

Keyfactor Orchestrators 11.1

Installation and Configuration Guide

Table of Contents

1.0 Introduction	1
2.0 Installing Orchestrators	2
2.1 Orchestrator Job Overview	4
2.2 Universal Orchestrator	6
2.2.1 Preparing for the Universal Orchestrator	7
2.2.1.1 System Requirements	7
2.2.1.2 Create Service Accounts for the Universal Orchestrator	11
2.2.1.3 Configure Certificate Root Trust for the Universal Orchestrator	15
2.2.1.4 Grant the Orchestrator Service Account Permissions on the CAs	16
2.2.1.5 Acquire a Certificate for Client Certificate Authentication (Optional)	18
2.2.1.6 Upgrading the Universal Orchestrator	22
2.2.2 Install the Universal Orchestrator on Windows	25
2.2.3 Install the Universal Orchestrator on a Linux Server	39
2.2.4 Install the Universal Orchestrator in a Linux Container	49
2.2.5 Optional Configuration	61
2.2.5.1 Configure Windows Targets for Remote Management	62
2.2.5.2 Configure the Universal Orchestrator for Remote CA Management	65
2.2.5.3 Installing Custom-Built Extensions	67
2.2.5.4 Configuring Script-Based Certificate Store Jobs	73
2.2.5.5 Configure Logging for the Universal Orchestrator	77
2.2.5.6 Start the Universal Orchestrator Service	80
2.2.5.7 Change Service Account Passwords	81
2.2.5.8 Register a Client Certificate Renewal Extension	88
2.3 Java Agent	95
2.3.1 Preparing for the Java Agent	96
2.3.1.1 Create Service Accounts for the Java Agent	96
2.3.1.2 Create a Group for Java Agent Auto-Registration (Optional)	97
2.3.1.3 Configure Certificate Root Trust for the Java Agent	98
2.3.1.4 Create Environment Variables for the Java Agent on Windows	98
2.3.2 Install the Java Agent on Windows	101
2.3.3 Install the Java Agent on Linux	106
2.3.4 Optional Configuration	113
2.3.4.1 Configure Logging for the Java Agent	113
2.3.4.2 Start the Keyfactor Java Agent Service	116
2.3.4.3 Uninstall the Java Agent	117
2.4 Bash Orchestrator	118
2.4.1 Preparing for the Keyfactor Bash Orchestrator	119
2.4.1.1 System Requirements	119
2.4.1.2 Create a Service Account for the Keyfactor Bash Orchestrator	121
2.4.1.3 Create a Group for Auto-Registration (Optional)	121
2.4.1.4 Certificate Root Trust for the Keyfactor Bash Orchestrator	122
2.4.2 Install the Keyfactor Bash Orchestrator	122
2.4.3 Install Remote Control Targets	127
2.4.4 Optional Configuration	128
2.4.4.1 Configure Logging for the Keyfactor Bash Orchestrator	129
2.4.4.2 Start the Keyfactor Bash Orchestrator Service	130
2.5 Troubleshooting	130
2.6 Appendices	148
2.6.1 Appendix - Generate New Credentials for the Java Agent	148
2.6.2 Appendix - Set up the Universal Orchestrator to Use Client Certificate Authentication via a Reverse Proxy: Citrix ADC	150
2.6.3 Appendix - Set up the Universal Orchestrator to Use Client Certificate Authentication with Certificates Stored in Active Directory	162
2.6.4 Appendix - Set up the Universal Orchestrator to Use a Forwarding Proxy	176
3.0 Glossary	178

4.0 Copyright Notice	188
----------------------------	-----

List of Tables

Table 1: Linux Container Parameters	59
Table 2: Remote CA Configuration Parameters	66

List of Figures

Figure 1: Orchestrator Job Flow	5
Figure 2: Select the Download x64 Option Under Run Console Apps	8
Figure 3: Client Secret for Orchestrator Client in Keyfactor Identity Provider	14
Figure 4: Local Security Policy	15
Figure 5: CA Permissions	17
Figure 6: Microsoft Certificate Template Application Policies for Client Authentication Certificate	19
Figure 7: Microsoft Certificate Template Request Handling for Client Authentication Certificate	20
Figure 8: Installation Files Blocked after Download	25
Figure 9: CA Configuration Settings	66
Figure 10: View Packages as Part of a List	68
Figure 11: View Packages on Individual Pages	68
Figure 12: Find the Latest Version of the Package	69
Figure 13: Download the Package Zip File	69
Figure 14: Universal Orchestrator on Windows NLog.config File	79
Figure 15: Universal Orchestrator on Linux NLog.config File	80
Figure 16: Universal Orchestrator Service	81
Figure 17: Change Service Account Password in Services MMC	82
Figure 18: Application Settings for Client Certificate Renewal	92
Figure 19: Keyfactor Command Permissions Required for Automatic Renewal and Revocation of Client Authentication Certificates	94
Figure 20: Search for System Environment Variables	99
Figure 21: Edit the System Path Environment Variable to Add the Path to Java	100
Figure 22: Add JAVA_HOME System Environment Variable	101
Figure 23: Keyfactor Java Agent Local Installation on Windows	105
Figure 24: Keyfactor Java Agent Local Installation on Linux	110
Figure 25: Configure Logging for Keyfactor Java Agent on Windows	115
Figure 26: Configure Logging for Keyfactor Java Agent on Linux	116
Figure 27: Keyfactor Java Agent Service on Windows	117
Figure 28: SSH Key Discovery Flow	118
Figure 29: SSH User Key Management Flow	119
Figure 30: Find the Server Group ID	124
Figure 31: Configure Logging for the Keyfactor Bash Orchestrator	130
Figure 32: Orchestrator Management for a Keyfactor Bash Orchestrator	131
Figure 33: Orchestrator Management for a Keyfactor Bash Orchestrator	131
Figure 34: Status for the Keyfactor Bash Orchestrator Service	137
Figure 35: Certificate Incorrectly in the Trusted Root Certificate Store	145
Figure 36: Find the Certificate for the Keyfactor Command Web Site	147
Figure 37: Configure Keyfactor Command for Client Certificate Authentication	156
Figure 38: IIS Module for Client Certificate Authentication	157
Figure 39: Configure only Anonymous Authentication at the Server Level in IIS	158
Figure 40: Disable Authentication Methods at the Application Level in IIS	158
Figure 41: Configure SSL Settings in IIS for Client Certificate Authentication	159
Figure 42: Configure IIS Client Certificate Mapping Authentication for the Default Web Site	160
Figure 43: Configure Authorization Credentials for Keyfactor Orchestrators	160
Figure 44: Configure Application Setting in Keyfactor Command to use the Header Certificate	161
Figure 45: Client Certificate Authentication with AD Storage Does Not Require Certificate Authentication Configuration in Keyfactor Command	163
Figure 46: IIS Module for Client Certificate Authentication with AD Storage	164
Figure 47: Configure Client Certificate Authentication at the Server Level in IIS	165
Figure 48: Disable Authentication Methods at the Application Level in IIS	165
Figure 49: Configure SSL Settings in IIS for Client Certificate Authentication	166
Figure 50: Microsoft Certificate Template General for Client Authentication Certificate	167
Figure 51: Microsoft Certificate Template Request Handling for Client Authentication Certificate	168

Figure 52: Microsoft Certificate Template Application Policies for Client Authentication Certificate	169
Figure 53: Microsoft Certificate Template Security for Client Authentication Certificate	170
Figure 54: System Environment Variable to Define a Proxy URL for Use by the Universal Orchestrator on Windows	177

1.0 Introduction

The *Keyfactor Command Documentation Suite* includes:

- *Keyfactor Command Reference Guide*
- *Keyfactor API Reference Guide*
- *Keyfactor Command Server Installation Guide*
- *Keyfactor Orchestrators Installation and Configuration Guide*
- *Keyfactor Command Release Notes & Upgrading*

In addition, Keyfactor offers documentation for products that are not part of the *Keyfactor Command Documentation Suite*, including the *Keyfactor Command Upgrade Overview* and installation guides for third-party CA gateways that interface with Keyfactor, which are available upon request.

2.0 Installing Orchestrators

Keyfactor offers several orchestrators (a.k.a. agents) that may be used to interact with and enhance the functionality of the Keyfactor Command Server.



Tip: Keyfactor recommends that you check the Keyfactor GitHub Site (<https://keyfactor.github.io/integrations-catalog/>) with each release that you install to check if you will need to download the updated orchestrators to work with that version of Keyfactor Command.

This guide covers installation of the following orchestrators:

- Keyfactor Universal Orchestrator
The Keyfactor Universal Orchestrator runs on Windows Servers, Linux servers, and in Linux containers. It can be used to:
 - Run SSL discovery and monitoring tasks.
 - Manage synchronization of certificate authorities in remote forests (installations on Windows only).
 - Collect logs from the orchestrator for central review (full server installations only).
 - Run custom jobs to provide certificate management capabilities on a variety of platforms and devices.
 - Run custom jobs to execute tasks outside the standard list of certificate management functions. This powerful feature can execute just about any job that requires processing on the orchestrator and submitting data back to Keyfactor Command.

As of this release, the following functions, some of which were part of the Keyfactor Windows Orchestrator, are now included among the custom extensions supported for the Keyfactor Universal Orchestrator:

- Interact with Amazon Web Services (AWS) resources for certificate management.
- Interact with Citrix NetScaler devices for certificate management.
- Interact with F5 devices for certificate management.
- Interact with Windows servers (a.k.a. IIS certificate stores), create new bindings for IIS web sites and manage certificates in both the Web Hosting certificate store and the Personal certificate store.
- Remote Java keystore certificate management.
- Remote PEM store certificate management.
- Remote PKCS12 store certificate management.

These custom extensions and more are publicly available at:

<https://keyfactor.github.io/integrations-catalog/content/orchestrator>

The final release of the Keyfactor Windows Orchestrator was version 8.7. This version of the Keyfactor Windows Orchestrator is not compatible with Keyfactor Command version 11.0. Customers should migrate to the Keyfactor Universal Orchestrator with custom extensions as needed.

- **Keyfactor Bash Orchestrator**
The Keyfactor Bash Orchestrator runs on Linux servers and is used to perform discovery and management of SSH public keys, including installation of new keys and automated removal of unauthorized keys.
- **Keyfactor Java Agent**
The Keyfactor Java Agent runs on Windows or Linux servers and is used to perform discovery of Java keystores and PEM certificate stores, to inventory discovered stores, and to push certificates out to stores as needed. In addition, the Keyfactor Java Agent can be extended to create custom certificate store jobs.



Important: The Keyfactor Java Agent will be deprecated in a future version of Keyfactor Command. Customers are encouraged to begin planning a migration to the Keyfactor Universal Orchestrator with the Remote File custom extension publicly available at:

<https://github.com/Keyfactor/remote-file-orchestrator>

For more information, see *Installing Custom-Built Extensions* in the *Keyfactor Orchestrators Installation and Configuration Guide*.

Keyfactor also offers a variety of tools to allow users to develop custom orchestrators and extensions, including:

- **Keyfactor AnyAgent Framework**
The AnyAgent capability of the Keyfactor Universal Orchestrator and Java Agent allows management of certificates regardless of source or location by allowing customers to implement custom agent functionality.
- **Keyfactor Integration SDK**
The Keyfactor Integration SDK (software development kit) includes a variety of tools for building a custom orchestrator, including the Keyfactor Native Agent, which is a reference implementation intended for customers wanting to include Keyfactor Command certificate store management functionality in embedded or other platforms.
- **Keyfactor Orchestrator NuGet Package**
The Keyfactor Orchestrator NuGet package is designed to allow customers to build custom extensions for the Keyfactor Universal Orchestrator.
- **Keyfactor GitHub Site**
Keyfactor offers several publicly available integrations and plugins for the Keyfactor platform in the Keyfactor GitHub. Find all the latest developer tools and resources to integrate the Keyfactor platform with your PKI, Cloud, and DevOps infrastructure.

<https://keyfactor.github.io/>

These tools for developing custom orchestrators and extensions are not documented in this guide. For more information about these and other custom orchestrator solutions, contact your Keyfactor representative.



Important: The Keyfactor Java Agent will be deprecated in a future version of Keyfactor Command. Customers are encouraged to begin planning a migration to the Keyfactor Universal Orchestrator with the Remote File custom extension publicly available at:

<https://github.com/Keyfactor/remote-file-orchestrator>

For more information, see *Installing Custom-Built Extensions* in the *Keyfactor Orchestrators Installation and Configuration Guide*.

2.1 Orchestrator Job Overview

Keyfactor orchestrators can be used to perform a wide variety of jobs. Out of the box, orchestrators can manage certificate stores, manage SSH keys, perform SSL scanning, fetch system logs, and synchronize certificates from CAs in remote forests. Orchestrator jobs fall into these broad types:

- **Certificate Store Jobs**

This type of job includes the built-in jobs for managing certificate stores, based on the type(s) of certificate stores supported by the orchestrator, and custom-built certificate store jobs that can be added with an extension (see [Installing Custom-Built Extensions on page 67](#)) or script (see [Configuring Script-Based Certificate Store Jobs on page 73](#)).

Certificate store jobs (built-in or custom-built), are managed in Keyfactor Command with certificate store types. If you're adding a custom-built certificate store job, you'll need to add a user-defined certificate store type to go with it (see *Certificate Store Types* in the *Keyfactor Command Reference Guide* and *Certificate Store Types* in the *Keyfactor API Reference Guide*).

- **Custom Jobs**

This type of job is intended to implement just about anything else you need an orchestrator to do other than manage certificate stores. The built-in fetch logs job is an example of a custom job.

Custom jobs are managed in Keyfactor Command with custom job types. If you're adding a custom job, you'll need to add a custom job type to go with it (see *Custom Job Types* in the *Keyfactor API Reference Guide*).

Custom jobs are supported only by the Keyfactor Universal Orchestrator.

- **Other Jobs**

This type of job includes the built-in jobs for SSL scanning and certificate synchronization from remote CAs.

Prescripts and Postscripts

All of the job types supported by the Keyfactor Universal Orchestrator—including the built-in jobs—support executing a prescript and/or postscript as part of the job. A prescript might be used to fetch credentials from a privilege access management (PAM) solution to pass in to the username and

password parameters for a certificate store. A postscript might be used to restart the web service (e.g. Apache) after performing a management job to replace the certificate in the certificate store for the web server. Prescripts and postscripts for all types of jobs are configured similarly to the description provided for installing custom-built extensions (see [Installing Custom-Built Extensions on page 67](#)).



Note: The prescript and postscript functionality of the Keyfactor Universal Orchestrator has been replaced by other functionality in Keyfactor Command such as that provided by Keyfactor Command workflows (see *Workflow Definitions in Keyfactor Command Reference Guide*). As a result, prescript and postscript functionality has been deprecated and will be removed from a future release.

Orchestrator Job Flow

An orchestrator job begins when an orchestrator queries Keyfactor Command to ask for jobs and the Keyfactor Command orchestrator API returns a list of the jobs the orchestrator needs to run. The flow continues as shown in the following chart.

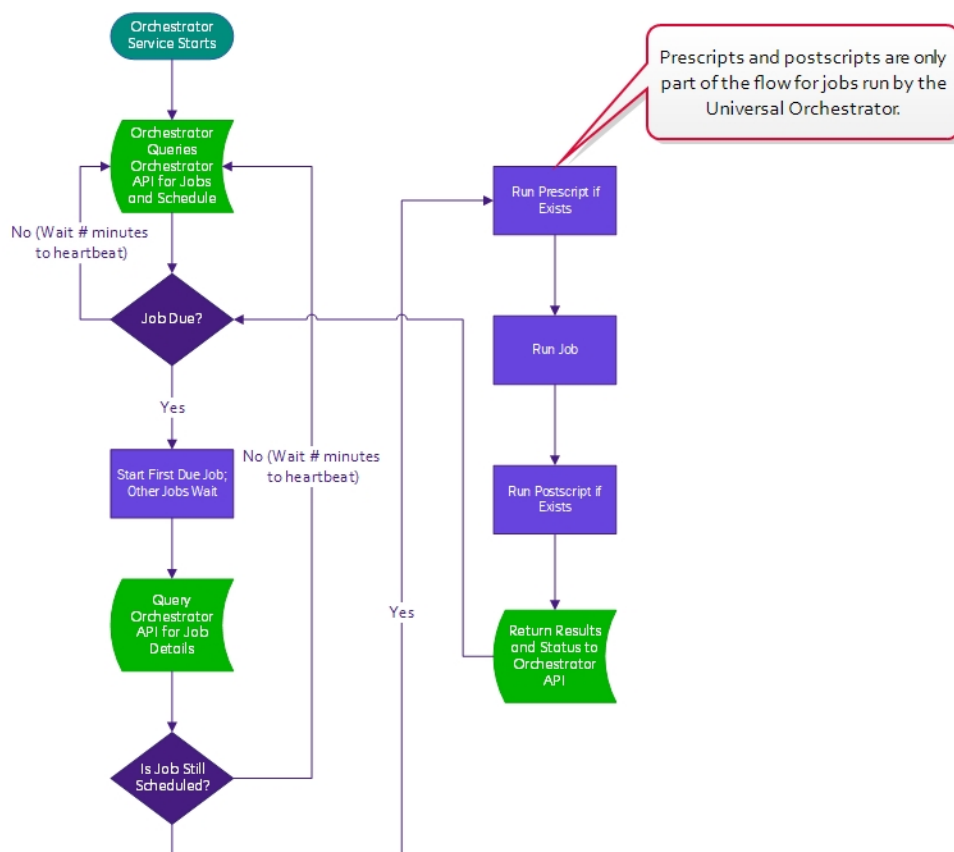


Figure 1: Orchestrator Job Flow

2.2 Universal Orchestrator

The Keyfactor Universal Orchestrator is designed to run jobs at the request of the Keyfactor Command server. Jobs primarily perform certificate management tasks, but other types of operations are also supported. The orchestrator operates as a .NET Core based service on a Windows server, Linux server, or in a Linux container and communicates with a Keyfactor Command server to receive job tasks and report job results. Along with the job results, data can be returned to the Keyfactor Command server and stored in the Keyfactor Command SQL database. Extensions are hosted by the orchestrator and implement the jobs to be executed.

