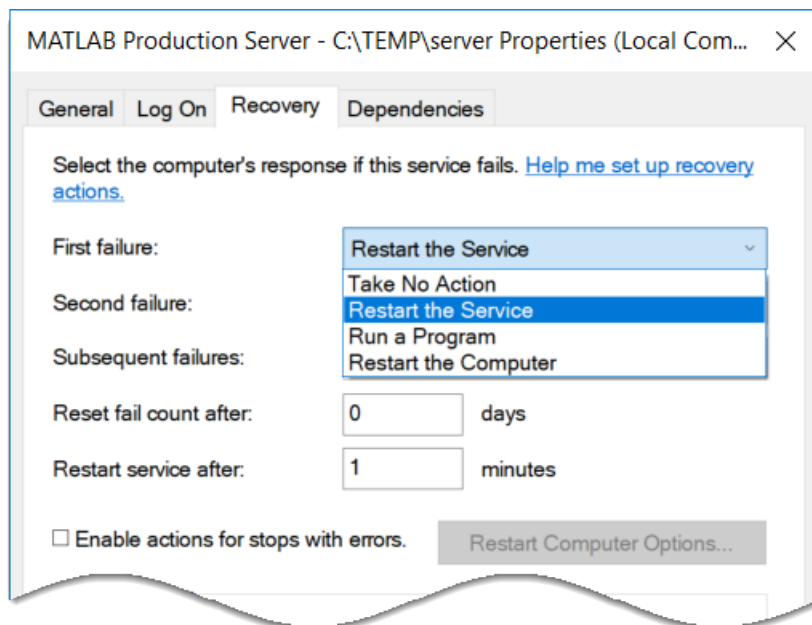
- “Recovery Options for a Server Instance Running as a Windows Service” on page 1-17

Recovery Options for a Server Instance Running as a Windows Service

You can install a MATLAB Production Server instance in an on-premise installation to run as a Windows service. For more information on how to set this up, see “Install a Server Instance as a Windows Service” on page 1-16.

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.



See Also

mps-service

More About

- “Server Overview” on page 1-2
- “Enable HTTPS” on page 3-3

Manage Licenses for MATLAB Production Server

- “Specify or Verify License Server Options in Server Configuration File” on page 2-2
- “Verify Status of License Server using mps-status” on page 2-3
- “Force a License Checkout Using mps-license-reset” on page 2-4

Specify or Verify License Server Options in Server Configuration File

Specify or verify values for License Server options in the server configuration file (`main_config`). You create a server by using the `mps - new` command.

Edit the configuration file for the server. Open the file `server_name/config/main_config` and specify or verify parameter values for the following options. See the comments in the server configuration file for complete instructions and default values.

- `license` — Configuration option to specify the license servers and/or the license files. You can specify multiple license servers including port numbers (`port_number@license_server_name`), as well as license files, with one entry in `main_config`. List where you want the product to search, in order of precedence, using semicolons (;) as separators on Windows or colons (:) as separators on Linux.

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

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

The system searches these resources in this order:

- 1 `27000@hostA:` (hostA configured on port 27000)
 - 2 `/opt/license/license.dat` (local license data file)
 - 3 `27001@hostB:` (hostB configured on port 27001)
 - 4 `./license.dat` (local license data file)
- `license-grace-period` — The maximum length of time MATLAB Production Server responds to HTTP requests, after license server heartbeat has been lost. See the network license manager documentation for more on heartbeats and related license terminology.
 - `license-poll-interval` — 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 checkout a valid license. Polling occurs at the interval specified by `license-poll-interval` until license has been successfully checked-out. See the network license manager documentation for more on heartbeats and related license terminology.

Verify Status of License Server using mps-status

When you enter an `mps - status` command, the status of the server *and* the associated license is returned.

For detailed descriptions of these status messages, see “License Server Status Information”.

Force a License Checkout Using `mps-license-reset`

Use the `mps-license-reset` command to force MATLAB Production Server to checkout a license. You can use this command at any time, providing you do not want to wait for MATLAB Production Server to verify and checkout a license at an interval established by a server configuration option such as `license-grace-period` or `license-poll-interval`.

Secure a Server

- “Security Overview” on page 3-2
- “Enable HTTPS” on page 3-3
- “Configure Client Authentication” on page 3-5
- “Specify Access to MATLAB Programs” on page 3-6
- “Adjust Security Protocols” on page 3-7
- “Improve Startup Time When Security Is Activated” on page 3-8
- “Access Control” on page 3-9
- “Use Kerberos and Kerberos Delegation” on page 3-14

Security Overview

MATLAB Production Server uses HTTPS to establish secure connections between server instances and clients. The HTTPS layer provides certificate-based authentication for both clients and server instances. It also provides an encrypted data path between the clients and server instances. For more information, see “Enable HTTPS” on page 3-3.

The default security settings enable all security protocols and cipher suites, except for the eNULL cipher suite. You can configure the level of security provided by the HTTPS layer and the security protocols it supports. For more information, see “Adjust Security Protocols” on page 3-7.

The default security settings allow all clients to access all programs hosted by the server instance. The server instance does not authenticate the clients, nor does it perform any authorization. MATLAB Production Server provides a certificate-based authorization mechanism for restricting access to specific programs. Using this mechanism, you specify the MATLAB programs that a client can access. To configure client authorization, see “Specify Access to MATLAB Programs” on page 3-6. To ensure that only trusted client applications have access to a server instance, configure the server instance to require client authentication. For more information, see “Configure Client Authentication” on page 3-5.

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 3-5. HTTPS also provides an encrypted data path between the clients and server instances.

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 stored in a PEM-format certificate chain
- `x509-private-key`: Valid private key stored in PEM format

For more information about the server configuration file, see “Edit the 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/my-cert.pem` and the unencrypted private key stored in `./x509/my-key.pem`.

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

Starting in R2019b, if `https` is enabled on the server, you must set both the `x509-cert-chain` and `x509-private-key` properties; otherwise, the server fails to start.

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/my-cert.pem
--x509-private-key ./x509/my-key.pem
--x509-passphrase ./x509/my-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

[client-credential-delegation](#) | [ssl-protocols](#) | [ssl-tmp-ec-param](#)

More About

- “Configure Client Authentication” on page 3-5
- “Specify Access to MATLAB Programs” on page 3-6
- “Adjust Security Protocols” on page 3-7

Configure Client Authentication

To ensure that only trusted client applications have access to a server instance, configure the server instance to require client authentication:

- 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 adding the `x509-use-crl` property to the server's configuration. If this property is not specified, the server instance ignores the CRLs and may authenticate clients using revoked credentials.

Caution You must add a CRL list to the server's CA store 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-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-private-key](#)

More About

- “Enable HTTPS” on page 3-3
- “Adjust Security Protocols” on page 3-7
- “Specify Access to MATLAB Programs” on page 3-6

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 3-3
- “Configure Client Authentication” on page 3-5
- “Adjust Security Protocols” on page 3-7

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

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

[https](#) | [ssl-ciphers](#) | [ssl-tmp-ec-param](#)

More About

- “Enable HTTPS” on page 3-3
- “Configure Client Authentication” on page 3-5
- “Specify Access to MATLAB Programs” on page 3-6
- “Adjust Security Protocols” on page 3-7

Access Control

Access Control Configuration File

To provide an identity to each user, you define an access control configuration file in JSON format. Each identity provider has a different configuration file to enable authorization. The default name for the JSON file for Azure® Active Directory is `azure_ad.json`.

Azure Active Directory configuration parameters are as follows:

- `tenantId` (Required): Azure Active Directory tenant ID. To locate your tenant ID, go to <https://portal.azure.com>. On the left panel, select **Azure Active Directory**, then on the **Overview** panel, select **Properties**. The hexadecimal code under **Directory ID** is your tenant ID.
- `serverAppId` (Required): MATLAB Production Server application ID as registered in Azure Active Directory. To locate your `serverAppId`, go to <https://portal.azure.com>. On the left panel, select **Azure Active Directory**, then on the **Overview** panel, select **App registrations**. Then select **MPS server** to find the **Application ID**, which is your `serverAppId`.
- `jwtUri` (Optional): Used to get Azure Active Directory JSON Web Key Set that is used to verify token signature. Default is <https://login.microsoftonline.com/common/discovery/keys>.
- `issuerBaseUri` (Optional): Used with `tenantId` to validate issuer of the token. For Azure Active Directory, default is <https://sts.windows.net/>.
- `jwtTimeout` (Optional): Maximum time the `jwtUri` request is allowed to take. Default is 120 seconds.

