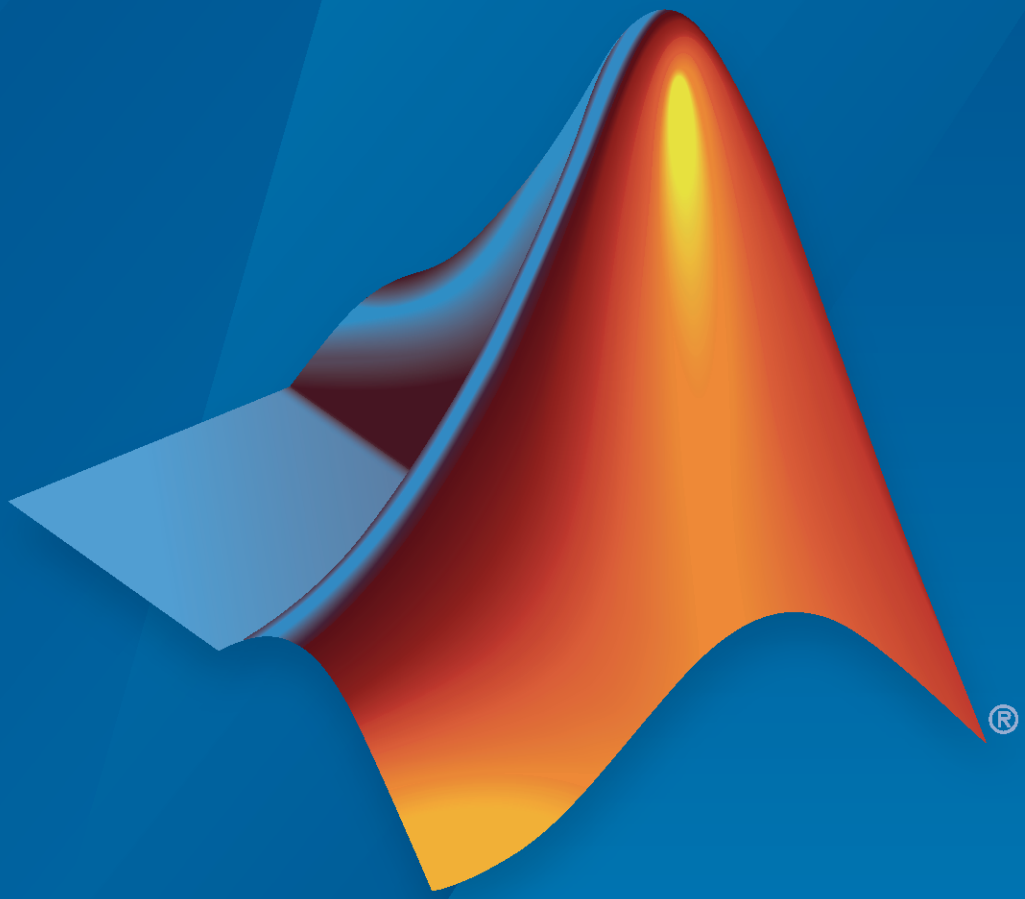
# MATLAB® Compiler™

## Hadoop® Integration Guide



MATLAB®

R2023a

MathWorks®

# How to Contact MathWorks

Latest news: `www.mathworks.com`

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

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

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

Phone: 508-647-7000

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

*MATLAB® Compiler™ Hadoop® Integration Guide*



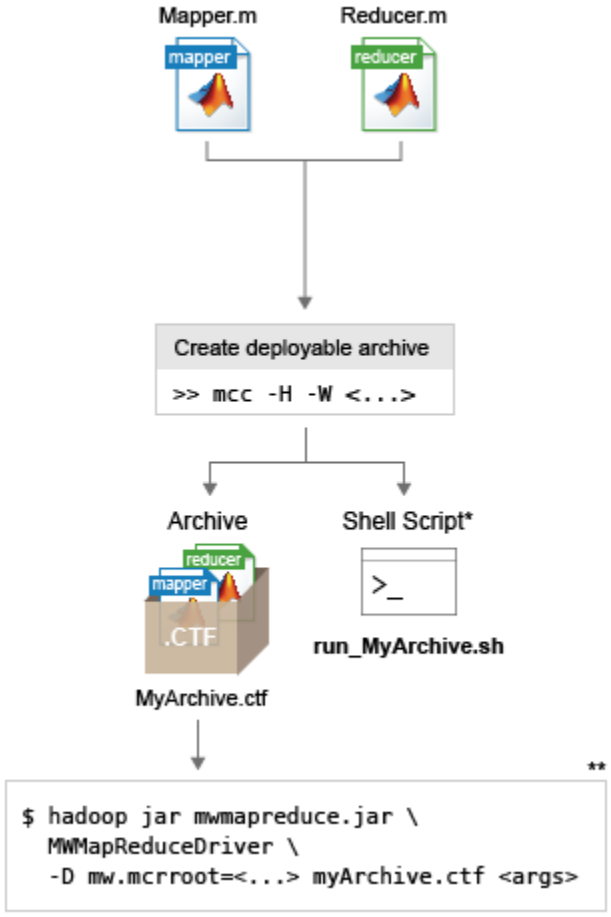
**Revision History**

| | | |
|---|---|---|
| October 2014 | Online only | New for Version 5.2 (Release 2014b) |
| March 2015 | Online only | Revised for Version 6.0 (Release 2015a) |
| September 2015 | Online only | Revised for Version 6.1 (Release 2015b) |
| October 2015 | Online only | Rereleased for Version 6.0.1 (Release 2015aSP1) |
| March 2016 | Online only | Revised for Version 6.2 (Release 2016a) |
| September 2016 | Online Only | Revised for Version 6.3 (Release 2016b) |
| March 2017 | Online only | Revised for Version 6.4 (Release R2017a) |
| September 2017 | Online only | Revised for Version 6.5 (Release R2017b) |
| March 2018 | Online only | Revised for Version 6.6 (Release R2018a) |
| September 2018 | Online only | Revised for Version 7.0 (Release R2018b) |
| March 2019 | Online only | Revised for Version 7.0.1 (Release R2019a) |
| September 2019 | Online only | Revised for Version 7.1 (Release R2019b) |
| March 2020 | Online only | Revised for Version 8.0 (Release R2020a) |
| September 2020 | Online only | Revised for Version 8.1 (Release R2020b) |
| March 2021 | Online only | Revised for Version 8.2 (Release R2021a) |
| September 2021 | Online only | Revised for Version 8.3 (Release R2021b) |
| March 2022 | Online only | Revised for Version 8.4 (Release R2022a) |
| September 2022 | Online only | Revised for Version 8.5 (Release R2022b) |
| March 2023 | Online only | Revised for Version 8.6 (Release R2023a) |

# Contents

# Deployable Archives

# Workflow to Incorporate MATLAB Map and Reduce Functions into a Hadoop Job




* You can use automatically generated shell scripts to execute applications from the terminal.

