# MATLAB Production Server Guide

## Planning, Deployment, Operations, and Application Development

MathWorks®

## Table of Contents

## Why Might You Need MATLAB Production Server?

- You need to give users (who do not have MATLAB®) access to the algorithms you create.
- You need to easily manage and update a single reference version of your algorithm or application.
- You need to protect the intellectual property of your code.
- You need hot deployment of code, high availability, or load balancing to maximize the uptime of your application.
- You want to control over who can use certain functions.
- You want to scale your algorithms to support hundreds or thousands of requests.

## What Is MATLAB Production Server?

MATLAB Production Server™ is an application server that lets you publish MATLAB algorithms and applications as APIs that can be called from a broad array of enterprise applications, web applications, and mobile apps, as well as from MATLAB itself (Figure 1).

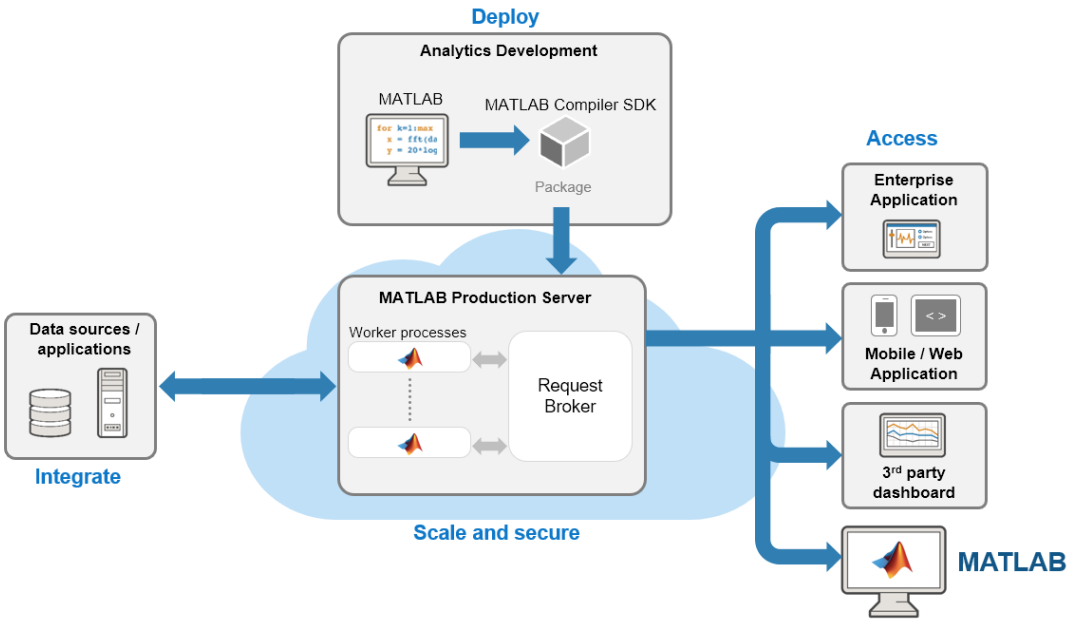


*Figure 1. Deploying MATLAB algorithms with MATLAB Production Server.*

Typical uses of MATLAB Production Server include:

- Accessing analytics hosted on MATLAB Production Server through a dashboard in their web browser, designed either in-house or using out-of-the-box solutions such as *Spotfire*, *Tableau*, or *Kibana*.

- Accessing MATLAB functions as APIs hosted on MATLAB Production Server through enterprise applications you have developed in other programming languages.
- Accessing centralized remote MATLAB functions hosted on MATLAB Production Server through MATLAB applications.
- Performing near-real-time anomaly detection or predictive analytics on streaming data from connected assets ingested through Kafka or Azure® EventHub.

## Benefits of MATLAB Production Server

- Time savings and elimination of errors from manual transcoding. IT developers can focus on more value-added activities.
- Intellectual property protection. Your MATLAB algorithms are deployed in encrypted packages.
- Easier maintenance and updates. There is a centralized single version of algorithms and functions.
- High performance, high availability, and simple deployment. Load balance as needed for additional performance.
- Easy integration into your existing on-premises or cloud architectures.
- Straightforward integration with a wide variety of front-end client technologies through the included client libraries (C/C++, C#, Java®, Python®, and RESTful client applications).

## Is MATLAB Production Server the Right Tool for You?

MathWorks has several server products: MATLAB Parallel Server™, MATLAB Production Server, and MATLAB Web App Server. Each of these servers serves different use cases. Table 1 provides a high-level picture of each product to help you decide.

| Product | Use Case |
|---|---|
| MATLAB Parallel Server | • Decrease run time for parameter sweeps, parameter optimization, Monte Carlo runs, and other iterative tasks<br>• Leverage the memory of multiple computers to solve larger problems with distributed arrays<br>• Scale workflows with tall arrays and datastores: reduce run time by using more parallel workers |
| MATLAB Production Server | • Integrate MATLAB algorithms with C/C++, Java, .NET, Python, and RESTful applications<br>• Apply MATLAB analytics to streaming time-series data<br>• Centralize hosting of MATLAB functions |
| MATLAB Web App Server | • Host and share MATLAB apps created in App Designer |

*Table 1. MATLAB server products and their use cases.*

Each of these server products is best suited for different application types. Table 2 provides additional detail on how to select the best server product depending on your application type.

| Use Case | MATLAB Parallel Server | MATLAB Production Server | MATLAB Web App Server |
|---|---|---|---|
| Speed up parameter sweeps, optimizations, and Monte Carlo runs | ✓ | | |
| Speed up big data workflows (tall arrays, datastores) | ✓ | | |
| Large memory distributed array calculation | ✓ | | |
| Integrate with streaming data/IIoT application | | ✓ | |
| Integrate with third-party web application (e.g., HTML/ Javascript, LAMP, Angular, React, ASP.NET, JSP, JSF, Python) | | ✓ | |
| Integrate with enterprise application (C/C++, Java, .NET) | | ✓ | |
| Integrate with mobile app (iOS, Android, Phonegap, Cordova) | | ✓ | |
| Publish MATLAB web app authored in App Designer | | | ✓ |

*Table 2. MATLAB server products and recommended applications.*

Apart from deciding between the various MATLAB server products, you might also have to decide between the different deployment options available from MathWorks. The most suitable deployment product to employ depends on the application being developed. Table 3 shows which product is best suited various applications to help you decide between MATLAB Compiler™, MATLAB Compiler SDK™, and MATLAB Production Server. In certain cases, there is more than one possible solution. Note: This table does not show options for real-time embedded applications. Embedded application scenarios are typically better suited to other products, such as MATLAB Coder.

| | Type of application | MATLAB Compiler | MATLAB Web Apps | MATLAB Compiler SDK | MATLAB Production Server |
|---|---|---|---|---|---|
| Non-Interactive | Business logic components (Beans, EJBs, Facades, Factories) | N/A | N/A | Best | Not recommended |
| | Long running batch jobs | Doable but not ideal | Not recommended | Consider MATLAB Parallel Server | Doable but ties up worker |
| | Streaming data / IoT application | N/A | N/A | N/A | Best |
| | Hadoop jobs | Best | Not recommended | Not recommended | Not recommended |
| Interactive | Standalone executable | Best | N/A | N/A | N/A |
| | Excel add-inn | Best | N/A | N/A | Best |
| | MATLAB App Designer application | Best | Best | N/A | N/A |
| | Classic client server Java/C/C++/.NET/COM desktop client application | N/A | N/A | Best if interfacing with non MATLAB server | Best |
| | Thin client Java/C/C++/.NET/COM desktop client application where the application logic and data reside on a server | N/A | N/A | Best if interfacing with non MATLAB server | Best |
| | Web application written in LAMP stack | N/A | N/A | Not recommended | Best |
| | Web application written in MEAN stack | N/A | N/A | Not recommended | Best |
| | Web application written in HTML/Javascript | N/A | N/A | Not recommended | Best |
| | Web application written in ASP.NET/Razor | N/A | N/A | Good | Best |
| | Web application written in Java Servlet/JSP/JSF | N/A | N/A | Good | Best |
| | Web application written in Python | N/A | N/A | Good | Best |
| | Native Mobile application (Obj-C, Swift, Android/Java) | N/A | N/A | Not recommended | Best |
| | Hybrid Mobile application (Cordova, PhoneGap, Xamarin) | N/A | N/A | Not recommended | Best |
| | Hybrid Desktop Electron application | N/A | N/A | Not recommended | Best |

*Table 3. MATLAB deployment products and recommended applications.*

**Key point:** MATLAB Production Server does not generate a front-end UI for your interactive application; it exposes the algorithms as function APIs that are called by the front-end UI. The front-end application is often developed by software engineers or developers in the IT department.

## How Does MATLAB Production Server Fit into My IT Environment?

Engineers and scientists package their algorithms into a deployable archive using MATLAB Compiler SDK. This deployable archive is copied into the MATLAB Production Server `auto_deploy` folder. Client applications can then be written to access the deployed algorithms. MATLAB Production Server supports a wide variety of client applications: Java, .NET, C/C++, Python, Web, and mobile apps. Web and mobile applications are supported via a RESTful interface with JSON payloads. Meanwhile, the native applications use MathWorks provided royalty-free client libraries to communicate with MATLAB Production Server. When a request is received by MATLAB Production Server, the request broker dispatches the request to a corresponding MATLAB worker process. MATLAB Production Server supports applications written in multiple versions of MATLAB dynamically without having to manage MATLAB Runtime versions on a user's PC. The MATLAB applications may access enterprise or cloud data sources to perform their processing or analytic tasks before returning the results to the calling application.

MATLAB Production Server fits into the middleware layer of your IT ecosystem fabric as an application server whose APIs can be published through your API gateway (Figure 2).

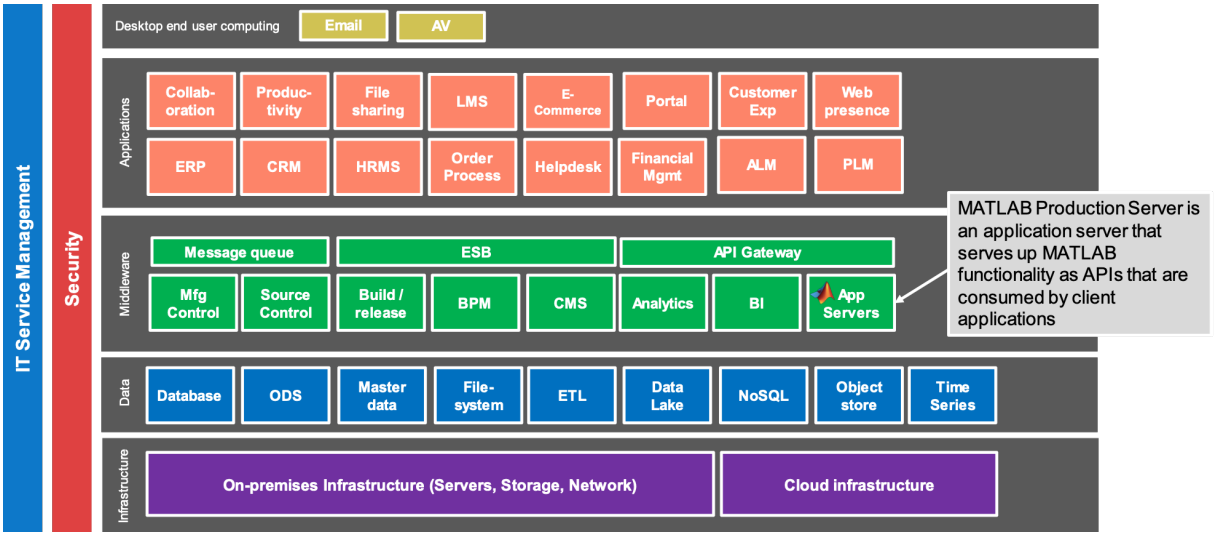


*Figure 2. IT ecosystem, with MATLAB Production Server areas shown outlined in red. MATLAB Production Server is an application server that serves up MATLAB functionality as APIs that are consumed by client applications.*

## High-Level Workflow for Using MATLAB Production Server for Application Deployment

Engineers, scientists, and quants use MATLAB to write algorithms that access, explore, and analyze data (Figure 3).

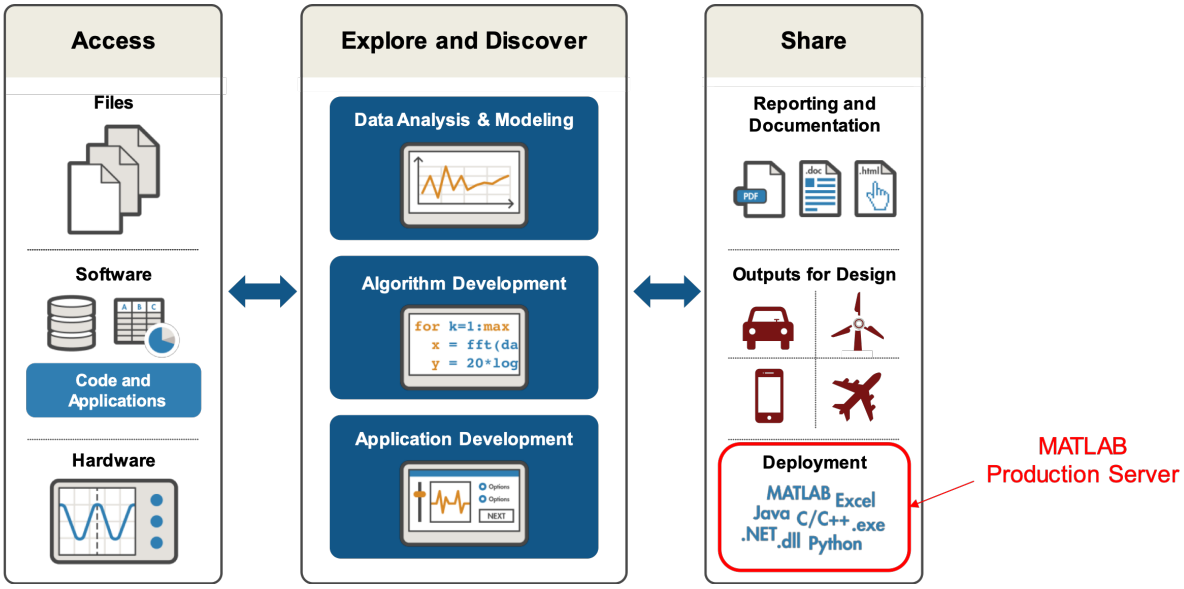


*Figure 3. Developing and sharing MATLAB algorithms that access, explore, and analyze data.*

Once these MATLAB algorithms are developed, they are packaged into a deployable archive using MATLAB Compiler SDK (Figure 4). MATLAB Compiler SDK walks engineers through the process of packaging the algorithms into a .ctf file, which can then be handed over to IT, who manage MATLAB Production Server. The .ctf file is analogous to a .war/.ear file in Java applications.

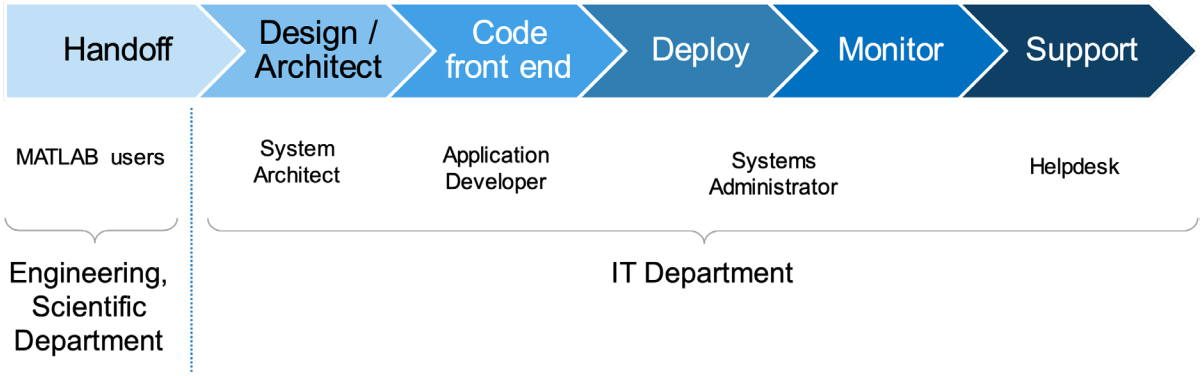


*Figure 4. Algorithm development and deployment workflow involving engineering and IT departments.*

The deployable archive .ctf file is copied to the MATLAB Production Server `auto_deploy` folder, which automatically hot deploys the code. Figure 5 shows the entire process.