The orchestrator includes these built-in extensions:

- Discover and monitor certificates at TLS 1.3 endpoints either within the local network or across the internet using any of the 5 ciphersuites mentioned in appendix B.4 of RFC 8446. Certificates from the results of SSL discovery and monitoring are imported into Keyfactor Command for viewing, reporting and alerting purposes. Scanning using server name indication (SNI) is supported.
- Retrieve logs generated on the orchestrator via the Keyfactor Command Management Portal. This task returns up to 2 MB of log data from the end of the orchestrator log file to be viewed in the Management Portal. This feature is supported only on full server installations.
- Manage certificates from remote Microsoft Certificate Authorities (CAs) using the Management Portal. Certificates from remote CAs can be imported into Keyfactor Command for viewing, reporting and alerting purposes. This feature is supported only on Windows installations.

If the remote CA is domain-joined to a domain in the remote forest, the Universal Orchestrator may be installed on the CA itself or on a separate server joined to a domain in the same forest (generally a server in the same domain as the CA). Multiple CAs in the same remote forest can be managed with a single Universal Orchestrator server. However, if the remote CA is not domain-joined, the Universal Orchestrator must be installed on the remote CA server.



Note: The Universal Orchestrator does not support certificate enrollment for remote CAs. If you need this capability, you will need to use the *Explicit Credentials* option in the Management Portal CA configuration (see *Adding or Modifying a CA Record* in the *Keyfactor Command Reference Guide*).

In addition, two types of custom extensions are supported:

- Manage and deliver certificates to certificate stores on various platforms and devices using custom certificate store types and orchestrator jobs in the Keyfactor Command Management Portal. Custom extensions may be developed by Keyfactor or end users. Keyfactor offers several publicly available custom extensions for the Universal Orchestrator in the Keyfactor GitHub:

<https://keyfactor.github.io/integrations-catalog/content/orchestrator>

With the custom extensions available from the Keyfactor GitHub, you can manage Windows certificate stores (IIS), JKS stores, PEM stores, F5 devices, Citrix NetScaler devices, AWS resources and more (see [Installing Custom-Built Extensions on page 67](#)).

For more information about custom extensions, contact your Keyfactor representative.

- Run custom jobs on the orchestrator that fall outside the standard certificate management tasks. With custom jobs, you can perform operations locally on the orchestrator—or initiate them remotely across the network—and then report results back to Keyfactor Command along with data collected from the jobs, if any.

2.2.1 Preparing for the Universal Orchestrator

This section describes the steps that need to be taken prior to a Keyfactor Universal Orchestrator installation to install the prerequisites, create the required supporting components, and gather the necessary information to complete the Universal Orchestrator installation and configuration process.

2.2.1.1 System Requirements

The Keyfactor Universal Orchestrator is supported on the following operating systems:

- Windows Server 2019 or Windows Server 2022
- Oracle Linux 7 or higher
- Red Hat Enterprise 7 or higher
- Ubuntu 16 or higher



Note: Older versions of the Universal Orchestrator will work with newer versions of Keyfactor Command, but not the other way around (see the *Compatibility Matrix* in the *Keyfactor Command Documentation Suite*). The current version of the Universal Orchestrator requires Keyfactor Command version 10.0 or greater.



Important: Microsoft support for .NET Runtime version 3.1 was deprecated at the end of 2022. Instructions for upgrading to version 6.0 are included in the [tip](#), below.

Windows Server Application Requirements

The Universal Orchestrator has the following requirements on Windows.

- The orchestrator requires the Microsoft .NET Runtime version 6.0 (x64). Version 6.0 is available for download from Microsoft:

<https://dotnet.microsoft.com/download/dotnet/6.0/runtime>

You need only the .NET Runtime (x64), not the ASP.NET Core Runtime or ASP.NET Core Hosting Bundle. At the above link, this would be the **Download x64** option under the *Run console apps* heading.

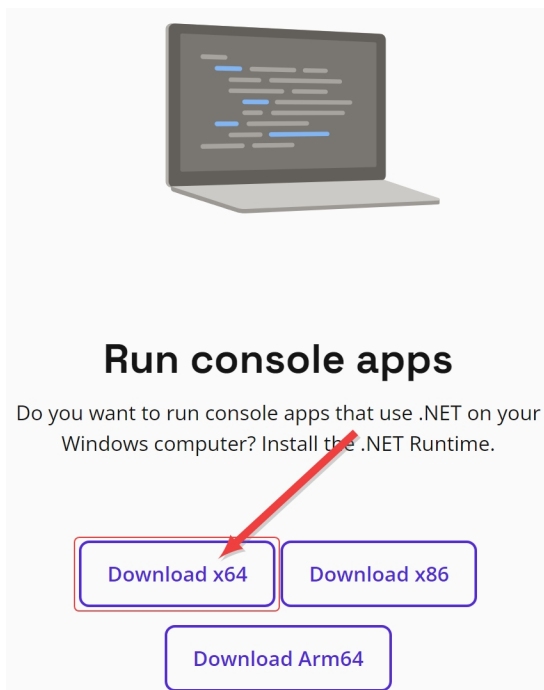


Figure 2: Select the Download x64 Option Under Run Console Apps

You can use the following PowerShell command to check the .NET core version(s) installed on a server (if any):

```
dotnet --list-runtimes
```

Output from this command will look something like this if you have the correct 6.0 x64 version of the .NET Runtime installed (notice the path is in C:\Program Files, not C:\Program Files (x86), indicating this is the x64 version):

```
Microsoft.NETCore.App 6.0.11 [C:\Program Files\dotnet\shared\Microsoft.NETCore.App]
```

- If you intend to use the orchestrator to manage certificates in remote Windows machine certificate stores (servers other than the server on which the orchestrator is installed) using the IIS Certificate Store Manager extension or Java Keystores, PKCS12 files, PEM files, DER files, or IBM Key Database files on Windows servers with the Remote File Certificate Store Management extension (see [Installing Custom-Built Extensions on page 67](#)), make sure that TCP port 5985 or 5986 is open between the orchestrator and the remote servers (see [Configure Windows Targets for Remote Management on page 62](#)).
- If you intend to use the orchestrator to manage certificates from remote Microsoft CAs (CAs outside the forest in which Keyfactor Command is installed or forests in a two-way trust with this

forest), the orchestrator requires the Microsoft Visual C++ 2019 (or later) redistributable for x64. This is available for download from Microsoft:

https://aka.ms/vs/16/release/vc_redist.x64.exe

The Microsoft Visual C++ Redistributable appears as an application in the Windows Apps & features.

Linux Server Application Requirements

The following applications are required in order to install the Universal Orchestrator on Linux servers.

Microsoft .NET 6.0 Runtime

The orchestrator requires the Microsoft .NET Runtime version 6.0 (x64). Information about this is available from Microsoft:

<https://docs.microsoft.com/en-us/dotnet/core/install/linux>

You need only the .NET Runtime (x64), not the ASP.NET Core Runtime, but it won't hurt anything to install both the .NET and ASP.NET Core runtimes as suggested in the Microsoft documentation for installing .NET on Linux.

For the most part, it can be installed via the OS package manager. The method to complete this varies depending on the Linux operating system. For example, for Ubuntu 20.04, the following commands will install the correct version of .NET:

```
wget https://packages.microsoft.com/config/ubuntu/20.04/packages-microsoft-prod.deb
sudo dpkg -i packages-microsoft-prod.deb
sudo apt-get update
sudo apt-get install apt-transport-https
sudo apt-get install dotnet-runtime-6.0
```

You can use the following command to check the .NET version installed on a server (if any):

```
dotnet --list-runtimes
```

Output from this command will look something like this if you have the correct 6.0 version of the .NET Runtime installed:

```
Microsoft.NETCore.App 6.0.6 [/usr/share/dotnet/shared/Microsoft.NETCore.App]
```

jq

The orchestrator can only be installed on a Linux server that has jq installed. You can use the following command to check the jq version of a server:

```
jq --version
```

systemd

The orchestrator requires a Linux server that uses the systemd service manager. You can use the following command to test whether a system is running systemd:

```
ps -p 1
```

bash

The orchestrator can only be installed on a Linux server that is running bash version 4.3 or higher. You can use the following command to check the bash version of a server:

```
bash --version
```

curl

The orchestrator can only be installed on a Linux server that has curl installed. You can use the following command to check the curl version of a server:

```
curl --version
```

Linux Container Application Requirements

The following applications are required in order to install the Universal Orchestrator in Linux containers.

Containerization Solution

The orchestrator needs a containerization solution in which to run. Keyfactor has tested with Docker and Kubernetes.



Tip: If you have an existing installation of the Universal Orchestrator using the older Microsoft .NET Runtime version 3.1, you do not need to reinstall the orchestrator to upgrade the .NET version.

To update your existing Universal Orchestrator to the latest .NET version:

1. On the Universal Orchestrator machine, browse to locate the Orchestrator.runtimeconfig.json file in your installation directory. By default, this is:

Windows: C:\Program Files\Keyfactor\Keyfactor

Orchestrator\Orchestrator.runtimeconfig.json

Linux: /opt/keyfactor/orchestrator/Orchestrator.runtimeconfig.json



2. Using a text editor, open the `Orchestrator.runtimeconfig.json` file for editing and add the following property to the `runtimeOptions` section:

```
"rollForward": "LatestMajor"
```

Being sure to add a comma at the end of the previous row, resulting in a final file that looks something like:

```
{
  "runtimeOptions": {
    "tfm": "netcoreapp3.1",
    "framework": {
      "name": "Microsoft.NETCore.App",
      "version": "3.1.0"
    },
    "rollForward": "LatestMajor"
  }
}
```

3. Save the `Orchestrator.runtimeconfig.json` file.
4. Uninstall the Microsoft .NET Runtime version 3.1 (x64) and install the 6.0 version.
5. Restart the Universal Orchestrator service (see [Start the Universal Orchestrator Service on page 80](#)).

2.2.1.2 Create Service Accounts for the Universal Orchestrator

The Keyfactor Universal Orchestrator makes use of up to two service accounts to allow it to communicate with the Keyfactor Command server. These two service accounts work together to transfer information from the Universal Orchestrator to the Keyfactor Command server. The two service accounts can be thought of as players on two sides of a fence, with the service account that the Universal Orchestrator runs as lobbing information over the fence to the service account that communicates with the Keyfactor Command server side to catch and hand to the Keyfactor Command server. Below, these are referred to as the Universal Orchestrator service account and the Keyfactor Command connect service account.

The service accounts need to be created prior to installation of the Universal Orchestrator software (except as noted below for installations on Linux), and the person installing the Universal Orchestrator software needs to know the domain (if applicable), username and password of each service account.



Important: Keyfactor highly recommends that you use strong passwords for any accounts or certificates related to Keyfactor Command and associated products, especially when these have elevated or administrative access. A strong password has at least 12 characters (more



is better) and multiple character classes (lowercase letters, uppercase letters, numeral, and symbols). Ideally, each password would be randomly generated. Avoid password re-use.

Universal Orchestrator Service Account

Your choice of service account may vary depending on the operating system on which you are installing the orchestrator:

Universal Orchestrator on a Windows Server

When the Universal Orchestrator is installed on Windows, you may use either the built-in Network Service account or a custom service account as the Universal Orchestrator service account. Keyfactor recommends using the default of Network Service unless you have a need to use a custom service account. If you choose to use a custom service account, it may be a standard Active Directory service account, an Active Directory group managed service account (gMSA), or a local machine account. Of the custom service account choices, an Active Directory account is more typically used unless the machine is not domain-joined. If you use an Active Directory service account, it needs to be a service account in the forest in which the Universal Orchestrator is installed. This is not necessarily the same forest as the forest in which the Keyfactor Command server is installed. The Universal Orchestrator on Windows has several possible roles, and the choice of service account may vary depending on these roles:

SSL Management

If your Universal Orchestrator SSL discovery and monitoring, you may choose to run the orchestrator as the built-in Network Service account or as a custom service account.

CA Management

If your Universal Orchestrator will be providing certificate synchronization from a remote CA, the Universal Orchestrator service account needs to be able to read the CA(s) in the forest in which the Universal Orchestrator is installed to retrieve certificates and templates from them. When the Universal Orchestrator is used in this configuration, this is typically a forest other than the forest in which the Keyfactor Command server is installed. For domain-joined CAs, you would typically use an Active Directory service account in the remote forest (the forest where the Universal Orchestrator is installed). For a non-domain-joined CA, you may use a local account created on the CA as the Universal Orchestrator service account instead of a domain account.

Custom Extensions

Keyfactor offers several publicly available custom extensions for the Universal Orchestrator in the Keyfactor GitHub. Many of these will operate correctly with a Universal Orchestrator service account running as Network Service, but some may require a custom account. Check the specific documentation for each custom extension for more information:

<https://keyfactor.github.io/integrations-catalog/content/orchestrator>

The Keyfactor Orchestrator Service on the server on which the Universal Orchestrator is installed runs as the Universal Orchestrator service account. This service account requires local “Log on as a service” permissions; this permission is granted automatically during installation.

Universal Orchestrator on a Linux Server

For the purposes of this documentation, it is assumed that Linux machines will be non-domain joined and will use a local account to run the Universal Orchestrator.

For Linux systems, Keyfactor recommends running the service as an account other than root. The default Universal Orchestrator service account of *keyfactor-orchestrator* will be created automatically during the install if the *force* option is used. If you prefer not to use the *force* option, you may create a local service account before running the installation script.

Universal Orchestrator in a Linux Container

This service account is not relevant for the orchestrator run in a container, since the container build is self-contained.

Keyfactor Command Connect Service Account

For the Keyfactor Command connect service account, the service account you use depends on the identity provider you’re using:

- If you’re using Active Directory as an identity provider, a standard Active Directory service account in the primary Keyfactor Command server forest is used. Group managed service accounts are not supported in this role.



Tip: If the Universal Orchestrator is installed on Windows in the same forest as the Keyfactor Command server, the same Active Directory service account may be used as both the Universal Orchestrator service account and the Keyfactor Command connect service account, if desired.

- If you’re using an identity provider other than Active Directory, a client (not user) in the identity provider is used. The client should be configured with a secret and have *Client authentication* and *Service account roles* enabled (see *Using Keyfactor Identity Provider: Service Accounts* in the *Keyfactor Command Server Installation Guide*). The user installing the orchestrator will need the client ID and secret.

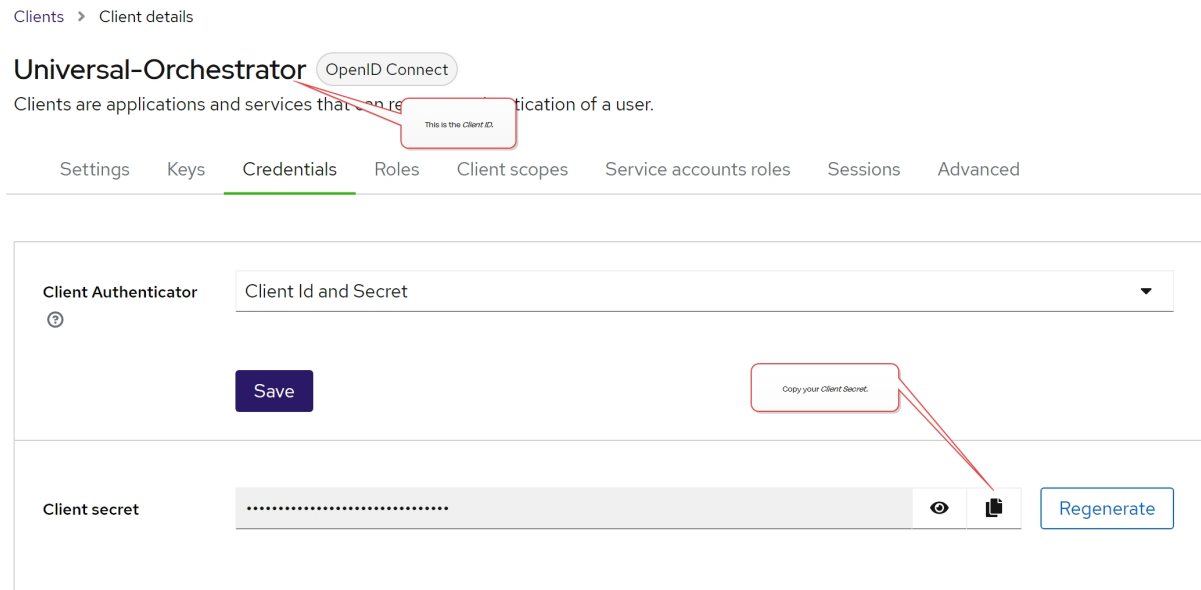


Figure 3: Client Secret for Orchestrator Client in Keyfactor Identity Provider

This service account appears in the Management Portal Orchestrator Management grid as the Identity for the Universal Orchestrator.

Permissions

The user installing the orchestrator must have the `SeBackupPrivilege` and `SeRestorePrivilege` rights on the Keyfactor Universal Orchestrator server. Normally, administrators are granted these permissions by default, but you should confirm the permissions prior to starting the install. These permissions can be set through Group Policy or Local Security Policy, and can be found under *Local Policies\User Rights Assignment* as *Back up files and directories* and *Restore files and directories*.