** Commands are not exact. For complete commands, see the auto-generated shell scripts.

1. Write mapper and reducer functions in MATLAB.
2. Create a MAT-file that contains a datastore that describes the structure of the data and the names of the variables to analyze. The datastore in the MAT-file can be created from a test data set that is representative of the actual data set.
3. Create a text file that contains Hadoop settings such as the name of the mapper, reducer, and the type of data being analyzed. This file is automatically created if you are using the **Hadoop Compiler** app.
4. Use the **Hadoop Compiler** app or the `mcc` command to package the components into a deployable archive. Both options generate a deployable archive (.ctf file) that can be incorporated into a Hadoop mapreduce job.

5 Incorporate the deployable archive into a Hadoop mapreduce job using the `hadoop` command and syntax.

**Execution Signature**

```
A  $ hadoop \
B    jar \
C    /<MATLAB_Runtime_Location>/v91/toolbox/mlhadoop/jar/a2.2.0/mwmapreduce.jar \
D    com.mathworks.hadoop.MWMapReduceDriver \
E    -D mw.mcrroot=<MATLAB_Runtime_Location> \
F    MapRedDeployableArchive.ctf \
G    <inputFolderOnHDFS> \
H    <outputFolderOnHDFS>
```

*Key*

| Letter | Description |
|---|---|
| A | Hadoop command |
| B | JAR option |
| C | The standard name of the JAR file. All applications have the same JAR: `mwmapreduce.jar`.The path to the JAR is also fixed relative to the MATLAB Runtime location. |
| D | The standard name of the driver. All applications have the same driver name: `MWMapReduceDriver` |
| E | A generic option specifying the MATLAB Runtime location as a key-value pair. |
| F | Deployable archive (`.ctf` file) generated by the **Hadoop Compiler** app or `mcc` is passed as a payload argument to the job. |
| G | Location of input files on HDFS™. |
| H | Location on HDFS where output can be written. |

To simplify the inclusion of the deployable archive (`.ctf` file) into a Hadoop mapreduce job, both the **Hadoop Compiler** app and the `mcc` command generate a shell script alongside the deployable archive. The shell script has the following naming convention: `run_<deployableArchiveName>.sh`

To run the deployable archive using the shell script, use the following syntax:

```
$./run_myDeployableArchive.sh\
<MATLAB_Runtime_Location>\
[hadoop_specific_properites] \
<inputFolderOnHDFS>\
<outputFolderOnHDFS>
```

## See Also

## Related Examples

- “Example Using the Hadoop Compiler App Workflow” on page 1-5
- “Include MATLAB Map and Reduce Functions into Hadoop Job” on page 1-9

# Example Using the Hadoop Compiler App Workflow

**Supported Platform:** Linux® only.

This example shows you how to use the **Hadoop Compiler** app to create a deployable archive consisting of MATLAB map and reduce functions and then pass the deployable archive as a payload argument to a job submitted to a Hadoop cluster.

**Goal:** Calculate the maximum arrival delay of an airline from the given dataset.

| | |
|---|---|
| **Dataset:** | `airlinesmall.csv` |
| **Description:** | Airline departure and arrival information from 1987-2008. |
| **Location:** | `/usr/local/MATLAB/R2023a/toolbox/matlab/demos` |

## Prerequisites

1. Start this example by creating a new work folder that is visible to the MATLAB search path.
2. Before starting MATLAB, at a terminal, set the environment variable `HADOOP_PREFIX` to point to the Hadoop installation folder. For example:

| Shell | Command |
|---|---|
| *csh / tcsh* | `% setenv HADOOP_PREFIX /usr/lib/hadoop` |
| *bash* | `$ export HADOOP_PREFIX=/usr/lib/hadoop` |

**Note** This example uses `/usr/lib/hadoop` as directory where Hadoop is installed. Your Hadoop installation directory maybe different.

If you forget setting the `HADOOP_PREFIX` environment variable prior to starting MATLAB, set it up using the MATLAB function `setenv` at the MATLAB command prompt as soon as you start MATLAB. For example:

```
setenv('HADOOP_PREFIX','/usr/lib/hadoop')
```

3. Install the MATLAB Runtime in a folder that is accessible by every worker node in the Hadoop cluster. This example uses `/usr/local/MATLAB/MATLAB_Runtime/R2023a` as the location of the MATLAB Runtime folder.

   If you don’t have the MATLAB Runtime, you can download it from the website at: `https://www.mathworks.com/products/compiler/mcr`.

**Note** For information about MATLAB Runtime version numbers corresponding MATLAB releases, see this list.

4. Copy the map function `maxArrivalDelayMapper.m` from `/usr/local/MATLAB/R2023a/toolbox/matlab/demos` folder to the work folder.

**maxArrivalDelayMapper.m**

```
function maxArrivalDelayMapper (data, info, intermKVStore)
partMax = max(data.ArrDelay);
add(intermKVStore,'PartialMaxArrivalDelay',partMax);
```

For more information, see “Write a Map Function”.

5 Copy the reduce function `maxArrivalDelayReducer.m` from *`matlabroot`*`/toolbox/matlab/demos` folder to the work folder.

**maxArrivalDelayReducer.m**

```
function maxArrivalDelayReducer(intermKey, intermValIter, outKVStore)
maxVal = -inf;
while hasnext(intermValIter)
   maxVal = max(getnext(intermValIter), maxVal);
end
add(outKVStore,'MaxArrivalDelay',maxVal);
```

For more information, see “Write a Reduce Function”.

6 Create the directory `/user/<`*`username`*`>/datasets` on HDFS and copy the file `airlinesmall.csv` to that directory. Here <*`username`*> refers to your user name in HDFS.

```
$ ./hadoop fs -copyFromLocal airlinesmall.csv hdfs://host:54310/user/<username>/datasets
```

## Procedure

1 Start MATLAB and verify that the `HADOOP_PREFIX` environment variable has been set. At the command prompt, type:

```
>> getenv('HADOOP_PREFIX')
```

If `ans` is empty, review the **Prerequisites** section above to see how you can set the `HADOOP_PREFIX` environment variable.

2 Create a `datastore` to the file `airlinesmall.csv` and save it to a `.mat` file. This `datastore` object is meant to capture the structure of your actual dataset on HDFS.