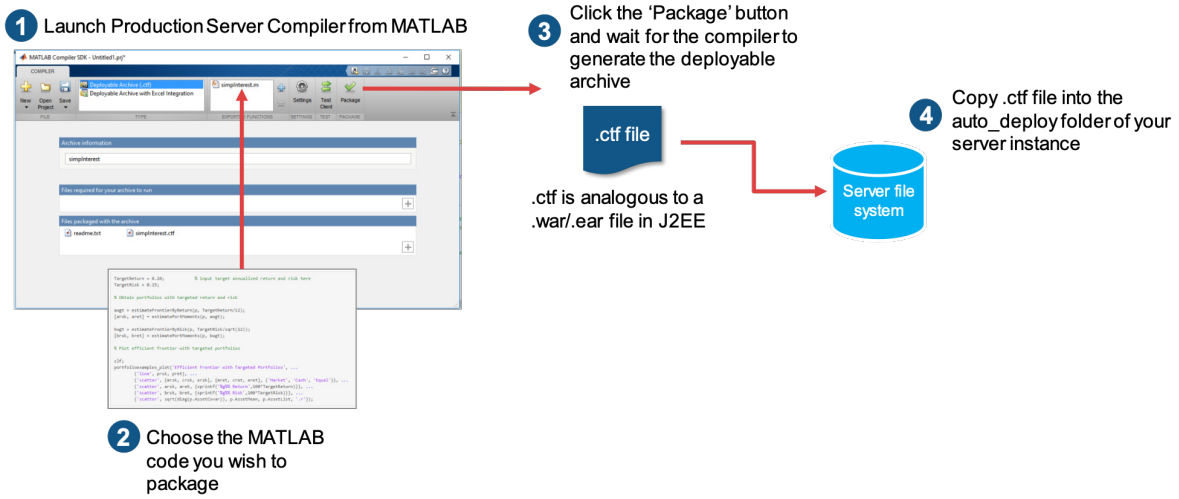


*Figure 5. Deploying code using MATLAB Production Server.*

For this step:

- MATLAB Compiler SDK and MATLAB Compiler are required.
- In addition to manually compiling the deployable archive file, you can use a script-based automated CI/CD workflow. See the ***Integration with CI/CD pipelines section*** for more details.
- There is no need to translate the MATLAB code to C, Java, C#, Python, or other language.

# Planning for Deployment

## Change Management

Currently, in most organizations the deployment of algorithms developed by engineers in MATLAB is unlike what was described in the prior section. Instead, the MATLAB model is developed using iterative analysis, but when the time comes for deployment, an IT team is engaged to translate the MATLAB code into Java, C#, or Python so that it can be run on a web or application server (Figure 6). Often, the programmers converting code from one language to another are unfamiliar with the problem domain. Their lack of background and understanding can lead to inaccurate transcoding and errors or lost time. The transcoding process also requires time and effort from programmers, QA, system engineers, and others. Each time the MATLAB model is updated, the code has to be transcoded again to reflect the changes.

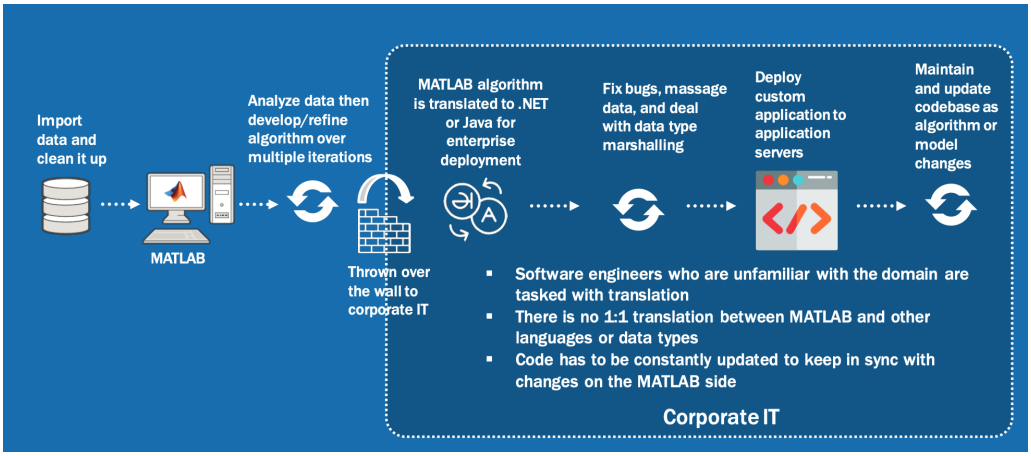


*Figure 6. Typical change management scenario.*

Introducing MATLAB Production Server into this process helps streamline the workflow because it eliminates the manual transcoding step (Figure 7). The MATLAB users simply deliver the .ctf file to IT, who copy it to the MATLAB Production Server filesystem. The MATLAB model can then be called via any enterprise application. A continuous deployment workflow could also be employed to automatically copy the file artifact over using scp, sftp, or network file copy.

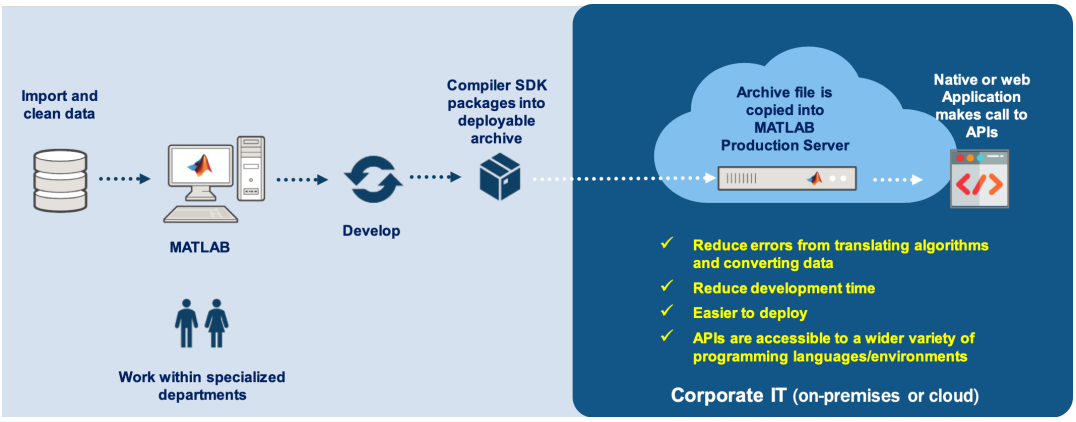


*Figure 7. Change management with MATLAB Production Server.*

This process change may require some reassignment of IT staff who previously performed transcoding tasks. From the IT perspective, the amount of effort required to deploy an application is significantly reduced. In a resource-constrained organization, this should be a source of relief and the staff involved may see increased morale if they are deployed to more interesting or value-added projects. If there are teams previously dedicated to transcoding (unlikely except in very large organizations), they should be provided with timely, transparent communications about the change of responsibilities. Note that while MATLAB Production Server surfaces the MATLAB code as an API, a front-end application still needs to be developed for end user interaction. These teams may be retrained and retasked to develop the UI applications.

## Service Desk and Support Processes

Your service desk should escalate MATLAB Production Server issues or inquiries to the MathWorks support resources as shown in Table 4. A Software Maintenance Service subscription is required in order to receive support (software maintenance service is included with a new product license). Technical support engineers with advanced degrees in EE, ME, and CS are on hand to resolve your problem.

Technical support is available in 10 languages via telephone, email, or through the MathWorks web site.

| Support Need | Contact |
|---|---|
| Installation issues | *Contact Installation Support* |
| License issues | *Contact Installation Support for FLEX License Server or Customer Support for license renewal* |
| Technical issues | *Contact Technical Support* |
| Community help | *Ask questions and browse answers at MATLAB Answers* |
| Implementation help | *Contact MathWorks Consulting* |
| Training | *Contact MathWorks Training* |

*Table 4. MathWorks support resources.*

## Skills Development

Certain skills are required to successfully develop, integrate, and deploy MATLAB applications in your production environment (Table 5).

| Persona | Skills required |
|---|---|
| Solution/System Architect | • Understanding of MATLAB Production Server architecture, worker management, and security requirements so that it can be deployed effectively within the organization<br>• Understanding of MATLAB Production Server development endpoints so as to successfully integrate with enterprise data sources and operational systems |
| Application Developer | • No additional skills required beyond current .NET, Java, JavaScript, C/C++, Python knowledge<br>• Familiarity with encoding JSON payloads, calling REST APIs and decoding JSON return values required if using REST interface |
| Systems Administrator | • Familiarity with MATLAB Production Server configuration parameters, monitoring thresholds, and log messages |
| Helpdesk/Support | • Ability to identify when incident is caused by MATLAB Production Server rather than the calling application<br>• Ability to differentiate between installation, configuration, licensing, technical issues and escalate the incident to the correct MathWorks support channel |

*Table 5. Skills required for developing, integrating, and deploying MATLAB applications.*

## Technology Planning

### System Requirements and Dependencies

Table 6 shows system requirements for MATLAB Production Server.

| | Requirements |
|---|---|
| Server hardware | One physical or virtual core per worker<br>Minimum 512 MB and recommended 2 GB of RAM per worker<br>Minimum of 10 GB disk space for server binaries and log files |
| Operating systems | Microsoft® Windows® 7 SP1 or later<br>Windows Server 2008 R2 SP1 or later<br>OS X El Capitan (10.11) or later<br>Ubuntu® 14.04 LTS, 16.04 LTS, and 17.04<br>Red Hat® Enterprise Linux® 6 and 7<br>SUSE® Linux Enterprise Desktop 12 SP2 or later<br>Debian® 8.x, 9.x |
| Server software | MATLAB Runtime 2012b or later (note: more than one MATLAB Runtime may have to be installed on the server) |
| Network | By default, MATLAB Production Server listens on TCP port 9910 (http) and 9920 (https); firewalls, reverse proxies, etc. should be set to forward traffic on these ports.<br>Default port may be changed based on the organization's preferences. |

*Table 6. MATLAB Production Server system requirements.*

MATLAB Production Server requires the MATLAB Runtime (***free download***). Each specific release of MATLAB Production Server supports that release version of the MATLAB Runtime as well as five prior release versions. For example, MATLAB Production Server R2019b supports the following MATLAB Runtime releases:

- R2019b (the same version as the MATLAB Production Server in this example)
- R2019a
- R2018b
- R2018a
- R2017b
- R2017a

While MATLAB code written and compiled in older versions may run in MATLAB Production Server, MathWorks does not test or support these configurations.

### Licensing Management

MATLAB Production Server is licensed based on the number of worker processes:

- A worker process is a headless runtime instance of MATLAB.
- MATLAB Production Server is priced on a per-worker basis. The default size is 24 workers, but customers may purchase other sizes based on their needs, with a minimum order size of 4 workers.
- MathWorks recommends one worker per core; which is analogous to a per-core licensing model.
- Workers may be split among multiple physical or virtual machines. For example, in Figure 8, Scenario 2, 24-workers are split in an 8, 4, 12 configuration between three physical machines.
- Licensing is managed by a license server, which can be installed on the same machine as MATLAB Production Server or on a separate machine. MathWorks recommends a separate machine (minimum requirements dual core, 4 GB RAM). Figure 8 depicts the license server on a separate machine connected via a network link.
- All MATLAB Production Server instances must have constant communications with the licensing server. If communication is lost, there is a grace period before MATLAB Production Server will cease responding to requests. The grace period is configurable; the default is set to 2.5 hours, which is the maximum period allowed.

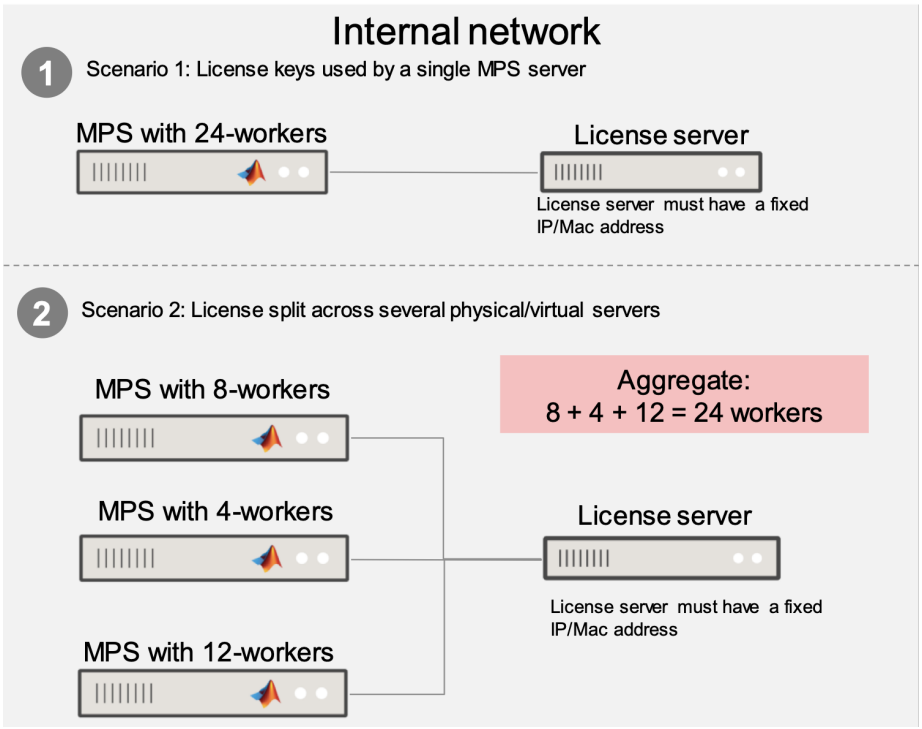


*Figure 8. MATLAB Production Server license management options.*

## Capacity Management and Server Sizing

MATLAB Production Server scales and performs extremely well under load given the appropriate hardware resources.

- MathWorks recommends one core and 2 GB of RAM per worker.
- MATLAB Production Server scales almost linearly as the number of workers is ramped up.

Figure 9 shows the results of a benchmark test for a medium-load function. MATLAB Production Server continued serving approximately 400 requests per second even as the number of concurrent users hit 1000.

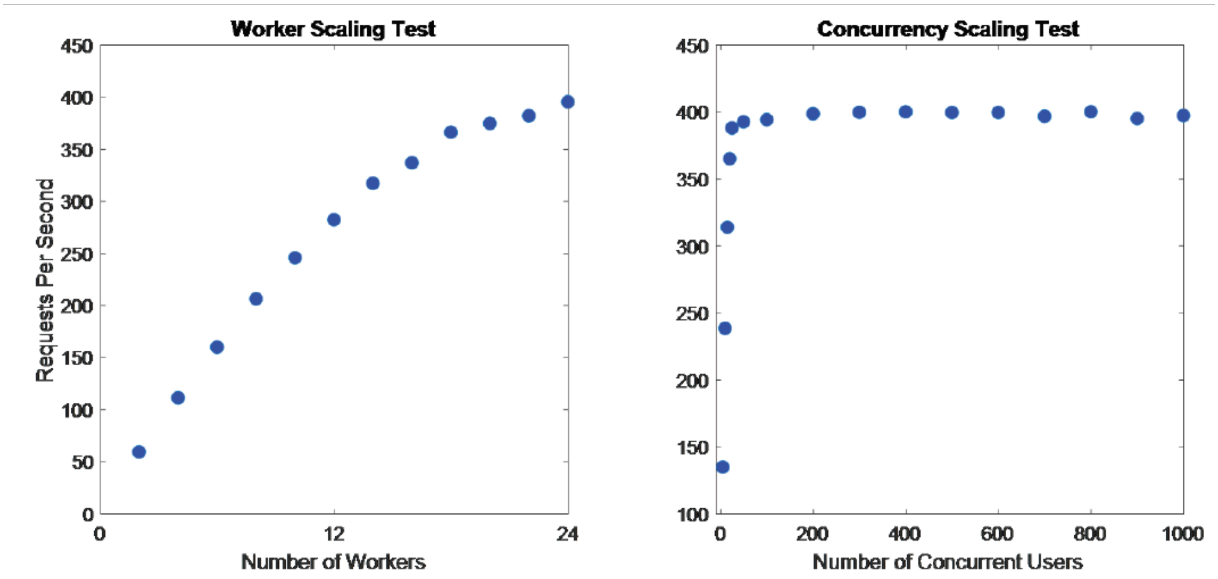


*Figure 9. MATLAB Production Server performance with a medium load (Fast Fourier transform) running on a 24-core Intel Xeon with 128GB RAM.*

To determine how many workers you need, answer these three questions:

1. How many concurrent requests to the server will there be? If you don't know, 10% of the total users is a good rule of thumb.
2. How long does your function take to run? If you have written the function, time it within MATLAB using `tic` and `toc`. If you have not written the function yet, you can estimate using these average times: a simple function typically takes < 1 sec, an intermediate function takes 1–4 seconds, and a complex function takes 5–8 seconds.
3. How long will your users be willing to wait to view the results? A best practice is usually 5 seconds or less. Note: This value cannot be less than the value in Question 2.

You can then calculate how many workers you need for a particular deployed function using the equation:

$$\#\ workers\ required = \frac{\#\ of\ concurrent\ requests \times how\ long\ the\ function\ takes\ to\ run}{how\ long\ your\ users\ are\ willing\ to\ wait\ to\ see\ the\ results}$$

Note that you will have to repeat this process for each function you wish to deploy to MATLAB Production Server and then aggregate the number of workers for all the functions to get a final tally.