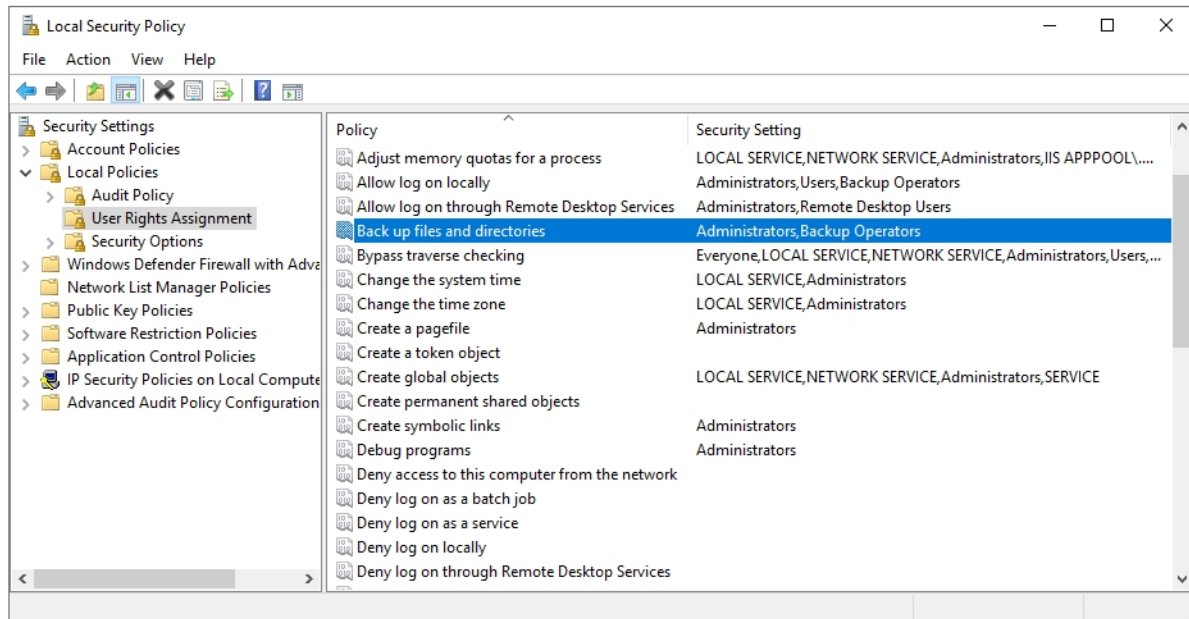


Figure 4: Local Security Policy

For more information on this from Microsoft, see:

<https://docs.microsoft.com/en-us/windows/win32/api/userenv/nf-userenv-load-userprofilea#remarks>

2.2.1.3 Configure Certificate Root Trust for the Universal Orchestrator

Keyfactor recommends using HTTPS to secure the channel between each Keyfactor Universal Orchestrator and the Keyfactor Command server(s). This requires an SSL certificate configured in IIS on the Keyfactor Command server(s). This certificate can either be a publicly-rooted certificate (e.g. from DigiCert, Entrust, etc.), or one issued from a private certificate authority (CA). If your Keyfactor Command server is using a publicly rooted certificate, the orchestrator server may already trust the certificate root for this certificate. However, if you have opted to use an internally-generated certificate, your orchestrator server may not trust this certificate. In order to use HTTPS for communications between the orchestrator and the Keyfactor Command server with a certificate generated from a private CA, you may need to import the certificate chain for the certificate into either the local machine certificate store on the orchestrator server on Windows or the root certificate store on Linux.



Note: The CRL(s) for the Keyfactor Command certificate need to be available to the orchestrator (see [Troubleshooting on page 130](#)).

Installations on Windows Servers

If the public key infrastructure (PKI) that issued the certificate has only a root CA, the root certificate from this CA must be installed in the Trusted Root Certification Authorities store under Local Computer on the orchestrator server. If the PKI that issued the certificate has both a root and issuing CA, the root certificate must be installed in the Trusted Root Certification Authorities store under Local Computer on the orchestrator server and the issuing CA certificate must be installed in the Intermediate Certification Authorities store under Local Computer on the orchestrator server.

Installations on Linux Servers and in Linux Containers

The location of the OpenSSL trusted root store varies depending on your Linux implementation. The root certificate must be installed in the appropriate location for the operating system before beginning the installation.

2.2.1.4 Grant the Orchestrator Service Account Permissions on the CAs

This step only needs to be completed if you plan to use the Keyfactor Universal Orchestrator for remote Microsoft CA synchronization.

In order for the Universal Orchestrator to be able to synchronize certificates from the remote Microsoft CA(s) to the Keyfactor Command database, the Universal Orchestrator service account—the identity under which the orchestrator in the remote forest runs—must have permissions to read the CA database(s) in the remote forest.

In the management console for each CA that the orchestrator will interact with, open the properties for the CA and grant the service account that the orchestrator runs as (see [Create Service Accounts for the Universal Orchestrator on page 11](#)) read permissions before continuing.

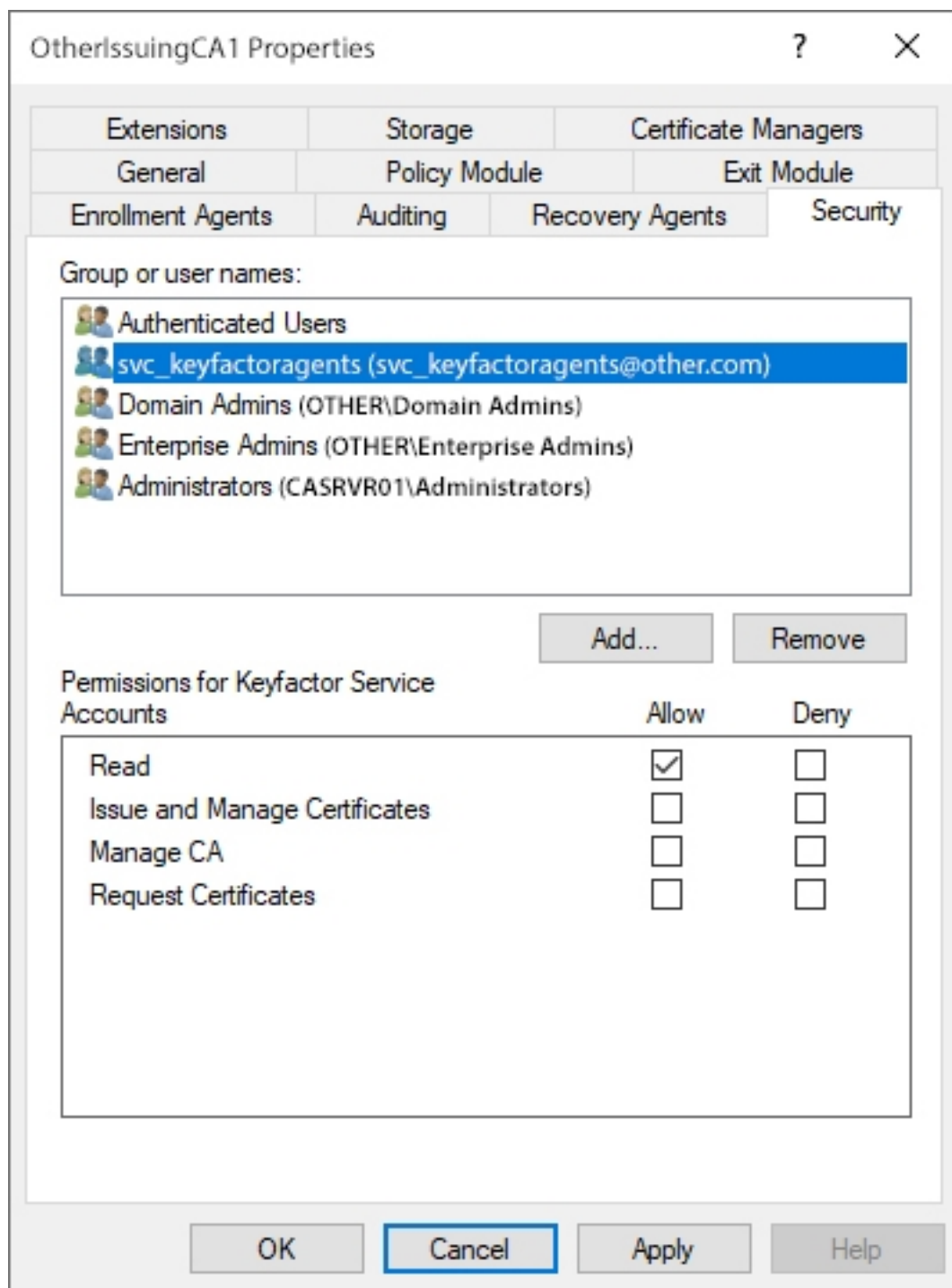


Figure 5: CA Permissions

2.2.1.5 Acquire a Certificate for Client Certificate Authentication (Optional)

The Keyfactor Universal Orchestrator supports client certificate authentication to allow you to authenticate via client certificates from individual orchestrator machines to either a centralized proxy, such as a network load balancer, which would in turn authenticate to the Keyfactor Command server using either a username and password or client ID and secret that was stored securely on the proxy or another client certificate, or directly using IIS on the Keyfactor Command to manage the certificate authentication and Active Directory to manage the mapping of client certificates to service accounts. The proxy approach allows orchestrator credentials to be assigned and managed outside the Active Directory forest in which Keyfactor Command is installed. The web proxy's job is to confirm the validity of the certificate and to provide Active Directory or an identity provider other than Active Directory credentials known to Keyfactor Command (if configured in this manner). Typically the proxy would be configured to accept all certificates issued from a given PKI implementation—even a PKI that is unknown to the Keyfactor Command Active Directory forest—thus delegating orchestrator access control to that PKI. For more information, see:

- [Appendix - Set up the Universal Orchestrator to Use Client Certificate Authentication via a Reverse Proxy: Citrix ADC on page 150](#)
- [Appendix - Set up the Universal Orchestrator to Use Client Certificate Authentication with Certificates Stored in Active Directory on page 162](#)



Important: The Universal Orchestrator supports automated client certificate renewal using an extension point interface on the orchestrator that can be implemented by the end-user. The custom extension will generate a CSR with private key and submit the CSR to Keyfactor Command for enrollment. Keyfactor Command will return the certificate to the orchestrator, which will pair it with its private key and use that certificate for authentication. See [Register a Client Certificate Renewal Extension on page 88](#) for more information.



Note: Client certificate authentication is not supported when using the Universal Orchestrator installed in a Linux container (see [Install the Universal Orchestrator in a Linux Container on page 49](#)).

There are several situations in which using certificate authentication for the Universal Orchestrator may be helpful, including:

- **Scale**—To allow orchestrator numbers to scale (e.g. the IoT case) where it isn't practical to have a unique Active Directory account for each orchestrator.
- **Untrusted Environments**—To support environments (e.g. a “hostile” network) where policy doesn't allow the password for an Active Directory account to be stored on the orchestrator.

The certificate that the Universal Orchestrator uses for authentication needs:

- An extended key usage (EKU) of Client Authentication

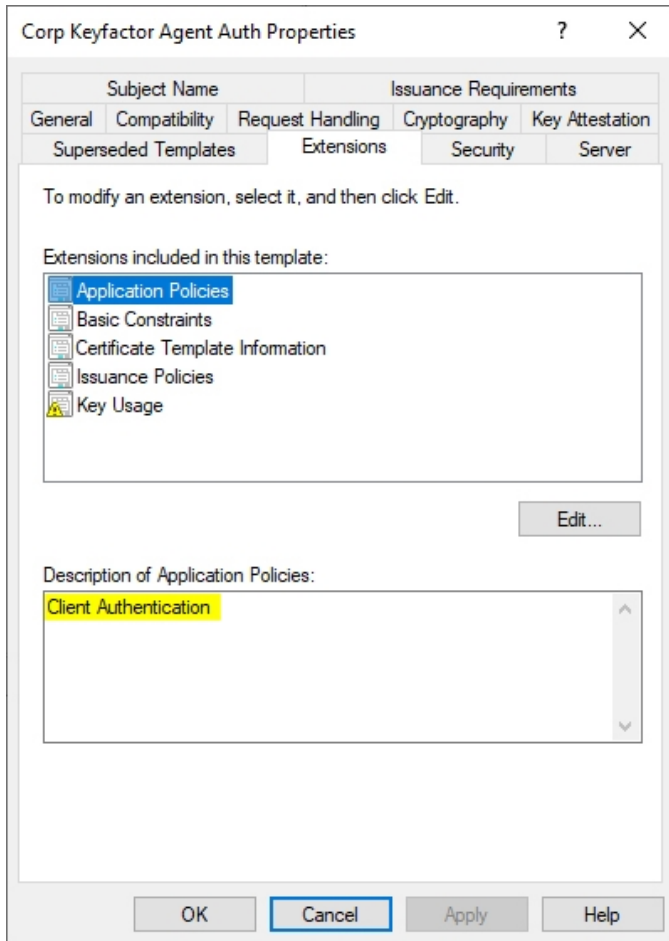


Figure 6: Microsoft Certificate Template Application Policies for Client Authentication Certificate

- A key usage that includes Digital Signature

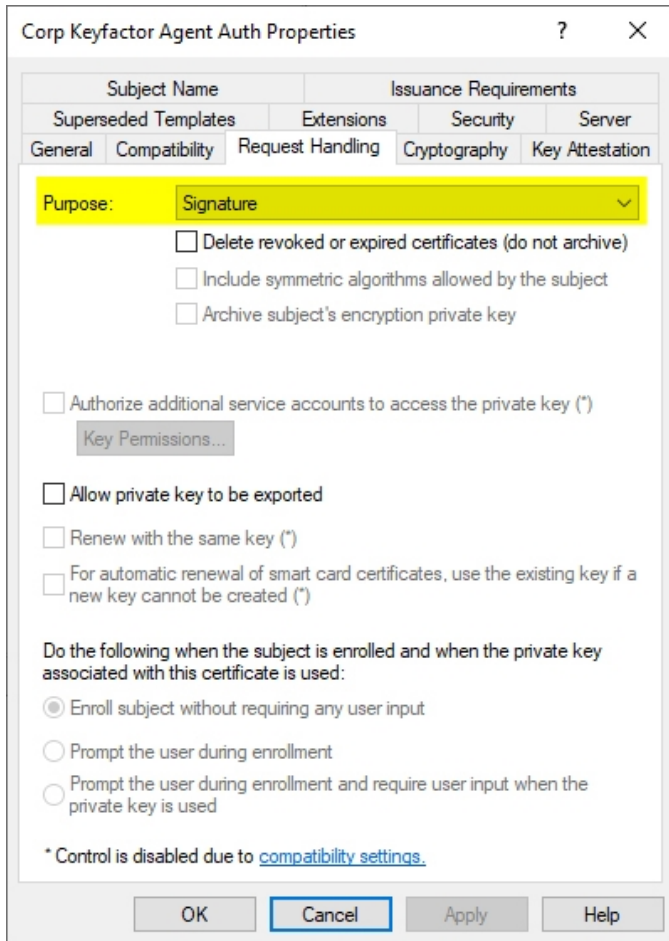


Figure 7: Microsoft Certificate Template Request Handling for Client Authentication Certificate

On Windows servers, the certificate may be referenced either as a PKCS12 file stored in the file system or may be placed either in the local machine's personal store (*My*), or, if you opt to run the Universal Orchestrator service as a domain service account rather than the default of *Network Service*, in the personal store of the Universal Orchestrator service account user. If you opt to place the certificate in the local machine store, you need to grant the service account under which the Universal Orchestrator service will run (including *Network Service* if you will use this option) read permissions to the private key of the certificate. If you opt to place the certificate in the personal store of the Universal Orchestrator service account user, it also needs to be placed in the personal store of the user running the installation for the duration of the installation to allow it to be read during initial configuration. It may be removed from the installing user's store after installation is complete.

On Linux servers, the certificate is referenced as a PKCS12 file stored in the file system.



Important: Keyfactor highly recommends that you use strong passwords for any accounts or certificates related to Keyfactor Command and associated products, especially when these have elevated or administrative access. A strong password has at least 12 characters (more is better) and multiple character classes (lowercase letters, uppercase letters, numeral, and symbols). Ideally, each password would be randomly generated. Avoid password re-use.

To acquire a certificate for use by the Universal Orchestrator using a Microsoft CA, first create a template using the appropriate configurations as described above and make it available for enrollment on the CA from which you will request the certificate. The simplest way to acquire a certificate as a PKCS12 file for either Linux or Windows use is with PFX enrollment in Keyfactor Command. There are multiple ways to acquire a certificate and place it in the machine store on the Windows server where the Universal Orchestrator will be installed, including:

- Enroll through the Microsoft certificates MMC.
- Generate a CSR through the Microsoft certificates MMC and take the CSR to Keyfactor Command to issue a certificate using the CSR enrollment option in the Keyfactor Command Management Portal. You will need to return to the Microsoft certificates MMC to marry the certificate with the private key.
- Enroll for a certificate through Keyfactor Command using the PFX enrollment method and deploy it to the certificate store using an already installed Universal Orchestrator managing the store as an IIS store.
- Enroll using the command-line `certreq` command with a `request.inf` file on the Universal Orchestrator server.

Several of the above methods can also be used if you opt to enroll into the Universal Orchestrator service account user's personal store, though this option requires a few extra steps.

To enroll for a certificate using the certificates MMC into the local machine store:

1. On the Universal Orchestrator machine, do one of following:
 - Using the GUI:
 - a. Open an empty instance of the Microsoft Management Console (MMC).
 - b. Choose **File->Add/Remove Snap-in....**
 - c. In the *Available snap-ins* column, highlight **Certificates** and click **Add**.
 - d. In the Certificates snap-in popup, choose the radio button for Computer account, click **Next**, accept the default of Local computer, and click **Finish**.
 - e. Click **OK** to close the Add or Remove Snap-ins dialog.
 - Using the command line:
 - a. Open a command prompt using the "Run as administrator" option.
 - b. Within the command prompt type the following to open the certificates MMC:
`certlm.msc`

2. Drill down to the Personal folder under **Certificates** for the Local Computer, right-click, and choose **All Tasks->Request New Certificate...**
3. Follow the certificate enrollment wizard, selecting the template you created or identified for use for this purpose, and providing any required information.
4. When the enrollment completes, locate the certificate in the Personal store (you may need to refresh), highlight it, and choose **All Tasks->Manage Private Keys...**
5. In the Permissions for private keys dialog, click **Add**, add the Universal Orchestrator service account—the account under which the Universal Orchestrator is running (created as per [Create Service Accounts for the Universal Orchestrator on page 11](#))—and grant that service account **Read** but not **Full control** permissions. Click **OK** to save.

2.2.1.6 Upgrading the Universal Orchestrator

There are two possible paths for upgrading from an earlier implementation of the Keyfactor Universal Orchestrator to a newer implementation:

- If your newer orchestrator will be installed in the same path as the older orchestrator, you may install the newer orchestrator over the older orchestrator using the *-Force* (Windows) or *--force* (Linux) option to overwrite the existing implementation.
- You may uninstall the older implementation using the provided uninstall script (uninstall.ps1 on Windows or uninstall.sh on Linux) and install the newer version using the standard installation steps (see [Install the Universal Orchestrator on Windows on page 25](#) or [Install the Universal Orchestrator on a Linux Server on page 39](#)).

If you have an existing instance of the Keyfactor Windows Orchestrator and wish to migrate to the Keyfactor Universal Orchestrator, you may either install the two orchestrators side-by-side and then uninstall the Keyfactor Windows Orchestrator or uninstall the Keyfactor Windows Orchestrator and then install the Keyfactor Universal Orchestrator.



Important: Before following any of these upgrade paths, be sure to save off a copy of any custom extensions for the Keyfactor Universal Orchestrator (found in C:\Program Files\Keyfactor\Keyfactor Orchestrator\extensions by default) or plugins for the Keyfactor Windows Orchestrator (found in C:\Program Files\Keyfactor\Keyfactor Windows Orchestrator\plugins by default).