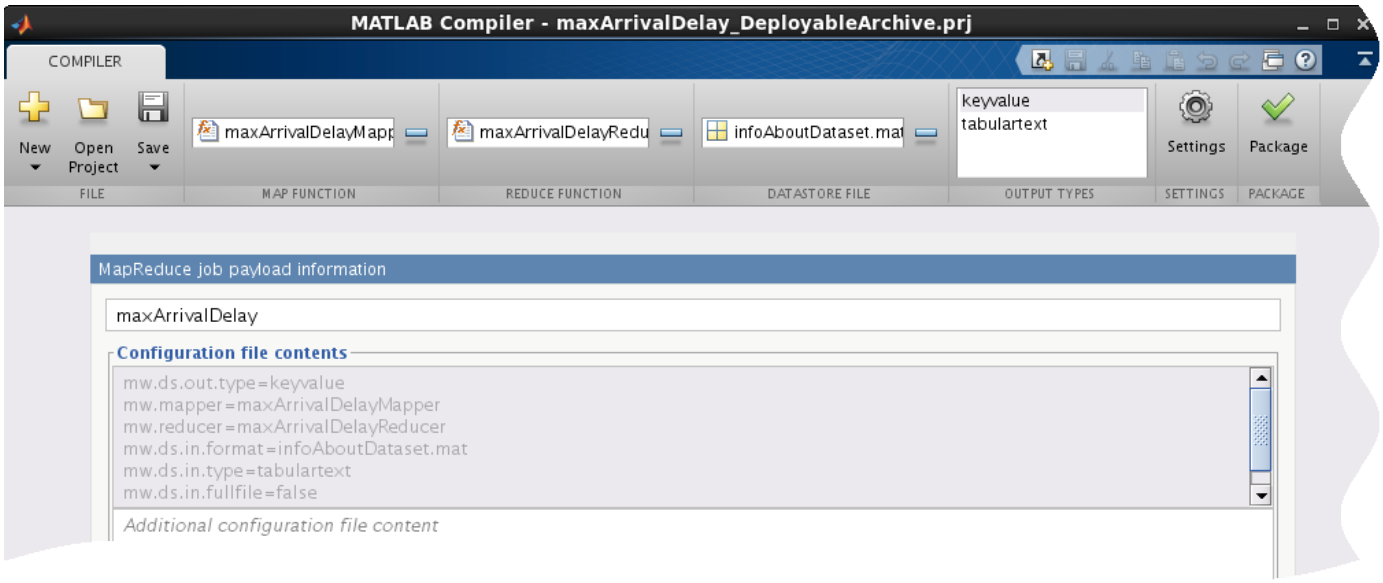
```
ds = datastore('airlinesmall.csv','TreatAsMissing','NA',...
     'SelectedVariableNames','ArrDelay','ReadSize',1000);

save('infoAboutDataset.mat','ds')
```

In most cases, you will start off by working on a small sample dataset residing on a local machine that is representative of the actual dataset on the cluster. This sample dataset has the same structure and variables as the actual dataset on the cluster. By creating a `datastore` object to the dataset residing on your local machine you are taking a snapshot of that structure. By having access to this `datastore` object, a Hadoop job executing on the cluster will know how to access and process the actual dataset residing on HDFS.

**Note** In this example, the sample dataset (local) and the actual dataset on HDFS are the same.

3 Launch the **Hadoop Compiler** app through the MATLAB command line (`>> hadoopCompiler`) or through the apps gallery.




4. In the **Map Function** section of the toolstrip, click the plus button to add mapper file `maxArrivalDelayMapper.m`.
5. In the **Reduce Function** section of the toolstrip, click the plus button to add reducer file `maxArrivalDelayReducer.m`.
6. In the **Datastore File** section, click the plus button to add the `.mat` file `infoAboutDataset.mat` containing the `datastore` object.
7. In the **Output Types** section, select `keyvalue` as output type. Selecting `keyvalue` as your output type means your results can only be read within MATLAB. If you want your results to be accessible outside of MATLAB, select output type as `tabulartext`.
8. Rename the **MapReduce job payload information** to `maxArrivalDelay`.
9. Click **Package** to build a deployable archive.

   The **Hadoop Compiler** app creates a log file `PackagingLog.txt` and two folders `for_redistribution` and `for_testing`.

| for_redistribution | for_testing |
|---|---|
| readme.txt | readme.txt |
| maxArrivalDelay.ctf | maxArrivalDelay.ctf |
| run_maxArrivalDelay.sh | run_maxArrivalDelay.sh |
| | mccExcludedFiles.log |
| | requiredMCRProducts.txt |

   You can use the log file `PackagingLog.txt` to see the exact `mcc` syntax used to package the deployable archive.
10. From a Linux shell navigate to the `for_redistribution` folder.
11. **a** Incorporate the deployable archive containing MATLAB map and reduce functions into a Hadoop mapreduce job from a Linux shell using the following command:

```
$ hadoop \
jar /usr/local/MATLAB/MATLAB_Runtime/R2023a/toolbox/mlhadoop/jar/a2.2.0/mwmapreduce.jar \
com.mathworks.hadoop.MWMapReduceDriver \
-D mw.mcrroot=/usr/local/MATLAB/MATLAB_Runtime/R2023a \
maxArrivalDelay.ctf \
```

```
hdfs://host:54310/user/<username>/datasets/airlinesmall.csv \
hdfs://host:54310/user/<username>/results
```

**b** Alternately, you can incorporate the deployable archive containing MATLAB map and reduce functions into a Hadoop mapreduce job using the shell script generated by the **Hadoop Compiler** app. At the Linux shell type the following command:

```
$ ./run_maxArrivalDelay.sh \
/usr/local/MATLAB/MATLAB_Runtime/R2023a \
-D mw.mcrroot=/usr/local/MATLAB/MATLAB_Runtime/R2023a \
hdfs://host:54310/user/username/datasets/airlinesmall.csv \
hdfs://host:54310/user/<username>/results
```

**12** To examine the results, switch to the MATLAB desktop and create a `datastore` to the results on HDFS. You can then view the results using the `read` method.

```
d = datastore('hdfs:///user/<username>/results/part*');
read(d)
```

```
ans =

          Key            Value
    ________________    ______

    'MaxArrivalDelay'    [1014]
```

Other examples of map and reduce functions are available at `toolbox/matlab/demos` folder. You can use other examples to prototype similar deployable archives to run on a Hadoop cluster. For more information, see “Build Effective Algorithms with MapReduce”.

## See Also

`datastore` | `TabularTextDatastore` | `KeyValueDatastore` | `deploytool`

## Related Examples

- “Include MATLAB Map and Reduce Functions into Hadoop Job” on page 1-9

# Include MATLAB Map and Reduce Functions into Hadoop Job

**Supported Platform:** Linux only.

This example shows you how to use the `mcc` command to create a deployable archive consisting of MATLAB map and reduce functions and then pass the deployable archive as a payload argument to a job submitted to a Hadoop cluster.