Let's say you wish to deploy two functions to your MATLAB Production Server:

1. Function A will be used by 100 users, takes 1 second to execute, and you don't want users to wait more than 2 seconds to get a response.
2. Function B will be used by 50 users, takes 4 seconds to execute, and you don't want to users to wait more than 5 seconds to get a response.

Number of workers required by Function A:

$$\#\ workers\ required = \frac{(100 \times 10\%) \times 1}{2} = 5$$

Number of workers required by Function B:

$$\#\ workers\ required = \frac{(50 \times 10\%) \times 4}{5} = 4$$

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

## Availability Management

There are many possible topologies when it comes to high availability for MATLAB Production Server (Table 7). The choice of high availability and disaster recovery topology must be balanced with the associated cost and complexity.

| Topology | Cost | Performance | Wasted resource | Ease of failover | Single point of failure | Complex setup |
|---|---|---|---|---|---|---|
| Active-Active with independent licenses | ☹ | ☺ | ☺ | ☺ | ☺ | ☺ |
| Active-Active Backup license | ☺ | ☺ | 😐 | ☺ | ☺ | 😐 |
| Active-Active Shared license | ☺ | ☺ | 😐 | ☹ | ☹ | ☹ |
| Active-Passive Backup license | ☺ | 😐 | ☹ | 😐 | ☺ | 😐 |
| Hot Standby | ☺ | 😐 | ☹ | ☺ | ☺ | ☺ |

*Table 7. MATLAB Production Server topologies and associated tradeoffs.*

A load balancer is typically placed in front of the MATLAB Production Server pool to distribute the requests. Most MATLAB Production Server calls are stateless[1]; a round-robin scheme is typically employed. Table 8 shows some of the load balancers that have been employed with MATLAB Production Server.

| Usage Scenario | Software Load Balancers | Hardware Appliances |
|---|---|---|
| Departmental, SMB | NGINX (Plus)<br>Kemp LoadMaster<br>Zevenet Zen | Kemp LoadMaster<br>Zevenet Zen |
| Enterprise | F5 BIG-IP<br>Radware Alteon | F5 BIG-IP<br>Citrix NetScaler<br>Radware Alteon |
| Cloud | Azure® Load Balancer/Application Gateway<br>AWS® Elastic Load Balancer | |

*Table 8. Using load balancers with MATLAB Production Server.*

[1] You can store state in between calls using the built-in Redis™ cache (*see detailed instructions*).

If the highest availability and performance is paramount and cost is less of a concern, MathWorks recommends an active/active independent license topology (Figure 10). Note that this topology requires more software licenses as well as hardware resources.

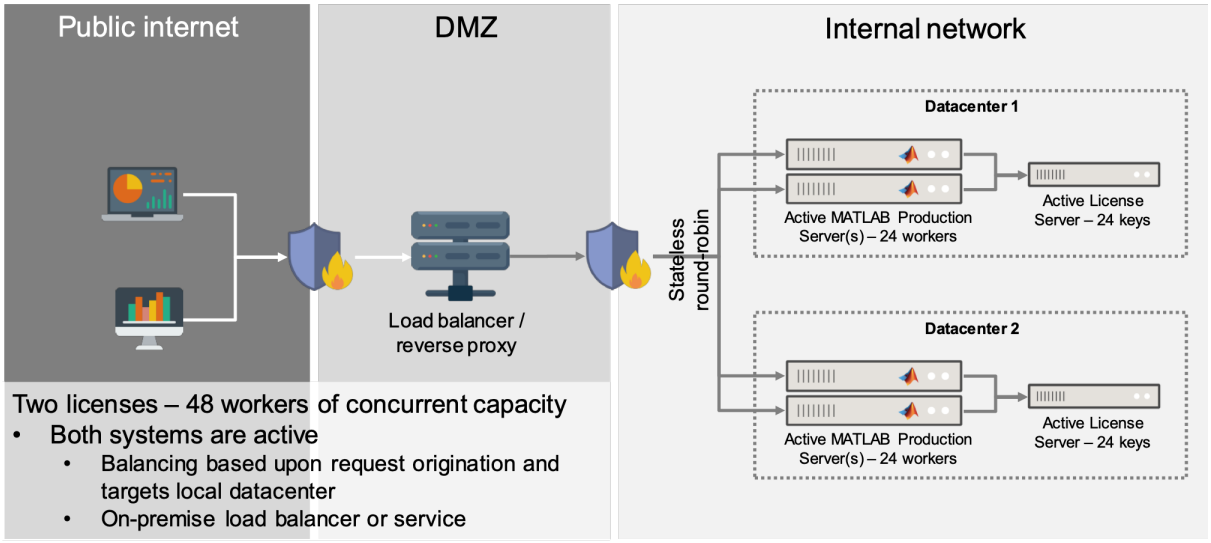


*Figure 10. Using an active/active independent license topology for MATLAB Production Server.*

The next best choice is a good balance of price and availability: the active/active with standby license topology (Figure 11).

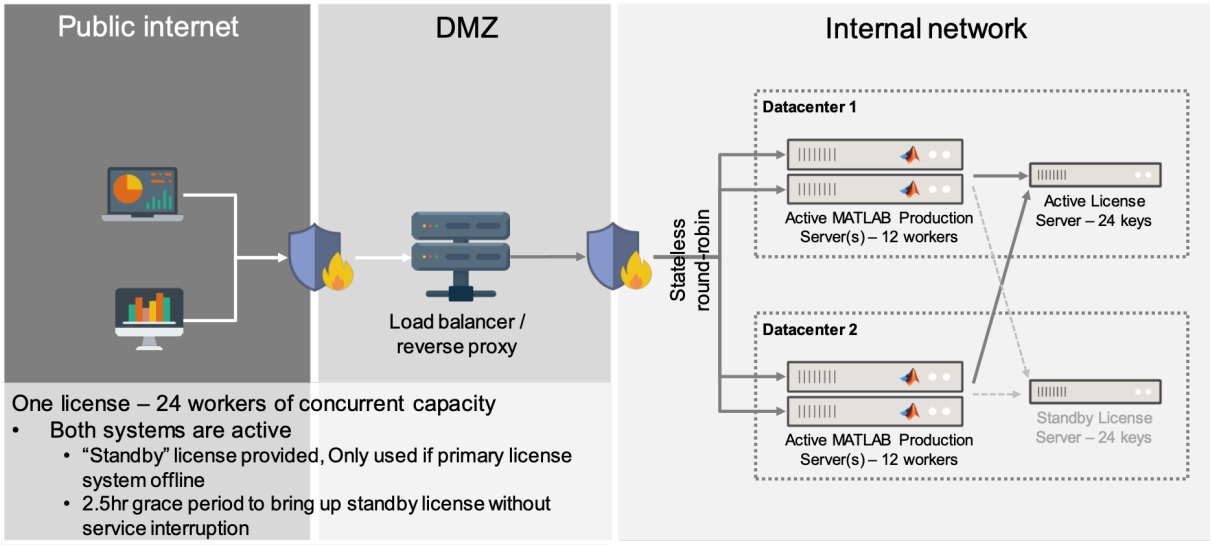


*Figure 11. Using an active/active with standby license topology for MATLAB Production Server.*

To configure MATLAB Production Server to operate with a specific number of workers, edit the `main _ config` file and set the following parameter (in this example, with 12 workers for each server):

```
--num-workers 12
```

on each of the servers. A restart is required for the setting to take effect.

Contact MathWorks installation support to acquire, configure, and deploy the standby license server. See the section on *service desk* for contact information.

**Note:** FlexNet® license servers can be operated in a clustered (quorum) type configuration, but MathWorks does not recommend this configuration due to its somewhat fragile nature.

### Security Management

Figure 12 details how MATLAB Production Server deals with authentication, authorization, access control, administration, audit, and encryption.

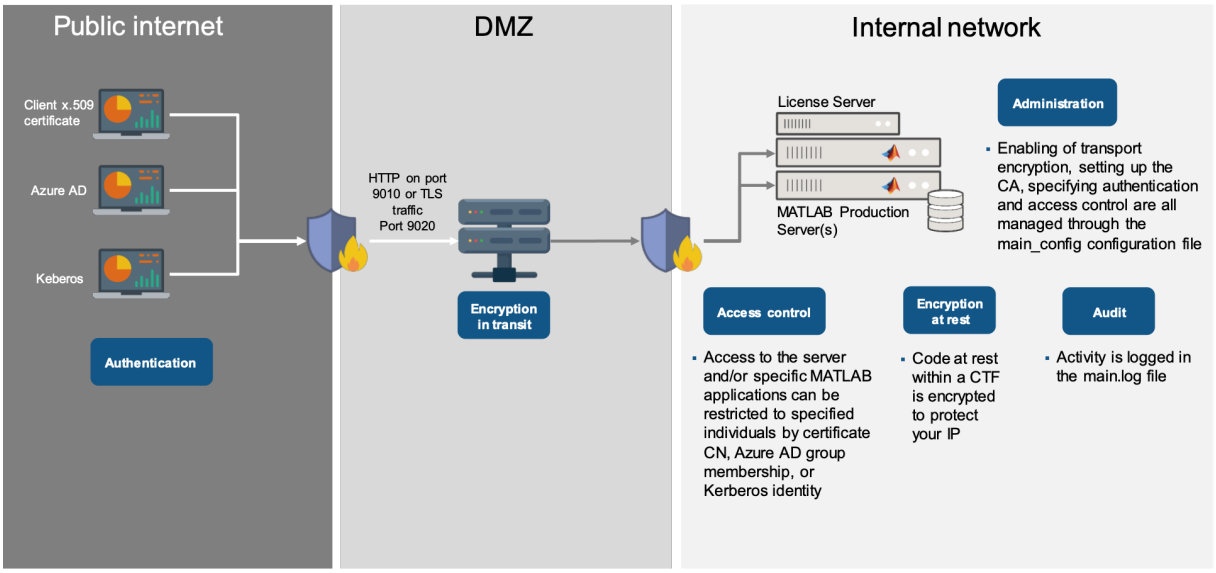


*Figure 12. Security management with MATLAB Production Server.*

**Authentication:** Client identification is performed by one of these methods:

1. Issuing x.509 client certificates to each desktop, laptop, or mobile device that will be accessing MATLAB Production Server. These client certificates are used to authenticate the user and provide access. Client certificates are a strong form of authentication especially when compared with the basic authentication that is used in many other systems.
2. Utilizing Microsoft Azure Active Directory as the Identity Provider for single sign-on.
3. Leveraging Kerberos authentication in Microsoft Windows.

**Encryption in transit:** Industry standard TLS is used to encrypt network traffic. By default, this uses port 9020.

**Encryption at rest:** The MATLAB application code is encrypted to protect your intellectual property. If your application accesses data from a database, data warehouse, NoSQL, filesystem, block storage, or other source, it is your responsibility to ensure that it is encrypted.

**Access Control:** Access to the .ctf files (MATLAB applications and functions) can be restricted to named individuals as identified by their client certificate or Azure Active Directory group. This access control is managed through the `main _ config` configuration file.

**Administration:** Security administration—for example, configuring SSL or specifying access control—is managed through the `main _ config` configuration file.

**Audit:** Activity is logged in the main log file, which can be analyzed using standard log analyzer tools.

Figure 13 shows an example of an end-to-end security architecture.

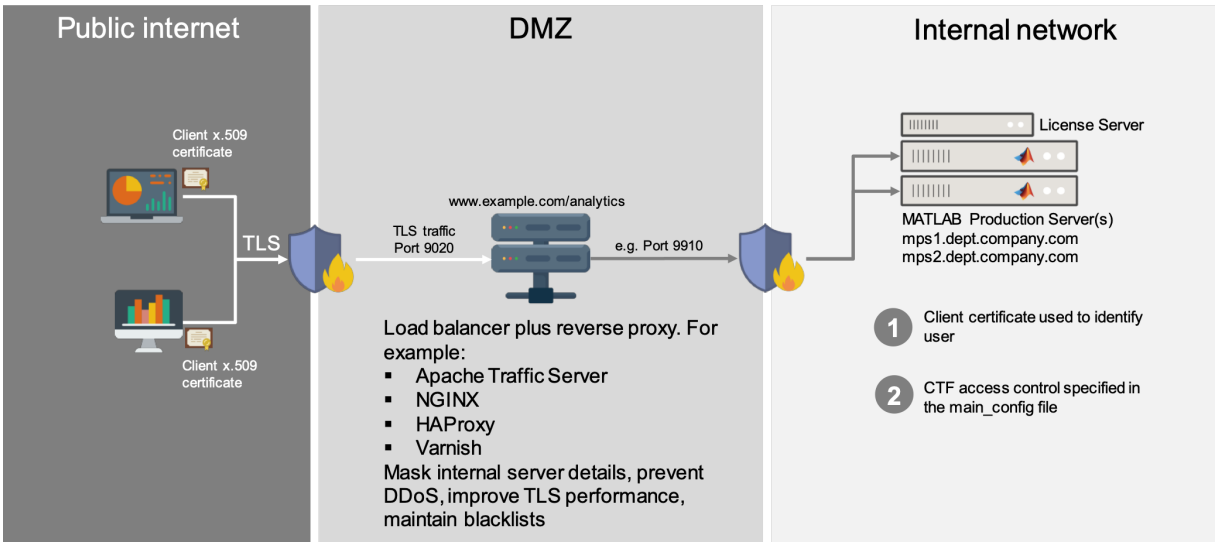


*Figure 13. An end-to-end security architecture with MATLAB Production Server.*

Specific security configuration steps and daily management are covered in the ***Operations Runbook*** section.

## Deployment Patterns

This section contains many component diagrams grouped by use case. These component diagrams are high-level conceptual/logical architectures and do not go into physical architectures. They should be used as conceptual guides as your design your solution.

## General On-Premises Pattern

Figure 14 shows MATLAB Production Server in a disaster recovery topology across multiple servers, sharing a single license server. On the left, you will find the myriad client applications that can connect to MATLAB Production Server either through language-specific client libraries or through a generic RESTful API. On the right are the resources within the organization such as compute (MATLAB Parallel Server), data, or operational systems that may be called by the functions in MATLAB Production Server to generate the results.

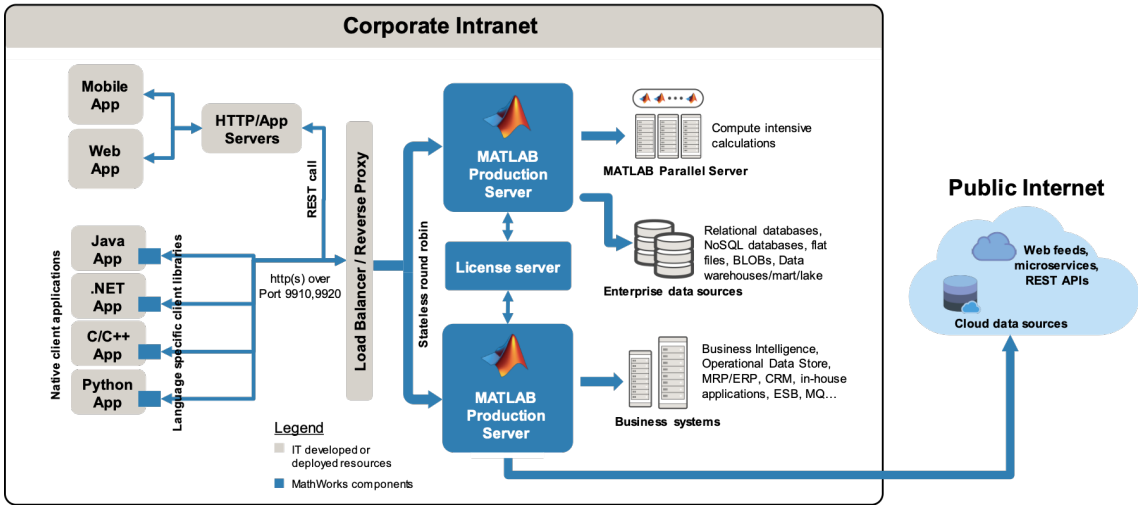


*Figure 14. A general on-premises pattern for MATLAB Production Server.*

## General Cloud Pattern

The cloud reference architecture shown in Figure 15 is not specific to any cloud platform, although at this point in time MathWorks focus is Microsoft Azure and Amazon Web Services. This pattern assumes a bring-your-own license approach. MathWorks provides cloud reference architectures to simplify deployment to these platforms:

- *MATLAB Production Server cloud reference architecture for AWS*
- *MATLAB Production Server cloud reference architecture for Azure*

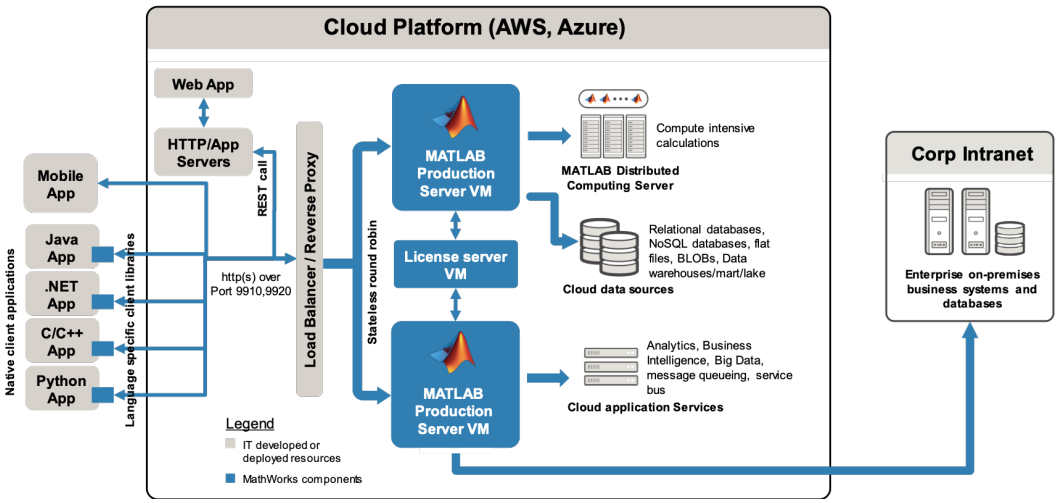


*Figure 15. A general cloud pattern for MATLAB Production Server.*

### Streaming Data Pattern

For Azure, MathWorks has an EventHub connecter that listens to events in Event Hub or IoT Hub and sends that streaming data to MATLAB Production Server for analysis and processing (Figure 16). The results of the analysis can be depicted in a custom-developed front-end application or through third-party visualization applications such as Tableau, Spotfire, Qlik, or Power BI. Email us at *mwlab@mathworks.com* for more information regarding these visualization dashboards.