Keyfactor's suggested upgrade process is:

1. Review your installed orchestrator and gather this information:
 - Is this a Windows Orchestrator or a Universal Orchestrator?

There are a number of ways to tell the difference. For example, the default install directory for the Universal Orchestrator is *Keyfactor Orchestrator* which the default install directory for the Windows Orchestrator is *Keyfactor Windows Orchestrator*. The Universal Orchestrator has several subdirectories, including an *extensions* directory. The Keyfactor Windows Orchestrator has only one subdirectory—*plugins*—by default.

- What user account is being used to run the orchestrator service?

To check this, you can open the Windows Services tool (services.msc), look for the *Keyfactor Orchestrator Service*, and check the account that's configured as the *Log On As*.

- If this installation is on Windows, is the user account being used to run the orchestrator service a group managed service account (gMSA)?
- What type of authentication is being used to make the connection to Keyfactor Command?

To check this, you can open the appsettings.json configuration file, which is found in the following location by default:

Windows: C:\Program Files\Keyfactor\Keyfactor Orchestrator\configuration\appsettings.json

Linux: /opt/keyfactor/orchestrator/configuration/appsettings.json

If you're using client certificate authentication, the CertPath and AuthCertThumbprint fields will be populated. If you're using token authentication, the BearerTokenUrl and ClientId fields will be populated. If none of the aforementioned fields are populated, you're using Basic authentication.

- What user account, client ID, or other authentication information is being used to make the connection to Keyfactor Command?

Check in Keyfactor Command for the Identity that the orchestrator indicates on the Orchestrator Management page.

- What secret information (user password, client secret, etc.) is used to make the connection to Keyfactor Command?

This information cannot be retrieved from your existing installation. It is stored in an encrypted state.

- What plugins (Windows Orchestrator) or extensions (Universal Orchestrator) are you using?

The plugins for the Keyfactor Windows Orchestrator are found in C:\Program Files\Keyfactor\Keyfactor Windows Orchestrator\plugins by default. The extensions for the Keyfactor Universal Orchestrator are found in C:\Program Files\Keyfactor\Keyfactor Orchestrator\extensions by default.

2. If you're using plugins or extensions:

- a. Before beginning the upgrade, save off a copy of any existing plugins or extensions in use.
- b. If you're using plugins or extensions from the Keyfactor Git Hub, check for the latest version:

<https://keyfactor.github.io/integrations-catalog/content/orchestrator>

Plugins for the Keyfactor Windows Orchestrator are not compatible with the Keyfactor Universal Orchestrator, so check for extensions that replace your plugins.

- c. Review the documentation of each extension you will be using to determine if there are any changes needed to the certificate store type definition in Keyfactor Command. Only the following fields in a certificate store type that's in use may be edited:

- Name
- Short Name
- Supported Job Types
- Entry Parameters

If changes to any other fields of the certificate store type are needed, you will need to create a new certificate store type.

- d. Create new certificate store types or edit existing certificate store types as needed (see the previous step). If you're creating a new certificate store type to replace an existing one, it cannot have the same Short Name as an existing certificate store type. This means that you will most likely be using a non-standard Short Name for your extension (e.g. CitrixAdc2) and will need to modify the extension configuration to point it to the correct certificate store type. To do this, in the directory for your extension (the version that you will later copy into the upgraded orchestrator's directory), locate the manifest.json file and open it for editing. Change the existing capability to map to your new certificate store type(s). For example, CitrixAdc becomes CitrixAdc2:

```
{
  "extensions": {
    "Keyfactor.Orchestrators.Extensions.IOrchestratorJobExtension": {
      "CertStores.CitrixAdc2.Inventory": {
        "assemblypath": "Keyfactor.Extensions.Orchestrator.CitricAdc.dll",
        "TypeFullName": "Keyfactor.Extensions.Orchestrator.CitricAdc.Inventory"
      },
      "CertStores.CitrixAdc2.Management": {
        "assemblypath": "Keyfactor.Extensions.Orchestrator.CitricAdc.dll",
        "TypeFullName": "Keyfactor.Extensions.Orchestrator.CitricAdc.Management"
      }
    }
  }
}
```

3. Install the new Universal Orchestrator, either uninstalling the previous version or installing over the previous version using the -force option, following the standard installation steps (see [Install the Universal Orchestrator on Windows on the next page](#), [Install the Universal Orchestrator on a Linux Server on page 39](#), or [Install the Universal Orchestrator in a Linux Container on page 49](#)) and referencing the information you gathered in step 1. If you're installing in a container, you'll need to stage your selected extensions as part of the install.
4. If you're using extensions and didn't install in a container, copy your extensions into the extensions directory of the new orchestrator (see [Installing Custom-Built Extensions on page 67](#)).

Restart the orchestrator service to pick up the extension changes.

5. In the Keyfactor Command Management Portal, review the orchestrator, confirm that the capabilities are as expected, and approve the orchestrator.
6. If you're using extensions and added a new certificate store type, you will need to recreate your certificate store. The certificate store type associated with a certificate store cannot be edited. Review your current certificate stores and recreate them with the new certificate store type, and then delete the versions with the old certificate store type.
7. Confirm that your certificate stores and/or SSL scanning are functioning as expected.

2.2.2 Install the Universal Orchestrator on Windows

To install the Keyfactor Universal Orchestrator on Windows, copy the zip file containing installation files to a temporary working directory on the Windows server and unzip it.



Note: In some instances, downloading a compressed file on Windows can cause the file to be marked as *blocked*. If you unzip a blocked file and proceed with the installation, the installation may fail with an error about missing files or dependencies (e.g. “Could not load file or assembly [filename] or one of its dependencies...”). Before beginning the installation, check the zip file *before* unzipping it to confirm that it is not blocked and unblock it if it is blocked.

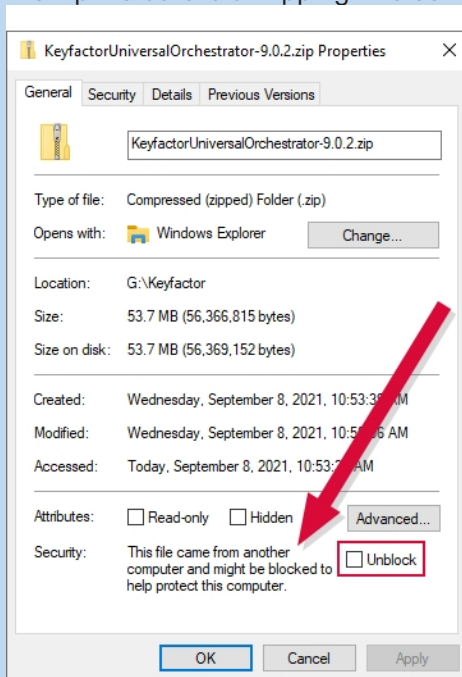


Figure 8: Installation Files Blocked after Download

To begin the installation:

1. On the Windows machine on which you wish to install the orchestrator, open a PowerShell window using the “Run as Administrator” option and change to change to *InstallationScripts* subdirectory under the temporary directory where you placed the installation files.
2. In the PowerShell window, select from the following commands to run based on the identity provider you’re using for Keyfactor Command, the desired orchestrator service accounts, and the desired install experience to prepare for the install.

In these examples, *credKeyfactor* is used for the Keyfactor Command connect service account that the orchestrator uses to connect to Keyfactor Command and *credService* is used for the Universal Orchestrator service account that the service runs as. Usernames should be given in DOMAIN\username format for Active Directory domain accounts or hostname\username format for local user accounts.

- If you’re using Active Directory as an identity provider, will be running the service as an Active Directory domain or local account rather than Network Service, and do not want to provide the usernames and passwords in the command, run the following commands to populate a variable with the user credentials for the Keyfactor Command connect service account (see [Create Service Accounts for the Universal Orchestrator on page 11](#)) and populate a variable with the user credentials for the Universal Orchestrator service account:

```
$credKeyfactor = Get-Credential  
$credService = Get-Credential
```

Enter the appropriate username and password when prompted.

- Active Directory as the identity provider, running the service as Network Service, and not providing the username and password for the Keyfactor Command connect service account in the command:

```
$credKeyfactor = Get-Credential
```

- Active Directory as the identity provider, running the service as Network Service, and providing the username and password for the Keyfactor Command connect service account in the command:

```
$keyfactorUser = "DOMAIN\mykeyfactorconnectusername"  
$keyfactorPassword = "MySecurePassword"  
$secKeyfactorPassword = ConvertTo-SecureString $keyfactorPassword -AsPlainText  
-Force  
$credKeyfactor = New-Object System.Management.Automation.PSCredential  
($keyfactorUser, $secKeyfactorPassword)
```

- Active Directory as the identity provider, running the service as a domain or local account, and providing the username and password for the Keyfactor Command connect service account and the service account the orchestrator runs as in the command:

```

$serviceUser = "DOMAIN\myserviceusername"
$keyfactorUser = "DOMAIN\mykeyfactorconnectusername"
$keyfactorPassword = "MyFirstSecurePassword"
$servicePassword = "MySecondSecurePassword"
$secKeyfactorPassword = ConvertTo-SecureString $keyfactorPassword -AsPlainText
-Force
$secServicePassword = ConvertTo-SecureString $servicePassword -AsPlainText -
Force
$credKeyfactor = New-Object System.Management.Automation.PSCredential
($keyfactorUser, $secKeyfactorPassword)
$credService = New-Object System.Management.Automation.PSCredential
($serviceUser, $secServicePassword)

```

- Active Directory as the identity provider, running the service as a group managed service account (gMSA), and providing the username and password for the Keyfactor Command connect service account in the command:

```

$serviceUser = "DOMAIN\myGMSAserviceusername$"
$keyfactorUser = "DOMAIN\mykeyfactorconnectusername"
$keyfactorPassword = "MySecurePassword"
$secKeyfactorPassword = ConvertTo-SecureString $keyfactorPassword -AsPlainText
-Force
$credKeyfactor = New-Object System.Management.Automation.PSCredential
($keyfactorUser, $secKeyfactorPassword)
$credService = New-Object System.Management.Automation.PSCredential
($serviceUser, (New-Object System.Security.SecureString))

```



Note: Group managed service accounts are not supported for use in making the connection to Keyfactor Command.

- An identity provider other than Active Directory and running the service as Network Service:

```
none
```

- An identity provider other than Active Directory, running the service as a domain or local account, and not providing the username and password for the service account the orchestrator runs as in the command:

```
$credService = Get-Credential
```

- An identity provider other than Active Directory, running the service as a domain or local account, and providing the username and password for the service account the orchestrator runs as in the command:

```
$serviceUser = "DOMAIN\myserviceusername"
$servicePassword = "MySecondSecurePassword"
$secServicePassword = ConvertTo-SecureString $servicePassword -AsPlainText -
Force
$credService = New-Object System.Management.Automation.PSCredential
($serviceUser, $secServicePassword)
```



Tip: In some cases, you may be using the same service account for both the Universal Orchestrator service account role and the Keyfactor Command connect service account role. If this is the case, you may use a single variable for both passwords in the next step.

3. In the PowerShell window, run the `install.ps1` script using the following syntax to begin the installation:

-URL (Required)

This is the URL to the Agent Services endpoint on the Keyfactor Command server running the Keyfactor Command Agent Services role. If you installed all the Keyfactor Command server roles together, this is the URL for your Keyfactor Command server with `/KeyfactorAgents` after the server's IP or FQDN (e.g. `https://keyfactor.keyexample.com/KeyfactorAgents`). If you choose to use SSL to connect to the Keyfactor Command server, you'll need to enter a URL that contains a hostname that is found in the SSL certificate.

This parameter sets the local orchestrator application setting *AgentsServerUri* to the specified value.

This parameter is **required**.



Note: If you've opted to use client certificate authentication for the orchestrator, the value you use for the **URL** will vary depending on the method you select to implement client certificate authentication. You may choose to route client certificate authentication through a proxy (see [Appendix - Set up the Universal Orchestrator to Use Client Certificate Authentication via a Reverse Proxy: Citrix ADC on page 150](#)), in which case you would use the proxy server name here (whatever name you're using to route traffic through the proxy). You may choose to publish client certificates to Active Directory and access the Keyfactor Command server directly (see [Appendix - Set up the Universal Orchestrator to Use Client Certificate Authentication with Certificates Stored in Active Directory on page 162](#)), in which case you would use the Keyfactor Command server name here.



Tip: If your Keyfactor Command server was configured with an alternate virtual directory for the Keyfactor Command Agents Services endpoint, you will need to enter that in the URL rather than `/KeyfactorAgents`.

Client Authentication Parameters (Required)

The Keyfactor Universal Orchestrator supports authenticating to the Keyfactor Command server using Basic authentication, token authentication, or client certificate authentication. The method you choose depends in part on the identity provider in use for the Keyfactor Command the orchestrator will be communicating with. If you're using Active Directory as an identity provider, you may choose Basic authentication or client certificate authentication. If you're using an identity provider other than Active Directory, you may choose token authentication or client certificate authentication.

When you configure the orchestrator with Basic authentication (*WebCredential*), you provide a username and password as a *PSCredential* object. With token authentication (*BearerTokenUrl*), you provide a client ID and secret that allows the orchestrator to acquire a bearer token. With client certificate authentication (either *ClientCertificateThumbprint* or *ClientCertificate* and *ClientCertificatePassword*), the orchestrator uses a client certificate to authenticate to either a proxy or IIS on the Keyfactor Command server. You cannot configure multiple types of authentication together.

One of the following authentication methods is **required**:

- Basic Authentication: *WebCredential*
- Token Authentication: *BearerTokenUrl*, *ClientId*, *ClientSecret*, and *TokenLifetime*
- Client Certificate Authentication: *ClientCertificate* and *ClientCertificatePassword*
- Client Certificate Authentication: *ClientCertificateThumbprint*



Important: Choosing a client certificate authentication method for the orchestrator may require additional configuration on your Keyfactor Command server. For more information, see [Appendix - Set up the Universal Orchestrator to Use Client Certificate Authentication with Certificates Stored in Active Directory on page 162](#), [Appendix - Set up the Universal Orchestrator to Use Client Certificate Authentication via a Reverse Proxy: Citrix ADC on page 150](#), and *Install the Keyfactor Command Components on the Keyfactor Command Server(s)* in the *Keyfactor Command Server Installation Guide*.



Tip: For information about rotating passwords and client authentication certificates, see [Change Service Account Passwords on page 81](#).

-WebCredential (Basic Authentication)

This is the credential object of the Keyfactor Command connect service account that the orchestrator uses to communicate with Keyfactor Command that you created as per [Create Service Accounts for the Universal Orchestrator on page 11](#). It is provided as a *PSCredential* object.

This parameter is **required** if Basic authentication will be used.

This parameter cannot be used in conjunction with the *BearerTokenURL*, *ClientCertificateThumbprint*, *ClientCertificate*, or *ClientCertificatePassword* parameter.

-BearerTokenURL (Token Authentication)

Specifying this parameter causes the installation to be done using token authentication for the connection to Keyfactor Command.

This parameter **requires** that *TokenLifetime*, *ClientId*, and *ClientSecret* also be specified.

This parameter is **required** if token authentication will be used.

This parameter cannot be used in conjunction with the *WebCredential*, *ClientCertificateThumbprint*, *ClientCertificate*, or *ClientCertificatePassword* parameter.

-ClientId (Token Authentication)

This parameter is used to specify the ID of the identity provider client that should be used to authenticate the session when *BearerTokenUrl* authentication is used (see [Create Service Accounts for the Universal Orchestrator on page 11](#)).

This parameter **requires** that *TokenLifetime* and *ClientSecret* also be specified.

This parameter is only supported if the *BearerTokenUrl* parameter is specified.

-ClientSecret (Token Authentication)

This parameter is used to specify the secret of the identity provider client that should be used to authenticate the session when *BearerTokenUrl* authentication is used.

This parameter **requires** that *TokenLifetime* and *ClientId* also be specified.

This parameter is only supported if the *BearerTokenUrl* parameter is specified.

-TokenLifetime (Token Authentication)

The number of seconds for which the bearer token is valid. The *TokenLifetime* should be set to the same value as the Keyfactor Command *Cookie Expiration* (see *Install the Main Keyfactor Command Components on the Keyfactor Command Server(s): Authentication Tab* in the *Keyfactor Command Server Installation Guide*). For example, if the Keyfactor Command *Cookie Expiration* is 5 minutes, the *TokenLifetime* should be 300 seconds.

The *Cookie Expiration* value determines the length of time the authentication cookie is considered valid. After half of the setting's duration, Keyfactor Command will attempt to use a refresh token to update the cookie. If this fails, the orchestrator's session will be terminated.

This parameter **requires** that *ClientId* and *ClientSecret* also be specified.

This parameter is only supported if the *BearerTokenUrl* parameter is specified.

-ClientCertificate (Client Certificate Authentication)

The path and file name on the orchestrator of a PKCS12 file containing the client authentication certificate used to authenticate to Keyfactor Command created as per [Acquire a Certificate for Client Certificate Authentication \(Optional\) on page 18](#). The certificate must have a Client Authentication EKU.

The account under which the Universal Orchestrator service will run (see [-ServiceCredential on page 34](#)) needs read and write permissions on the PKCS12 file you specify with this parameter.

This parameter **requires** that *ClientCertificatePassword* also be specified.

You may specify either the thumbprint of the certificate with the *ClientCertificateThumbprint* parameter or specify a path and password to a PKCS12 file containing the certificate on the orchestrator using *ClientCertificate* and *ClientCertificatePassword*. You do not need to specify both a thumbprint and a PKCS12 file; if you do, the certificate stores will take precedence.

Specifying this parameter sets the local orchestrator application setting *CertPath* to the specified value.

This parameter cannot be used in conjunction with the *BearerTokenURL* or *WebCredential* parameter.

-ClientCertificatePassword (Client Certificate Authentication)

The password for the PKCS12 file specified with the *ClientCertificate* parameter.

Specifying this parameter **requires** that *ClientCertificate* also be specified.

This parameter cannot be used in conjunction with the *BearerTokenURL* or *WebCredential* parameter.

-ClientCertificateThumbprint (Client Certificate Authentication)

The thumbprint of the client authentication certificate used to authenticate to Keyfactor Command created as per [Acquire a Certificate for Client Certificate Authentication \(Optional\) on page 18](#). The certificate must have a Client Authentication EKU, have a private key readable by the account under which the Universal Orchestrator service will run (see [-ServiceCredential on page 34](#)), and be located in either the orchestrator local machine's personal certificate store (*My*) or the Universal Orchestrator service account user's (see [-ServiceCredential on page 34](#)) personal certificate store. If the certificate is stored in the local machine's store, the Universal Orchestrator service account user must be granted permissions to read the private key of the certificate (see the final steps under [Acquire a Certificate for Client](#)

[Certificate Authentication \(Optional\) on page 18](#)).