The format of the configuration file is as follows:

```
{
  "tenantId": "54ss4lk1-8428-7256-5fvh-d5785gfhkjh6",
  "serverAppId": "j21n12bg-3758-3r78-v25j-35yj4c47vhmt",
  "jwtUri": "https://login.microsoftonline.com/common/discovery/keys",
  "issuerBaseUri": "https://sts.windows.net/",
  "jwtTimeout": 120
}
```

Access Control Policy File

To use access control for MATLAB Production Server, the server admin should define an access control policy file in JSON format. The default name for the JSON file is `ac_policy.json`.

The policy file is read on server startup. If it does not exist or contains errors, the server does not start, and an error message is written to `main.log` file found in the log-root directory.

Once the server has started, the policy file is scanned every five seconds for changes. If the policy file is deleted or contains errors, the server continues to run, but all requests are denied. Again, an error message is written to the `main.log` file.

The JSON 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 and consists of a *Subject Block*, a *Resource Block*, and an *Action Block*.



The schema version has a value that is a JSON string in the format `<major#>.<minor#>.<patch#>`, with each number specified as a nonnegative integer.

Policy Block

The policy block contains a list of policies required for access control. Currently, only a single policy can be specified in a policy file.

```

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

```

An ID is required for each policy. `<policy_id>` must be unique for each policy. Any leading or trailing white space is 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>
  }
]
```

Multiple rules can exist in a rule block, for example: `"rule": [<rule>, <rule>, ...]`.

An ID is required for each rule. `<rule_id>` must be unique for each rule. Any leading or trailing white space is removed.

The `description` is optional for a rule.

Subject Block

The subject block of a rule defines who can access the resources. Currently, only the `groups` attribute is supported.

```
"subject" : {"groups": ["<group_id>", "<group_id>", ...]}
```

For Azure Active Directory, a list of group IDs can be specified to control which groups can access the resources defined in the rule.

Get Group ID from Azure Active Directory Based on Group Display Name

- 1 Open Azure Active Directory graph explorer on <https://graphexplorer.azurewebsites.net>, and login.
- 2 Use query `https://graph.windows.net/<tenant>/groups?$filter=startswith(displayName, '<groupname>')` where `<tenant>` is the tenant name, and `<groupname>` is the name of a specific group.
- 3 Search for `objectId` of the specific group in the response.

Get All Group IDs for a Certain User from Azure Active Directory

- 1 Open Azure Active Directory graph explorer on <https://graphexplorer.azurewebsites.net>, and login.
- 2 Use query `https://graph.windows.net/<tenant>/users/<username>@<tenant>/memberOf` where `<tenant>` is the tenant name, and `<username>` is the name of a specific user.
- 3 For all groups where **securityEnabled** is true, search for `objectId` in the response.

Resource Block

The resource block of a rule describes the object being accessed. Currently, only a `ctf` file can be accessed.

```
"resource" : {"ctf": ["<ctf_name>", "<ctf_name>", ...]}
```

You can use `ctf_name` to access multiple `ctf` files by using the wildcard character `*`. For example, if you want to access all `ctf` files whose names start or end with 'test', you would specify `<ctf_name>` as `test*` or `*test`, respectively. If you use `*` as the `<ctf_name>`, you can access all the `ctf` files.

Action Block

The action block of a rule describes the action being attempted on the resource. Currently, only the action `execute` is supported.

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

Example of a JSON Policy File

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

- All users belonging to a group with ID `aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa` can execute the `ctf` file `magic`.
- All users belong to groups with id `aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa` and `bbbbbbbbb-bbbb-bbbb-bbbb-bbbbbbbbbbbb` can execute the `ctf` files `monteCarlo` and `fastFourier`.
- All users belong to Quality Engineering group `cccccccc-cccc-cccc-cccc-cccccccccccc` can execute all `ctfs` starting with `test`.

Access is denied for all other requests.

```
{
  "version": "1.0.0",
  "policy" : [
    {
      "id": "policy1",
      "description": "MPS Access Control policy for XYZ Corp.",
      "rule": [
        {
          "id": "rule1",
          "description": "group A can execute ctf magic",
          "subject": { "groups": ["aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa"] },
          "resource": { "ctf": ["magic"] },
          "action": ["execute"]
        },
        {
          "id": "rule2",
          "description": "group A and group B can execute ctf monteCarlo and fastFourier",
          "subject": { "groups": ["aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa", "bbbbbbbbb-bbbb-bbbb-bbbb-bbbbbbbbbbbb"] },
          "resource": { "ctf": ["monteCarlo", "fastFourier"] },
          "action": ["execute"]
        },
        {
          "id": "rule3",
          "description": "QE group C can execute any ctf starts with test",
          "subject": { "groups": ["cccccccc-cccc-cccc-cccc-cccccccccccc"] },
          "resource": { "ctf": ["test*"] },
          "action": ["execute"]
        }
      ]
    }
  ]
}
```



```
]
}
```

See Also

access-control-policy

External Websites

- <https://docs.microsoft.com/en-us/azure/active-directory/develop/active-directory-graph-api-quickstart>

Use Kerberos 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 requires domain administrator privileges. Currently, you can use Kerberos and Kerberos Delegation with MATLAB Production Server instances running on Windows Server® operating systems with a Windows Key Distribution Center. To configure Kerberos and Kerberos delegation, consult your IT / Windows System Administrator, and follow these steps:

- 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 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`) support using Kerberos Delegation:

- `webread`
- `webwrite`
- “HTTP Interface” (MATLAB) functions
- Database Toolbox™ functions (requires an ODBC driver)

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

Supported Environment

Option	Requirement
Operating system	Windows Server
Kerberos Delegation	Constrained delegation without protocol transition
Key distribution center	Windows Server 2003 or later
Client	<ul style="list-style-type: none"> • RESTful client over HTTP/HTTPS (HTTP 1.1) with JSON payload • The RESTful client must be one that supports SPNEGO/Kerberos—for example, <code>curl</code> with the <code>--negotiate</code> option or <code>.NET HttpClient</code>
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

See Also

[client-credential-delegation](#) | [http-authentication-method](#)

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 a server in an on-premise MATLAB Production Server installation.

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 at the system command prompt

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

Output:

- If `prod_server_1` is running and operating with a valid license.

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

- If `prod_server_1` is unable to check out valid license.

```
\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.
```

To verify whether the server has started or stopped after issuing `mps -restart` and `mps -stop` commands, use `mps -status`.

License Server Status Information

In addition to the status of the server, `mps - status` also displays the status of the license server associated with the server you are querying.

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 the required number of license keys are checked out.
WARNING: lost connection to license server - request processing will be disabled at <i>time</i> 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 <i>time</i> . At <i>time</i> , if connectivity to the license server has not been restored, request processing will be disabled 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. Request processing has been 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 - restart` | `mps - stop`

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 `main_config`.

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 9-17.

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-premise 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.

Manage Log Files

In this section...

“Best Practices for Log Management” on page 4-7

“Log Retention and Archive Settings” on page 4-7

“Setting Log File Detail Levels” on page 4-8

Best Practices for Log Management

Use these recommendations as a guide when defining values for the options listed in “Log Retention and Archive Settings” on page 4-7.

- Avoid placing `log-root` and `log-archive-root` 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 best MATLAB coding practices. See *MATLAB Programming Fundamentals* for guidelines.

Log Retention and Archive Settings

Log data is written to the server’s `main.log` file for as long as a specific server instance is active, or until midnight. When the server is restarted, log data is written to an archive log, located in the archive log folder specified by `log-archive-root`.

You can set parameters that define when `main.log` is archived using the following options in each server’s `main_config` file.

- `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 options using the following units and notations:

Represent these units of measure...	Using this 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.

Setting Log File Detail Levels

The log level provides different levels of information for troubleshooting:

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

The log level is set using the `log-severity` configuration property.

Before you call support, you should set logging levels to `trace`.

Common Error Messages and Resolutions

In this section...

“(404) Not Found” on page 4-9

“Error: Bad MATLAB Runtime Instance” on page 4-9

“Error: Server Instance not Specified” on page 4-9

“Error: invalid target host or port” on page 4-9

“Error: HTTP error: HTTP/x.x 404 Component not found” on page 4-10

(404) Not Found

Commonly caused by requesting a component that is not deployed on the server, or trying to call a function that is not exported by the given component.

Verify that the name of the deployable archive specified in your Uri is the same as the name of the deployable archive hosted in your `auto_deploy` folder.

Error: Bad MATLAB Runtime Instance

Common causes of this message include:

- You are not properly qualifying the path to the MATLAB Runtime. You must include the version number. For example, you need to specify:

```
C:\Program Files\MATLAB\MATLAB Runtime\vn.n
```

not

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

Error: Server Instance not Specified

MATLAB Production Server can't find the server you are specifying.

Ensure you are either entering commands from the folder containing the server instance, or are using the `-C` command argument to specify a precise location of the server instance.

For example, if you created `server_1` in `C:\tmp\server_1`, you would issue 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 a Server Instance” on page 1-10.

Error: invalid target host or port

The port number specified has not been properly defined to your computer. 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. Consult the “Log Files” on page 4-6 for further details on the precise cause of the problem.

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-limit` | `response-archive-root`

More About

- “Edit the 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 6-5

Set Up and Log In to MATLAB Production Server Dashboard

In this section...

“Set Up the Dashboard” on page 6-2

“Log In to the Dashboard” on page 6-4

“Reset the Admin Password” on page 6-4

Follow these instructions to set up the dashboard for an on-premise installation of MATLAB Production Server.

Set Up the Dashboard

Warning You must have admin privileges on Windows to complete setup.

To set up an instance of the MATLAB Production Server dashboard:

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

Platform	Default Directory Where the MATLAB Production Server dashboard is Installed
Windows (Administrator)	C:\Program Files\MATLAB\MATLAB Production Server\R2020a\dashboard
Linux®	/usr/local/MATLAB/MATLAB_Production_Server/R2020a/dashboard

- 2 Execute the script `mps -dashboard` with the `setup` option, and when prompted, specify the directory for dashboard setup. You must have write privileges to the directory from where you are running the `mps -dashboard` script, and to the directory where the dashboard is going to be set up.

Platform	Script for Dashboard Setup
Windows (Administrator)	<pre>> mps-dashboard.bat setup</pre> <p>For example:</p> <pre>> mps-dashboard.bat setup Specify a workspace directory for MATLAB Production Server Dashboard: C:\mp</pre>
Linux	<pre>\$./mps-dashboard.sh setup</pre> <p>For example:</p> <pre>\$./mps-dashboard.sh setup Specify a workspace directory for MATLAB Production Server Dashboard: /opt/mps/dashboard</pre>

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, in Windows: > mps-dashboard.bat setup -C D:\mps\dashboard

For example, in Linux: \$./mps-dashboard.sh setup -C /opt/mps/dashboard

Note For a complete list of options that can be passed to the mps-dashboard script, pass a ? as an option to the mps-dashboard script.

For example, in Windows type:

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

In Linux, type:

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

The complete list of options are as follows.

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

Tip Troubleshooting (Windows only): If the mps-dashboard.bat setup command fails with an error message that mentions a missing MSVCR DLL, add \$MPS_ROOT\bin\win64 to your system PATH, where \$MPS_ROOT is the directory where you have MATLAB Production Server installed. The \$MPS_ROOT\bin\win64 folder contains DLLs that MATLAB Production Server uses.

- 3 Execute the mps-dashboard script with the start option to start the dashboard.

Platform	Script to Start Dashboard
Windows (Administrator)	> mps-dashboard.bat start
Linux	\$./mps-dashboard.sh start

You will get a message 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\R2020a\dashboard\config\config.txt
Linux	/usr/local/MATLAB/MATLAB_Production_Server/R2020a/dashboard/config/config.txt

Other customizations to the setup process can be made by editing relevant parts of the config.txt file.

- 4 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 the 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 the Admin Password

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

Platform	Script to Reset the Admin Password
Windows (Administrator)	<code>> mps-dashboard.bat reset_admin_password</code>
Linux	<code>\$./mps-dashboard.sh reset_admin_password</code>

Warning The `reset_admin_password` option should not be executed while dashboard is still running. First, stop dashboard execution using the `mps-dashboard` script with the `stop` option and then reset the admin password.

See Also

Related Examples

- “Remove MATLAB Production Server Dashboard” on page 6-5

Remove MATLAB Production Server Dashboard

Follow these steps to remove the MATLAB Production Server dashboard in an on-premise 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\R2017b\dashboard
Linux	/usr/local/MATLAB/MATLAB_Production_Server/R2017b/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 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 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 dashboard was set up:

```
data
mps_workspace
.pid
```

If you run into any issues while removing 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 don't need to open a new Terminal. The & control operator makes command run in the background.

In Windows, if 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 dashboard does not uninstall it from the system. It removes the 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 a MATLAB Production Server instance for problems

Syntax

```
mps-check [--timeout seconds] host:port
```

Description

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

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

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

Before using `mps-check`, you must deploy `mcrroot/bin/arch/mps_check.ctf` to the server instance.

- `mcrroot` is the path to the MATLAB Runtime installation folder.
- `arch` is standard abbreviation for the system's operating system and hardware architecture.

Input Arguments

- `--timeout seconds` — The time, in seconds, to wait for a response from the server before timing out. The default is two minutes.
- `host` — The host name of the machine running the server instance.
- `port` — The port number on which the server instance listens for requests.

Examples

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

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

```
Connecting to localhost:9910
Connected
Sending HTTP request
HTTP request sent
HTTP response received
MPS status check completed successfully
```


More About

Server Instance

Server instance is an instance of the MATLAB Production Server. The files contained in the folder created by `mps - new`, defined by *path/*, comprise one configuration of the MATLAB Production Server product.

Introduced in R2012b

mps-license-reset

Force a server instance to immediately attempt license checkout

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:

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

See Also

`mps-status`

Topics

“Force a License Checkout Using `mps-license-reset`” on page 2-4

Introduced in R2012b

mps-new

Create a server instance

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 a Server Instance

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

```
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 a Windows Service

Create a new server instance, and register it as a Windows service:

```
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.

See Also

`mps-start` | `mps-status`

Topics

“Create a Server” on page 1-4

“Install a Server Instance as a Windows Service” on page 1-16

“Server Overview” on page 1-2

Introduced in R2012b

mps-profile

Turn profiling on or off

Syntax

```
mps-profile [-C [path/]instance_name] {on|off} [object...]
```

Description

`mps-profile` turns profiling on or off for specified objects.

Input Arguments

- `-C` — 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.
- `on` — Activate profiling.
- `off` — Deactivate profiling.
- *object* — The list of objects whose profiling behavior is changed.

Valid object values are:

- `requests`
- `worker_pool`

If no object is specified the command changes all objects.

Examples

Turn profiling on.

```
mps-profile on
```

Turn request profiling on without turning on worker pool profiling.

```
mps-profile on requests
```

Introduced in R2015b

mps-restart

Stop and start a server instance

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

Restart a server instance named `server_1`, located in folder `tmp`. Force successful completion of `mps-restart`.

```
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.

See Also

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

Introduced in R2012b

mps-service

Create or modify a Windows service for a server instance

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.

`mps-service delete service_name [--force][[-f]]` deletes the Windows service entry by name.

`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 a Windows Service

Create a default Windows service for the server instance `server_1`:

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

Delete a Windows Service

Delete the Windows service entry for the server instance `server_1`:

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

List Existing Windows Services

List the Windows service entries for all the server instances installed on the local machine:

```
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
```

See Also

`mps -new`

Topics

“Install a Server Instance as a Windows Service” on page 1-16

Introduced in R2015a

mps-setup

Set up a server environment

Syntax

```
mps-setup [mcrroot]
```

Description

`mps-setup [mcrroot]` sets location of MATLAB Runtime and other start-up options.

The `mps-setup` command sets the default path to the MATLAB Runtime for all server instances you create with the product. This is equivalent to presetting the `--mcr-root` option in each server's `main_config` configuration file.

If a default value already exists in `server_name/config/mcrroot`, it is updated with the value specified when you run the command line wizard.