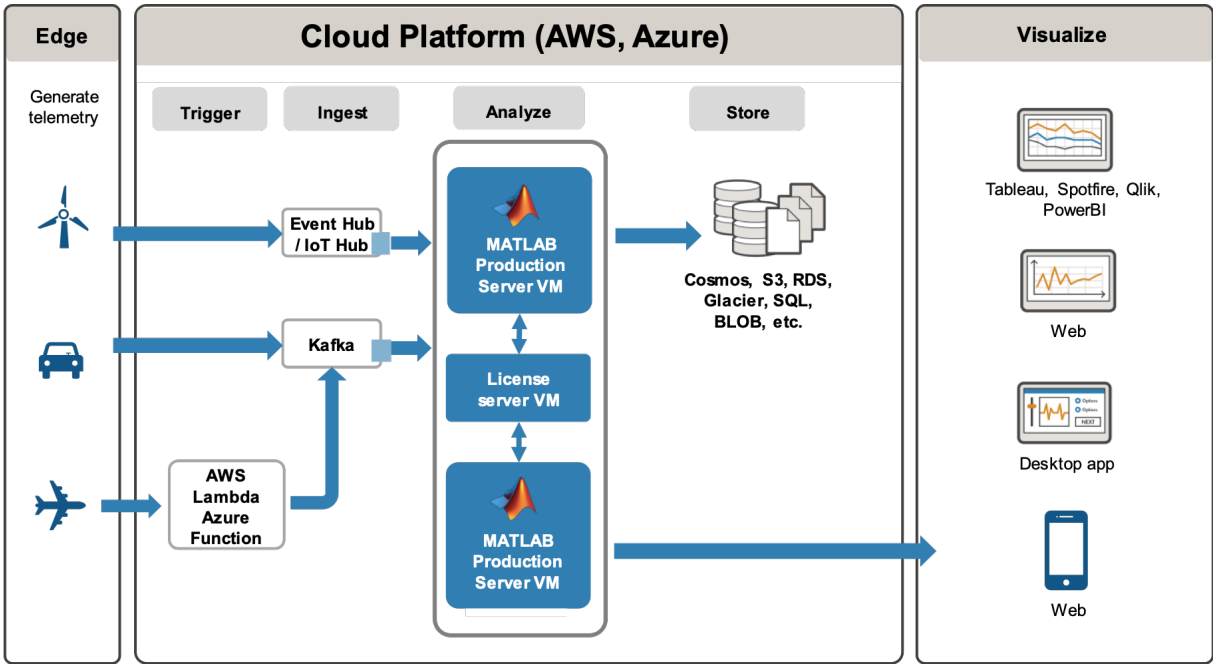


*Figure 16. A streaming data pattern for MATLAB Production Server.*

For AWS, the Kafka service is the preferred mode of telemetry ingestion. The device telemetry triggers a Lambda function that transmits the data to Kafka, and a Kafka connector sends the streaming data to MATLAB Production Server for analysis and processing. Similar to Azure, the results of the analysis are depicted in a custom-developed front-end application or through third-party visualization applications such as Tableau, Spotfire, Qlik, or Power BI. Learn more about integrating with dashboards:

- *MATLAB and PowerBI*
- *MATLAB and SpotFire*
- *MATLAB and Tableau*

### Machine Learning or Deep Learning Pattern

For a machine learning or deep learning scenario, you would first develop the model in MATLAB using the necessary text, audio, and video data. The trained model would then be exported as a .mat file and incorporated into the .ctf artifact that is copied onto MATLAB Production Server. During runtime, the application will use the trained model to make the prediction against data being read from either streaming sources or databases. The results of the prediction can then be displayed in the front-end application (Figure 17).

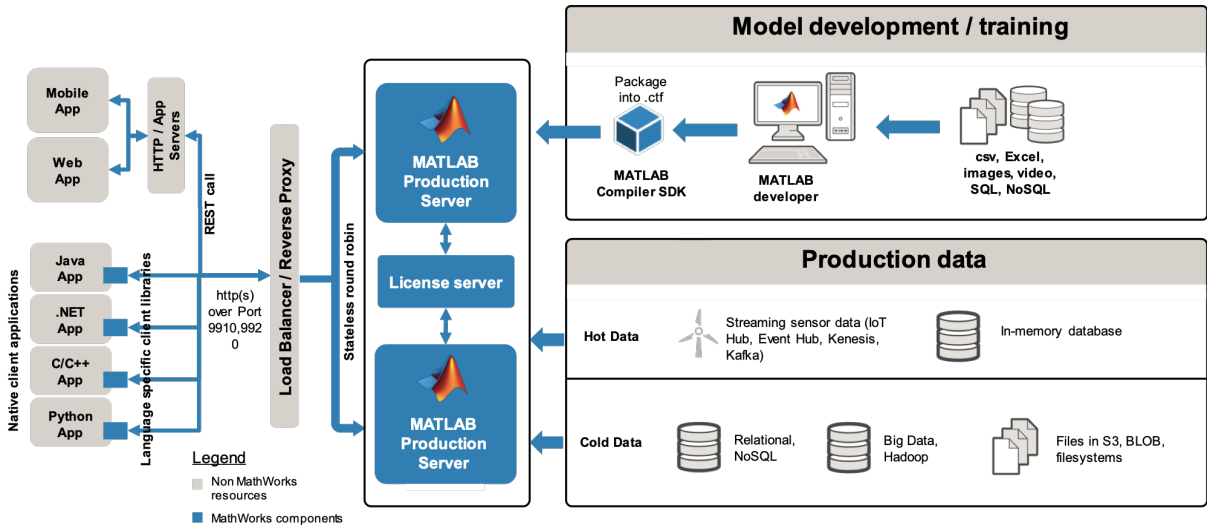


*Figure 17. A machine learning or deep learning pattern for MATLAB Production Server.*

## Enterprise Data Integration Pattern

If your application is primarily accessing a database and performing analytics on the data, you probably will use Database Toolbox™ during development to access the data. When the code is deployed to MATLAB Production Server, the associated Database Toolbox functionality will be packaged with your application-specific analytics (Figure 18). This approach allows the application to access the same database and read the data in real time when it is called by the front-end application.

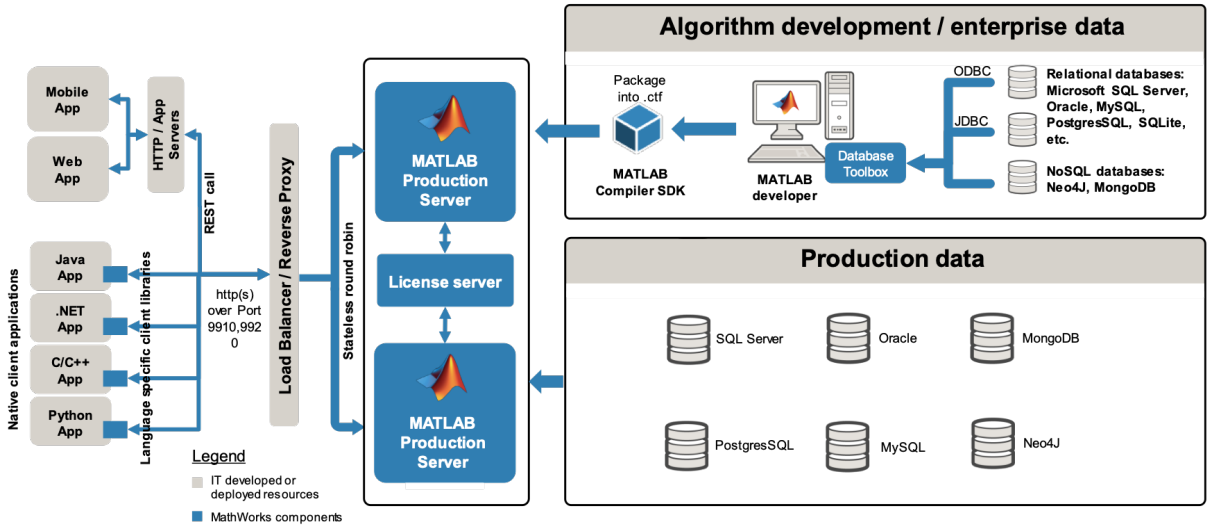


*Figure 18. An enterprise data integration pattern for MATLAB Production Server.*

Connectors to a variety of data sources can be found in the *MathWorks GitHub repository*. Examples of data connectors include:

- *AWS S3*
- *AWS DynamoDB*
- *AWS Athena*
- *Azure Data Lake*
- *Azure BLOB*
- *Azure CosmosDB*

## Application Server Integration Pattern

A MATLAB application may occasionally need to call other services, especially if it is part of a larger microservice architecture framework. These services may be exposed or advertised via an API gateway or Enterprise Service Bus. The recommended approach to calling these services from your MATLAB application is to make a *REST-style call with* `webread()` *or* `webwrite()`. These services may be provided by application servers such as WebSphere, WebLogic, Tomcat, or Microsoft IIS. The results of these services can be incorporated into the analytics of the MATLAB algorithm. Once deployed to MATLAB Production Server, the code uses the same REST call to the underlying services during runtime to service calls from front-end applications (Figure 19).

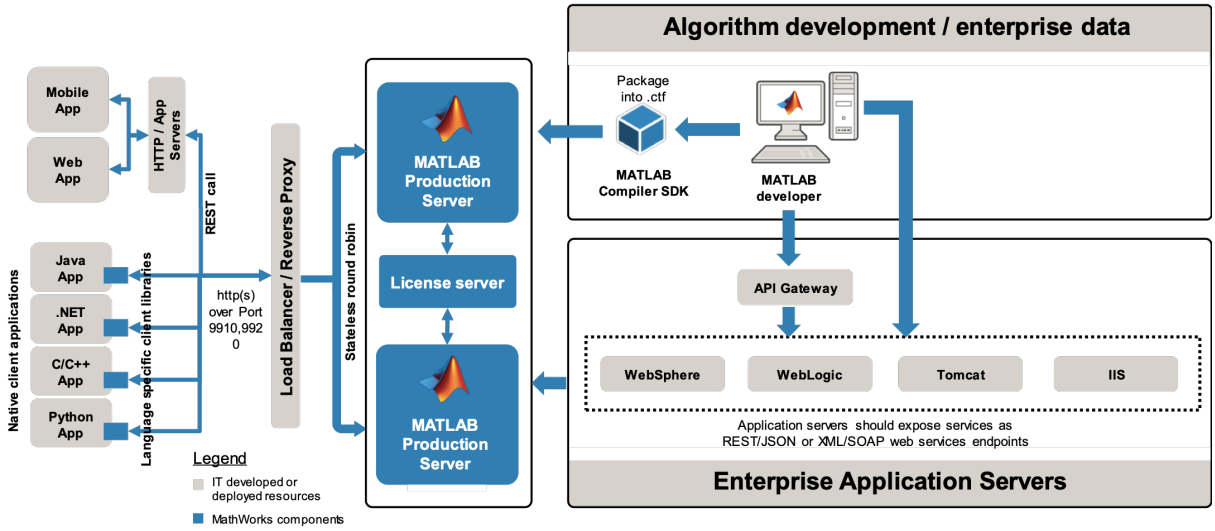


*Figure 19. An application server integration pattern for MATLAB Production Server.*

## Transactional System Integration Pattern

Operational and transactional systems come in many shapes and sizes. Often they provide APIs or language libraries that can be called from C/C++, Java, or .NET. More modern systems may provide a REST-style interface with JSON message payloads. It is possible to call these legacy systems from within your MATLAB code. The following functions are used in MATLAB to load the libraries/ assemblies/jar files so that their methods can called:

- C/C++: `loadlibrary()` or `clib` *interface*
- Java: `import`
- NET: `NET.addAssembly()`
- Python: prepend `py.` in front of the Python module and function name within your MATLAB code

These interfaces will define their own data types and parameters, or they may use XML as a payload. Newer REST-style interfaces typically use JSON data types (Figure 20).

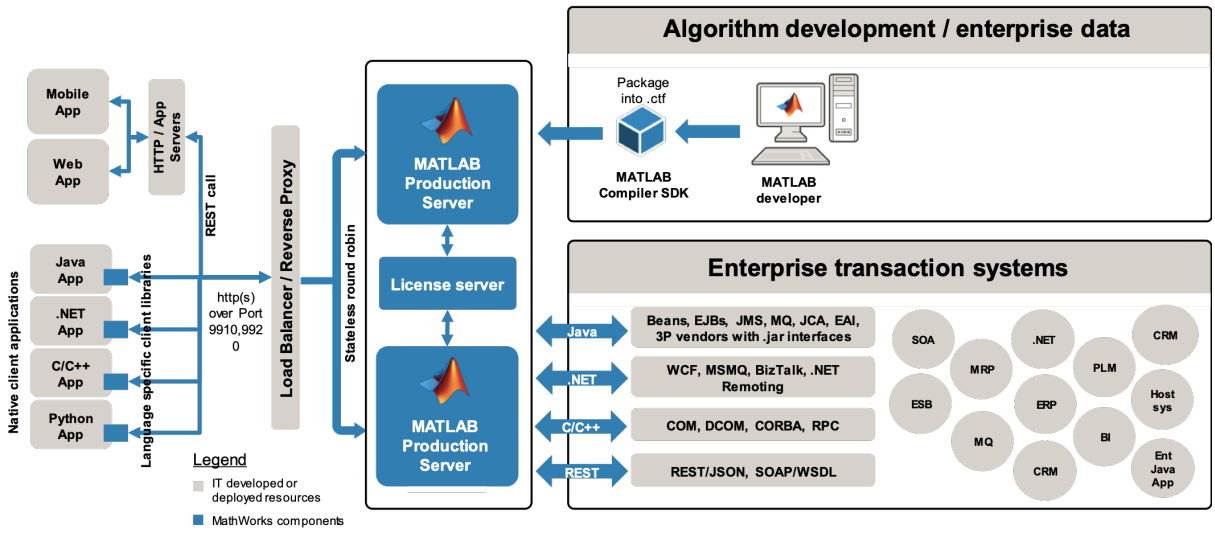


*Figure 20. A transactional server integration pattern for MATLAB Production Server.*

## Operational System (OSIsoft PI System) Integration Pattern

OSIsoft is a popular data archive and operational system used in many process industries to collect data from a wide variety of assets (Figure 21).

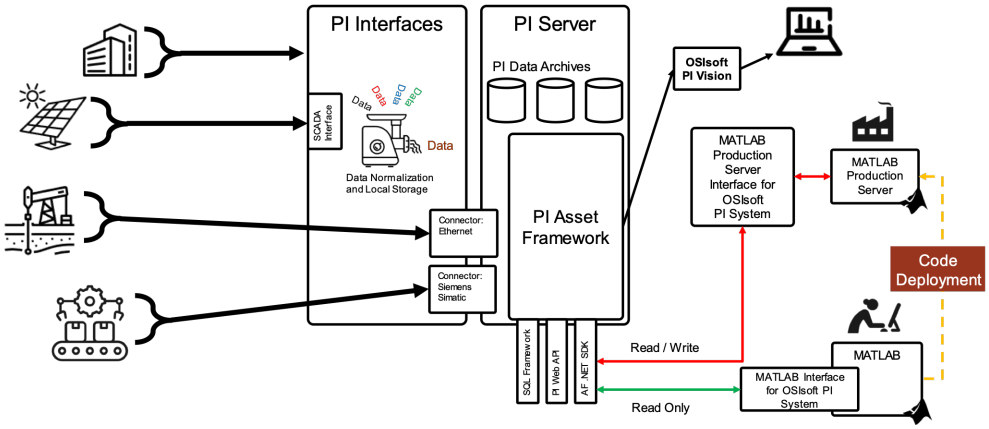


*Figure 21. An operational system integration pattern for MATLAB Production Server.*

The MATLAB Production Server Interface for OSIsoft PI System is a Microsoft Windows Service that periodically communicates with PI Asset Framework (AF) through the PI AF SDK to read asset attribute data into MATLAB deployed functions where the data can be analyzed. The output of the function can be written back to PI AF. For more details, email *mwlab@mathworks.com*.