**Goal:** Calculate the maximum arrival delay of an airline from the given dataset.

| | |
|---|---|
| **Dataset:** | `airlinesmall.csv` |
| **Description:** | Airline departure and arrival information from 1987-2008. |
| **Location:** | `/usr/local/MATLAB/R2023a/toolbox/matlab/demos` |

**Note** When compared to the **Hadoop Compiler** app workflow, this workflow requires the explicit creation of a Hadoop settings file. Follow the example for details.

### Prerequisites

1. Start this example by creating a new work folder that is visible to the MATLAB search path.
2. Before starting MATLAB, at a terminal, set the environment variable `HADOOP_PREFIX` to point to the Hadoop installation folder. For example:

| Shell | Command |
|---|---|
| *csh / tcsh* | `% setenv HADOOP_PREFIX /usr/lib/hadoop` |
| *bash* | `$ export HADOOP_PREFIX=/usr/lib/hadoop` |

**Note** This example uses `/usr/lib/hadoop` as directory where Hadoop is installed. Your Hadoop installation directory maybe different.

If you forget setting the `HADOOP_PREFIX` environment variable prior to starting MATLAB, set it up using the MATLAB function `setenv` at the MATLAB command prompt as soon as you start MATLAB. For example:

```
setenv('HADOOP_PREFIX','/usr/lib/hadoop')
```

3. Install the MATLAB Runtime in a folder that is accessible by every worker node in the Hadoop cluster. This example uses `/usr/local/MATLAB/MATLAB_Runtime/R2023a` as the location of the MATLAB Runtime folder.

   If you don’t have the MATLAB Runtime, you can download it from the website at: `https://www.mathworks.com/products/compiler/mcr`.

**Note** For information about MATLAB Runtime version numbers corresponding MATLAB releases, see this list.

4. Copy the map function `maxArrivalDelayMapper.m` from `/usr/local/MATLAB/R2023a/toolbox/matlab/demos` folder to the work folder.

**maxArrivalDelayMapper.m**

```
function maxArrivalDelayMapper (data, info, intermKVStore)
partMax = max(data.ArrDelay);
add(intermKVStore,'PartialMaxArrivalDelay',partMax);
```

For more information, see “Write a Map Function”.

5 Copy the reduce function `maxArrivalDelayReducer.m` from *`matlabroot`*`/toolbox/matlab/demos` folder to the work folder.

**maxArrivalDelayReducer.m**

```
function maxArrivalDelayReducer(intermKey, intermValIter, outKVStore)
maxVal = -inf;
while hasnext(intermValIter)
   maxVal = max(getnext(intermValIter), maxVal);
end
add(outKVStore,'MaxArrivalDelay',maxVal);
```

For more information, see “Write a Reduce Function”.

6 Create the directory `/user/<`*`username`*`>/datasets` on HDFS and copy the file `airlinesmall.csv` to that directory. Here *<username>* refers to your user name in HDFS.

```
$ ./hadoop fs -copyFromLocal airlinesmall.csv hdfs://host:54310/user/<username>/datasets
```

**Procedure**

1 Start MATLAB and verify that the `HADOOP_PREFIX` environment variable has been set. At the command prompt, type:

```
>> getenv('HADOOP_PREFIX')
```

If `ans` is empty, review the **Prerequisites** section above to see how you can set the `HADOOP_PREFIX` environment variable.

2 Create a `datastore` to the file `airlinesmall.csv` and save it to a `.mat` file. This `datastore` object is meant to capture the structure of your actual dataset on HDFS.

```
ds = datastore('airlinesmall.csv','TreatAsMissing','NA',...
     'SelectedVariableNames','ArrDelay','ReadSize',1000);

save('infoAboutDataset.mat','ds')
```

In most cases, you will start off by working on a small sample dataset residing on a local machine that is representative of the actual dataset on the cluster. This sample dataset has the same structure and variables as the actual dataset on the cluster. By creating a `datastore` object to the dataset residing on your local machine you are taking a snapshot of that structure. By having access to this `datastore` object, a Hadoop job executing on the cluster will know how to access and process the actual dataset residing on HDFS.

**Note** In this example, the sample dataset (local) and the actual dataset on HDFS are the same.

3 Create a configuration file (`config.txt`) that specifies the input type of the data, the format of the data specified by the `datastore` created in the previous step, the output type of the data, the name of map function, and the name of reduce function.

```
mw.ds.in.type = tabulartext
mw.ds.in.format = infoAboutDataset.mat
```

```
mw.ds.out.type = keyvalue
mw.mapper = maxArrivalDelayMapper
mw.reducer = maxArrivalDelayReducer
```

For more information, see “Configuration File for Creating Deployable Archive Using the mcc Command” on page 3-2.

4 Use the `mcc` command with the `-H` and `-W` flags to create a deployable archive. However, the `mcc` command cannot package the results in an installer. The command must be entered as a single line.

```
mcc -H -W 'hadoop:maxArrivalDelay,CONFIG:config.txt'
  maxArrivalDelayMapper.m maxArrivalDelayReducer.m
   -a infoAboutDataset.mat
```

For more information, see `mcc`.

MATLAB Compiler creates a shell script `run_maxarrivaldelay.sh`, a deployable archive `airlinesmall.ctf`, and a log file `mccExcludedfiles.log`.

5 **a** Incorporate the deployable archive containing MATLAB map and reduce functions into a Hadoop mapreduce job from a Linux shell using the following command:

```
$ hadoop \
jar /usr/local/MATLAB/MATLAB_Runtime/R2023a/toolbox/mlhadoop/jar/a2.2.0/mwmapreduce.jar \
com.mathworks.hadoop.MWMapReduceDriver \
-D mw.mcrroot=/usr/local/MATLAB/MATLAB_Runtime/R2023a \
maxArrivalDelay.ctf \
hdfs://host:54310/user/<username>/datasets/airlinesmall.csv \
hdfs://host:54310/user/<username>/results
```

**b** Alternately, you can incorporate the deployable archive containing MATLAB map and reduce functions into a Hadoop mapreduce job using the shell script generated by the **Hadoop Compiler** app. At the Linux shell type the following command:

```
$ ./run_maxArrivalDelay.sh \
/usr/local/MATLAB/MATLAB_Runtime/R2023a \
-D mw.mcrroot=/usr/local/MATLAB/MATLAB_Runtime/R2023a \
hdfs://host:54310/user/username/datasets/airlinesmall.csv \
hdfs://host:54310/user/<username>/results
```

6 To examine the results, switch to the MATLAB desktop and create a `datastore` to the results on HDFS. You can then view the results using the `read` method.

```
d = datastore('hdfs:///user/<username>/results/part*');
read(d)

ans =

          Key            Value
    ___________________    ______

    'MaxArrivalDelay'    [1014]
```

Other examples of `map` and `reduce` functions are available at `toolbox/matlab/demos` folder. You can use other examples to prototype similar deployable archives that run against Hadoop. For more information, see “Build Effective Algorithms with MapReduce”.