Tips

- Run `mps-setup` from the script folder. Alternatively, add the `script` folder to your system `PATH` environment variable to run `mps-setup` from any folder on your system.
- Run `mps-setup` without arguments and it will search your system for MATLAB Runtime instances you may want to use with MATLAB Production Server.
- Run `mps-setup` by passing the path to the MATLAB Runtime as an argument. This method is ideal for non-interactive (silent) installations.

Input Arguments

mcrroot

Specify a path to the MATLAB Runtime if running `mps-setup` in non-interactive, or silent, mode.

Examples

Run `mps-setup` non-interactively, by passing in a path to the MATLAB Runtime instance that you want MATLAB Production Server to use.

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

mcrver is the version of the MATLAB Runtime to use.

See Also

`mps-new` | `mps-start` | `mps-status`

Introduced in R2012b

mps-start

Start a server instance

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

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`. Force successful completion of `mps-start`.

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

Tips

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

See Also

`mps-new` | `mps-restart` | `mps-status` | `mps-stop`

Topics

“Start a Server Instance” on page 1-10

“Server Overview” on page 1-2

Introduced in R2012b

mps-status

Display status of a server instance

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 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 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 utilization, 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 a Server is Running

Display status of server instance `server_1`, residing in `tmp` folder.

```
mps-status -C /tmp/server_1
```

If server is running and running with a valid license:


```

'/tmp/server_1' STARTED
license checked out

```

If server is not running:

```

'/tmp/server_1' STOPPED

```

Report Statistics in a Human Readable Format

Display statistics for the server instance `server_1`, residing in `tmp` folder.

```

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.

```

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
  }
}

```

See Also

`mps-restart` | `mps-start` | `mps-stop` | `mps-which`

Topics

“Start a Server Instance” on page 1-10

“Server Overview” on page 1-2

“License Server Status Information” on page 4-3

Introduced in R2012b

mps-stop

Stop a server instance

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. All function requests that were executing when the command was issued are allowed to complete before the server shuts down.

Input Arguments

-C *path/*

Specify a path to the server instance.

If you are stopping 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 current working folder and its parents are searched to find the server instance.

server_name

Name of the server to be stopped.

-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.

Starting in R2020a, 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.

-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.

Starting in R2020a, 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.

```
mps-stop -f -v -C /tmp/server_1 --timeout 00:03:00
```

Example Output

```
waiting for stop... (timeout = 00:03:00)
```

- Immediately terminate all running server instance processes by force using the `-k` option.

```
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.

```
mps-stop /tmp/server_1
```

Tips

- After issuing `mps - stop`, issue the `mps - status` command to verify that the server instance has stopped.

See Also

`mps - new` | `mps - restart` | `mps - start` | `mps - status` | `server-termination-grace-period`

Introduced in R2012b

mps-support-info

Display licensing and configuration information of a MATLAB Production Server instance

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.

```
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 )
```

Introduced in R2012b

mps-which

Display path to server instance that is currently using the configured port

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 instance on the same port, the latter server instance will fail 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` for both servers:

```
mps-which -C /tmp/server_1
```

```
mps-which -C /tmp/server_2
```

Example Output

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. Only specify the server name.

See Also

`mps-status`

Introduced in R2012b

mps-cache

Control persistence service

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 a Remote Persistence Service” on page 7-28.

--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 a Persistence Service

Start a persistence service on Windows assuming a connection name `myConnection` has been defined in the file `mps_cache_config`.

```
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 is:

```
{
  "Connections": {
```

```
    "myRedisConnection": {
      "Provider": "Redis",
      "Host": "localhost",
      "Port": 9710
    }
  }
}
```

Ping a 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 is:

```
{
  "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**.

The screenshot displays the Microsoft Azure portal interface for a Redis Cache resource named 'mps'. The left sidebar shows the navigation menu with 'Overview' selected. The main pane shows the resource details, including the resource group 'vick', status 'Running - Basic 250 MB', location 'East US 2', and subscription ID '2fe59ed0-c022-4411-b2c3-4a2021c1df9c'. The 'Keys' section is highlighted, showing the primary key 'C4TekjUBZwzqhamDhPAIEE648oY6+VDXGu+xMwT' and the secondary key 'SpajIyffIII1fq76MYGf0uEjTzyk62F2o1z4IokxeY='. The 'Manage keys' pane on the right shows the primary and secondary keys and connection strings, with the primary key highlighted.

See Also

Topics

“Use a Data Cache to Persist Data”

Introduced in R2018b

Configuration Properties

access-control-provider

Identity management service provider name

Syntax

```
--access-control-provider provider
```

Description

--access-control-provider *provider* enables access control by identity provider.

ssl-allowed-clients and --access-control-provider are incompatible flags, and only one of them can be enabled. If both are enabled, MATLAB Production Server fails to start.

Parameters

provider specifies the identity provider that is used by the MATLAB Production Server instance for access control. Supported values for *provider* include: AzureAD.

Examples

Enable Access Control Using Azure Active Directory

```
--access-control-provider AzureAD
```

See Also

access-control-config | access-control-policy

Introduced in R2018b

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. This syntax is optional. The default path for AzureAD is `./config/azure_ad.json`. If access control is enabled by specifying `access-control-provider`, the access control configuration file must exist in *path*, otherwise MATLAB Production Server fails to start.

Parameters

path specifies the path to access the configuration file.

Examples

Specify Path to AzureAD.

```
--access-control-config ./config/azure_ad.json
```

See Also

`access-control-policy` | `access-control-provider`

Topics

“Access Control Configuration File” on page 3-9

Introduced in R2018b

access-control-policy

Path to the access control policy file

Syntax

```
--access-control-policy path
```

Description

--access-control-policy *path* specifies the path to the access control policy file. This syntax is optional. The default path is `./config/ac_policy.json`. If access control is enabled by specifying `access-control-provider`, the access control policy file must exist in *path*, otherwise MATLAB Production Server fails to start.

Once the server has started, the policy file is scanned every five seconds for changes. If the policy file is deleted or contains errors, the server continues to run, but all requests are denied. An error message is written to the `main.log` file.

Parameters

path specifies the path to access the policy file.

Examples

Specify Path to the JSON Access Control Policy File.

```
--access-control-policy ./config/ac_policy.json
```

See Also

`access-control-config` | `access-control-provider`

Topics

“Access Control Policy File” on page 3-9

Introduced in R2018b

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
```

See Also

`auto-deploy-root` | `mps-start`

Topics

“Server Overview” on page 1-2

“Share the Deployable Archive” on page 1-11

“Modifying Deployed Functions”

Introduced in R2020a

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

“Share the Deployable Archive” on page 1-11

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
```

See Also

`http-authentication-method`

Topics

“Use Kerberos and Kerberos Delegation” on page 3-14

Introduced in R2019b

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

Requests from any domain origin are allowed access to the sever.

```
--cors-allowed-origins *
```

Requests from a specific list of domain origins are allowed access to the server.

```
--cors-allowed-origins http://www.w3.org, https://www.apache.org
```

See Also

http

disable-control-c

Disable keyboard interruptions for the 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

“Edit the 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
```

See Also

`disable-control-c` | `mps-stop`

Topics

“Edit the Configuration File” on page 1-6

Introduced in R2020a

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
```


hide-matlab-error-stack

Hide the MATLAB stack from the clients

Syntax

```
--hide-matlab-error-stack
```

Description

`hide-matlab-error-stack` controls whether the MATLAB stack is exposed to the client. The stack can be sent to the client during development and debug phase, but can be turned off in production.

Examples

Do not transmit the error stack to clients.

```
--hide-matlab-error-stack
```

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

Restrict access to the HTTP interface for local clients only on port 9910.

```
--http localhost:9910
```

Bind to any free port. The bound address is written to `$INSTANCE/endpoint/https`.

```
--http 0
```

Bind to a specific IP address and port.

```
--http 234.27.101.3:9920
```

Bind to a specific host name on any free port

```
--http my.hostname.com:0
```

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 3-5.

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
```

See Also

`client-credential-delegation`

Topics

“Use Kerberos and Kerberos Delegation” on page 3-14

Introduced in R2019b

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

Restrict access to the HTTPS interface for local clients only on port 9920.