## Third-Party Visualization Application Integration Pattern

MATLAB Production Server analytics and application results can be visualized using a variety of third-party dashboard tools such as Tableau, Spotfire, Qlik, and PowerBI. Each tool requires its own specific integration mechanism (Figure 22). These integrations are provided by MathWorks outside of the core MATLAB Production Server product as pilot support packages.
Email *mwlab@mathworks.com* for more information.

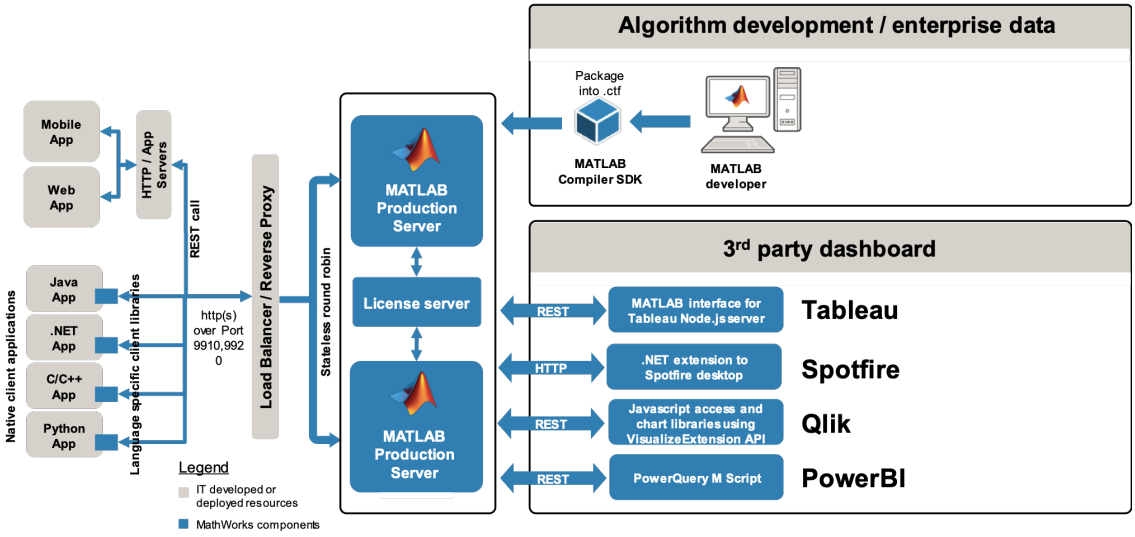


*Figure 22. A third-party visualization application integration pattern for MATLAB Production Server.*

# Operations Runbook

## Installation

Refer to the *MATLAB Production Server Installation Quick Start Guide* for instructions on how to setup and configure your MATLAB Production Server and supporting infrastructure.

## Post-Installation Steps

### Setting Up a Reverse Proxy

MathWorks recommends a reverse proxy (often serviced by the same appliance and software as a load balancer) in front of the MATLAB Production Server instances. This configuration allows you to mask the actual physical addresses of your server instances. This configuration also allows you to create secure transport/SSL connections from the client applications to the reverse proxy while keeping the connections from the reverse proxy to the server instances inside the firewall as unencrypted http traffic. This approach helps to improve performance.

### Setting Up Load Balancing and Failover

A load balancer (doing double duty as a reverse proxy) can be placed in front of multiple MATLAB Product Server instances or hosts to spread the load and provide failover. The load balancer should be configured for stateless round-robin policy. For example, if you are using NGINX, your `load-balancer.conf` file should look similar to this (change to addresses/hostnames that reflect your network configuration):

```
upstream backend {
     Server 192.168.0.1
     Server 192.168.0.2
     Server 192.168.0.3
}
server {
     listen 443;
     location / {
            proxy_pass http://backend;
     }
}
```

## Post-Install Configuration

Once installed, the server is configured using the `main_config` file and running commands on the command line, or by using the web management dashboard (only available on Windows and Linux platforms). Choosing between using the command line and the web dashboard depends on your needs, as shown in Table 9.

| | Command Line | Web Dashboard |
|---|---|---|
| Ease of use | Requires some familiarity | Easiest |
| Automate with scripting | Yes | No |
| Create server instances | Yes* | Yes* |
| Monitor instances/performance | Static numeric data | Dynamic graphical chart |
| Manage applications | Yes | Yes |
| Configure settings | Yes | Yes |
| Start/stop servers | Yes | Yes |
| Run as a service | Yes | No** |

* Server instances created using the command line must be managed using the command line. Similarly, server instances created in the web management dashboard must be managed using the web dashboard. Note that the web management dashboard only manages instances of one physical or virtual machine. It cannot manage instances across multiple machines.

**The web management dashboard runs as a user process, not a service. It is possible to run the dashboard as a Windows service using a third-party utility such as *nssm*, but this is not officially supported.

*Table 9. Choosing between the command line and web dashboard to configure MATLAB Production Server.*

Figure 23 shows an example of configuring MATLAB Production Server using the command line.

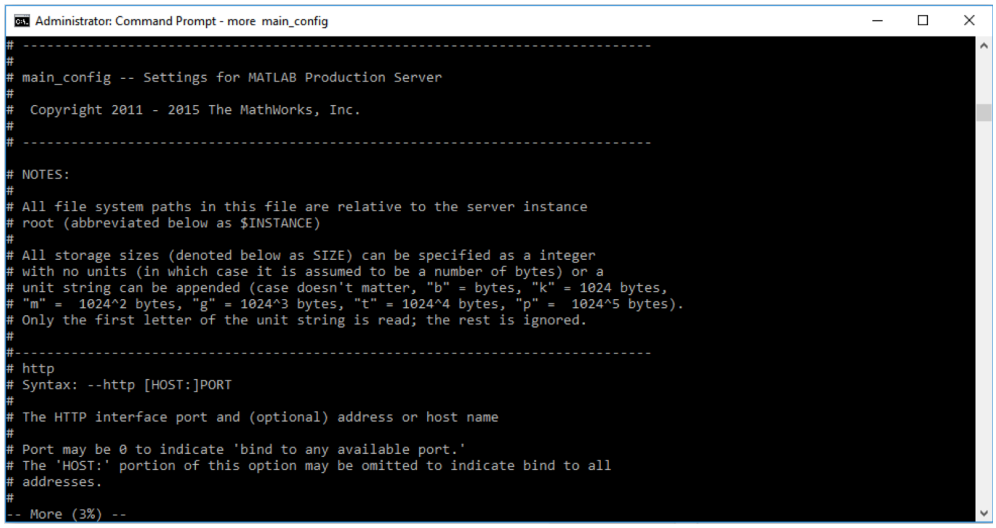


*Figure 23. Configuring MATLAB Production Server using the command line.*

The main configuration steps after installation are as follows:

- Set the MCR Root to point to the MATLAB Runtime version you wish to support. This is done initially using the `mps-setup` utility and subsequently by editing `main _ config` and adding a new `-mcr-root` property for each runtime.
- Create server instances.
- Edit `main _ config` to meet your requirements:
  - http host and port settings, SSL settings
  - X.509 certificate settings
  - `--num-workers` should be set to the number of physical and virtual cores minus 1–2 (reserved for the master process)
  - `--num-threads` should be set to half the num-workers value
  - `--worker-restart-interval` should be set to a time that will trigger outside of normal working hours
  - `--log-rotation-size` can be increased from 100 MB if you wish to keep more data
  - Set `--cors-allowed-origins` to the hosts that will be allowed to access MATLAB Production Server

## Monitoring

The web dashboard provides a variety of monitoring metrics (Figure 24). The two metrics to look out for are:

- Requests in queue
- Throughput

If your requests in queue are increasing and throughput has leveled out or is decreasing, you probably need to increase the amount of hardware resources on your host server to handle the increased demand.

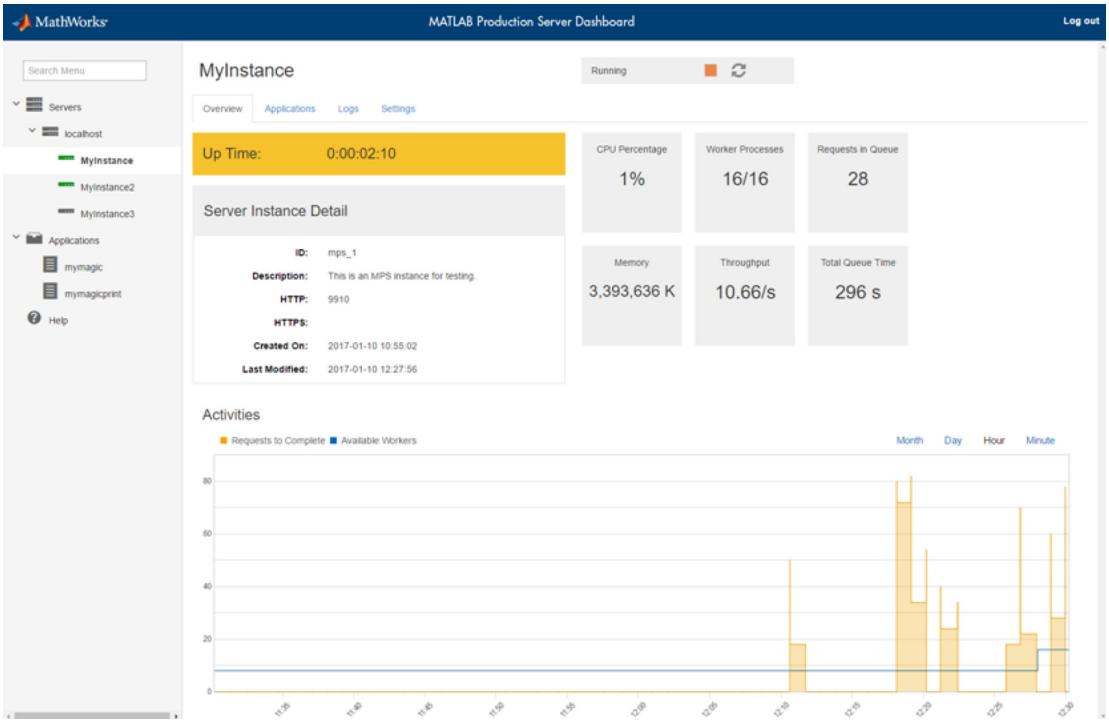


*Figure 24. Monitoring metrics using the MATLAB Production Server web dashboard.*

**Special Note:** Asynchronous REST calls that do not retrieve and/or delete their requests will cause the MATLAB Production Server master process memory usage to spike. One way to alleviate this is to set `request-timeout` to a low interval such as 1 hour, or to limit `server-memory-threshold` to a specified amount (the default is 2 GB).

## Event Management

The default log settings are appropriate for most situations. As a best practice, the log files should be located on a local drive instead of a network drive (since the loss of network drive connectivity will significantly drive up the MATLAB Production Server master process memory usage as it continuously tries unsuccessfully to write to `stdout`).

MathWorks does not recommend writing your own log handler (although it is supported). An improperly written log handler can significantly impact memory usage.

## Creating New Server Instances

A server instance must be created to host MATLAB deployable archives (.ctf files). A physical or virtual machine may host one or more server instances. Server instances can be created either using the web management dashboard or through the `mps-new` command. Note that server instances created in the web management dashboard can only be managed from the web management dashboard, and server instances created from the `mps-new` command can only be managed from the command line. When you create a new server instance, you give it a path where that instance will store its configuration and data files.

## Starting/Stopping the Server

The MATLAB Production Server instance can be started or stopped using either the web management dashboard or the `mps-start` and `mps-stop` commands.

## Checking the State of the Server

The state of the server can be checked using the `mps-status -s` command or through the web management dashboard. The web management dashboard provides a near-real-time view of the running state of the server including metrics and monitoring data, whereas `mps-status` provides a point-in-time snapshot.

## License Management

Licenses are handled by a license server that is installed and configured separately from MATLAB Production Server (see instructions).

Certain license management tasks are performed by editing the license entries in the `main_config` of MATLAB Production Server. These include:

- `--license` specifies the license servers and/or license files that will be referenced. More than one license server or files may be specified. A combination of license servers and license files may also be used. Separate each entry with a colon (:). MATLAB Production Server will search for licenses in the order that you add the entries.
- `--license-grace-period` specifies the maximum length of time in minutes in which MATLAB Production Server responds to HTTP requests, after license server heartbeat has been lost. The longest period that can be specified is 2.5 hours (which is also the default value).
- `--license-poll-interval` specifies the interval of time that must pass, after license server heartbeat has been lost and MATLAB Production Server stops responding to HTTP requests, before license server is polled, to verify and check out a valid license. Polling occurs at the interval specified by `license-poll-interval` until a license has been successfully checked out.

## Adding New Applications

Once the MATLAB users are done creating their algorithm, model, or application, they use the MATLAB Production Server Compiler app to generate the deployable archive. The generated file is found in the `for_redistribution` folder of the project. In the `for_redistribution` folder you should be provided with the file with the extension .ctf. Copy the .ctf file to the `auto_deploy` folder of the server instance. MATLAB Production Server monitors the folder and hot deploys the application without any intervention on your part and without any server restart.

It is possible to use the `mcc -W` command-line utility in a CI/CD pipeline to build the .ctf and then automate the copying of the file to MATLAB Production Server using scp/sftp/network file copy.

## Configuring Security

### Transport Layer Security

First obtain a server certificate in PEM format from your certificate authority. MathWorks recommends that you use an encrypted private key. Store the passphrase for your private key in a file with restricted access (preferably on a full disk encrypted volume, or at the least an encrypted folder). Now you can modify the `main _ config` configuration file with the following lines:

```
--https 9920
--x509-cert-chain ./x509/my-cert.pem
--x509-private-key ./x509/my-key.pem
--x509-passphrase ./x509/my-passphrase
```

where `my-key.pem` is your certificate file and `my-passphrase` is a file containing your passphrase.

### Authentication

Authentication can be performed in three ways:

**x.509 Client Certificates**

Each connecting client must have a client certificate installed. The Common Name (CN) in the client certificate is used to identify each client connecting to MATLAB Production Server. A few steps need to be performed on the server in order to enable client authentication. First you need to enable peer verification by adding the `-ssl-verify-peer-mode` switch to `verify-peer-require-peer-cert` in `main _ config` as in the following:

```
--ssl-verify-peer-mode verify-peer-require-cert
--x509-use-system-store
--x509-use-crl
```

If you are using a Linux machine, you may choose to use your operating system's certificate authority (CA) store or provide a custom specific CA store. Windows users must specify a specific CA store using `-x509-ca-file-store /path/ca _ file.pem`. If you wish you may also specify a certificate revocation list using the `-x509-use-crl` property. The CA store is checked for any CRLs.

**Azure Active Directory**

Azure Active Directory can be used to authenticate users by creating an access control configuration file. This file is named `azure _ ad.json` and at a minimum you need to specify your Azure tenant ID and server app ID. For example:

```
{
   "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
}
```

*Learn more about how to configure the Azure Active Directory file.*

**Kerberos**

Currently Kerberos authentication only works when running on Windows Server operating systems with a Windows Key Distribution Center (KDC). Once a user logs into his or her Windows desktop, those credentials can be passed to MATLAB Production Server for delegation purposes. The user's credentials are delegated to the next hop web server or database server that is called from the MATLAB code (`webread`, `webwrite`, HTTP interface, and Database Toolbox functions) hosted on MATLAB Production Server.

*Learn more about how to set up Kerberos authentication.*

## Access Control

By default, users have access to all MATLAB functions in CTFs installed on a server. Access control can be enforced using either the client certificate identities or through Azure Active Directory policies.

**Client Certificate Identity**

*Restrict access to the server:* You can authorize only users with specific common names as specified in their x.509 client certificate to access the server. To restrict access to the server, add this parameter to the configuration file:

```
--ssl-allowed-client johnsmith,maryjones,michaeljordan
```

Restrict access to individual CTFs: If you require more fine-grained access control, you can specify which individuals have access to each CTF installed on the server. You do so by adding the following parameters to the configuration file:

```
--ssl-allowed-client johnsmith:app1,app2
--ssl-allowed-client maryjones,michaeljordan:app3,app4
```

In this example, `johnsmith` is allowed access to `app1` and `app2`, while `maryjones` and `michaeljordan` are allowed access to `app3` and `app4`.