You may specify either the thumbprint of the certificate with the *ClientCertificateThumbprint* parameter or specify a path and password to a PKCS12 file containing the certificate on the orchestrator using *ClientCertificate* and *ClientCertificatePassword*. You do not need to specify both a thumbprint and a PKCS12 file; if you do, the certificate stores will take precedence.

Specifying this parameter sets the local orchestrator application setting *AuthCertThumbprint* to the specified value.

This parameter cannot be used in conjunction with the *BearerTokenURL* or *WebCredential* parameter.

-Audience

This parameter is used to specify an audience value to be included in token requests delivered to the identity provider when using an identity provider other than Active Directory.

-Capabilities

This parameter is used to specify the capabilities the orchestrator will support if a capability set other than the default set is desired. Supported options are:

- all
All the capabilities supported by the orchestrator will be enabled and reported to Keyfactor Command.
- none
The orchestrator will be installed with no capabilities and will not be registered with Keyfactor Command. This is primarily used for implementations that will support only custom capabilities (see [Installing Custom-Built Extensions on page 67](#) and [Configuring Script-Based Certificate Store Jobs on page 73](#)).
- ssl
Only the SSL discovery and monitoring capability will be enabled and reported to Keyfactor Command.

If the *InPlace* parameter is specified, this parameter must be set to *all*.

If this parameter is not specified, the default set of capabilities for the orchestrator will be used. For the Universal Orchestrator, the default capability set is *IIS*, *CA* and *LOG* (log fetching).

One installation of the orchestrator can be enabled with multiple capabilities to perform more than one function, but there are best practices for locating orchestrators that should be considered. For example, Keyfactor recommends against performing the SSL discovery and monitoring function using an orchestrator installed on the main Keyfactor Command server due to the resource requirements of this function and against using the same orchestrator for the SSL function and other functions, again due to the resource requirements. The CA management

function is typically used on remote servers and not collocated with other orchestrator functions.

-Destination

This parameter specifies a location in which to install the orchestrator that is other than the default. The default installation location is:

```
C:\Program Files\Keyfactor\Keyfactor Orchestrator
```

This parameter cannot be used in conjunction with the *InPlace* parameter.

-Force

Specifying this parameter causes the installation to warn and continue on certain potential problems, including:

- A service with either the default service name or the service name specified with the *ServiceSuffix* parameter already exists. The service will be overwritten if *Force* is specified.
- Either the default installation location or the location specified with the *Location* parameter is not empty. The install will occur to the specified or default location anyway and files may be overwritten if *Force* is specified.

If this parameter is not specified and any of these problems are encountered, the installation will terminate prematurely.

-InPlace

This parameter is used to indicate that the installation should occur in the current directory where the install files are located and no files should be copied to another location on the machine.

This parameter cannot be used in conjunction with the *Destination* parameter. This parameter is only supported if the *Capabilities* parameter is set to *all*.

-NoRevocationCheck

This parameter is used to indicate that the revocation status (CRL) of the SSL certificate on the Keyfactor Command server should not be checked when connecting to Keyfactor Command.

Specifying this parameter sets the local orchestrator application setting *Check-ServerCertificateRevocation* to false. The default for this parameter is *true* (CRL checking will be done).

-NoService

This parameter is used to indicate that no Windows service should be created. The orchestrator will be installed but will need to be started manually or added as a service at a later time.

This parameter cannot be used in conjunction with the *ServiceSuffix* or *ServiceCredential* parameter.

-OrchestratorName

Specifying this parameter allows you to override the name the orchestrator would by default use to register itself in Keyfactor Command.

Specifying this parameter sets the local orchestrator application setting *OrchestratorName* to the specified value.

By default, the orchestrator uses the value of the `COMPUTERNAME` environment variable for the orchestrator's name.

-ServiceCredential

This is the credential object of the Universal Orchestrator service account the orchestrator service will run as (see [Create Service Accounts for the Universal Orchestrator on page 11](#)). It is provided as a `PSCredential` object.

This parameter cannot be used in conjunction with the *NoService* parameter.

If this parameter is not specified, the built-in Network Service account will be used.

-ServiceSuffix

This parameter is used to add a suffix to the root service name of *KeyfactorOrchestrator* (e.g. *Instance1* for a resulting service name of *KeyfactorOrchestrator-Instance1*). This is used primarily for implementations where the orchestrator will be installed multiple times on the same server.

This parameter cannot be used in conjunction with the *NoService* parameter.

If this parameter is not specified, the default service name of *KeyfactorOrchestrator-Default* will be used—with a display name of *Keyfactor Orchestrator Service (Default)*.

-Scope

This parameter is used to specify one or more scopes that should be included in token requests delivered to the identity provider when using an identity provider other than Active Directory. Multiple scopes should be separated by spaces.

-Source

Specify this parameter to point to a directory containing the installation files other than the directory in which the install.ps1 file is found. This parameter is used primarily if a copy of the install.ps1 file is made in an alternate directory, updated with some customizations, and then used for installation without being copied back to the directory where the remaining installation files are located.

Installation example with expected output using Basic authentication and Network Service to run the local service:

```
$keyfactorUser = "KEYEXAMPLE\svc_kyforch1"
$keyfactorPassword = "MySecurePassword123!"
$secKeyfactorPassword = ConvertTo-SecureString $keyfactorPassword -AsPlainText -Force
$credKeyfactor = New-Object System.Management.Automation.PSCredential ($keyfactorUser, $secKeyfactorPassword)

.\install.ps1 -URL https://keyfactor.keyexample.com/KeyfactorAgents -WebCredential $credKeyfactor -OrchestratorName websrvr42.keyexample.com -Capabilities all

Copying files
Setting configuration data
Installing Windows Service
Granting necessary file permissions to NT AUTHORITY\NETWORK SERVICE for configuration file
Starting service KeyfactorOrchestrator-Default
```

Installation example with expected output using Basic authentication and a standard Active Directory service account to run the local service:

```
$serviceUser = "KEYEXAMPLE\svc_kyforch1"
$keyfactorUser = "KEYEXAMPLE\svc_kyforch2"
$servicePassword = "MyFirstSecurePassword123!"
$keyfactorPassword = "MySecondSecurePassword456#"
$secServicePassword = ConvertTo-SecureString $servicePassword -AsPlainText -Force
$secKeyfactorPassword = ConvertTo-SecureString $keyfactorPassword -AsPlainText -Force
$credService = New-Object System.Management.Automation.PSCredential ($serviceUser, $secServicePassword)
$credKeyfactor = New-Object System.Management.Automation.PSCredential ($keyfactorUser, $secKeyfactorPassword)

.\install.ps1 -URL https://keyfactor.keyexample.com/KeyfactorAgents -WebCredential $credKeyfactor -ServiceCredential $credService -OrchestratorName websrvr42-IIS.keyexample.com -Capabilities all

Copying files
```

```
Setting configuration data
Installing Windows Service
Granting necessary file permissions to KEYEXAMPLE\svc_kyforch1 for configuration file
Granting Log on as a Service permission to KEYEXAMPLE\svc_kyforch1
Starting service KeyfactorOrchestrator-Default
```

Installation example with expected output using Basic authentication and an Active Directory gMSA to run the local service:

```
$serviceUser = "KEYEXAMPLE\GMSA_kyforch$"
$keyfactorUser = "KEYEXAMPLE\svc_kyforch"
$keyfactorPassword = "MySecurePassword123!"
$secKeyfactorPassword = ConvertTo-SecureString $keyfactorPassword -AsPlainText -Force
$credService = New-Object System.Management.Automation.PSCredential ($serviceUser,(New-Object
System.Security.SecureString))
$credKeyfactor = New-Object System.Management.Automation.PSCredential ($keyfactorUser, $secKey-
factorPassword)

.\install.ps1 -URL https://keyfactor.keyexample.com/KeyfactorAgents -WebCredential $credKeyfactor
-ServiceCredential $credService -OrchestratorName webservr42-IIS.keyexample.com -Capabilities all

Copying files
Setting configuration data
Installing Windows Service
Granting necessary file permissions to KEYEXAMPLE\GMSA_kyforch$ for configuration file
Granting Log on as a Service permission to KEYEXAMPLE\GMSA_kyforch$
Starting service KeyfactorOrchestrator-Default
```



Important: Prior to using a gMSA in the installation, you need to have installed the account on the Universal Orchestrator server using the *Install-ADServiceAccount* PowerShell command. For example:

```
Install-ADServiceAccount -Identity GMSA_kyforch$
```

This requires the *Active Directory module for Windows PowerShell*, which is installed as a feature as part of the *Remote Server Administrator Tools*.

Installation example with expected output using token authentication and Network Service to run the local service:

```
.\install.ps1 -URL https://keyfactor.keyexample.com/KeyfactorAgents -BearerTokenUrl https://appsrvr18.keyexample.com:1443/realms/Keyfactor/protocol/openid-connect/token -TokenLifetime 300 -
```

```
ClientId Universal-Orchestrator -ClientSecret m1aE6RErW5cezSPmv0PJcFdFp152HFqK -OrchestratorName  
websrvr42-U0.keyexample.com -Capabilities all
```

```
Copying files  
Setting configuration data  
Installing Windows Service  
Granting necessary file permissions to NT AUTHORITY\NETWORK SERVICE for configuration file  
Starting service KeyfactorOrchestrator-Default
```

Installation example with expected output using token authentication and a local account on the machine to run the local service:

```
$serviceUser = "websrvr42\kyforch"  
$servicePassword = "MySecurePassword123!"  
$secServicePassword = ConvertTo-SecureString $servicePassword -AsPlainText -Force  
$credService = New-Object System.Management.Automation.PSCredential ($serviceUser, $secServicePassword)  
  
.install.ps1 -URL https://keyfactor.keyexample.com/KeyfactorAgents -BearerTokenUrl https://appsrvr18.keyexample.com:1443/realms/Keyfactor/protocol/openid-connect/token -TokenLifetime 300 -  
ClientId Universal-Orchestrator -ClientSecret m1aE6RErW5cezSPmv0PJcFdFp152HFqK -ServiceCredential  
$credService -OrchestratorName websrvr42-U0.keyexample.com -Capabilities all  
  
Copying files  
Setting configuration data  
Installing Windows Service  
Granting necessary file permissions to websrvr42\kyforch for configuration file  
Granting Log on as a Service permission to websrvr42\kyforch  
Starting service KeyfactorOrchestrator-Default
```

Installation example with expected output using client certificate authentication with the certificate stored in the local machine store:

```
$serviceUser = "KEYEXAMPLE\svc_kyforch"  
$servicePassword = "MySecurePassword123!"  
$secServicePassword = ConvertTo-SecureString $servicePassword -AsPlainText -Force  
$credService = New-Object System.Management.Automation.PSCredential ($serviceUser, $secServicePassword)  
  
.install.ps1 -URL https://keyfactor.keyexample.com/KeyfactorAgents -ClientCertificateThumbprint  
29b21df7403b4afe6daf44762e5c47fb73c07ce7 -ServiceCredential $credService -OrchestratorName  
websrvr42-IIS.keyexample.com -Capabilities all
```

```
Copying files
Setting configuration data
Installing Windows Service
Granting necessary file permissions to KEYEXAMPLE\svc_kyforch for configuration file
Granting Log on as a Service permission to KEYEXAMPLE\svc_kyforch
Starting service KeyfactorOrchestrator-Default
```



Tip: The client certificate authentication example shown here references a certificate stored in the local machine store. Because of this, the service account that will run the Universal Orchestrator service needs to be granted permissions to read the private key of the certificate before the installation is run. If the certificate had been acquired into the Universal Orchestrator service account user's personal store rather than the local machine store, the step of granting private key read permissions would not have been necessary.

Installation example with expected output using client certificate authentication with the certificate stored as a file:

```
.\install.ps1 -URL https://keyfactor.keyexample.com/KeyfactorAgents -ClientCertificate
C:\Certs\kyforch.pfx -ClientCertificatePassword MySecurePassword123! -OrchestratorName webservr42-
IIS.keyexample.com -Capabilities all

Copying files
Setting configuration data
Installing Windows Service
Granting necessary file permissions to KEYEXAMPLE\svc_kyforch for configuration file
Granting Log on as a Service permission to KEYEXAMPLE\svc_kyforch
Starting service KeyfactorOrchestrator-Default
```



Tip: The client certificate authentication example shown here does not use the -ServiceCredential parameter. This will cause the Universal Orchestrator service to run as Network Service. If you prefer to run the service as a domain service account, you will need to include the -ServiceCredential parameter and specify the PSCredential value for the service credentials appropriately, as shown in the previous examples. Network Service will need to be granted read and write permissions on the PFX file before the script is executed.

4. Review the output from the installation to confirm that no errors have occurred.

The script creates a directory, C:\Program Files\Keyfactor\Keyfactor Orchestrator by default, and places the orchestrator files in this directory. Log files are found in C:\Program Files\Keyfactor\Keyfactor Orchestrator\logs by default, though this is configurable (see [Configure Logging for the Universal Orchestrator on page 77](#)).

The orchestrator service, by default given a display name of *Keyfactor Orchestrator Service (Default)*, should be automatically started at the conclusion of the install and configured to restart on reboot unless you have selected the *NoService* parameter.



Tip: Once the installation of the orchestrator is complete, you need to use the Keyfactor Command Management Portal to approve the orchestrator and configure certificate stores or SSL jobs as per the *Keyfactor Command Reference Guide*:

- Orchestrator Management Operations: Approving or Disapproving Orchestrators
- Certificate Store Operations
- SSL Discovery

If you've opted to enable remote CA management for the orchestrator, further configuration is needed (see [Configure the Universal Orchestrator for Remote CA Management on page 65](#)).

2.2.3 Install the Universal Orchestrator on a Linux Server

To install the Keyfactor Universal Orchestrator on a Linux server, copy the zip file containing installation files to a temporary working directory on the Linux server and unzip it.

To begin the installation:

1. On the Linux machine on which you wish to install the orchestrator, in a command shell change to *InstallationScripts* subdirectory under the temporary directory where you placed the installation files.
2. Use the `chmod` command to make the `install.sh` script file executable. The file ships in a non-executable state to avoid accidental execution. For example:

```
sudo chmod +x install.sh
```

3. In the command shell, run the `install.sh` script as root using the following parameters to begin the installation:

--url (Required)

This is the URL to the Agent Services endpoint on the Keyfactor Command server running the Keyfactor Command Agent Services role, which is installed as part of the Keyfactor Command Services role. If you installed all the Keyfactor Command server roles together, this is the URL for your Keyfactor Command server with `/KeyfactorAgents` after the server's IP or FQDN (e.g. `https://keyfactor.keyexample.com/KeyfactorAgents`). If you choose to use SSL to connect to the Keyfactor Command server, you'll need to enter a URL that contains a hostname that is found in the SSL certificate.

This parameter sets the local orchestrator application setting *AgentsServerUri* to the specified value.

This parameter is **required**.



Tip: If your Keyfactor Command server was configured with an alternate virtual directory for the Keyfactor Command Agents Services endpoint, you will need to enter that in the URL rather than /KeyfactorAgents.

Client Authentication Parameters (Required)

The Keyfactor Universal Orchestrator supports authenticating to the Keyfactor Command server using Basic authentication, token authentication, or client certificate authentication. The method you choose depends in part on the identity provider in use for the Keyfactor Command the orchestrator will be communicating with. If you're using Active Directory as an identity provider, you may choose Basic authentication or client certificate authentication. If you're using an identity provider other than Active Directory, you may choose token authentication or client certificate authentication.

When you configure the orchestrator with Basic authentication (*username* and *password*), you provide a username and password. With token authentication (*bearer-token-url*, *client_id*, and *client_secret*), you provide a client ID and secret that allows the orchestrator to acquire a bearer token. With client certificate authentication (*client-auth-certificate* and *client-auth-certificate-password*), the orchestrator uses a client certificate to authenticate to either a proxy or IIS on the Keyfactor Command server. You cannot configure multiple types of authentication together.

One of the following authentication methods is **required**:

- Basic Authentication: *username* and *password*
- Token Authentication: *bearer-token-url*, *client_id*, *client_secret*, and *token_lifetime*
- Client Certificate Authentication: *client-auth-certificate* and *client-auth-certificate-password*



Important: Choosing to use client certificate authentication for the orchestrator may require additional configuration on your Keyfactor Command server. For more information, see *Install the Main Keyfactor Command Components on the Keyfactor Command Server(s)* in the *Keyfactor Command Server Installation Guide* and [Appendix - Set up the Universal Orchestrator to Use Client Certificate Authentication with Certificates Stored in Active Directory on page 162](#), [Appendix - Set up the Universal Orchestrator to Use Client Certificate Authentication via a Reverse Proxy: Citrix ADC on page 150](#).



Tip: For information about rotating passwords and client authentication certificates, see [Change Service Account Passwords on page 81](#).

`--username` (Basic Authentication)


This is the Keyfactor Command connect service account that the orchestrator uses to communicate with Keyfactor Command that you created as per [Create Service Accounts for the Universal Orchestrator on page 11](#). It may be entered either as `username@domain` (e.g. `svc_kyforch@keyexample.com`) or `DOMAIN\username` (e.g. `KEYEXAMPLE\svc_kyforch`).

This parameter is **required** if Basic authentication will be used.

This parameter cannot be used in conjunction with the *bearer-token-url* or *client-auth-certificate* and *client-auth-certificate-password* parameters.


`--password` (Basic Authentication)

This is the password for the Keyfactor Command connect service account that the orchestrator uses to communicate with Keyfactor Command specified with the *username* parameter.

 **Important:** The password for the Keyfactor Command connect service account is stored in clear text in the `orchestratorsecrets.json` file in the configuration directory under the installation directory for the orchestrator. By default, this file is granted read/write permissions for the Universal Orchestrator service account running the service on the Linux machine (*keyfactor-orchestrator* by default) and no permissions for any other users. Access to this file should be strictly controlled. If you prefer to avoid the use of a password in a file, consider using client certificate authentication.

This parameter is **required** if the *username* parameter is specified.

This parameter cannot be used in conjunction with the *bearer-token-url* or *client-auth-certificate* and *client-auth-certificate-password* parameters.

 **Important:** Your password may be preserved in command history and may be visible on the process listing when providing a password using this parameter. See the *--secret-file-path* and *--secret-std-in* parameters for alternatives.