```
--https localhost:9920
```

Bind to any free port. The bound address is written to `$INSTANCE/endpoint/https`.

```
--https 0
```

Bind to a specific IP address and port.

```
--https 234.27.101.3:9920
```

Bind to a specific host name on any free port.

```
--https my.hostname.com:0
```

See Also

[x509-cert-chain](#) | [x509-private-key](#)

Topics

“Enable HTTPS” on page 3-3

license

Locations searched for valid licenses

Syntax

```
--license pathList
```

Description

`license` specifies the license servers or the license files used by the server instance. You can specify multiple license sources with this option.

If this option is not specified, the server searches in the default locations for the license files.

Parameters

pathList

Path to one or more license servers or license files. Multiple entries are separated by the appropriate path separator for the platform.

Examples

A Unix server looks for licenses using a license server hosted on port 27000 of `hostA` and in `/opt/license/license.dat`.

```
--license 27000@hostA  
--license /opt/license/license.dat
```

The same configuration in one line.

```
--license 27000@hostA:/opt/license/license.dat
```

A Windows server looks for licenses using a license server hosted on port 27000 of `hostA` and in `c:\license\license.dat`.

```
--license 27000@hostA  
--license c:\license\license.dat
```

The same configuration in one line.

```
--license 27000@hostA;c:\license\license.dat
```

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 the log file folder

Syntax

```
--log-root path
```

Description

--log-root *path* specifies the location for log files.

When a server instance starts, the following log files are created:

- `main__DATE__SERIAL.log` — The head process main log
- `main.log` — A link to the mostly recently written main log file
- `main.out` — Captured standard output from the main process
- `main.err` — Captured standard error output from the main process

When the server instance stops, the head process main log is moved to the log archive folder.

Note Omitting this property disables all logging except for `stdout` and `stderr` capture of `main`.

Parameters

path

Path to the folder where log files are stored relative to the root folder of the server instance.

Examples

Archive logs to `server_root/logs`.

```
--log-root ./logs
```

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

Enable all log messages.

```
--log-severity trace
```

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
```

See Also

`log-handler` | `log-root`

Topics

“Log Files” on page 4-6

“Manage Log Files” on page 4-7

Introduced in R2020a

mcr-root

Location of a MATLAB Runtime installation

Syntax

```
--mcr-root path
```

Description

`mcr-root` specifies the location of an installed MATLAB Runtime instance. If multiple MATLAB Runtime installations are available, then specify each installation on a separate line, starting from the latest version to the oldest.

Note Specifying multiple MATLAB Runtime installations allows one MATLAB Production Server instance to support multiple versions of the MATLAB Runtime. Specifying multiple MATLAB Runtime installations of the same version has no effect on performance.

If multiple `mcr-root` settings are present, then the server uses dynamic worker pool management, where worker processes are started in response to demand and shut down in response to system resource utilization.

The server instance scans the list of provided MATLAB Runtime installations 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 deployable archive containing the function being evaluated.

Note

- A server instance should only be configured to use MATLAB Runtime roots on a local file system. Otherwise, a network partition may cause worker processes to fail.
 - All values for `mcr-root` must be for the same OS/hardware combination.
-

Parameters

path

Path to the root folder of the MATLAB Runtime installation.

Note The special value `mCRuNSETtOKEN` indicates to the `mps-start` command that there is no MATLAB Runtime installation configured for this instance. Running the `mps-start` command results in an error.

Examples

Use the v80 version of the MATLAB Runtime.


```
--mcr-root /usr/local/MCR/v80
```

Use the v81 and v80 versions of the MATLAB Runtime.

```
--mcr-root /usr/local/MCR/v81
```

```
--mcr-root /usr/local/MCR/v80
```

See Also

Topics

“Specify the MATLAB Runtime for a Server Instance” on page 1-9

“Support Multiple MATLAB Versions” on page 1-12

num-threads

Number of request-processing threads within the server instance

Syntax

```
--num-threads count
```

Description

`num-threads` sets the size of the thread pool available to process 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 server main 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

Create a pool of 10 threads for processing requests.

```
--num-threads 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 can be processed simultaneously. 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, this setting determines the fixed size of the worker pool.

If you specify more than one value for the `mcr-root`, this setting specifies a maximum limit on the size of each subpool specific to MATLAB Runtime. There can be more than specified number of worker processes at a time, but at a maximum only the specified number of workers are allowed to be processing a request.

Parameters

count

Number of workers available 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

Allow 10 workers to process requests at a time.

```
--num-workers 10
```

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

Turn profiling on or off

Syntax

```
--profile state object
```

Description

`profile` turns profiling on or off for different objects.

Note Activating profiling has a negative impact on performance.

In some circumstances, `worker_2.pid` may be present when `worker_1.pid` is not. This is a strong indication that `worker_1` stopped and was restarted automatically. You can confirm this by checking the main log file.

When profiling is activated, messages similar to the following are included in the log.

```
12 [2014.02.27 10:13:28.075126] [profile] [SERVICE:http-connection]
[endpoint:[::]:9910] [client:163.72.158.2:57611] [request-id:0:1:5]
[type:arrive] [component:mymagic] [function:magic]
Request arrived and was placed in the queue
13 [2014.02.27 10:13:28.087752] [profile] [SERVICE:http-connection]
[endpoint:[::]:9910] [client:163.72.158.2:57611] [request-id:0:1:5]
[type:start] [worker:3] Request started executing on worker-3
...
15 [2014.02.27 10:13:31.397266] [profile] [SERVICE:http-connection]
[endpoint:[::]:9910] [client:163.72.158.2:57611] [request-id:0:1:5]
[type:finish] [status:200] Request completed with HTTP status 200
```

Parameters

state

Specifies if profiling is active. Valid values are:

- `on` — Activate profiling.
- `off` — Activate profiling.

object

The list of objects to change. Supported objects are:

- `requests`
- `worker_pool`

If no object is specified, all objects are changed.

Examples

Turn on request profiling.

```
--profile on requests
```

Turn on profiling for all objects.

```
--profile on
```

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 specified by *size*. The default request size is 64MB.

Parameters

size

Size, in bytes, of the request.

Examples

Set the request size to 128MB.

```
--request-size-limit 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.

If there are no archive names following the common name, the client can access all of the deployed archives. Otherwise, the client can access only the specified archives.

Parameters

client

Common name of the client.

archive

Name of an archive the clients 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
```


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 3-3

“Adjust Security Protocols” on page 3-7

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

Allow only TLSv1.

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

See Also

[https](#) | [ssl-ciphers](#)

Topics

“Enable HTTPS” on page 3-3

“Adjust Security Protocols” on page 3-7

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

Require clients to provide a certificate.

```
--ssl-verify-peer-mode verify-peer-require-peer-cert
```

See Also

[https](#) | [x509-ca-file-store](#) | [x509-use-crl](#) | [x509-use-system-store](#)

Topics

“Configure Client Authentication” on page 3-5

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

Start the MATLAB Runtime with a single computational thread.

```
--use-single-comp-thread
```

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

Topics

“Control Worker Restarts” on page 1-14

worker-restart-interval

Time interval at which a server instance stops and restarts its workers

Syntax

```
--worker-restart-interval hr:min:sec.fractSec
```

Description

`worker-restart-interval` specifies the interval at which the server instance stops and restarts its worker processes. If this setting is not given, the workers are not restarted 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.

```
--worker-restart-interval 1:29:05
```

Restart workers at intervals of 10 minutes and 250 ms.

```
--worker-restart-interval 00:10:00.25
```

See Also

Topics

“Control Worker Restarts” on page 1-14

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-memory-check-interval` | `worker-restart-memory-limit-interval`

Topics

“Control Worker Restarts” on page 1-14

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-memory-check-interval` | `worker-restart-memory-limit`

Topics

“Control Worker Restarts” on page 1-14

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

The instance loads the CA store from `ca_file.pem` which is located at `instance_root/x509`.

```
--x509-ca-file-store ./x509/ca_file.pem
```

See Also

[https](#) | [ssl-verify-peer-mode](#) | [x509-use-crl](#) | [x509-use-system-store](#)

Topics

“Configure Client Authentication” on page 3-5

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

The instance loads the certificate chain from `cert_chain.pem` which is located at `instance_root/x509`.