**Azure Active Directory Policy**

*Restrict access to individual CTFs:* An admin creates a policy JSON file, which by default is called `ac _ policy.json`. Within the file, you create rule blocks that specify who (which individual or group) has access to which CTF. For example:

```
{
  "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", ...
            "bbbbbbbb-bbbb-bbbb-bbbb-bbbbbbbbbbbb"]  },
          "resource": { "ctf": ["monteCarlo", "fastFourier"] },
          "action": ["execute"]
        }
      ]
    }
  ]
}
```

*Learn more about how to create a policy JSON file.*

## Configuration Changes

Any changes made to the `main _ config` file require a restart of the server in order to pick up the changes.

## Upgrading

Upgrades to the server require a shutdown and restart. Upgrading typically involves running the setup/installation for the newer version on top of the existing server. In order to prevent downtime, it is recommended that you run a high-availability topology with a load balancer that can redirect traffic to the alternate server while you are upgrading.

## Running as a Windows Service

When creating a new server instance, you may specify the `--service` flag to specify that you want the instance to be a Windows service instead of a standard process. This allows your instance to automatically start/restart when the machine boots/reboots. An existing server instance can also be converted to a Windows service using the `mps-service -C <your instance path> create` command. Note that these steps only work for server instances created using the command-line interface.

The dashboard runs as a standard process by default, and server instances created using the dashboard do not automatically start/restart on boot. There is a workaround to make the dashboard run as a Windows service using the NSSM utility, but MathWorks does not officially support this configuration.

## Restarts

Under normal circumstances, there is no need to restart MATLAB Production Server on a regular basis. A restart should only be performed if there is anomalous behavior. However, you may configure regular worker process restarts.

As worker processes run 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 a set period or when they have consumed a predetermined amount of memory. These reset parameters can be set in the `main _ config` file.

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

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

Alternately, check every hour and restart workers when they consume 1 GB of memory:

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

## Backup/Recovery

The MATLAB Production Server binaries, data (applications/functions), and configuration files are stored underneath the folder that you specified when you ran the installer. Your backup system should be configured to back up this entire folder at regular intervals.

## Application Development

MATLAB Production Server supports a broad variety of client applications through either language-specific libraries or REST endpoints, as shown in Figure 25.

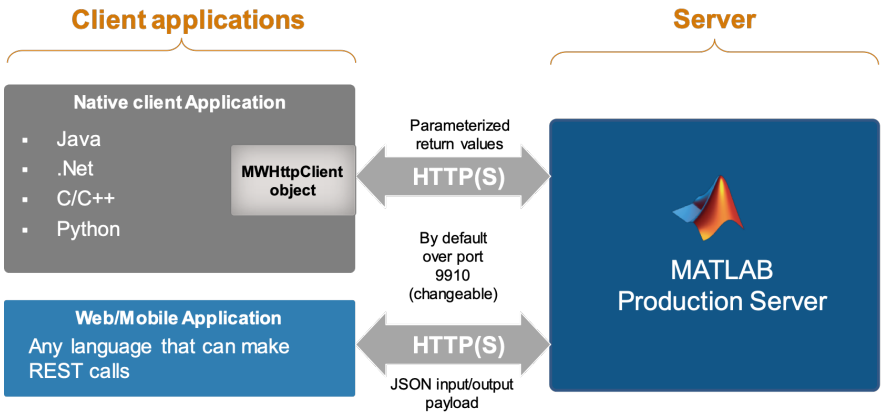


*Figure 25. MATLAB Production Server support for client applications.*

Note that MATLAB Production Server does not generate a user interface for your MATLAB application; instead, it exposes the functions within the MATLAB application as APIs that can be called by your custom application. Your custom application can be based on Java, C/C++, .NET, or Python. It can also be a web application or mobile app that makes REST calls to MATLAB Production Server. Rapid application development/low-code UI builders can also be employed as long as they incorporate the client libraries provided by MathWorks or make REST calls.

### Language-Specific Guidelines

This section highlights several examples that demonstrate how to call a MATLAB function from within your custom application. All the examples will all be based off the same, simple MATLAB function that takes four double arguments and returns a double as the result. The MATLAB function is depicted here:

```
function price = pricecalc(maturity_value, coupon_yield, interest_rate, ...
  num_payments)

    C = coupon_yield;
    N = num_payments;
    i = interest_rate;
    M = maturity_value;

    price = C * ( (1 - (1 + i)^-N) / i ) + M * (1 + i)^-N;
end
```

### .NET

*Download the client library that corresponds to your version of MATLAB Production Server*. Unzip the file and locate the library for .NET. Add a reference to `dotnet\MathWorks.MATLAB.ProductionServer.client.dll` to your project in Visual Studio:

1. Define function signature for the MATLAB function as an interface
2. Bind to server using URI of host and deployable archive
3. Call function to be run and pass in parameters; the function will return result(s)

Note that the .NET interface must have the same name, same method name, and same number of inputs, outputs as the MATLAB function.

```
using Mathworks.MATLAB.ProductionServer.Client;

    public interface BondTools
    {
             double pricecalc(double maturityValue, double couponYield, ...
               double interestRate, double numPayments);
    }

    using (MWClient client = new MWHttpClient())
    {
             double maturityValue;
             double couponYield;
             double interestRate;
             double numPayments

             BondTools bondtool = client.CreateProxy<BondTools>...
               (new Uri("http://toolserve.mathworks.com:9910/BondToolsPkg"));

             double price = bondtool.pricecalc(maturityValue, couponYield, ...
               interestRate, numPayments);
    }
```

### C/C++

*Download the client library that corresponds to your version of MATLAB Production Server*. Unzip the file and locate the library for C. Include the header file found in `\client\c\include\mps\client.h` to your source file.

```
#include <mps/client.h>
   mpsClientRuntime* mpsruntime = mpsInitializeEx(MATLAB Production ...
     Server _ CLIENT _ 1 _ 1);
   mpsClientConfig* config;
   mpsStatus status = mpsruntime->createConfig(&config);
   mpsClientContext* context;
   status = mpsruntime->createContext(&context, config);

   //Get the input arguments
   double maturityValue = <get the value from somewhere>;
   double couponYield = <get the value from somewhere>;
   double interestRate = <get the value from somewhere>;
   double numPayments = = <get the value from somewhere>;

   //Pack the input arguments into an mpsArray
   int numIn = 4;
   mpsArray **inVal = new mpsArray[numIn];
   inVal[0] = maturityValue;
   inVal[1] = couponYield;
   inVal[2] = interestRate;
   inVal[3] = numPayments;

   //Create an array to hold the result
   int numOut = 1;
   mpsArray **outVal = new mpsArray[numOut];

   //Call the function
   status = mpsruntime-
>feval(context,http://toolserve.mathworks.com:9910/BondToolsPkg,numOut,...
  outVal,numIn,(const mpsArray**)inVal);

   //free memory
   delete[] inVal;
   delete[] outVal;
   mpsruntime->destroyConfig(config);
   mpsruntime->destroyContext(context);
   mpsTerminate();
```

The compiler will need access to `client.h` and the linker will need access to the libraries in `\client\c\<arch>\lib`. Depending on your architecture, you will need different libraries:

- Windows: .dll
- Linux: .so
- Mac: .dylib

At runtime, the libraries in the `\client\<arch>\lib` directory will need to be added the system PATH environment variable.

### Java (Static Invocation)

*Download the client library that corresponds to your version of MATLAB Production Server*. Unzip the downloaded file and locate the library for Java. Add `\client\java\mps _ client.jar` to the classpath for your project:

1. Define function signature/interface
2. Bind to server using URI of host and deployable archive
3. Call function to be run and pass in parameters; the function will return result(s)
4. Close the connection to the server and deployed archive

Note that the interface can be named any valid Java name, but the method name must be exactly the same as the MATLAB function name as well as support the same number of input and output parameters.

```
import com.mathworks.mps.client.MWClient;
import com.mathworks.mps.client.MWHttpClient;
import com.mathworks.mps.client.MATLABException;

   interface BondTools
   {
            double pricecalc(double maturityValue, double couponYield, ...
              double interestRate, double numPayments) throws IOException,...
              MATLABException;
   }

   MWClient client = new MWHttpClient()

   double maturityValue;
   double couponYield;
   double interestRate;
   double numPayments

   BondTools bondtool = client.createProxy<BondTools>...
     (new URL("http://toolserve.mathworks.com:9910/BondToolsPkg"), ...
     BondTools.class);

   double price = bondtool.pricecalc(maturityValue, couponYield, ...
     interestRate, numPayments);

   client.close();
```

### Java (Dynamic Invocation)

With dynamic invocation, there is no need to define an interface; instead, you define signatures and function at runtime (the `mps _ client.jar` must be added to the classpath of your project; see instructions in Java Static Invocation section to obtain the JAR):

1. No need to define interface
2. Define URL using `MWInvokable` and `createComponentProxy` method
3. Call invoke method and pass function name and parameters

```
import com.mathworks.mps.client.MWClient;
import com.mathworks.mps.client.MWHttpClient;
import com.mathworks.mps.client.MATLABException;

   MWClient client = new MWHttpClient()

   double maturityValue;
   double couponYield;
   double interestRate;
   double numPayments;

   MWInvokable bondtool = client.createComponentProxy...
     (new URL("http://toolserve.mathworks.com:9910/BondToolsPkg"));

   double price = bondtool.invoke("pricecalc", maturityValue, ...
     couponYield, interestRate, numPayments);

   client.close();
```

### Python

Python integration is slightly different in that you have to run an installer first:

1. *Download the client library that corresponds to your version of MATLAB Production Server*
2. Unzip the downloaded file and copy the contents of the `\client\python` folder to your development environment, then run `python setup.py install`
3. Import the MATLAB client library in your python code using `import matlab`
4. Open a connection to the server/deployed archive
5. Call the function

```
import matlab
from production_server import client

client_obj = client.MWHttpClient("http://localhost:9910")
Coupon = matlab.double(5.51)
Num_payments = matlab.double(12.0)
Interest_rate = matlab.double(6.25)
Maturity_value = matlab.double(500.0)

Result = client_obj.BondTools.pricecalc(Coupon, Num_payments, ...
  Interest_rate, Maturity_value)
```

### Synchronous REST

Web applications are typically written in JavaScript and call MATLAB Production Server using the REST interface with JSON payloads:

1. Enable Cross Origin Resource Sharing (CORS) on MATLAB Production Server by editing the `main_config` file.
2. Make a synchronous REST call using `XMLHttpRequest()`. Alternatively you can use JQuery. `ajax()` or `Jquery.post()`.

```
var cp = parseFloat(document.getElementById('coupon_payment_value').value);
var np = parseFloat(document.getElementById('num_payments_value').value);
var ir = parseFloat(document.getElementById('interest_rate_value').value);
var vm = parseFloat(document.getElementById('facevalue_value').value);

var request = new XMLHttpRequest();
var url = "http://localhost:9910/BondTools/pricecalc";
var params = { "nargout":1,"rhs": [vm, cp, ir, np] };

request.open("POST", url);
request.setRequestHeader("Content-Type", "application/json");
request.onload = function()
 {    //Use MATLAB Production Server RESTful API to check HTTP Status
                if (request.status == 200)
                 {
                      // Get the lhs return value
          var result = JSON.parse(request.responseText);
          // display result from result.lhs[0].mwdata;
     }
     request.send(JSON.stringify(params));
 }
```

### Asynchronous REST Call

Asynchronous calls take a little bit more work to perform:

1. Like synchronous requests, you must enable Cross Origin Resource Sharing (CORS) on MATLAB Production Server by editing the `main _ config` file.
2. While synchronous calls block until a result is return, an asynchronous request to MATLAB Production Server will immediately return a result with the correlation ID for that request.
3. You would typically use a jQuery Ajax request to make an asynchronous call and then poll the server using `setTimeout()` to determine if the call has finished execution.
4. In the polling function, make additional recursive AJAX calls using the 'UP' and 'ID' results to form the request URI.

```
$.ajax(url, {
                  data: JSON.stringify(params),
     contentType: 'application/json',
                  method: 'POST',
                  dataType: 'json',
                  success: function(response) {
     pollServer(response);
                  }
              });

              function pollServer(request) {
                    setTimeout(function() {
     var newSeq = parseInt(request.lastModifiedSeq) + 1;
                    var queryURI = hostname + request.up + "?since=" + ...
                      newSeq + "&ids=" + request.id;
                    $.ajax({
                          url: queryURI,
                          method: 'GET',
                          dataType: 'json',
                          success: function(response) {
                               //Poll again if no data about the request was
                               //received.
                               if (response.data.length == 0) {
                                    pollServer(request);
                                    return;
                               }
     .200);
}
```

### MATLAB as the Client App

You can call functions deployed to MATLAB Production Server from the MATLAB desktop. Consolidating your functions on MATLAB Production Server allows you to maintain a single reference version that everyone in the organization can call. It eliminates the burden of distributing and updating your code since you only have to deploy it to one central location. A `File Exchange support package` allows you to "install" MATLAB Production Server remotely deployed functions to your MATLAB desktop. Run the add-on manager and select the functions you want to use, then call those remotely deployed functions as if they were locally installed.

Note: Using MATLAB as the client app and using the Test Client within MATLAB Compiler SDK can have unpredictable results since both are running within the same MATLAB process.

### Testing Your Deployed Function with Postman

The `Postman app` lets you easily send, receive, and examine REST API requests. Once it is installed, you will find an icon for Postman on your desktop. Postman allows you to construct REST API requests and inspect the results. This makes it very useful to test your deployed applications on MATLAB Production Server. Create a new request (optionally add it to a collection), then specify a few basic fields:

1. Set the request to HTTP POST
2. Provide the URL to your MATLAB Production Server server, e.g.,
   `http://<mps_hostname>:9910/<ctf_name>/<function_name>`
3. In the Headers tab, add a key called `content-type` and provide the value `application/json`
4. In the body tab, click on the raw radio button and specify the JSON input payload, e.g.,
   `{ "nargout":1, "rhs":[10000,0.08,0.06,40] }`

**Note:** The `mps.json.encoderequest` and `mps.json.decoderequest` helper functions allow you easily create and parse the JSON payloads you send and receive from MATLAB Production Server.

Ensure your MATLAB Production Server server is running and click Send to send the request. You should see the return results in the Response panel.

Figure 26 shows you the JSON formatted response that MATLAB Production Server is returning. The format and values will help you debug your function as well as understand how to parse and extract the result data for display in your application. For example, in this case you want to grab `result.lhs[0].mwdata` to display in your application.

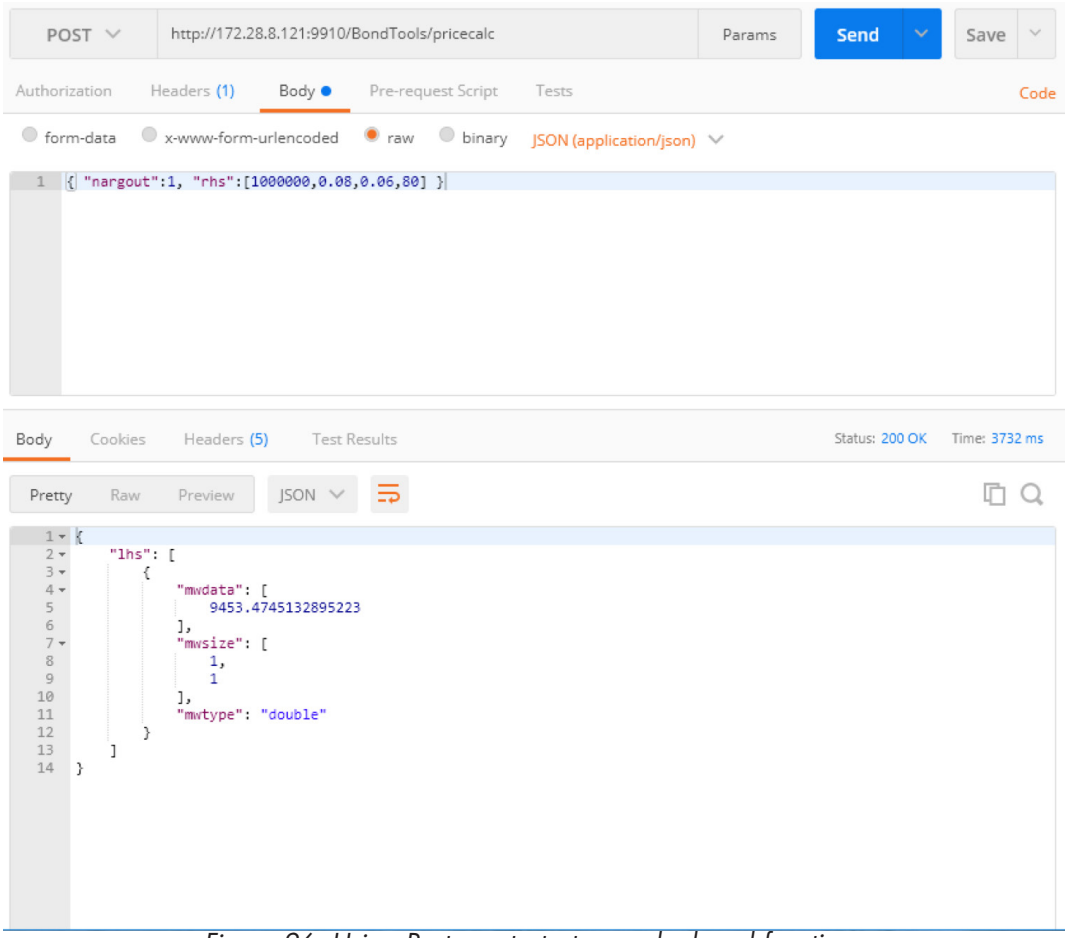


*Figure 26. Using Postman to test your deployed function.*

## Mobile App Development Guidelines

Most mobile apps today call services via a REST API and pass data using JSON payloads.