`--bearer-token-url` (Token Authentication)

Specifying this parameter causes the installation to be done using token authentication for the connection to Keyfactor Command. Set this to the URL of the token endpoint for your identity provider. For example:

```
https://my-keyidp-server.keyexample.com/realms/Keyfactor/protocol/openid-connect/token
```

For Keyfactor Identity Provider, this is included among the information that can be found on the OpenID Endpoint Configuration page, a link to which can be found on the Realm Settings page (see *Gathering Keyfactor Identity Provider Data for the Keyfactor Command Installation* in the *Keyfactor Command Server Installation Guide*).

This parameter **requires** that *token-lifetime*, *client-id*, and *client-secret* also be specified.

This parameter is **required** if token authentication will be used.

This parameter cannot be used in conjunction with the *username* and *password* or *client-auth-certificate* and *client-auth-certificate-password* parameters.

--client-id (Token Authentication)


This parameter is used to specify the ID of the identity provider client that should be used to authenticate the session when *bearer-token-url* authentication is used (see [Create Service Accounts for the Universal Orchestrator on page 11](#)).

This parameter **requires** that *token-lifetime* and *client-secret* also be specified.

This parameter is only supported if the *bearer-token-url* parameter is specified.


--client-secret (Token Authentication)

This parameter is used to specify the secret of the Keyfactor Identity Provider client that should be used to authenticate the session when *bearer-token-url* authentication is used.

 **Important:** The client secret for the Keyfactor Command connect service account is stored in clear text in the `orchestratorsecrets.json` file in the configuration directory under the installation directory for the orchestrator. By default, this file is granted read/write permissions for the Universal Orchestrator service account running the service on the Linux machine (*keyfactor-orchestrator* by default) and no permissions for any other users. Access to this file should be strictly controlled. If you prefer to avoid the use of a password in a file, consider using client certificate authentication.

This parameter **requires** that *token-lifetime* and *client-id* also be specified.

This parameter is only supported if the *bearer-token-url* parameter is specified.

 **Important:** Your secret may be preserved in command history and may be visible on the process listing when providing a secret using this parameter. See the `--secret-file-path` and `--secret-std-in` parameters for alternatives.

--token-lifetime (Token Authentication)

The number of seconds for which the bearer token is valid. The *token-lifetime* should be set to the same value as the Keyfactor Command*CookieExpiration*. For example, if the Keyfactor Command*CookieExpiration* is 5 minutes, the *token-lifetime* should be 300 seconds.

The *Cookie Expiration* value determines the length of time the authentication cookie is considered valid. After half of the setting's duration, Keyfactor Command will attempt to use a refresh token to update the cookie. If this fails, the orchestrator's session will be terminated.

This parameter **requires** that *client-id* and *client-secret* also be specified.

This parameter is only supported if the *bearer-token-url* parameter is specified.

--client-auth-certificate (Client Certificate Authentication)

The path and file name on the orchestrator of a PKCS12 file containing the client authentication certificate used to authenticate to Keyfactor Command created as per [Acquire a Certificate for Client Certificate Authentication \(Optional\) on page 18](#). The certificate must have a Client Authentication EKU.

The account under which the Universal Orchestrator service will run (see [--service-user on page 47](#)) needs read and write permissions on the PKCS12 file you specify with this parameter.

This parameter **requires** that *client-auth-certificate-password* also be specified.

Specifying this parameter sets the local orchestrator application setting *CertPath* to the specified value.

This parameter cannot be used in conjunction with the *bearer-token-url* or *username* and *password* parameters.

--client-auth-certificate-password (Client Certificate Authentication)

The password for the PKCS12 file specified with the *client-auth-certificate* parameter.

Specifying this parameter **requires** that *client-auth-certificate* also be specified.

This parameter cannot be used in conjunction with the *bearer-token-url* or *username* parameters.



Important: Your password may be preserved in command history and may be visible on the process listing when providing a password using this parameter. See the *--secret-file-path* and *--secret-std-in* parameters for alternatives.


--secret-file-path (All Authentication Types)

This parameter specifies a path and filename to provide a plain text secret for the Keyfactor Command connect service account that the orchestrator uses to communicate with Keyfactor Command. For example:

```
sudo ./install.sh --secret-file-path /opt/apps/my_secret_file [other parameters here]
```

This parameter can be used with the *username*, *client-auth-certificate*, or *client-id* parameter to provide the authentication secret from a file rather than the command line to avoid storing it in command history.

This parameter cannot be used in conjunction with the *password*, *client_secret*, or *client-auth-certificate-password* parameter.

 **Tip:** Be sure to delete your secret file at the conclusion of the installation.

--secret-std-in (All Authentication Types)

This parameter allows you to provide a plain text secret via standard in for the Keyfactor Command connect service account that the orchestrator uses to communicate with Keyfactor Command. For example:

```
echo "MySuperSecretPassword" | sudo ./install.sh --secret-std-in [other parameters here]
```

This parameter can be used with the *username*, *client-auth-certificate*, or *client-id* parameter to provide the authentication secret from a file rather than the command line to avoid storing it in command history.

This parameter cannot be used in conjunction with the *password*, *client_secret*, or *client-auth-certificate-password* parameter.

--audience

This parameter is used to specify an audience value to be included in token requests delivered to the identity provider when using an identity provider other than Active Directory.

--capabilities

This parameter is used to specify the capabilities the orchestrator will support if a capability set other than the default set is desired. Supported options are:

- all

All the capabilities supported by the orchestrator will be enabled and reported to Keyfactor Command.

- none

The orchestrator will be installed with no capabilities and will not be registered with Keyfactor Command. This is primarily used for implementations that will support only custom capabilities (see [Installing Custom-Built Extensions on page 67](#) and [Configuring Script-Based Certificate Store Jobs on page 73](#)).

- ssl

Only the SSL discovery and monitoring capability will be enabled and reported to Keyfactor Command.

If the *in-place* parameter is specified, this parameter must be set to *all*.

If this parameter is not specified, the default set of capabilities for the orchestrator will be used. For the Linux orchestrator, the default capability set is *LOG* (log fetching).



Important: The Linux orchestrator does not support the CA (remote CA management) or IIS (Windows server certificate store) capabilities due to the Windows-specific nature of the authentication requirements for these methods.

—destination

This parameter specifies a location in which to install the orchestrator that is other than the default. The default installation location is:

```
/opt/keyfactor/orchestrator
```

This parameter cannot be used in conjunction with the *in-place* parameter.

—force, -f

Specifying this parameter causes the installation to warn and continue on certain potential problems, including:

- The local Universal Orchestrator service account does not exist. The default user will be created if *force* is specified.
- The local application settings (appsettings.json) file does not exist. A new one will be created if *force* is specified.
- A service with either the default service name or the service name specified with the *service-suffix* parameter already exists. The service will be overwritten if *force* is specified.
- Either the default installation location or the location specified with the *location* para-

meter is not empty. The install will occur to the specified or default location anyway and files may be overwritten if *force* is specified.

If this parameter is not specified and any of these problems are encountered, the installation will terminate prematurely. See also the *what-if* parameter.

--in-place

This parameter is used to indicate that the installation should occur in the current directory where the install files are located and no files should be copied to another location on the machine.

This parameter cannot be used in conjunction with the *destination* parameter. This parameter is only supported if the *capabilities* parameter is set to *all*.

--no-revocation-check

This parameter is used to indicate that the revocation status (CRL) of the SSL certificate on the Keyfactor Command server should not be checked when connecting to Keyfactor Command.

Specifying this parameter sets the local orchestrator application setting *Check-ServerCertificateRevocation* to false. The default for this parameter is *true* (CRL checking will be done).

--no-service

This parameter is used to indicate that no service should be created and added to the server's service control manager. The orchestrator will be installed but will need to be started manually or added to the server's service control manager manually.

This parameter cannot be used in conjunction with the *service-suffix* or *service-user* parameter.

--orchestrator-name

Specifying this parameter allows you to override the name the orchestrator would by default use to register itself in Keyfactor Command.

Specifying this parameter sets the local orchestrator application setting *OrchestratorName* to the specified value.

By default, the orchestrator uses the results from a hostname lookup for the orchestrator's name.

--service-suffix

This parameter is used to add a suffix to the root service name of *keyfactor-orchestrator* (e.g. *instance1* for a resulting service name of *keyfactor-orchestrator-instance1*). This is used primarily for implementations where the orchestrator will be installed multiple times on the same

server.

This parameter cannot be used in conjunction with the *no-service* parameter.

If this parameter is not specified, the default service name of *keyfactor-orchestrator-default* will be used.

--service-user

This is the local Linux Universal Orchestrator service account that the service will run as (see [Create Service Accounts for the Universal Orchestrator on page 11](#)). It should be entered as just the user name. Entry of a password for this service account is not required. You may either create this account prior to running the installation script (or use an existing account) or use the *force* parameter to generate the account automatically during the installation process.

This parameter cannot be used in conjunction with the *no-service* parameter.

If this parameter is not specified, the default service account name of *keyfactor-orchestrator* will be used.

--scope

This parameter is used to specify one or more scopes that should be included in token requests delivered to the identity provider when using an identity provider other than Active Directory. Multiple scopes should be separated by spaces.

--source

Specify this parameter to point to a directory containing the installation files other than the directory in which the *install.sh* file is found. This parameter is used primarily if a copy of the *install.sh* file is made in an alternate directory, updated with some customizations, and then used for installation without being copied back to the directory where the remaining installation files are located.

--verbose, -v

Specify this parameter to output verbose installation messages.

--what-if

This parameter is used to test the installation command without actually installing in order to see any errors that might arise and correct them before installing.

Installation example with expected output using Basic authentication (the password for the *svc_kyforch* service account is saved in *my_password_file*):

```
vi my_password_file
```

```

sudo ./install.sh --url https://keyfactor.keyexample.com/KeyfactorAgents --username svc_
kyforch@keyexample.com --secret-file-path my_password_file --orchestrator-name appsrvr16-
ssl.keyexample.com --capabilities all --force

Creating user keyfactor-orchestrator
Copying files from /tmp/KeyfactorOrchestrator to /opt/keyfactor/orchestrator
Saving app settings
Setting file permissions
Installing systemd service keyfactor-orchestrator-default
Created symlink /etc/systemd/system/multi-user.target.wants/keyfactor-orchestrator-default.ser-
vice → /etc/systemd/system/keyfactor-orchestrator-default.service.
Starting systemd service keyfactor-orchestrator-default

```

Installation example with expected output using token authentication (the secret for the client is provided at standard in):

```

echo "WcHlahyku6wmD0a6rj0XC1rkz0Jw9sGh" | sudo ./install.sh --url https://key-
factor.keyexample.com/KeyfactorAgents --bearer-token-url https://appsr-
vr18.keyexample.com:1443/realms/Keyfactor/protocol/openid-connect/token --token-lifetime 300 --
client-id Universal-Orchestrator --secret-stdin --orchestrator-name appsrvr16-ssl.keyexample.com
--capabilities all --force

Creating user keyfactor-orchestrator
Copying files from /tmp/KeyfactorOrchestrator to /opt/keyfactor/orchestrator
Setting file permissions and saving app settings
Installing systemd service keyfactor-orchestrator-default
Created symlink /etc/systemd/system/multi-user.target.wants/keyfactor-orchestrator-default.ser-
vice → /etc/systemd/system/keyfactor-orchestrator-default.service.
Starting systemd service keyfactor-orchestrator-default

```

Installation example with expected output using client certificate authentication (the password for the client certificate is saved in cert_password_file):

```

vi cert_password_file

sudo ./install.sh --url https://keyfactor.keyexample.com/KeyfactorAgents --client-auth-certi-
ficate /opt/certs/kyforch.p12 --secret-file-path cert_password_file --orchestrator-name
appsrvr16-ssl.keyexample.com --capabilities all --force

Creating user keyfactor-orchestrator
Copying files from /tmp/KeyfactorOrchestrator to /opt/keyfactor/orchestrator
Saving app settings
Setting file permissions

```

```
Installing systemd service keyfactor-orchestrator-default
Created symlink /etc/systemd/system/multi-user.target.wants/keyfactor-orchestrator-default.service → /etc/systemd/system/keyfactor-orchestrator-default.service.
Starting systemd service keyfactor-orchestrator-default
```

4. Review the output from the installation to confirm that no errors have occurred.

The script creates a directory, `/opt/keyfactor/orchestrator` by default, and places the orchestrator files in this directory. Log files are found in `/opt/keyfactor/orchestrator/logs` by default, though this is configurable (see [Configure Logging for the Universal Orchestrator on page 77](#)).

The orchestrator service, by default named `keyfactor-orchestrator-default.service`, should be automatically started at the conclusion of the install and configured to restart on reboot unless you have selected the `no-service` parameter.



Tip: Once the installation of the orchestrator is complete, you need to use the Keyfactor CommandManagement Portal to approve the orchestrator and configure certificate stores or SSL jobs as per the *Keyfactor Command Reference Guide*:

- Orchestrator Management Operations: Approving or Disapproving Orchestrators
- Certificate Store Operations
- SSL Discovery

2.2.4 Install the Universal Orchestrator in a Linux Container

When the Keyfactor Universal Orchestrator runs in a Linux container, it is typically installed in a containerization solution that sits on top of a Linux server or set of servers. There are a wide variety of containerization solutions for multiple operating systems. This document covers deploying the container to either Docker or Kubernetes on Linux.

The artifactory for the Universal Orchestrator images can be found here:

`keyfactor.jfrog.io/con-develop-us-engineering/command/`

Check with your Keyfactor Customer Success Manager for credentials.

Two different images are available, depending on the functionality you are looking for:

- `universal-orchestrator`

This image has no built-in functionality and is designed to be used with custom extensions.

- `universal-orchestrator-ssl`

This image provides the SSL capability to provide support for SSL discovery and monitoring.

Docker

If you plan to use Docker, you may find it helpful to first run the Universal Orchestrator in the foreground so that it will output log messages to assist in troubleshooting.



Tip: If your Docker implementation hasn't been configured to inject your DNS server(s) into running containers, you may wish to do this so that the Universal Orchestrator will be able to do name resolution. To do this, on the Linux server(s) where you are running Docker, create or update the `/etc/docker/daemon.json` file, and add an entry similar to the following:

```
{"dns": ["DNS_IP_Address_1", "DNS_IP_Address_2"]} }
```

To install the Universal Orchestrator in a Linux container and start the container using compose:

1. Create a directory from which you will run the Docker container (e.g. `/opt/kyf_uo`).
2. Select a Universal Orchestrator image, and from your Docker host, retrieve the Universal Orchestrator image from the artifactory with commands similar to the following (using credentials provided to you by Keyfactor; the password is saved in `my_password.txt`):

```
cat my_password.txt | docker login keyfactor.jfrog.io --username username --password-stdin
```

```
docker pull keyfactor.jfrog.io/con-develop-us-engineering/command/universal-orchestrator-ssl:11.0
```



Important: Remove the `my_password.txt` file when complete.

3. Create a Docker compose file (`compose.yaml`) in the directory for your Docker container similar to the following, using the inputs as per [Table 1: Linux Container Parameters](#), referencing the artifactory you pulled, selecting the appropriate authentication mechanism for your environment, and any additional volume mounts (see [Custom Extensions on page 52](#)). The fields highlighted in red below indicate fields that need to be edited or that you may wish to edit.



Important: When editing the file, be sure to preserve the indenting exactly as found. YAML requires a very specific file layout to function. If the indenting (multiples of two spaces) or layout is incorrect, you will receive an error when trying to install.

```
services:
  universal-orchestrator:
    image: keyfactor.jfrog.io/con-develop-us-engineering/command/universal-orchestrator-ssl:11.0
    container_name: universal_orchestrator_1
    environment:
```

```

COMMAND_AGENTS_URL: https://keyfactor.keyexample.com/KeyfactorAgents
ORCHESTRATOR_NAME: appsrvr19-U0-1

# Uncomment the next two lines to use Active Directory
#USERNAME: KEYEXAMPLE\svc_kyforch
#PASSWORD: MySuperSecretPassword

# Uncomment the next two lines to use OAuth
#BEARER_TOKEN_URL: https://appsrvr18.keyexample.com:1443/realms/Keyfactor/protocol/openid-
connect/token
#TOKEN_LIFETIME: 300
#CLIENTID: Universal-Orchestrator
#CLIENT_SECRET: Client-Secret-from-Keyfactor-IdP
volumes:
  - /etc/ssl/certs:/etc/ssl/certs:ro

```



Important: The password or secret for the Keyfactor Command connect service account is stored in clear text in this compose file. Access to this file should be strictly controlled.

4. Set the permissions on the *compose.yaml* file such that the file is owned by root and readable only by root (this assumes your Docker daemon is running as root, which is typical). For example:

```
sudo chown root:root compose.yaml
```

```
sudo chmod 400 compose.yaml
```



Tip: If you need to make edits to the compose file, you will need to make the file writable again. For example:

```
sudo chmod 600 compose.yaml
```

5. Execute the following command to install and run the container in the foreground:

```
sudo docker compose up
```

Press CTRL-C to stop it if it's running in the foreground. You can instead run it in the background by adding the *-d* flag like so, but it can sometimes be helpful to run it in the foreground initially so that you can easily review the log output live:

```
sudo docker compose up -d
```



Tip: To stop and start the container again after installation is complete, use the following commands:

```
sudo docker compose stop
```

```
sudo docker compose start
```

Or:

```
sudo docker compose restart
```

If you need to delete the container and try the install again, use this command:

```
sudo docker compose down
```

To review logs generated from the container, identify the container ID or name with this command:

```
sudo docker container ls
```

Then use the following command to output the current log (with the optional `--follow` to make output continuous):

```
sudo docker container logs [--follow] [container ID or name]
```

Custom Extensions

To use custom extensions with the orchestrators (see [Installing Custom-Built Extensions on page 67](#)), you can either build them into a custom-built orchestrator image or reference them as external volume mounts. The latter works well for quick testing in a development environment, but you'll probably want to use a custom-built orchestrator image for a production deployment.

To run the orchestrator referencing an extension as an external volume mount:

1. Create a directory on your Linux server to host the extension(s) you wish to use and copy each extension you wish to use into this directory. For example:

```
/opt/kyf_uo/exts/f5-ext
```

```
/opt/kyf_uo/exts/citrix-ext
```

Make note of whether the extension documentation indicates whether the files in the extension need to exist within a subdirectory or whether they should be placed in the root of the directory you're creating (e.g. f5-ext).