```
--x509-cert-chain ./x509/cert_chain.pem
```

See Also

`https` | `x509-private-key`

Topics

“Enable HTTPS” on page 3-3

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

The instance loads the private key from `private_key.pem`, which is located at `instance_root/x509`.

```
--x509-private-key ./x509/private_key.pem
```

See Also

[https](#) | [x509-cert-chain](#)

Topics

“Enable HTTPS” on page 3-3

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 3-5

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

The instance uses the system CA store.

```
--x509-use-system-store
```

See Also

[https](#) | [ssl-verify-peer-mode](#) | [x509-ca-file-store](#) | [x509-use-crl](#)

Topics

“Configure Client Authentication” on page 3-5

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

Set the request to time-out after 2 hours.

```
--request-timeout 2:00:00
```

See Also

`server-memory-threshold`

Introduced in R2016b

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 a server process's size exceeds `SIZE` set by `server-memory-threshold`, 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

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-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 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
```

See Also

`mps-stop`

Introduced in R2020a

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
```

See Also

`response-archive-root`

Introduced in R2016b

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."
```

See Also

Introduced in R2016a

Cloud Deployment

Azure Deployment

To deploy your MATLAB Production Server (BYOL) environment on Azure, launch the Azure solution template to provision cloud resources. Once deployment to Azure is complete, use MATLAB Production Server on Azure by logging in to the MATLAB Production Server cloud console and uploading your licensing file.

Note Before you can deploy resources on Azure and configure your server environment, you need an Azure account and 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 “Choose an Option to Activate Your License” (Licensing on the Cloud).

You are responsible for the cost of the Azure services and resources used in the deployment.

Follow these steps to deploy your server environment on Azure.

- 1 Click **Create** on the MATLAB Production Server (BYOL) offering page in the Azure portal. This 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 “Connect and Log In to Cloud Console” on page 9-7.
- 4 “Upload a License File” on page 9-14 using the cloud console.

You are now ready to use MATLAB Production Server (BYOL) on Azure.

To run applications on MATLAB Production Server, you will need to create applications using MATLAB Compiler SDK. For more information, see “Create a Deployable Archive for MATLAB Production Server”.

If your solution uses persistence, see “Edit the Azure Cache for Redis Configuration” on page 9-12.

Provision Cloud Resources

You need an Azure subscription before you start deploying cloud resources for MATLAB Production Server (BYOL) on Azure. Launch the MATLAB Production Server (BYOL) solution template to configure and deploy cloud resources. 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	<p>Choose a name for the resource group to hold the resources.</p> <p>Creating a new resource group for each deployment is recommended. Doing so enables you to delete all the resources for each deployment easily.</p>
Location	<p>Choose the location to start resources in.</p> <p>Select a location that supports your requested virtual machine (VM) instance types. For a list of resources that are supported, see Azure Region Services.</p>

Server

Next, you configure the server VM and data persistence. Each MATLAB Production Server instance runs on a VM and each instance runs multiple 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	<p>Specify the size of the VM to use for the server.</p> <p>Select a VM size where the number of cores on your VM matches the number of workers per VM that you plan on using. The template defaults to <code>Standard_D4s_v3</code>. This configuration has 4 vCPUs and 16 GB of memory. For more information on Azure VMs, see Azure documentation.</p>

Parameter Name	Value
Number of Server VMs	<p>Specify the number of VMs to run MATLAB Production Server instances.</p> <p>The deployment template sets the default to 2 VMs for load balancing. Because the number of server instances is limited to 24, you can provision a maximum of 24 VMs.</p> <p>For example, if you have a standard 24 worker MATLAB Production Server license and select <code>Standard_D4s_v3</code> VM (4 cores) as the Server VM Size, you need 6 VMs to fully utilize the workers in your license.</p> <p>You can always underprovision the number of VMs. If you do so, you can end up using fewer workers than you are licensed for.</p> <p>You can change the number of VMs after the initial deployment. For more information, see “Change the Number of Virtual Machines” on page 9-16.</p>
Server VM Operating System	<p>Choose the server VM operating system.</p> <p>Windows (Windows Server) and Linux (Ubuntu®) are the only available options.</p>
Create Azure Cache for Redis	<p>Choose whether you want to create an Azure cache for Redis.</p> <p>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 “Use a Data Cache to Persist Data”.</p> <p>You can provision an Azure cache for Redis after the initial deployment. For setup instructions, see Azure documentation.</p>

Cloud Console Login

After you deploy the server VMs, you can manage the server using the MATLAB Production Server cloud console, which provides a web-based interface to configure and manage server instances on the cloud. Specify the login credentials for the cloud console.

Parameter Name	Value
Admin Username	Specify the admin user name to log in to the MATLAB Production Server cloud console.

Parameter Name	Value
Admin Password	Specify the admin password to log in to the MATLAB Production Server cloud console.

Network

You can specify which IP addresses can access the cloud console, whether your solution should use a public IP address, and configure a virtual network (VNet).

Parameter Name	Value
Allow Connections From IP address	<p>Specify the IP address or range of IP addresses that is permitted to connect to the cloud console that manages the server.</p> <p>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.</p> <p>Example: 10.0.0.1/24</p> <p>You can use a CIDR calculator to determine the CIDR notation for a range of IP addresses.</p>
Make Solution Available over Internet	<p>Make your solution available over the Internet by setting this parameter to Yes. This will use public IP addresses.</p> <p>If you set this parameter to No, the ARM template does not assign a public IP address for the VM that hosts the cloud console and license server.</p>

Parameter Name	Value
Virtual Network	<p>Create a new VNet or choose an existing one.</p> <p>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.</p> <p>If you are using an existing VNet, you need to open the following ports.</p> <ul style="list-style-type: none"> • 443 - Required for communicating with the cloud console. • 8000, 8002, and 9910 - Required for communication between the cloud console and workers within the virtual network. These ports do not need to be accessible over the Internet. • 27000 and 50115 - Required for communication between the network license manager and the 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. • 3389 - Required for Remote Desktop functionality. Use remote desktop functionality for troubleshooting and debugging.
Subnets	<p>Specify the subnet name and subnet address prefix for a new or existing subnet.</p> <p>The first subnet hosts the cloud console. The second subnet hosts the application gateway.</p> <p>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.</p>

After you enter all the values required to configure the server environment, Azure runs a final validation. Click **OK** after validation passes.

Accept the Terms and Conditions. Click **Create** to begin the deployment. The deployment can take up to 30 minutes.

After the deployment finishes, the next step is to connect and login to the cloud console.

Connect and Log In to Cloud Console

The MATLAB Production Server cloud console is a web-based interface to configure and manage server instances on the cloud.

Note The Internet Explorer® web browser is not supported for interacting with the cloud console.

Connect and log in to the cloud console. Complete these steps only after your resource group has been successfully created. This workflow assumes that your solution uses public IP addresses.

- 1 In the Azure portal, click **Resource groups**.
- 2 Select the resource group you created for this deployment.
- 3 Select the resource labeled **servermachine-public-ip**. This resource contains the public IP address of the MATLAB Production Server cloud console.
- 4 Copy the IP address from the **Public IP address** field.
- 5 In your browser, connect to the cloud console using the IP address. Example: `https://11.22.135.137`
- 6 Use the admin user name and password that you specified in “Cloud Console Login” on page 9-4 to log in.

The cloud console uses a self-signed certificate, which you can change. For information on changing the self-signed certificate, see “Upload an HTTPS Certificate” on page 9-14.

The next step in the deployment is to upload a MATLAB Production Server license file.

Upload License File

Use the MATLAB Production Server cloud console to upload your MATLAB Production Server license file. You need a fixed MAC address to get a license file from the MathWorks® License Center. The license server MAC address for MATLAB Production Server on Azure is available only after you deploy the solution. For more information on configuring your license on the cloud, see “Choose an Option to Activate Your License” (Licensing on the Cloud).

Upload a new license file or replace an existing license file.

- 1 In the cloud console, select **Administration > License**.
- 2 Click **Browse License File**, select the license file you want to upload, and click **Open**.
- 3 Click **Upload**.