Third-party frameworks, for example for iOS apps using AlamoFire and SwiftyJSON, are readily available.

Either synchronous or asynchronous calls may be used.

### iOS Swift Example

This is a very basic example in Swift that shows how to construct a request, send it to MATLAB Production Server, and retrieve the result. Using third-party frameworks makes it much easier to package the JSON request and send the request.

```
import Alamofire
Import SwiftyJSON

Let parameters: Parameters = [ "nargout":1,"rhs": [vm, cp, ir, np] ]

Alamofire.request("http://localhost:9910/BondTools/pricecalc", ...
  parameters: parameters).responseJSON { response in

        let json = JSON(data: response.result.value)
        if let price = json["lhs"][0].double {
          //Display the price value
        }

}
```

## Test Locally on Your Workstation Using the MATLAB Compiler SDK Development and Test Environment

You can easily test the functionality of your deployed code without access to a live MATLAB Production Server by using the built-in server in MATLAB Compiler SDK. When you are ready to test your code, simply run the MATLAB Production Server Compiler app from the Apps drawer or by running `deploytool` from the command prompt. This will launch MATLAB Compiler SDK as shown in Figure 27.

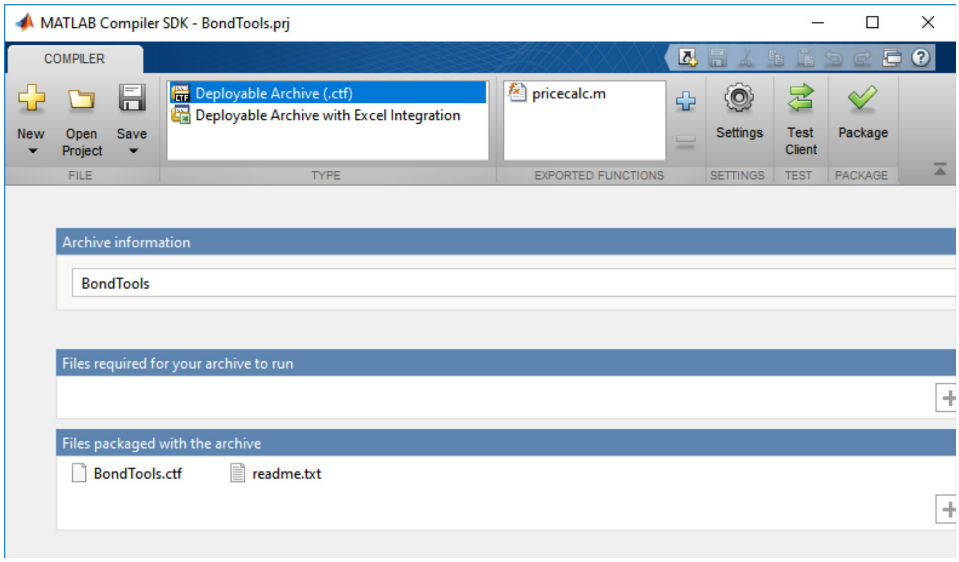


*Figure 27. Using the MATLAB Production Server Compiler app to launch MATLAB Compiler SDK.*

After you have specified the name and added the required files for your project, click **Test Client** to start the built-in MATLAB Production Server instance. This is a single-worker MATLAB Production Server instance that allows you to test the functionality of your code as if it were running in an actual MATLAB Production Server environment (Figure 28).

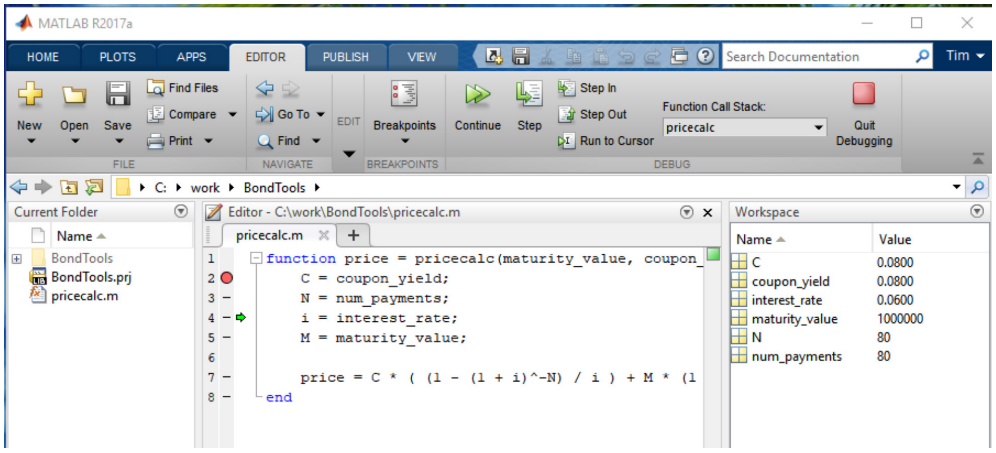


*Figure 28. Testing the functionality of your code using the built-in MATLAB Production Server instance in MATLAB Compiler SDK.*

An extremely useful feature is the ability to set breakpoints in your MATLAB code and step through it line by line as your application makes requests. Postman is a great way to exercise the function by passing it different parameters and checking to see if the result is correct.

This test MATLAB Production Server instance shows you which function is called as well as diagnostic information in the log (Figure 29). This can be very helpful to debug your code.

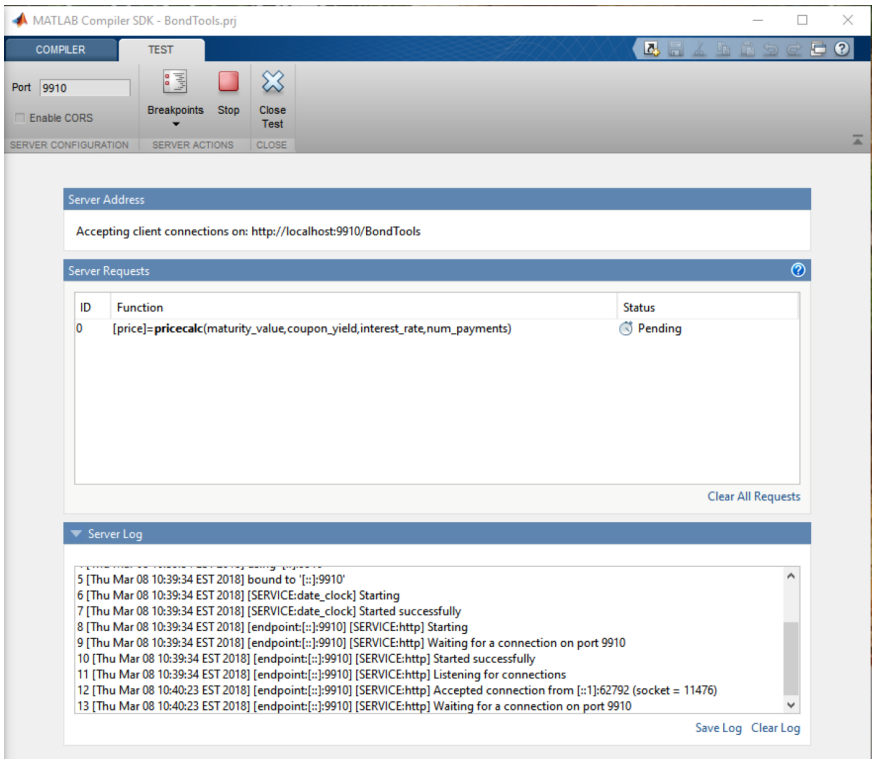


*Figure 29. Reviewing function calls and diagnostic information using the built-in MATLAB Production Server instance in MATLAB Compiler SDK.*

## Make Your Functions Discoverable by Specifying the Signature During Packaging

Previously, application developers need to know the archive name, function name, input parameters, and output parameters to invoke a function hosted on MATLAB Production Server. Typically, this information was provided by the developer of the MATLAB function to the team creating the front-end application. With the introduction of the service discovery RESTful API in R2018a, application developers can now use the returned results to determine which function to call and what input/output parameters are supported by that function. The service discovery RESTful API can be accessed at **`http://<mps _ hostname>:<port>/api/discovery`**.

To enable this discoverability, the MATLAB function developer creates a JSON-formatted descriptive file specifying the function's signature during the compilation phase by clicking **Create File** as shown in Figure 30.

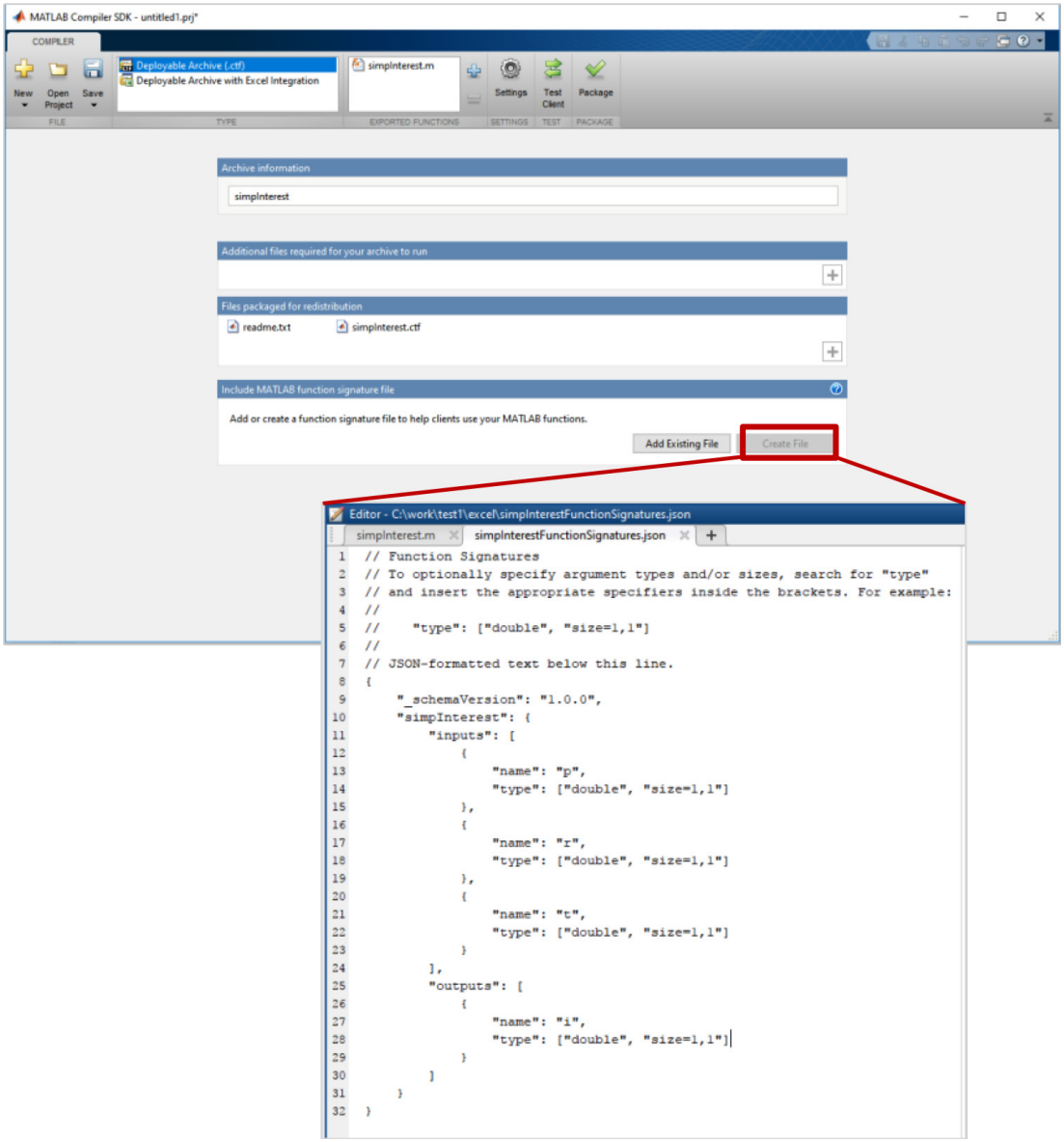


*Figure 30. Specifying a function's signature to enable discoverability.*

This opens a JSON-formatted skeletal template of the function being deployed. You should add details such as the data type for each input and output parameter. The `purpose` JSON key also allows you to add descriptive text for each parameter as well as for the function in general. Once you are done, save the JSON file and compile the deployable archive .ctf file. This embeds the descriptive function signature into the deployable archive, which can then be surfaced when the service discovery endpoint is queried.

*Learn more about the JSON attributes.*

This feature requires MATLAB Compiler SDK R2018a or later and MATLAB Production Server R2018a or later to work. A configuration flag `--enable-discovery` must also be set in the `main _ config` file for your MATLAB Production Server instance.

## Best Practices for MATLAB Application Development

The authors of the MATLAB code and functions should adhere to the following guidelines in order to optimize for production deployment.

### Test Code in MATLAB Thoroughly Before Deployment

It is generally much harder to test and debug deployed code than it is in MATLAB because:

- The deployed code is encrypted.
- Debugger functionality like breaking or stepping through is not available.
- Variables and data cannot be easily inspected.
- Problems with the code may crash the server instance.

Always test the MATLAB code as thoroughly as possible before deployment to reduce time and effort in the long run. The MATLAB client allows you to run your unit tests except in this case pointing to the remote function deployed on MATLAB Production Server.

### General Best Practices

1. Add a semicolon to the end of each statement to prevent extraneous log entries on the server.
2. Avoid MATLAB figure or UI code. If you wish to display figures for debugging or exploratory work, wrap calls to display figures with the `isDeployed` conditional so that it doesn't pop up a plot on the server. Any figures or UIs created during runtime will show up on the server machine, not the client machine. If figures or UIs are required to run to create the function results, make sure to close these figures at the end of your code to avoid leftover windows and leaking resources on the server.
3. If you need to display figures in your client application use `figToImStream()` to convert the figure into an image bytestream. Popular image formats such as jpg, png, bmp, and gif are supported. You will then have to render the image in your client application from the bytestream.

4. Wrap calls to client functions such as `dir` and `filebrowser` with the `isDeployed` conditional.
5. Avoid using `eval()` to execute code dynamically. *See a list of compilable functions.*
6. Refactor your function signature to only include parameters that are absolutely required to calculate the result for the calling application.
7. Avoid using global variables in MATLAB code, as well as static variables in MEX files or Java code.
8. Remember that functions cannot depend on nor change the MATLAB state. Functions deployed with MATLAB Production Server may not always execute on the same instance of the MATLAB Runtime. Each worker accesses a different MATLAB Runtime instance. *Learn more about these limitations.*

## Data

### Minimize Data Size of Function Inputs and Outputs

When a client calls a MATLAB function deployed on MATLAB Production Server, the input arguments and return data are transported over the network with TCP/IP. Thus, large data sizes can potentially slow down performance. The default upper limit for inputs and outputs is 64 MB.

### Store Static Data in Persistent Variables

If the MATLAB function has static data (data that does not change with repeated calls), it helps to store that data in persistent variables in the function, so that the data does not have to be recreated or passed in every time. This improves performance by either reducing the size of the input data going into the function or reducing time to calculate that data. The MATLAB persistent keyword can be used to create persistent data:

```
function foo(dynamicdata, varargin)
persistent staticdatastore;
if nargin == 2
staticdatastore = varargin{1};
end
%do work with staticdatastore and dynamicdata
```

In the above example, when the function is first called, the static data is passed in as the second optional argument and stored in the persistent variable `staticdatastore`. Subsequent calls to the function do not need to pass in the static data, and just calls the function with one argument, the `dynamicdata`.

Note: If you are running MATLAB Production Server R2019b or later, a better approach would be the use the Redis cache. Redis is an in-memory database that allows you to easily store and retrieve values using a put/get mechanism. *Learn more about how to use the Redis cache.*

### Use `varargin` and `varargout` Explicitly for Functions with Variable Inputs and Outputs

MATLAB allows implicitly optional arguments and outputs. For example, if you have the function:

```
function [a1, b1] = foo(a0, b0)
a1 = a0;
if false
b1 = b0;
end
```

You can call this function with just one argument and it works:

```
>> foo(1)
ans =
1
```

This is because `b0` is not utilized due to the conditional statement (a trivialized example of more complex code paths), and the way foo was called, it did not require the second output b1.