## See Also

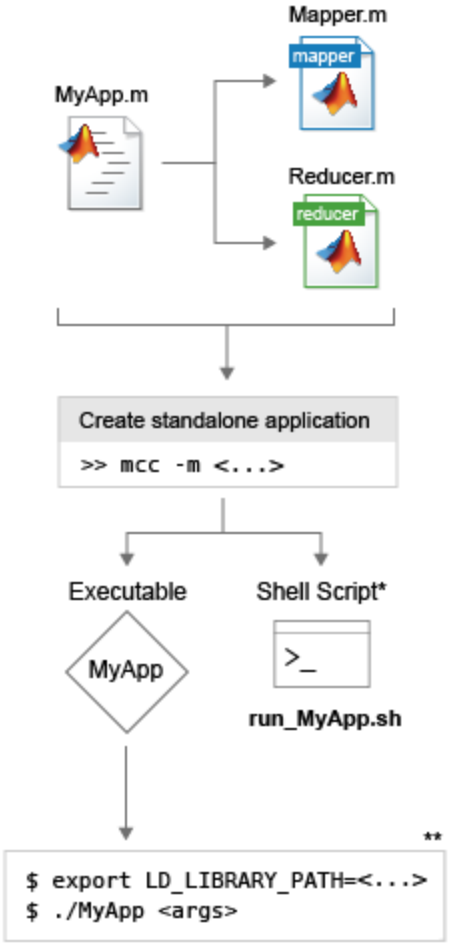
`datastore` | `TabularTextDatastore` | `KeyValueDatastore` | `mcc` | `deploytool`

## Related Examples

- “Example Using the Hadoop Compiler App Workflow” on page 1-5

# Standalone Applications

# Workflow to Run Compiled Standalone Applications Against a Hadoop Cluster




* You can use automatically generated shell scripts to execute applications from the terminal.

** Commands are not exact. For complete commands, see the auto-generated shell scripts.

1. Write mapper and reducer functions in MATLAB.
2. Write a MATLAB application script or function that calls the mapper and reducer functions. While writing applications it is preferable to structure them as MATLAB functions over scripts since functions accept inputs. End users can make use of this and pass inputs such as the location of the data to the application.
3. Use the **Application Compiler** app or the `mcc` command to package your application as a standalone application. Both options generate an executable and a shell script to run the executable.
4. Run the shell scripts at the terminal. Specify the location of MATLAB Runtime and any inputs the application takes.

**Execution Signature**

```
$./run_myStandaloneApp.sh \
  <MATLAB_Runtime_Location> \
  <inputFolderOnHDFS> \
  <outputFolderOnHDFS>
```

## See Also

## Related Examples

- “Run Standalone MATLAB MapReduce Application” on page 2-4

# Run Standalone MATLAB MapReduce Application

**Supported Platform:** Linux only.

This example shows you how to create a standalone MATLAB MapReduce application using the `mcc` command and run it against a Hadoop cluster.

**Goal:** Calculate the maximum arrival delay of an airline from the given dataset.

| | |
|---|---|
| **Dataset:** | `airlinesmall.csv` |
| **Description:** | Airline departure and arrival information from 1987-2008. |
| **Location:** | `/usr/local/MATLAB/R2023a/toolbox/matlab/demos` |

## Prerequisites

1. Start this example by creating a new work folder that is visible to the MATLAB search path.
2. Before starting MATLAB, at a terminal, set the environment variable `HADOOP_PREFIX` to point to the Hadoop installation folder. For example:

| Shell | Command |
|---|---|
| *csh / tcsh* | `% setenv HADOOP_PREFIX /usr/lib/hadoop` |
| *bash* | `$ export HADOOP_PREFIX=/usr/lib/hadoop` |

**Note** This example uses `/usr/lib/hadoop` as directory where Hadoop is installed. Your Hadoop installation directory maybe different.

If you forget setting the `HADOOP_PREFIX` environment variable prior to starting MATLAB, set it up using the MATLAB function `setenv` at the MATLAB command prompt as soon as you start MATLAB. For example:

```
setenv('HADOOP_PREFIX','/usr/lib/hadoop')
```

3. Install the MATLAB Runtime in a folder that is accessible by every worker node in the Hadoop cluster. This example uses `/usr/local/MATLAB/MATLAB_Runtime/R2023a` as the location of the MATLAB Runtime folder.

   If you don’t have the MATLAB Runtime, you can download it from the website at: `https://www.mathworks.com/products/compiler/mcr`.

**Note** For information about MATLAB Runtime version numbers corresponding MATLAB releases, see this list.

4. Copy the map function `maxArrivalDelayMapper.m` from `/usr/local/MATLAB/R2023a/toolbox/matlab/demos` folder to the work folder.

**`maxArrivalDelayMapper.m`**

```
function maxArrivalDelayMapper (data, info, intermKVStore)
partMax = max(data.ArrDelay);
add(intermKVStore,'PartialMaxArrivalDelay',partMax);
```

For more information, see “Write a Map Function”.

5 Copy the reduce function `maxArrivalDelayReducer.m` from *`matlabroot`*`/toolbox/matlab/demos` folder to the work folder.

**`maxArrivalDelayReducer.m`**

```
function maxArrivalDelayReducer(intermKey, intermValIter, outKVStore)
maxVal = -inf;
while hasnext(intermValIter)
   maxVal = max(getnext(intermValIter), maxVal);
end
add(outKVStore,'MaxArrivalDelay',maxVal);
```

For more information, see “Write a Reduce Function”.

6 Create the directory `/user/<`*`username`*`>/datasets` on HDFS and copy the file `airlinesmall.csv` to that directory. Here *<username>* refers to your user name in HDFS.

```
$ ./hadoop fs -copyFromLocal airlinesmall.csv hdfs://host:54310/user/<username>/datasets
```

## Procedure

1 Start MATLAB and verify that the `HADOOP_PREFIX` environment variable has been set. At the command prompt, type:

```
>> getenv('HADOOP_PREFIX')
```

If `ans` is empty, review the **Prerequisites** section above to see how you can set the `HADOOP_PREFIX` environment variable.

2 Create a new MATLAB script with the name `depMapRedStandAlone.m`. You will add the code listed in the steps listed below to this script file.