See Also

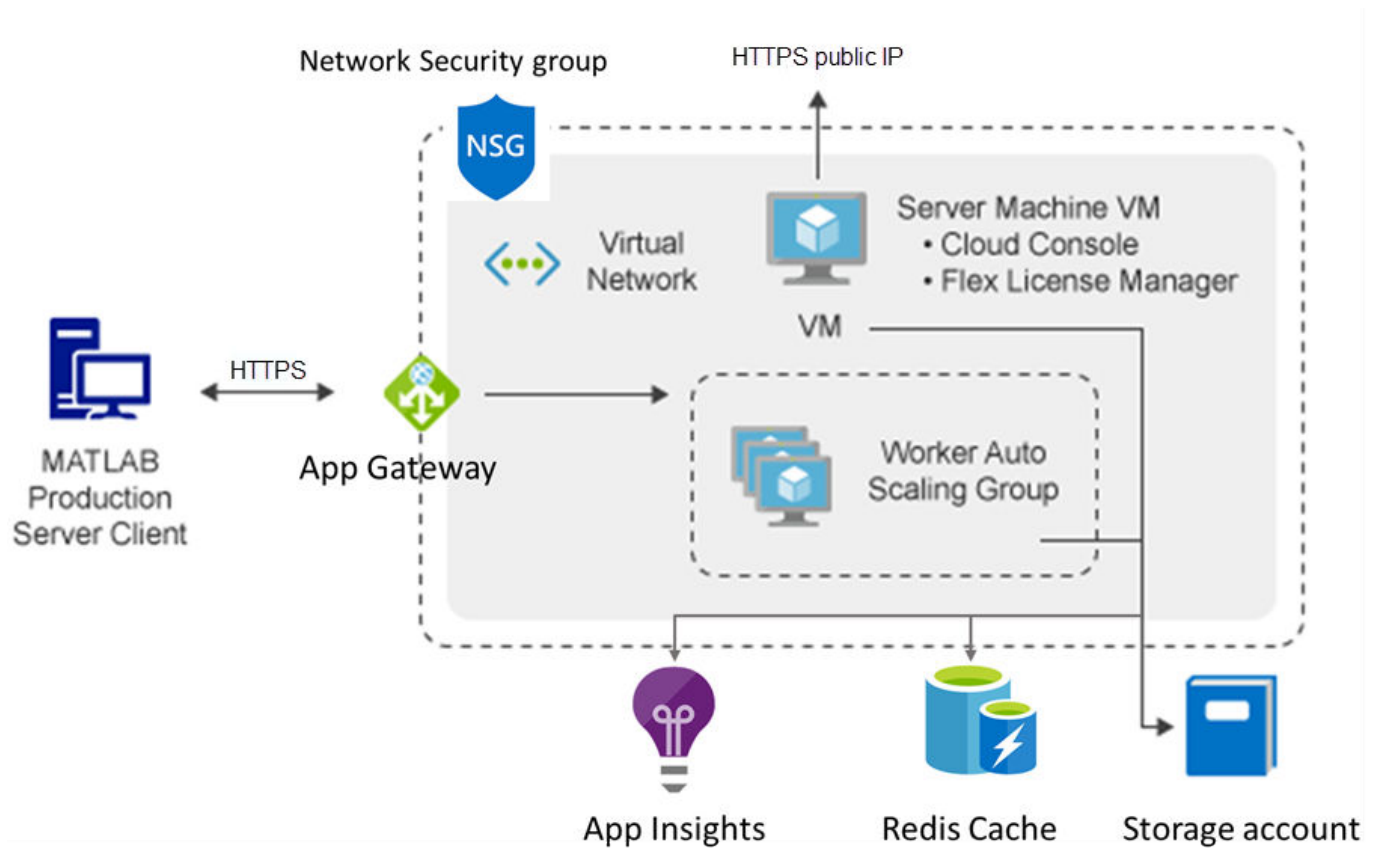
More About

- “Architecture and Resources on Azure” on page 9-8
- “Use MATLAB Production Server Cloud Console” on page 9-11
- “Configure Azure Resources” on page 9-16

Architecture and Resources on Azure

Deploying MATLAB Production Server on Azure creates several resources in your resource group. An architecture diagram and a description of resources follows.

MATLAB Production Server Architecture on Azure



Azure Resources

The following Azure resources are created in your resource group at the time of deployment.

Resource Name	Resource Name in Azure	Description
MATLAB Production Server cloud console and License Manager virtual machine	servermachine	<p>Virtual machine (VM) that hosts the MATLAB Production Server cloud console and License Manager. Use the cloud console to:</p> <ul style="list-style-type: none"> • Upload your license file and start using the server. • Get HTTPS endpoints to make requests. • Upload applications (CTF files) to the server. • Manage server configurations. • Manage the HTTPS certificate. <p>For more information about the cloud console, see “Use MATLAB Production Server Cloud Console” on page 9-11.</p>
MATLAB Production Server cloud console public IP	servermachine-public-ip	<p>Public IP address to connect to MATLAB Production Server cloud console.</p> <p>The address provides an HTTPS endpoint to the cloud console for managing server instances.</p> <p>If you choose to not use public IP addresses during deployment, the ARM template does not create this resource.</p>
Virtual machine scale set	vmss<uniqueID>	<p>Number of identical VMs to be deployed. Each VM runs an instance of MATLAB Production Server that in turn runs multiple MATLAB workers.</p> <p>For information on how to change the number of VMs, see “Change the Number of Virtual Machines” on page 9-16.</p>

Resource Name	Resource Name in Azure	Description
Application gateway	vmss<uniqueID>-agw	<p>Load balancer for routing traffic to MATLAB Production Server instances.</p> <p>The MATLAB Production Server cloud console 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.</p>
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 enables resources to communicate with each other.
Azure Cache for Redis	vmss<uniqueID>redis	<p>Redis cache that enables caching of data between calls to MATLAB code running on a server instance.</p> <p>For information on how to configure Azure Cache for Redis, see Azure documentation.</p>
Application Insights	logs-apmservice	<p>Application performance management service that enables storing and viewing of all logs associated with deployment.</p> <p>For information on how to view the logs, see “View Logs” on page 9-17.</p>

See Also

More About

- “Azure Deployment” on page 9-2
- “Use MATLAB Production Server Cloud Console” on page 9-11
- “Configure Azure Resources” on page 9-16

Use MATLAB Production Server Cloud Console

After you deploy a MATLAB Production Server environment in Azure, use the MATLAB Production Server cloud console to manage server instances. The cloud console is a web-based interface to configure and manage server instances on the cloud.

For information on how to connect to the cloud console, see “Connect and Log In to Cloud Console” on page 9-7.

View Information About Server Instances

View information about the server instance virtual machines (VMs) deployed on Azure, click **Home** on the cloud console navigation menu.

The screenshot shows the MATLAB Production Server Cloud Console interface. At the top, there is a navigation bar with the MathWorks logo and links for Home, Applications, Administration, and About. On the right side of the navigation bar, there are links for Hello and Log out. Below the navigation bar, the main content area displays the following information:

Status:	Running
Number of MATLAB Production Server VMs:	2
Number of MATLAB Production Server Workers per VM:	4
Total Number of Workers:	8
HTTPS Server Endpoint: i	https://mps23ikwokzajmxa.eastus.cloudapp.azure.com

Below the table, there is a section titled "Additional Information" with the following text:

To start using the server:

1. Get a license from the [MathWorks License Center](#) and upload it in the [Manage License](#) section.
2. Use the HTTPS server endpoint to make requests to the server.

View HTTPS Endpoint

When you run MATLAB Production Server on the cloud, you get two HTTPS endpoints that use self-signed certificates. One of the HTTPS endpoints provides access to the application gateway that connects the server instances. This endpoint is displayed on the home page of the cloud console and is used to make requests to the server.

- 1 On the cloud console navigation menu, click **Home**.
- 2 Copy the parameter value listed next to **HTTPS Server Endpoint**.

Upload a MATLAB Application

You can run an application on MATLAB Production Server that is created using MATLAB Compiler SDK. Upload the application using the cloud console.

- 1 On the cloud console navigation menu, click **Applications**.
- 2 Click **+Upload Application**.

- 3 Click **Browse CTF File**, select the file, and click **Upload**.

For information on how to create an application, see “Create a Deployable Archive for MATLAB Production Server”.

Edit the Server Configuration

You can edit the MATLAB Production Server configuration properties.