This should be avoided for deployed code. Because deployed code needs to generate function signatures for other stricter languages such as C#, Java or C, this type of construct can error when deployed. It is better to specifically follow the **`varargin`** and **`varargout`** constructs provided by MATLAB to handle optional arguments and outputs. For example, modifying the above code:

```
function [a1, varargout] = foo(a0, varargin)
%base operation
a1 = a0;
%initialize to default if second output requested
if nargout == 2;
        varargout{1} = 1;
end

if false
        %if second argument is given
        if nargin == 2;
                varargout{1} = varargin{1};
        end
end
```

### Data Type Marshalling

You can deploy only MATLAB functions to MATLAB Production Server. Classes cannot be deployed and called from your client applications. However, you can call classes from within your functions.

Data type best practices include:

1. Explicitly use `varargin` and `varargout` for functions with variable inputs and outputs.
2. Try to use primitive data types or simple vectors instead of complicated structs in your parameters. The specific data types supported by each language can be found in the documentation:
   a. *Java*
   b. *NET*
   c. *C/C++*
   d. *Python*
   e. *REST/JSON*
3. Avoid using these unsupported data types: MATLAB function handles, complex (imaginary) data, and sparse arrays.

## Language-Specific Best Practices

1. *Java*
2. *.NET and C#*

## Package the MATLAB Code Using the Same Operating System as the MATLAB Production Server Machine

Although MATLAB Production Server archives generated from MATLAB Compiler (.ctf files) are mostly platform independent, there are many cases where platform independence does not hold. If the MATLAB code uses any platform dependent files such as MEX files or native libraries (.dll, .so, .dylib), the resulting .ctf is not platform independent. Additionally, some MATLAB toolboxes, such as Parallel Computing Toolbox™ and Image Acquisition Toolbox™, use platform-dependent files extensively.

As such, avoid packaging on one operating system (for example, Windows) and then deploying on a different system (for example, Linux). If it needs to be done, the best way to ensure working order is to open the .ctf file manually (unzip it) and check to see if it includes any platform-dependent files. Afterwards, test it out extensively.

## Troubleshooting

- *Check the troubleshooting documentation for more information.*
- Make sure the client machine can communicate with the server regularly. Use `ping` or `telnet` to check the connection.

- Make sure the client specifies the correct MATLAB Production Server URL. There could be a typo in the ctf archive name or function name.
- Try using the IP address of the server machine instead of the hostname, to rule out DNS issues.
- Check the log files on the MATLAB Production Server instance. More information can be obtained from the log by specifying a higher logging level in the `main_config` file (for example, `--log-severity trace`).
- Use the `diary` command in the MATLAB code to output information from the MATLAB side during runtime into a file. This can help diagnose MATLAB errors after deployment.

## Parallel Computing

MATLAB Production Server can leverage parallel computing to speed up certain tasks. For example, if you have a function that takes a long time to execute, but it contains easily parallelizable code that can be wrapped in a `parfor` or `parsim`, this would be a good candidate for combining MATLAB Production Server and parallel computing. There are two ways to use parallel computing from MATLAB Production Server: submitting the job to a MATLAB Parallel Server cluster, or submitting the job to a local parallel pool (Figure 31).

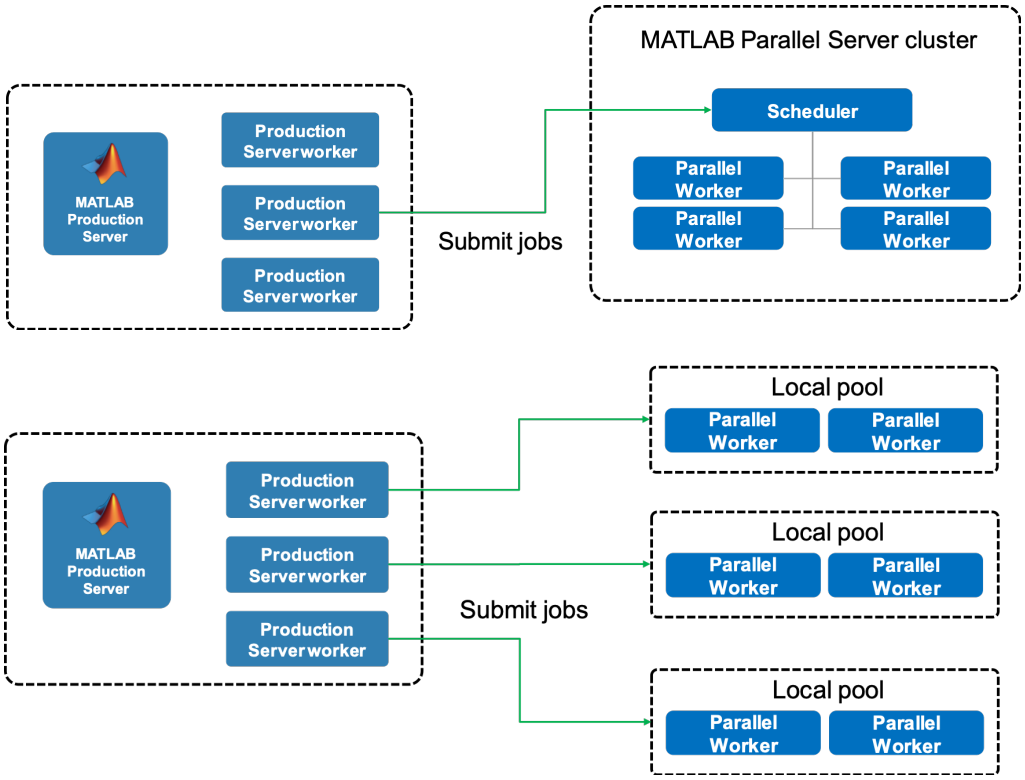


*Figure 31. Two methods for using parallel computing from MATLAB Production server: submitting the job to a MATLAB Parallel Server cluster (top) and submitting the job to a local parallel pool (bottom).*

Make sure to test your MATLAB code that submits to the MATLAB Parallel Server cluster or local pool in your MATLAB client before packaging into a MATLAB Production Server .ctf archive.

Deployed components that use parallel computing resources must be able to find the cluster profile that defines the parallel cluster being used. In MATLAB, these cluster profiles can be found through the Cluster Profile Manager (Figure 32).

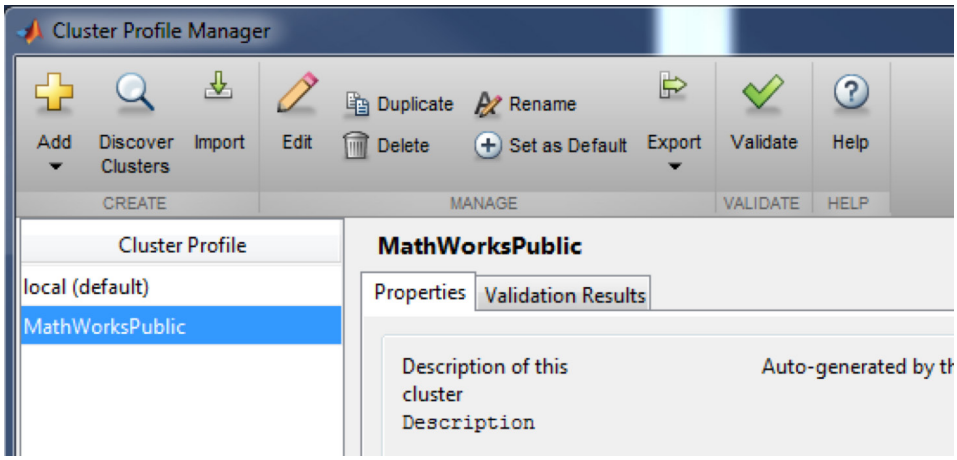


*Figure 32. Using the Cluster Profile Manager in MATLAB.*

Deployed components can find these cluster profiles through three ways:

1. **Automatically load the profile.**
   All profiles present within the MATLAB Cluster Profile Manager will be automatically added to the MATLAB Production Server archive during packaging. As such, the deployed code can use these profiles without any extra effort. This is the easiest method and most appropriate when the profile is not expected to change after deployment (e.g., the cluster name or number of workers or other configuration doesn't change, or new cluster profiles aren't added).
2. **Load the profile from a file after deployment.**
   If the cluster profile is expected to change, the MATLAB code being deployed can manually load the profile from a file when running. The profile description file is usually a .settings file, which can be exported through the Cluster Profile Manager from any MATLAB and can be imported with the `parallel.importProfile()` deployed MATLAB code. For example:

```
clustername = parallel.importProfile('EnkhMPSIntegrationTest.settings');
cluster = parcluster(clustername);
```

3. **Set the profile in the MATLAB Production Server instance configuration file.**
   In the MATLAB Production Server instance configuration file, `mps _ instance _ location/config/main _ config`, you can use the `--user-data` option to pass key-value parameters to the deployed code. This can be used to set a `ParallelProfile` key with a value pointing to where the .settings file for the profile is, and deployed code on this MATLAB Production Server instance will use that profile as the default. For example, the option might look like:

```
--user-data ParallelProfile
/sandbox/menkh/projects/mdcs _ mps _ integ/EnkhMPSIntegrationTest.settings
```

   Note that the profile automatically gets set as the default profile, so the MATLAB code doesn't have to specifically load it; it can just use the default cluster:

```
cluster = parcluster();
```

### Limitations

There are certain limitations when using parallel computing in MATLAB Production Server: Each MATLAB Production Server instance can have only one CTF deployed archive that makes use of Parallel Computing Toolbox functionality. If more than 1 CTF with Parallel Computing Toolbox functions is deployed, the server will experience errors and might crash. If you need to multiple deployed archive CTFs that call Parallel Computing Toolbox, there are some workarounds:

- Combine all the functions from the multiple CTFs into a single central CTF.
- When you want to use CTF #2, stop MATLAB Production Server, delete CTF #1, copy in CTF #2 and then restart the server.
- Create multiple instances of MATLAB Production Server (each running in its own physical or virtual machine). Each instance runs one CTF with Parallel Computing Toolbox functionality.

## Version Management

When deployable archive CTF files are copied to the `auto _ deploy` folder of MATLAB Production Server, they overwrite the previous version and the newest code is automatically hot deployed. Users will have access to the latest functionality immediately; there is no need to restart the server. Note that if you hot deploy a CTF while it is running, the current execution will run to completion since it is already loaded in memory. The next request will then trigger the newly deployed CTF. MathWorks recommends that you check to see that the request queue is empty before hot deploying new CTFs. The request queue status can be checked visually via the web management dashboard or from the `mps-status` command.

However, there may be situations where you wish to maintain multiple versions of your deployed algorithms or models. Out of the box, MATLAB Production Server does not have versioning of

artifacts or API endpoints. MathWorks recommends using file naming, potentially augmented with an API gateway to manage versions.

### Using File Naming

Appending a version number formatted using semantic versioning conventions to the CTF deployable archive is typically used (for example, `Riskmodelv1.0.0.ctf`). Since the CTF filename is part of the URI endpoint, the versions are explicitly callable when you execute the HTTP POST request Figure 33).

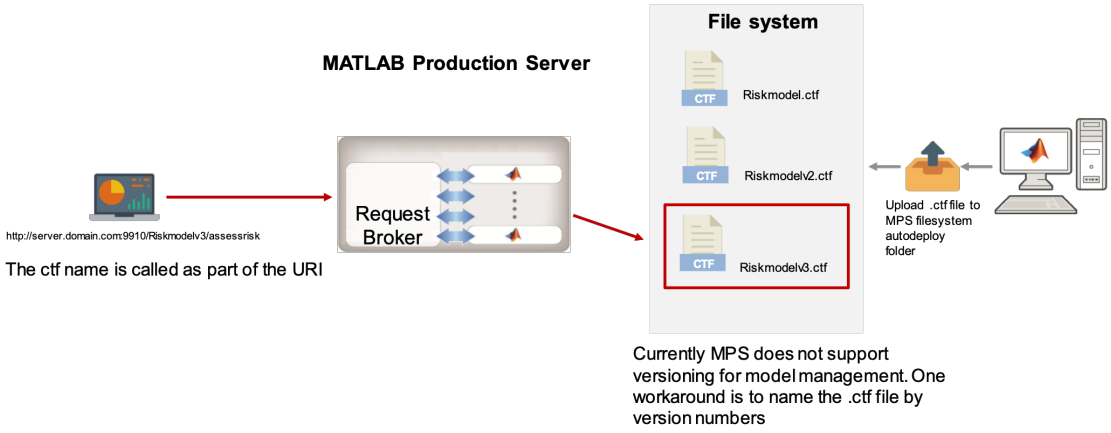


*Figure 33. Using file naming to maintain multiple versions of your deployed algorithms or models.*

### API Gateway

With file naming, your URI is constrained by the path provided by MATLAB Production Server appended by the file name (e.g., `http://mps_server:9010/Riskmodelv2.ctf/function_name`). If you want more flexibility with your URIs, you can introduce an API gateway such as MuleSoft AnyPoint, Apigee, or IBM API Connect to abstract and provide a more friendly URI path (e.g., `http://mps:9010/RiskModel/v2/function_name`).

## Integration with CI/CD Pipelines

MATLAB works with your continuous integration, continuous deployment (CI/CD) pipelines to enable you to setup up automated workflows (Figure 34).

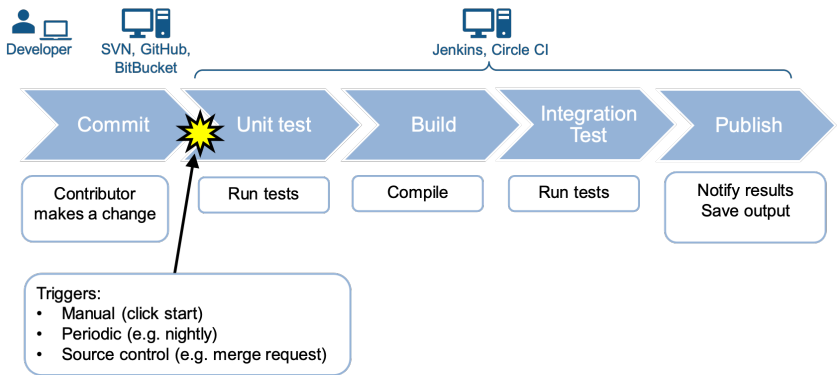


*Figure 34. Automated workflows for continuous integration and continuous development.*

For example, to compile your MATLAB Production Server deployable archive in a Jenkins pipeline:

1. Assuming the MATLAB code is managed in a software configuration management tool (SCM) such as Subversion (CVS and Git are also supported), configure Jenkins to point to your SCM repository (e.g., `http://myscmserver.mycorp.com/svn/project/trunk/...`).
2. Enter your SCM credentials to access your source code.
3. In your strategy, configure Jenkins to trigger based on your preferred criteria; the most common triggers are when a commit is pushed to the SCM (poll SCM), or build periodically.
4. Add an Execute Shell build step to launch MATLAB and run your unit tests (assuming you have written a script called `runALL_THE_TESTS`). For example: `"c:\Program Files\MATLAB\R2019b\bin\matlab.exe" -nodisplay -r "runALL_THE_TESTS"`
5. Add another batch command build step to call MATLAB Compiler SDK to generate the deployable archive:
   `"c:\Program Files\MATLAB\R2019b\bin\mcc.bat" -W CTF:<name_of_your_archive> -U <path\filename.m>`

6. Continue adding build steps to perform any necessary integration tests; for example, you may want to use Newman, the command line version of Postman, to test your RESTful API. Additionally, you could use test frameworks like selenium to test the user interface of your application.
7. Finally, if all the tests pass you should add a build step script to copy the artifacts to your requisite servers, including the deployable archive .ctf file to the `auto _ deploy` folder of MATLAB Production Server.

**Note:** In order to use MATLAB Compiler or MATLAB Compiler SDK in a build server (e.g., Jenkins), all of your engineers working on the project must be licensed to use MATLAB Compiler and MATLAB Compiler SDK (it is not enough to have a single licence of MATLAB Compiler and MATLAB Compiler SDK that lives on the build server). If you do not have sufficient MATLAB Compiler and MATLAB Compiler SDK licenses for all your engineers on the project, you must obtain a Client Access License (CAL) from MathWorks. Please contact your MathWorks account representative to obtain a CAL.

## Resources

- *MATLAB Production Server Overview*
- *MATLAB Production Server Quick Start Guide*
- *MATLAB Production Server Client Libraries*
- *Integrating with Enterprise Systems*
- Cloud reference architectures:
  - *MATLAB Production Server cloud reference architecture for AWS*
  - *MATLAB Production Server cloud reference architecture for Azure*
- Dashboard integrations:
  - *TIBCO Spotfire*
  - *Tableau*
  - *Microsoft Power BI*
- *Data Connectors*
- *Configuring the Redis Cache to Persist Data*

## Support

For product feature requests, licensing, general questions, technical inquiries, or assistance with enterprise integration, email *mwlab@mathworks.com*.

For Technical Support, *visit the Contact Support page.*