3 Create a `datastore` that points to the airline data in Hadoop Distributed File System (HDFS) .

```
ds = datastore('hdfs:///user/username/datasets/airlinesmall.csv',...
'TreatAsMissing','NA',...
'SelectedVariableNames',{'UniqueCarrier','ArrDelay'});
```

For more information, see “Work with Remote Data”.

4 Configure the application for deployment against Hadoop with default settings.

```
config = matlab.mapreduce.DeployHadoopMapReducer;
```

The class `matlab.mapreduce.DeployHadoopMapReducer` can be used to configure a standalone application based on the Hadoop environment where it is going to be deployed.

For example, if you want to specify the location of the MATLAB Runtime on each of the worker nodes on the cluster, include a line of code similar to this:

```
config = matlab.mapreduce.DeployHadoopMapReducer('MCRRoot','/opt/MATLAB/MATLAB_Runtime/R2023a
```

In this scenario, we assume that the MATLAB Runtime is installed in a non-default location such as `/opt/MATLAB/MATLAB_Runtime` on the worker nodes.

For information on specifying additional cluster specific properties, see `matlab.mapreduce.DeployHadoopMapReducer`.

**Note** Specifying a MATLAB Runtime location as part of the class `matlab.mapreduce.DeployHadoopMapReducer` will override any MATLAB Runtime location specified during the execution of the standalone application.

5. Define the execution environment using the `mapreducer`.

```
mr = mapreducer(config);
```

6. Apply the `mapreduce` function.

```
result = mapreduce(...
    ds,...
    @maxArrivalDelayMapper,@maxArrivalDelayReducer,...
    mr,...
    'OutputType','Binary', ...
    'OutputFolder','hdfs:///user/<username>/results/myresults');
```

**Note** An HDFS directory such as `.../myresults` can be written to only once. If you plan on running your standalone application multiple times against the Hadoop cluster, make sure you delete the `.../myresults` directory on HDFS prior to each execution. Another option is to change the name of the `.../myresults` directory in the MATLAB code and recompile the application.

7. Read the result from the resulting datastore.

```
myAppResult = readall(result)
```

8. Use the `mcc` command with the `-m` flag to create a standalone application.

```
mcc -m depMapRedStandAlone.m
```

The `-m` flag creates a standard executable that can be run from a command line. However, the `mcc` command cannot package the results in an installer.

9. Run the standalone application from a Linux shell using the following command:

```
$ ./run_depMapRedStandAlone.sh /usr/local/MATLAB/MATLAB_Runtime/R2023a
```

`/usr/local/MATLAB/MATLAB_Runtime/R2023a` is an argument indicating the location of the MATLAB Runtime.

Prior to executing the above command, verify that the `HADOOP_PREFIX` environment variable is set in the Terminal by typing:

```
$ echo $HADOOP_PREFIX
```

If `echo` comes up empty, see the **Prerequisites** section above to see how you can set the `HADOOP_PREFIX` environment variable.

Your application will fail to execute if the `HADOOP_PREFIX` environment variable is not set.

10. You will see the following output:

```
myAppResult =

          Key             Value
    ____________________    ______

    'MaxArrivalDelay'     [1014]
```

Other examples of `map` and `reduce` functions are available at `toolbox/matlab/demos` folder. You can use other examples to prototype similar standalone applications that run against Hadoop. For more information, see “Build Effective Algorithms with MapReduce”.

Complete code for the standalone application `depMapRedStandAlone` can be found here:

**`depMapRedStandAlone.m`**

```
%% Create datastore
ds = datastore(...
    'hdfs:///user/username/datasets/airlinesmall.csv',...
    'TreatAsMissing','NA',...
    'SelectedVariableNames',{'UniqueCarrier','ArrDelay'});

%% Configure application for deployment against Hadoop with default settings
config = matlab.mapreduce.DeployHadoopMapReducer;

%% Define the execution environment
mr = mapreducer(config);

%% Apply the mapreduce function
result = mapreduce(...
    ds,...
    @maxArrivalDelayMapper,@maxArrivalDelayReducer,...
    mr,...
    'OutputType','Binary', ...
    'OutputFolder','hdfs:///user/username/results/myresults');

%% Read the result from the resulting datastore
myAppResult = readall(result)
```

## See Also

`datastore` | `TabularTextDatastore` | `KeyValueDatastore` | `matlab.mapreduce.DeployHadoopMapReducer` | `mcc`

## Related Examples

- “Create Standalone Application from MATLAB Function”
- “Pass Parallel Computing Toolbox Profile at Run Time”

# Hadoop Configuration

# Configuration File for Creating Deployable Archive Using the mcc Command

When creating a deployable archive using the `mcc` command, you must create a text file containing the following information:

| Parameter Type | Description |
|---|---|
| `mw.ds.out.type` | Output type of data from Hadoop mapreduce job<br><br>The options are:<br><br>• `keyvalue`<br>• `tabulartext` |
| `mw.mapper` | Name of MATLAB map function |
| `mw.reducer` | Name of MATLAB reduce function |
| `mw.ds.in.format` | Name of MAT-file containing a `datastore` object representing the format of the data to be processed.<br><br>In most cases, you will start off by working on a small sample dataset residing on a local machine that is representative of the actual dataset on the cluster. This sample dataset has the same structure and variables as the actual dataset on the cluster. By creating a datastore object to the dataset residing on your local machine you are taking a snapshot of that structure. By having access to this datastore object, a Hadoop job executing on the cluster will know how to access and process the actual dataset residing on HDFS. |
| `mw.ds.in.type` | Input type of data to Hadoop mapreduce job<br><br>The options are:<br><br>• `keyvalue`<br>• `tabulartext` |
| `mw.ds.in.fullfile` | Default value is `false` |

## Sample Configuration File

**`config.txt`**

```
mw.ds.out.type = keyvalue
mw.mapper = maxArrivalDelayMapper
mw.reducer = maxArrivalDelayReducer
mw.ds.in.format = infoAboutDataset.mat
mw.ds.in.type = tabulartext
```

## See Also

## Related Examples

- “Include MATLAB Map and Reduce Functions into Hadoop Job” on page 1-9

# Functions

# deploytool

Open a list of application deployment apps

## Syntax

```
deploytool
deploytool project_name
```

## Description

`deploytool` opens a list of application deployment apps.

`deploytool project_name` opens the appropriate deployment app with the project preloaded.

## Examples

### Open a List of Application Deployment Apps

Open the list of apps.

```
deploytool
```

A list opens with the following options:

- **Application Compiler**
- **Hadoop Compiler**
- **Library Compiler**
- **Production Server Compiler** (if MATLAB Compiler SDK™ is installed)
- **Web App Compiler**

## Input Arguments

### `project_name` — name of the project to be opened

character array or string

Name of the project to be opened by the appropriate deployment app, specified as a character array or string. The project must be on the current path.