- 1 On the cloud console navigation menu, select **Administration > Server Configuration**.
- 2 Find the server property you want to change and enter the appropriate value. To assign a value to a property that has been commented out, remove the hash symbol and assign a value.

- Use this configuration to allow requests from specific domains.

```
--cors-allowed-origins http://www.w3.org, https://www.apache.org
```

- Use this configuration to set the number of workers to 4.

```
--num-workers 4
```

Note When setting the num-workers property, you need to carefully consider your cluster setup. Each VM in the cluster runs an instance of MATLAB Production Server and each instance runs multiple MATLAB workers. Using 1 core per MATLAB worker is recommended. For example, if the server size is set to *Standard_D4s_v3 Server*, which specifies 4 cores, set num-workers be no more than 4 workers per instance.

Edit the Azure Cache for Redis Configuration

MATLAB Production Server uses Redis for data persistence. Persistence allows caching of data between calls to MATLAB code running on a server instance.

To use the Azure Cache for Redis from your deployment, you must specify certain parameters in the configuration. To edit the Azure cache for Redis configuration, in the cloud console navigation menu, select **Administration > Persistence Configuration**.

Specify the cache configuration in JSON format as follows.

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

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.
<hostname> and <port_number>	<p>Specify the host name and port number for the 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.</p> <ul style="list-style-type: none"> • 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 <code>vmss<uniqueID>redis</code>. • Select Overview and copy the values listed under Host name and under Ports. <p>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.</p>
<access_key>	<p>To retrieve an access key to connect to an Azure cache for Redis resource, follow these steps.</p> <ul style="list-style-type: none"> • 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 <code>vmss<uniqueID>redis</code>. • 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 “Use a Data Cache to Persist Data”.

Get the License Server MAC Address

You can get the license server MAC address only after deploying the solution to the cloud. You need a fixed MAC address to get a license file from the MathWorks License Center.

- 1 In the cloud console navigation menu, select **Administration > License**.
- 2 Copy the license server MAC address displayed at the top.

Upload a License File

You can upload the license file only after provisioning the cloud resources. You can use MATLAB Production Server on Azure only after you upload a license file.

- 1 In the cloud console navigation menu, select **Administration > License**.
- 2 Click **Browse License File** and select your license file.
- 3 Click **Upload**.

Upload an HTTPS Certificate

You can change the self-signed certificate to the HTTPS endpoint that lets you connect to the cloud console. The server automatically restarts after uploading a certificate. You need to log out and log back in. When you upload a new certificate, you lose all pending requests.

- 1 In the cloud console navigation menu, select **Administration > HTTPS Certificate**.
- 2 Click **Browse Certificate** and select a certificate file. The solution supports PFX files only.
- 3 Enter the certificate password in the **Certificate Password** field.
- 4 Click **Upload**.

Set Up Authentication Using Azure Active Directory

The solution uses Azure Active Directory® (Azure AD) to provide an identity to each user. To use Azure AD, you must specify the following:

- Access control configuration file
 - Access control policy file
- 1 In the cloud console navigation menu, select **Administration > Azure AD**.
 - 2 Enter the details of each file and click **Save and Apply Configuration**.
 - 3 “Edit the Server Configuration” on page 9-12 and enable the option `access-control-provider`.

Sample code for the access control configuration file follows.

```
{
  "tenantId": "54ss4lk1-8428-7256-5fvh-d5785gfhkjh6",
  "serverAppId": "j21n12bg-3758-3r78-v25j-35yj4c47vhmt",
  "jwksUri": "https://login.microsoftonline.com/common/discovery/keys",
  "issuerBaseUri": "https://sts.windows.net/",
  "jwksTimeout": 120
}
```

Sample code for the access control policy file follows.

```
{
  "version": "1.0.0",
  "policy" : [
    {
      "id": "policy1",
      "description": "MPS Access Control policy for XYZ Corp.",
      "rule": [
```


Configure Azure Resources

After you deploy the MATLAB Production Server 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 workers. For example, if you have a standard 24 worker MATLAB Production Server license and select the `Standard_D4s_v3` VM (4 cores) 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 Self-Signed Certificate to the Application Gateway

When you run MATLAB Production Server on the cloud you get two HTTPS endpoints that use self-signed certificates. One of the HTTPS endpoints provides access to the application gateway that connects the server instances. Make requests to the server by using the application gateway HTTPS endpoint. To change the self-signed certificate to the application gateway, use the Azure portal to create a listener and add a rule.

Create a 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 **+Basic** on the top pane.
- 4 Enter values for these parameters.

Name	Value
Listener name	Enter a name for the listener.
Frontend IP	Choose Public if your solution uses public IPs; otherwise, choose Private .
Port	Enter any free port between 8001 and 10000.
Protocol	Select HTTPS .

- 5 Select **Create New** in **Choose a Certificate** dropdown, then select **Upload a certificate**.
- 6 Upload a PFX certificate, give it a name, and create a password. Click **Add**.

Add a Rule

- 1 Log in to the Azure portal and select the resource group associated with this deployment.
- 2 Select the application gateway resource with the name vmss<uniqueID>-agw.
- 3 Select **Rules** from the left pane and click **+Basic** on the top pane.
- 4 Enter values for these parameters.

Name	Value
Name	Enter a name for the rule.
Listener	Select the listener you created in "Create a Listener" on page 9-16.
Configure redirection	Select the box if appropriate.
Backend pool	Select the available value.
HTTP setting	Select the available value.

For information on changing the self-signed certificate to connect to the cloud console, see "Upload an HTTPS Certificate" on page 9-14.

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 **Performance** from the left pane of the resource.
- 4 Click the **Roles** tab from the top pane.
- 5 Select **View in Logs (Analytics)** from the top pane.
- 6 A new query tab opens with an existing query. Ignore this default query.
- 7 Create a new query by clicking the **+** icon next to the existing query.
- 8 Paste the following query in the empty section.

```
traces
| where customDimensions.source == "cloudconsole" or customDimensions.source == "mpsNative"
| limit 200
```

The query returns logs from three resources.

- mpsNative- Log of activity generated by the server instance.
- mpsSlaveController- Log of user logins, deployed archives (CTF files), certificate changes, and user interface actions.
- mpsInstances- Log of orchestrator code for server virtual machines.

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.

Reset Cloud Console Password

The password to log in to the cloud console is the same as the password to log in to the Azure VM that hosts the cloud console. You can reset your password.

- 1 Log in to the Azure portal and select the resource group associated with this deployment.
- 2 Select the resource `servermachine`.
- 3 Click **Reset Password** on the left pane.
- 4 Enter your new password and click **Update**. Enter the user name that you used during the initial deployment.
- 5 Click **Restart** on the top pane to restart the VM.
- 6 You can now log in to the cloud console using your new password.

See Also

More About

- “Azure Deployment” on page 9-2
- “Architecture and Resources on Azure” on page 9-8
- “Use MATLAB Production Server Cloud Console” on page 9-11

Calculate Number of Workers

In the MATLAB Production Server on Azure environment, each server instance runs on a virtual machine (VM) and each instance runs multiple workers. The size of the VM you choose for your server depends on the number of workers. We recommend you choose a server VM size where the number of cores on your VM match the number of workers per VM that you plan on using.

The number of workers you need depends on

- 1 Number of concurrent requests to the server.

If you do not know the number of concurrent requests, you can use 10%.

- 2 How long your deployed MATLAB function takes to execute.

If you have written the function, then time it within MATLAB using `tic` and `toc`.

If you have not written the function yet, then you can assume that a simple function typically takes less than one second, an intermediate function takes between one to four seconds, and a complex function takes between five to eight seconds to run.

- 3 How long will your users be willing to wait to view the results.

If you do not know, you can use five seconds or less. This value cannot be less than the value in the above condition.

You can then calculate the number of workers you need for a particular deployed function using

Repeat this process for each MATLAB function that you want to deploy to the server and then aggregate the number of workers for all the functions to get the total number of workers that you need.

For example, if you want to deploy two functions to your MATLAB Production Server

- 1 Function A will be used by 100 users, takes one second to execute and you do not want the users to wait more than two seconds to get a response.
- 2 Function B will be used by 50 users, takes four seconds to execute, and you do not want the users to wait more than 5 seconds to get a response.

In this case, the number of workers required for the server is $5 + 4 = 9$ workers.

See Also