2. Follow the instructions above, but modify the volumes section of your docker compose file to include the extension(s). For example (where /app/extensions is the path within the image where the extensions will live):

```
[See beginning above]
volumes:
  - /etc/ssl/certs:/etc/ssl/certs:ro
  - /opt/kyf_uo/exts/f5-ext:/app/extensions/f5-ext
  - /opt/kyf_uo/exts/citrix-ext:/app/extensions/citrix-ext
```

To create a custom build of the orchestrator referencing extensions:

1. Create a directory on your Linux server to host the extension(s) you wish to use and copy each extension you wish to use into this directory. This should be a subdirectory of the directory in which you will build your custom orchestrator image. For example:

```
/opt/kyf_uo/exts/f5-ext
```

```
/opt/kyf_uo/exts/citrix-ext
```

Make note of whether the extension documentation indicates whether the files in the extension need to exist within a subdirectory or whether they should be placed in the root of the directory you're creating (e.g. f5-ext).

2. In the directory above the extension directory (e.g. /opt/kyf_uo), create a file called *Dockerfile* and open it for editing. The entries in this file will vary depending on the extension(s) you wish to include in your build. Check the documentation for the specific extension for more information. The following example includes two extensions:

```
FROM keyfactor.jfrog.io/con-develop-us-engineering/command/universal-orchestrator:11.0
WORKDIR "/app/extensions/f5-ext"
COPY ./ext/f5-ext/ ./
WORKDIR "/app/extensions/citrix-ext"
COPY ./ext/citrix-ext/ ./
WORKDIR "/app"
```

This build script sets the source for the artifactory and then changes directories within the image to the f5-ext directory. It copies the contents of the ext/f5-ext subdirectory under the current working directory on the host to the current directory in the image. It then repeats this change directory and copy for the Citrix Netscaler extension. Finally, it changes directory back

to the /app directory within the image before leaving the build. This last step is important to be sure the image will later function as expected.

3. Build your custom image by executing the following command from the directory in which your *Dockerfile* is located (where `custom-uo-image` is the name you give to your image):

```
docker build -t custom-uo-image .
```

4. Create your compose file as above, but referencing your custom image like so:

```
services:
  universal-orchestrator:
    image: custom-uo-image
    container_name: universal_orchestrator_f5_ns
[see remainder above]
```

5. Complete the install as above.

Kubernetes

To install the Universal Orchestrator in a Linux container and start the container using Kubernetes:

1. Create a directory from which you will run the container (e.g. /opt/kyf_uo).
2. Create a secret in Kubernetes for the credentials that the orchestrator(s) will use to authenticate to Keyfactor Command (see [Create Service Accounts for the Universal Orchestrator on page 11](#)). For example, for Active Directory authentication:

```
echo -n 'keyexample\svc_kyforch' > ./username
```

```
echo -n 'MySuperSecretPassword' > ./password
```

```
kubectl create secret generic uo-credentials --from-file=./username --from-file=./password
```

For authentication with an identity provider other than Active Directory:

```
echo -n 'Universal-Orchestrator' > ./clientid
```

```
echo -n 'Client-Secret-from-Keyfactor-IdP' > ./clientsecret
```

```
kubectl create secret generic uo-credentials --from-file=./clientid --from-file=./clientsecret
```




Important: The password or secret for the Keyfactor Command connect service account is stored in clear text in the “password” or “clientsecret” file. Be sure to delete it after the Kubernetes secret has been created.

3. Create a secret in Kubernetes for the credentials you will use to authenticate to the Keyfactor artifactory. For example:

```
kubectl create secret docker-registry keyregcred --docker-server=keyfactor.jfrog.io
--docker-username=MyUsername --docker-password=MySuperSecretPassword --docker-
email=my.email@my-domain.com
```

4. On your Kubernetes server, create a configmap containing CA root certificates, including the chain certificates for the SSL certificate on the Keyfactor Command server (see [Configure Certificate Root Trust for the Universal Orchestrator on page 15](#)). For example:

```
kubectl create configmap ca-roots --from-file=/etc/ssl/certs/ca-certificates.crt
```



Note: The standard path to the trusted root store will vary depending on your Linux implementation.

5. Create a Kubernetes deployment file (e.g. `uo_ssl.yaml`) in the directory for your Kubernetes container similar to the following, using the inputs as per [Table 1: Linux Container Parameters](#), referencing the artifactory for the image you wish to install, selecting the appropriate authentication mechanism for your environment, and any additional volume mounts. The fields highlighted in red below indicate fields that need to be edited or that you may wish to edit.



Important: When editing the file, be sure to preserve the indenting exactly as found. YAML requires a very specific file layout to function. If the indenting (multiples of two spaces) or layout is incorrect, you will receive an error when trying to install.

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  # Give the pod a unique name if you plan to deploy more than one orchestrator to the same Kuber-
  netes server or cluster
  name: keyfactor-uo-1
  labels:
    app.kubernetes.io/name: keyfactor-universal-orchestrator
    app.kubernetes.io/instance: ssl-1
    app.kubernetes.io/version: "11.0"
spec:
```

```

# The universal orchestrator should not have replicas; instead use many different deployments
with different names if horizontal scaling is needed
replicas: 1
selector:
  matchLabels:
    app.kubernetes.io/name: keyfactor-universal-orchestrator
    app.kubernetes.io/instance: ssl-1
template:
  metadata:
    labels:
      app.kubernetes.io/name: keyfactor-universal-orchestrator
      app.kubernetes.io/instance: ssl-1
  spec:
    initContainers:
      # The below two commented out blocks provide examples of adding custom extensions
      #- env:
      #   - name: EXTENSION_NAME
      #     value: citrix-adc-orchestrator
      #   - name: EXTENSION_VERSION
      #     value: 2.0.0
      #   - name: INSTALL_PATH
      #     value: /app/extensions/citrix-adc-orchestrator
      # image: m8rmclarenkf/uo_extension_installer:1.0.5
      # imagePullPolicy: IfNotPresent
      # name: citrix-adc-orchestrator-installer
      # volumeMounts:
      #   - mountPath: /app/extensions
      #     name: command-pv-claim
      #     readOnly: false
      #     subPath: ""
      #- env:
      #   - name: EXTENSION_NAME
      #     value: f5-rest-orchestrator
      #   - name: EXTENSION_VERSION
      #     value: 1.4.4
      #   - name: INSTALL_PATH
      #     value: /app/extensions/f5-rest-orchestrator
      # image: m8rmclarenkf/uo_extension_installer:1.0.5
      # imagePullPolicy: IfNotPresent
      # name: f5-rest-orchestrator-installer
      # volumeMounts:
      #   - mountPath: /app/extensions
      #     name: command-pv-claim

```

```

#   readOnly: false
#   subPath: ""
containers:
- name: keyfactor-universal-orchestrator-ssl-1
  # Uncomment the desired image
  image: "keyfactor.jfrog.io/con-develop-us-engineering/command/universal-orchestrator-ssl:11.0"
  #image: "keyfactor.jfrog.io/con-develop-us-engineering/command/universal-orchestrator:11.0"
  imagePullPolicy: IfNotPresent
  # Universal orchestrator environment
  env:
    # The below block is the URL of the orchestrator API on the Keyfactor Command server
    - name: COMMAND_AGENTS_URL
      value: https://keyfactor.keyexample.com/KeyfactorAgents
    # The below block is the name the orchestrator will use when registering with
    Keyfactor Command
    - name: ORCHESTRATOR_NAME
      value: k8s-universal-orchestrator-ssl-1
    - name: LOG_LEVEL
      value: Info
    # Uncomment the next two blocks to use Active Directory authentication
    #- name: USERNAME
    #   valueFrom:
    #     secretKeyRef:
    #       name: uo-credentials
    #       key: username
    #- name: PASSWORD
    #   valueFrom:
    #     secretKeyRef:
    #       name: uo-credentials
    #       key: password
    # Uncomment the next four blocks to use OAuth authentication
    #- name: BEARER_TOKEN_URL
    #   value: https://appsrvr18.keyexample.com:1443/realms/Keyfactor/protocol/openid-connect/token
    #- name: TOKEN_LIFETIME
    #   value: "300"
    #- name: CLIENTID
    #   valueFrom:
    #     secretKeyRef:
    #       name: uo-credentials
    #       key: clientid

```

```

    #- name: CLIENT_SECRET
    # valueFrom:
    #   secretKeyRef:
    #     name: uo-credentials
    #     key: clientsecret
  volumeMounts:
    # Uncomment the next block if adding extensions
    #- mountPath: /app/extensions
    #   name: command-pv-claim
    #   readOnly: false
    #   subPath: ""
    - mountPath: /etc/ssl/certs/ca-certificates.crt
      name: root-ca
      readOnly: false
      subPath: ca-certificates.crt
  volumes:
    - configMap:
        items:
          - key: ca-certificates.crt
            path: ca-certificates.crt
        name: ca-roots
      name: root-ca
    # Uncomment the next block if adding extensions
    #- name: command-pv-claim
    #   emptyDir: {}
  imagePullSecrets:
    - name: keyregcred

```

6. Set the permissions on the deployment file such that the file is owned by root and readable only by root (this assumes your Kubernetes implementation is running as root, which is typical). For example:

```
sudo chown root:root uo_ssl.yaml
```

```
sudo chmod 400 uo_ssl.yaml
```



Tip: If you need to make edits to the compose file, you will need to make the file writable again. For example:

```
sudo chmod 600 uo_ssl.yaml
```

7. Execute the following command to install and run the container:

```
kubectl apply -f uo_ssl.yaml
```



Tip: To review logs generated from the container, identify the pod name with this command:

```
kubectl get pods
```

Then use the following command to output the current log:

```
kubectl logs [pod name] --follow
```

The optional follow parameter will continuously output the logs as they are generated until interrupted.

If you need to delete the container and try the install again, use this command:

```
kubectl delete -f uo_ssl.yaml
```

Table 1: Linux Container Parameters

Parameter	Description
AppSettings__Check-ServerCertificateRevocation	A Boolean that indicates whether the revocation status (CRL) of the SSL certificate on the Keyfactor Command server should be checked when connecting to Keyfactor Command (true) or not (false). The default is <i>true</i> (CRL checking will be done).
AUDIENCE	This parameter is used to specify an audience value to be included in token requests delivered to the identity provider when using an identity provider other than Active Directory.
BEARER_TOKEN_URL	<p>Required*. The URL of the token endpoint for your identity provider. For example:</p> <pre>https://my-keyidp-server.keyexample.com/realms/Keyfactor/protocol/openid-connect/token</pre> <p>For Keyfactor Identity Provider, this is included among the information that can be found on the OpenID Endpoint Configuration page, a link to which can be found on the Realm Settings page (see <i>Gathering Keyfactor Identity Provider Data for the Keyfactor Command Installation</i> in the <i>Keyfactor Command Server Installation Guide</i>).</p> <p>This parameter is required if you're using an identity provider other than Active Directory.</p>

Parameter	Description
CLIENTID	<p>Required*. For implementations using an identity provider other than Active Directory, the ID of the identity provider client that should be used to authenticate the session (see Create Service Accounts for the Universal Orchestrator on page 11).</p> <p>This parameter is required if you're using an identity provider other than Active Directory.</p>
CLIENT_SECRET	<p>Required*. For implementations using an identity provider other than Active Directory, the secret of the identity provider client that should be used to authenticate the session.</p> <p>This parameter is required if you're using an identity provider other than Active Directory.</p>
COMMAND_AGENTS_URL	<p>Required. The URL of the Orchestrators API on the Keyfactor Command server. For example:</p> <pre>https://keyfactor.keyexample.com/KeyfactorAgents</pre>
LOG_LEVEL	<p>The logging level for the orchestrator. The default value is <i>Info</i>. Possible values are the same as those described in Configure Logging for the Universal Orchestrator on page 77.</p>
ORCHESTRATOR_NAME	<p>The name the orchestrator uses to register itself with Keyfactor Command. By default, the container hostname is used, which is not ideal as this will create a new orchestrator entry with every container start. Although this parameter is not strictly required, Keyfactor strongly recommends using it.</p> <p>If you choose to uninstall and reinstall the orchestrator (using <code>compose down</code>), it is important to use the same orchestrator name for subsequent implementations so that Keyfactor Command will recognize the orchestrator when it is started again using <code>compose up</code>.</p>
PASSWORD	<p>Required*. The password for the Keyfactor Command Connect Service Account if you're using Active Directory as an identity provider (see <code>USERNAME</code>).</p> <p>This parameter is required if you're using Active Directory as an identity provider.</p>
SCOPE	<p>This parameter is used to specify one or more scopes that should be included in token requests delivered to the identity provider when using an identity provider other than Active Directory. Multiple scopes should be separated by spaces.</p>

Parameter	Description
TOKEN_LIFETIME	<p>For implementations using an identity provider other than Active Directory, the number of seconds for which the bearer token is valid. This should be set to the same value as the Keyfactor Command <i>Cookie Expiration</i>. For example, if the Keyfactor Command <i>Cookie Expiration</i> is 5 minutes, the <i>TOKEN_LIFETIME</i> should be 300 seconds. The default value is 60.</p> <p>The <i>Cookie Expiration</i> value determines the length of time the authentication cookie is considered valid. After half of the setting's duration, Keyfactor Command will attempt to use a refresh token to update the cookie. If this fails, the orchestrator's session will be terminated.</p>
USERNAME	<p>Required*. The username for service account used to connect to the Keyfactor Command server (see PASSWORD). This is the Keyfactor Command Connect Service Account described in Create Service Accounts for the Universal Orchestrator on page 11 if you're using Active Directory as an identity provider. The orchestrator uses Basic Authentication to authenticate to Keyfactor Command.</p> <p>This parameter is required if you're using Active Directory as an identity provider.</p>



Note: The Keyfactor Universal Orchestrator running in a container does not support client certificate authentication.



Tip: Once the installation of the orchestrator is complete, you need to use the Keyfactor CommandManagement Portal to approve the orchestrator and configure certificate stores or SSL jobs as per the *Keyfactor Command Reference Guide*:

- Orchestrator Management Operations: Approving or Disapproving Orchestrators
- Certificate Store Operations
- SSL Discovery

2.2.5 Optional Configuration

Once the installation is complete, the Keyfactor Universal Orchestrator should be running and ready to communicate with the Keyfactor Command server. The initial installation allows the orchestrator to register itself with Keyfactor Command and run jobs of the capability types configured during installation (after being approved in the Keyfactor Command Management Portal) unless you selected the NoService parameter.

This section details some post-install configuration steps that may need to be completed for some capabilities and some optional settings.



Important: Synchronization for the remote CA functionality of the orchestrator will not begin until you complete the configuration by making the appropriate configuration changes in the Keyfactor Command Management Portal. See *Orchestrator Management* in the *Keyfactor Command Reference Guide* for instructions on approving the orchestrator in the Keyfactor Command Management Portal on the *Orchestrators->Management* page and *Certificate Authority Operations: Adding or Modifying a CA Record* in the *Keyfactor Command Reference Guide* for instructions on configuring certificate and template synchronization for remote CAs on the *Locations->Certificate Authorities* page.

2.2.5.1 Configure Windows Targets for Remote Management

This step only needs to be completed if you plan to use one of the custom-built extensions for the Keyfactor Universal Orchestrator to manage certificate stores on Windows machines that relies on PowerShell remoting and WinRM. Keyfactor offers many custom-built extensions for the Universal Orchestrator on GitHub:

<https://keyfactor.github.io/integrations-catalog/content/orchestrator>

Packages that rely on PowerShell remoting and WinRM for store management on Windows include:

- [IIS Certificate Store Manager](#)
- [Remote File Certificate Store Management](#) (Java Keystores, PKCS12 files, PEM files, DER files, IBM Key Database files)

Permissions

On each target machine where you wish to manage certificate stores with the Universal Orchestrator, you need to grant the Active Directory or local service account the orchestrator is using to authenticate to the server sufficient permissions to read the directories where the certificate stores are located (for the Remote File extension) or local machine certificate store (for the IIS extension) and, if you plan to deploy certificates to it using Keyfactor Command and bind certificates to IIS, write to the directories (Remote File) and local machine store (IIS) and appropriate permissions to bind certificates in IIS. For the Remote File extension, granting read/write permissions on the given directories may be sufficient. For IIS, the Universal Orchestrator service account needs to be added to the local administrators group on each target machine.

PowerShell Remoting

The orchestrator uses PowerShell remoting to deliver certificates to targets and bind certificates to IIS web sites. This includes certificates delivered directly from the PFX enrollment option of the Keyfactor Command Management Portal or Keyfactor API to targets. If you wish to use any of these features, you will need to make sure that each target machine on which you want to use one of these features is running at least PowerShell version 3 and that PowerShell remoting has been enabled.

To check the PowerShell version on a given machine, open a PowerShell window, run the following command, and check the output CLRVersion:

```
$PSVersionTable
```

PowerShell version 3 is available for download from Microsoft.

To enable PowerShell remoting:

1. On the target machine, open a PowerShell window using the “Run as administrator” option.
2. On the target machine, run the following command to enable PowerShell remoting:

```
Enable-PSRemoting
```

Respond Yes to all the question prompts (or A for all).

3. On the target machine it may be necessary to run the following command to enable execution of unsigned local PowerShell scripts for some operating systems (e.g. Windows Server 2008 R2):

```
Set-ExecutionPolicy RemoteSigned
```

4. To test the PowerShell remoting, on the Universal Orchestrator server, open a PowerShell window and run the following command (where TARGET_MACHINE is the FQDN of the target machine you wish to manage with the orchestrator):

```
Enter-PSSession -ComputerName TARGET_MACHINE
```

Use the actual hostname of the target machine rather than a DNS alias (either A or CNAME records) when running this test. This is necessary because PowerShell remoting relies on Kerberos authentication, which requires that the target machine has a service principal name (SPN) in the HTTP/ format assigned to the target’s machine account. This will be present by default (as part of the HOST/ format record) as long as the HTTP/ format SPN has not been manually assigned elsewhere. Using an alias gets into complexities of setting up appropriate SPNs and assuring that there are not duplicate SPNs in the environment.

You should be connected to the target machine and be able to execute PowerShell commands on the target machine.

WinRM and Firewall Port Considerations

When you add a certificate store in Keyfactor Command using an extension from Keyfactor’s GitHub that relies on WinRM, you are given the option to choose whether to secure the channel to the target hosting the certificate store with SSL. If you select True, Microsoft Windows Remote Management (WinRM) on the target needs to be running on HTTPS and to have been configured with a certificate for WinRM. If you select False, WinRM on the target needs to be running on HTTP. By default, WinRM HTTP uses port 5985 and WinRM HTTPS uses port 5986. WinRM HTTPS is not enabled out-of-the box.

Make sure that any firewalls between the Universal Orchestrator, Keyfactor Command, and the remote target allow communications over port TCP 5985 or 5986, depending on your SSL selection,

or the alternate port you've configured for WinRM on the target if you're not using the default WinRM port(s).

You can use the Test-WSMan and Test-netConnection PowerShell cmdlets on the Universal Orchestrator to validate that communication can occur between the Universal Orchestrator and the remote target in the manner you are intending to configure it (SSL or not SSL). For example, for SSL using the default port (where `webservr38.keyexample.com` is your remote target):