# Version History

**Introduced in R2006b**

### R2020a: `-build` and `-package` options will be removed

*Warns starting in R2020a*

The `-build` and `-package` options will be removed. To build applications, use one of the `compiler.build` family of functions or the `mcc` command; and to package and create an installer, use the `compiler.package.installer` function.

# matlab.mapreduce.DeployHadoopMapReducer class

**Package:** `matlab.mapreduce`

Configure a MapReduce application for deployment against Hadoop

## Description

A `DeployHadoopMapReducer` object represents executing MapReduce on a Hadoop cluster with MATLAB Runtime.

## Construction

`config = matlab.mapreduce.DeployHadoopMapReducer` creates a `matlab.mapreduce.DeployHadoopMapReducer` object that specifies the default properties for Hadoop execution.

Use the resulting object as input to the `mapreducer` function to specify the configuration properties for Hadoop execution. For deploying a standalone application, pass the `matlab.mapreduce.DeployHadoopMapReducer` object as input to `mapreduce`.

`config = matlab.mapreduce.DeployHadoopMapReducer(Name,Value)` creates a `matlab.mapreduce.DeployHadoopMapReducer` object with properties specified by one or more name-value pair arguments.

### Input Arguments

#### Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

*Before R2021a, use commas to separate each name and value, and enclose* `Name` *in quotes.*

**`HadoopInstallFolder` — Path to Hadoop installation**
character vector | string scalar

Path to Hadoop installation, specified as the comma-separated pair consisting of the `HadoopInstallFolder` and a character vector or a string scalar.

The default value of Hadoop install folder is specified by the environment variables in the order of precedence of `MATLAB_HADOOP_INSTALL`, `HADOOP_PREFIX`, and `HADOOP_HOME`.

**`HadoopConfigurationFile` — Path to Hadoop application configuration files**
character vector | string scalar

Path to Hadoop application configuration files, specified as the comma-separated pair consisting of the `HadoopConfigurationFile` and a character vector or a string scalar.

### `MCRRoot` — MATLAB Runtime installation folder for Hadoop cluster

character vector | string scalar

MATLAB Runtime installation folder for Hadoop cluster, specified as the comma-separated pair consisting of the `MCRRoot` and a character vector or a string scalar.

MCRRoot specifies the MATLAB Runtime installation folder used by Hadoop when executing `mapreduce` tasks in Hadoop.

Example: `'MCRRoot','/hd-shared/hadoop-2.2.0/MCR/v84'`

### `HadoopProperties` — Job or application-specific Hadoop configuration properties

containers.Map

A `containers.Map` object of name-value pairs that specify Hadoop configuration properties for a specific job or application. Name-value pairs must be specified as character vectors.

The properties specified in the `containers.Map` object are passed as a [GENERIC_OPTION] consisting of name-value pairs signaled by a `-D` flag to the `hadoop` shell command.

**Example:**

```
setenv('HADOOP_PREFIX', '/usr/lib/hadoop') % replace with your Hadoop install location
name = {'mapreduce.map.maxattempts','mapreduce.job.reduces'};
value = {'2','1'};
prop = containers.Map(name,value);
obj = matlab.mapreduce.DeployHadoopMapReducer('HadoopProperties', prop)
```

## Examples

### Create a Deploy Hadoop MapReducer object

Create and use a `matlab.mapreduce.DeployHadoopMapReducer` object to deploy into a standalone application, and deploy against Hadoop.

```
config = matlab.mapreduce.DeployHadoopMapReducer('MCRRoot',...
    '/hd-shared/hadoop-2.2.0/MCR/v84');
mr = mapreducer(config);
```

## See Also

`mapreduce` | `mapreducer`

### Topics

“Run Standalone MATLAB MapReduce Application” on page 2-4

# hadoopCompiler

(Not recommended) Package MATLAB Compiler programs for deployment against Hadoop clusters as MapReduce programs

**Note** The `hadoopCompiler` function will be removed in a future release. To create standalone MATLAB® MapReduce applications, or deployable archives from MATLAB map and reduce functions, use the `mcc` command. For details, see "Compatibility Considerations".

## Syntax

```
hadoopCompiler
hadoopCompiler project_name
```

## Description

`hadoopCompiler` opens the **Hadoop Compiler** app.

`hadoopCompiler project_name` opens **Hadoop Compiler** app with the project preloaded.

## Examples

### Create a New Hadoop Compiler Project

Open the Hadoop compiler app to create a new project.

```
hadoopCompiler
```

## Input Arguments

### `project_name` — name of the project to be compiled

character array or string

Name of previously saved MATLAB Compiler project to be compiled, specified as a character array or string. The project must be on the current path.

# Version History

**Introduced in R2014b**

### R2020a: `hadoopCompiler` will be removed

*Not recommended starting in R2020a*

`hadoopCompiler` will be removed. To create standalone MATLAB MapReduce applications or deployable archives from MATLAB map and reduce functions use the `mcc` command.

## See Also

`deploytool` | `mcc`

# mapreducer

Define deployed execution for mapreduce

## Syntax

```
mapreducer(config)
mr = mapreducer(config)
```

## Description

Use this function with MATLAB Compiler to specify information about the execution environment for standalone applications that execute against Hadoop.

`mapreducer(config)` specifies execution environment. When deploying a standalone application against Hadoop, `config` is an object of `matlab.mapreduce.DeployHadoopMapReducer` class.

`mr = mapreducer(config)` returns a MapReducer object to specify the execution environment. You can define MapReducer objects, allowing you to swap execution environments by passing one as an input argument to `mapreduce`.

## Examples

### Create a mapreducer object in deployed mode

```
mr = mapreducer(...
    matlab.mapreduce.DeployHadoopMapReducer('MCRRoot',...
    '/hd-shared/hadoop-2.2.0/MCR/v84'))
```

## Input Arguments

### `config` — `mapreducer` object for running in deployed environment

`matlab.mapreduce.DeployHadoopMapReducer` object

`mapreducer` object for running in deployed environment, specified as a `matlab.mapreduce.DeployHadoopMapReducer` object.

Example: `config = mapreducer(matlab.mapreduce.DeployHadoopMapReducer('MCRRoot','/hd-shared/hadoop-2.2.0/MCR/v84'))`

## Output Arguments

### `mr` — Execution environment for `mapreduce`

`mapreducer` object

Execution environment for `mapreduce`, returned as a `mapreducer` object.

## Tips

- `mapreducer` and `mapreducer(0)` enables different configurations based on the products you have. In MATLAB, the `mapreduce` function automatically runs using a `SerialMapReducer`. For more information, see `mapreducer`.

  If you have Parallel Computing Toolbox™, see the function reference page for `mapreducer` for additional information.

# Version History

**Introduced in R2014b**

## See Also

**Functions**
`mapreduce` | `gcmr`

**Classes**
`matlab.mapreduce.DeployHadoopMapReducer`

**Topics**
“Run Standalone MATLAB MapReduce Application” on page 2-4

# Apps

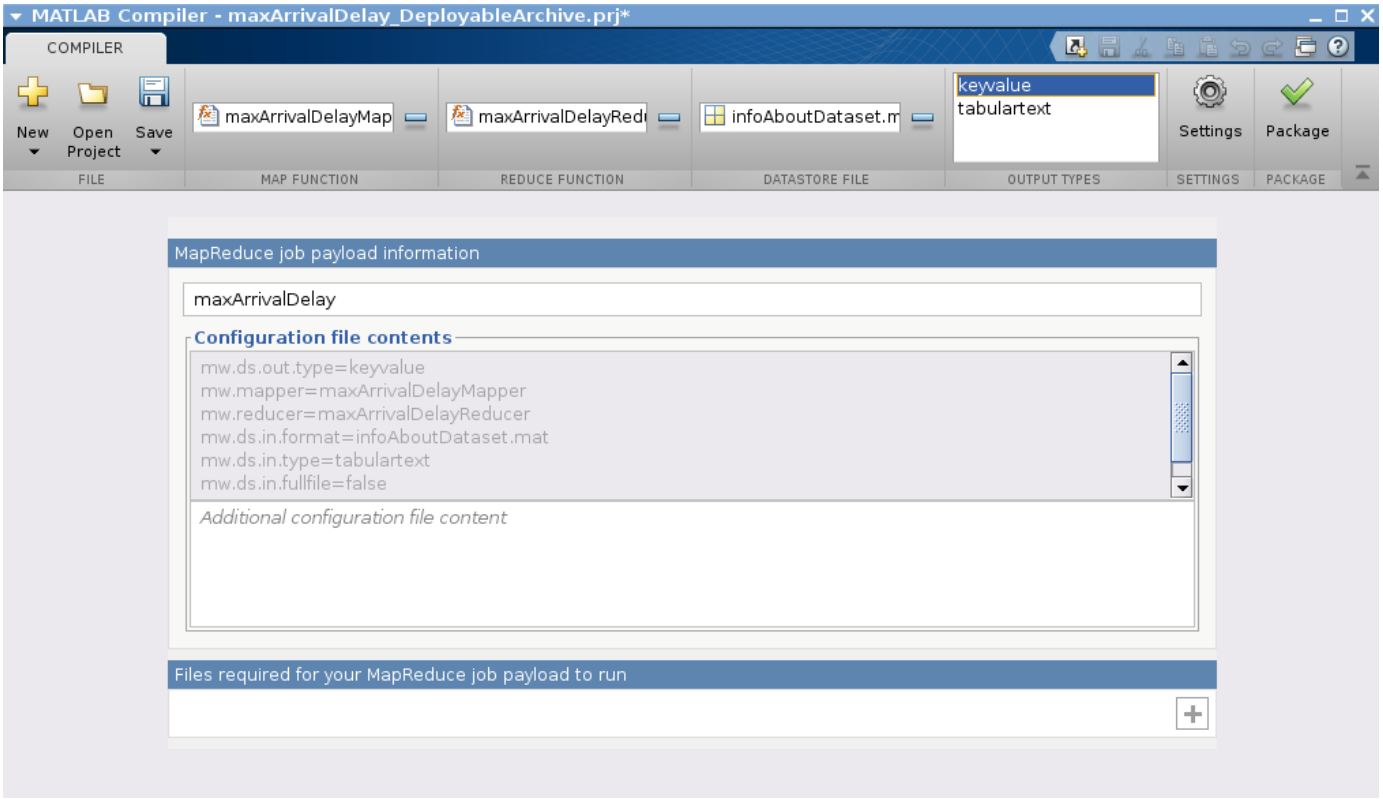
# Hadoop Compiler

Package MATLAB programs for deployment to Hadoop clusters as MapReduce programs

**Note** The **Hadoop Compiler** app will be removed in a future release. To create standalone MATLAB® MapReduce applications, or deployable archives from MATLAB map and reduce functions, use the `mcc` command. For details, see “Compatibility Considerations”.

## Description

The **Hadoop Compiler** app packages MATLAB map and reduce functions into a deployable archive. You can incorporate the archive into a Hadoop mapreduce job by passing it as a payload argument to job submitted to a Hadoop cluster.




## Open the Hadoop Compiler App

- MATLAB Toolstrip: On the **Apps** tab, under **Application Deployment**, click the app icon.
- MATLAB command prompt: Enter `hadoopCompiler`.

# Parameters

**`map function`** — mapper file
character vector

Function for the mapper, specified as a character vector.

**`reduce function`** — reducer file
character vector

Function for the reducer, specified as a character vector.

**`datastore file`** — file containing a datastore representing the data to be processed
character vector

A file containing a datastore representing the data to be processed, specified as a character vector.

In most cases, you will start off by working on a small sample dataset residing on a local machine that is representative of the actual dataset on the cluster. This sample dataset has the same structure and variables as the actual dataset on the cluster. By creating a datastore object to the dataset residing on your local machine you are taking a snapshot of that structure. By having access to this datastore object, a Hadoop job executing on the cluster will know how to access and process the actual dataset residing on HDFS.

**`output types`** — format of output
keyvalue (default) | tabulartext

Format of output from Hadoop mapreduce job, specified as a keyvalue or tabular text.

**`additional configuration file content`** — additional parameters configuring how Hadoop executes the job
character vector

Additional parameters to configure how Hadoop executes the job, specified as a character vector. For more information, see “Configuration File for Creating Deployable Archive Using the mcc Command” on page 3-2.

**`files required for your MapReduce job payload to run`** — files that must be included with generated artifacts
list of files

Files that must be included with generated artifacts, specified as a list of files.

#### Settings

**`Additional parameters passed to MCC`** — flags controlling the behavior of the compiler
character vector

Flags controlling the behavior of the compiler, specified as a character vector.

**`testing files`** — folder where files for testing are stored
character vector

Folder where files for testing are stored, specified as a character vector.

**`packaged files`** — folder where generated artifacts are stored
character vector

Folder where generated artifacts are stored, specified as a character vector.

## Version History

**Introduced in R2014b**

**R2020a: Hadoop Compiler will be removed**
*Not recommended starting in R2020a*

**Hadoop Compiler** app will be removed in a future release. To create standalone MATLAB MapReduce applications, or deployable archives from MATLAB map and reduce functions, use the `mcc` command.