```
Test-netConnection -ComputerName "webservr38.keyexample.com" port 5986
```

Output from this command should look something like this if the connection completes successfully:

```
ComputerName      : webservr38.keyexample.com
RemoteAddress     : 192.168.216.38
RemotePort        : 5986
InterfaceAlias    : Ethernet0
SourceAddress     : 192.168.216.42
TcpTestSucceeded  : True
```

And:

```
Test-WSMan -ComputerName webservr38.keyexample.com -UseSSL
```

Output from this command should look something like this if the connection completes successfully:

```
wsmid             : http://schemas.dmtf.org/wbem/wsman/identity/1/wsmanidentity.xsd
ProtocolVersion   : http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd
ProductVendor     : Microsoft Corporation
ProductVersion    : OS: 0.0.0 SP: 0.0 Stack: 3.0
```

On the remote target, you can use the following WinRM command to check the configuration of WinRM, whether it has been configured to support HTTPS, whether it has a certificate configured for HTTPS, and the ports in use:

```
winrm enumerate winrm/config/listener
```

Output from this command should look something like this if both HTTP and HTTPS are configured for WinRM (notice the port for HTTPS and the certificate thumbprint indicating a certificate has been configured for WinRM on HTTPS):

```
Listener
Address = *
```

```

Transport = HTTP
Port = 5985
Hostname
Enabled = true
URLPrefix = wsman
CertificateThumbprint
ListeningOn = 192.168.216.42, 127.0.0.1, ::1, fe80::21e1:ab7e:9c35:5550%3

Listener
Address = *
Transport = HTTPS
Port = 5986
Hostname = webservr42.keyexample.com
Enabled = true
URLPrefix = wsman
CertificateThumbprint = 79ee047d673da83cea87ba779761b0ec2b9217f8
ListeningOn = 192.168.216.42, 127.0.0.1, ::1, fe80::21e1:ab7e:9c35:5550%3

```

For troubleshooting help, see [Remote Management Helpful Tools on page 147](#). For more information about configuring WinRM for HTTPS, see:

<https://learn.microsoft.com/en-us/troubleshoot/windows-client/system-management-components/configure-winrm-for-https>

2.2.5.2 Configure the Universal Orchestrator for Remote CA Management

If you've opted to enable the remote CA management functionality for the Keyfactor Universal Orchestrator, further configuration is needed on the orchestrator to configure the CA(s) that the orchestrator will manage.

To configure CAs for the orchestrator:

1. On the orchestrator, open a text editor (e.g. Notepad) using the "Run as administrator" option.
2. In the text editor, browse to open the *extensionoptions.json* file for the Universal Orchestrator. The file is located in the configuration directory within the install directory, which is the following directory by default:

```
C:\Program Files\Keyfactor\Keyfactor Orchestrator\configuration
```

3. In the *extensionoptions.json* file, locate the CertificateAuthority section.

```


{
  "CertificateAuthority": {
    "BatchSize": 10000,
    "CacheHours": 3,
    "RecordCountLimit": 5000,
    "MaxErrorCount": 5,
    "AdditionalCertificateAuthoritiesAllowed": false,
    "CertificateAuthorities": [
      {
        "Forest": "keyother.com",
        "Hostname": "corpca01.keyother.com",
        "LogicalName": "KeyIssuing01"
      },
      {
        "Forest": "keyother.com",
        "Hostname": "corpca02.keyother.com",
        "LogicalName": "KeyIssuing02"
      }
    ]
  },
}

```

Figure 9: CA Configuration Settings

4. Either set the *AdditionalCertificateAuthoritiesAllowed* value to **true** or populate the *CertificateAuthorities* section with your CA information (see [Table 2: Remote CA Configuration Parameters](#)).
5. Save the file.
6. Restart the orchestrator service (see [Start the Universal Orchestrator Service on page 80](#)).

Table 2: Remote CA Configuration Parameters

Parameter	Description
BatchSize	<p>An integer that specifies the number of certificate cache records to read from the Keyfactor Command in each data retrieval batch. The default is 10,000.</p> <div>  Tip: Certificate cache information from Keyfactor Command is retrieved from and stored on the orchestrator to allow the orchestrator to calculate which records represent changes and return only those to Keyfactor Command on requests from Keyfactor Command for CA synchronization. </div>
CacheHours	An integer that specifies the number of hours for which to cache certificate information from Keyfactor Command on the orchestrator before clearing it. The default is 3.
RecordCountLimit	An integer that specifies the number of records to read from the CA(s) in each synchronization batch. The default is 5,000.
MaxErrorCount	An integer that specifies the number of times an attempt

Parameter	Description								
	should be made to read records from the CA before the synchronization job ends with a failure. The default is 5.								
AdditionalCertificateAuthoritiesAllowed	A Boolean that sets whether any CAs available to the orchestrator (to which the orchestrator has network access and sufficient permissions) should be considered as managed (<i>True</i>) or whether only those CAs specifically listed in the <i>CertificateAuthorities</i> parameter should be considered as managed (<i>False</i>). If you set this value to <i>True</i> , you do not need to populate the <i>CertificateAuthorities</i> value.								
CertificateAuthorities	<p>An array of the certificate authorities that should be considered managed by the orchestrator. The certificate authority information includes:</p> <table> <tr> <th>Parameter</th><th>Description</th></tr> <tr> <td>Forest</td><td>The name of the Active Directory forest in which the CA resides.</td></tr> <tr> <td>Hostname</td><td>The fully qualified domain name of the CA.</td></tr> <tr> <td>LogicalName</td><td>The logical name of the CA.</td></tr> </table>	Parameter	Description	Forest	The name of the Active Directory forest in which the CA resides.	Hostname	The fully qualified domain name of the CA.	LogicalName	The logical name of the CA.
Parameter	Description								
Forest	The name of the Active Directory forest in which the CA resides.								
Hostname	The fully qualified domain name of the CA.								
LogicalName	The logical name of the CA.								

2.2.5.3 Installing Custom-Built Extensions

Keyfactor offers many custom-built extensions for the Keyfactor Universal Orchestrator on GitHub:

<https://keyfactor.github.io/integrations-catalog/content/orchestrator>

Some packages that may be of special interest to long-term users of Keyfactor Command are:

- [AWS Certificate Store Manager](#)
- [Citrix NetScaler Certificate Store Manager](#)
- [F5 Certificate Store Manager](#)
- [IIS Certificate Store Manager](#)
- [Remote File Certificate Store Management](#) (Java Keystores, PKCS12 files, PEM files, DER files, IBM Key Database files)



Tip: Unlike the Keyfactor Java Agent, which must be installed directly on each machine holding Java keystores to be managed, the Keyfactor Universal Orchestrator with the Remote File extension is a centralized orchestrator, which is installed on just a single machine (or handful of machines) and then reaches out via remote management to each machine holding Java Keystores to be managed. For Java keystores on Windows servers, it uses PowerShell remoting and WinRM, typically over HTTPS, for this (see [Configure Windows Targets for Remote Management on page 62](#)). For Java keystores on Linux servers, it uses either SCP or SFTP (configurable on a per-orchestrator basis and can be configured to try both). Large-scale deployment of Java keystore management (or any of the other formats supported by this extension) involves enabling PowerShell remoting and WinRM with HTTPS on Windows targets (and a local login user if the servers aren't domain joined) or enabling SCP or SFTP and creating a login user for the orchestrator on Linux targets.



Note: For information about installing PAM extensions for the Universal Orchestrator, see *Installing Custom PAM Provider Extensions* in the *Keyfactor Command Reference Guide*.

To find a package on GitHub:

1. Visit one of the links above to find your desired package, and click either **Github Repository** or **View source on GitHub** to go to the package page on GitHub.

IIS Orchestrator

The IIS Orchestrator treats the certificates bound (actively in use) on a Microsoft Internet Information Server (IIS) as a Keyfactor certificate store. Inventory and Management functions are supported. The orchestrator replaces the IIS orchestrator that ships with Keyfactor Command (which did not support binding.)

[Github Repository](#)

Figure 10: View Packages as Part of a List

Remote File

Universal Orchestrator

The Remote File Orchestrator allows for the remote management of file-based certificate stores. Discovery, Inventory, and Management functions are supported. The orchestrator performs operations by first converting the certificate store into a BouncyCastle PKCS12Store.

[View source on GitHub](#)



Figure 11: View Packages on Individual Pages

2. On the GitHub page, on the right-hand side, click the link for the **Latest** version.

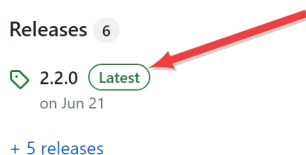


Figure 12: Find the Latest Version of the Package

3. On the GitHub version page in the Assets section, click the package name to download the zip file.

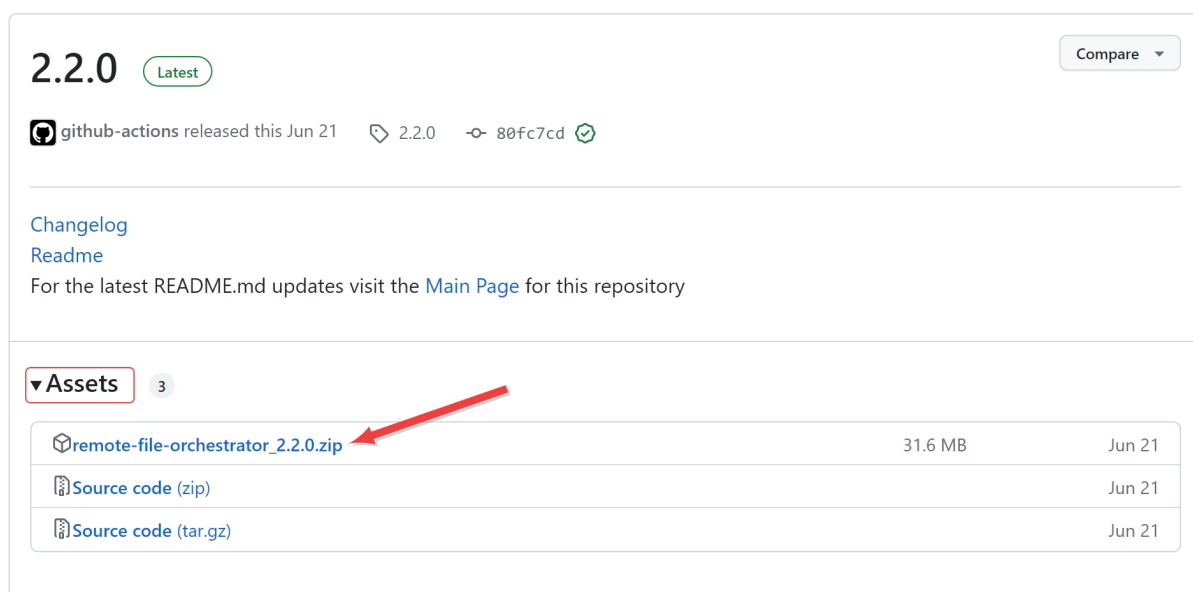


Figure 13: Download the Package Zip File

4. On the main extension GitHub page, review the documentation for the specific extension. Here you will find supported platforms, prerequisites, and extension-specific installation and configuration instructions. The below instructions only cover where to place the extension files on the orchestrator and building custom manifest.json files (changes to which aren't needed for extensions from GitHub unless you are customizing something), but not the details for creation of custom certificate store types for the extension or any other customization specific to a given extension.

Custom-built extensions can also be generated by end users using the Universal Orchestrator NuGet package. Custom-built extensions for certificate store jobs and custom jobs are both installed in the same way.

Once you have your custom-built extension ready, install it as follows:

1. In the Keyfactor Command Management Portal or using the Keyfactor API, add a certificate store type or custom job type for your custom-built extension, if applicable. See *Adding or*

Editing a Certificate Store Type in the Keyfactor Command Reference Guide or POST Custom Job Types in the Keyfactor API Reference Guide.

2. On the Universal Orchestrator server, locate the extensions directory within the install directory. By default, this is:

Windows: C:\Program Files\Keyfactor\Keyfactor Orchestrator\extensions

Linux: /opt/keyfactor/orchestrator/extensions

3. Under the extensions directory, create a new directory with an appropriate name for your custom-built extension (e.g. MyExtension). This name is for reference only and does not need to match any names used elsewhere.
4. Place the DLL(s) created for your custom-built extension along with any other supporting files needed for the extension in the new directory.
5. In the directory for your custom-built extension, create a file called manifest.json if one has not been provided with the extension. The manifest.json file must be placed in the same directory as the DLL(s) for your extension.
6. Using a text editor, edit the manifest.json file if needed and configure it appropriately for your application.



Tip: This step is generally not needed for extensions downloaded from GitHub unless you have opted to make customizations or not use the suggested short name when creating the certificate store type.

Some things to keep in mind are:

- The opening and closing lines of the file must match those shown in red here:

```
{
  "extensions": {
    "Keyfactor.Orchestrators.Extensions.IOrchestratorJobExtension":
    {
      "Custom.MyJob": {
        "assemblypath": "Keyfactor.Orchestrators.MyJob.dll",
        "TypeFullName": "Keyfactor.Orchestrators.MyJob.MyJobExtension"
      }
    }
  }
}
```

- Each customized section of the file starts with either a custom job reference (e.g. Custom.MyJob) or a certificate store reference (e.g. CertStores.MyStore.Inventory).

Custom jobs (beginning *Custom*) correspond to custom job types created with the Keyfactor API *POST /JobTypes/Custom* method. For example, a custom job type with a *JobTypeName* of *MyJob* would appear in the file as *Custom.MyJob*.

Certificate store jobs (beginning *CertStores*) correspond to certificate store types created with the Keyfactor API *POST /CertificateStoreType* method (see *POST Certificate Store Types* in the *Keyfactor API Reference Guide*) or in the Keyfactor Command Management Portal (see *Adding or Editing a Certificate Store Type* in the *Keyfactor Command Reference Guide*). For example, a certificate store type with a *Capability* of *MyStore* configured to do inventory, management and discovery, would have three separate sections in the file as *CertStores.MyStore.Inventory*, *CertStores.MyStore.Management*, and *CertStores.MyStore.Discovery*. An inventory section is required.

- The *assemblypath* referenced in each section points to the DLL in the extensions directory that corresponds to that job function. A single manifest file may include many different capabilities if the extension performs more than one type of job (e.g. inventory and management of certificates), such as is shown in the below example.
- The *TypeFullName* referenced in each section corresponds to the name of the type that resides inside of the DLL listed for the assembly path. A single manifest file may include many different capabilities if the extension performs more than one type of job (e.g. inventory and management of certificates), such as is shown in the below example.
- Each section may optionally have a *PreScript* reference, which points to a script file on the orchestrator machine that will run before the main job for the section executes.
 - For orchestrators installed on Windows, these will be PowerShell scripts. No special configuration is needed other than entry of a path to the PowerShell script in the *PreScript* field. The script may be placed anywhere on the orchestrator machine. The orchestrator will need read permissions to the script.
 - For orchestrators installed on Linux, these will be Bash scripts. In order to use a Bash script with the orchestrator, you must first register the Bash script driver in the *appsettings.json* file. This file is found in the *configuration* directory. Edit the file and add the following below the existing *AppSettings* configuration section in the file (before the final closing bracket):

```
"extensions": {
  "Keyfactor.Orchestrators.ScriptDrivers.IScriptDriver": {
    "RegisteredScriptDriver": {
      "assemblypath": "Keyfactor.Orchestrators.BashDriver.dll",
      "TypeFullName": "Keyfactor.Orchestrators.ScriptDrivers.BashDriver"
    }
  }
}
```

After the Bash script driver is registered, you may enter a path to the Bash script in the orchestrator manifest.json file *PreScript* section. The script may be placed

anywhere on the orchestrator machine. The orchestrator will need read permissions to the script.



Tip: If your script fails, this will cause the entire job to fail. You can use this to your advantage if you'd like to fail the job under certain conditions by doing a *Write-Error* on Windows or *exit <error code>* on Linux.

For more information about calling scripts from the orchestrator, contact your Keyfactor representative.



Note: The prescript and postscript functionality of the Keyfactor Universal Orchestrator has been replaced by other functionality in Keyfactor Command such as that provided by Keyfactor Command workflows (see *Workflow Definitions* in *Keyfactor Command Reference Guide*). As a result, prescript and postscript functionality has been deprecated and will be removed from a future release.

- Each section may optionally have a *PostScript* reference, which points to a script file on the orchestrator machine that will run after the main job for the section executes. See the notes for script use under *PreScript*.
- User-defined certificate store jobs support up to four job types—Inventory, Management, Discovery, and Reenrollment. Each one of these job types should have a separate section in the file.

```
{
  "extensions": {
    "Keyfactor.Orchestrators.Extensions.IOrchestratorJobExtension": {
      "CertStores.MyStore.Inventory": {
        "assemblypath": "Keyfactor.Orchestrators.MyStore.dll",
        "TypeFullName": "Keyfactor.Orchestrators.MyStore.MyStoreInventoryJobExtension"
      },
      "CertStores.MyStore.Management": {
        "assemblypath": "Keyfactor.Orchestrators.MyStore.dll",
        "TypeFullName": "Keyfactor.Orches-
trators.MyStore.MyStoreManagementJobExtension",
        "PreScript": "C:\\Program Files\\Keyfactor\\Keyfactor
Orchestrator\\extensions\\MyStoreManagementPreScript.ps1",
        "PostScript": "C:\\Program Files\\Keyfactor\\Keyfactor Orches-
trator\\extensions\\MyStoreManagementPostScript.ps1"
      },
      "CertStores.MyStore.Discovery": {
        "assemblypath": "Keyfactor.Orchestrators.MyStore.dll",
        "TypeFullName": "Keyfactor.Orchestrators.MyStore.MyStoreDiscoveryJobExtension"
      }
    }
  }
}
```

```
}  
}  
}
```

7. Restart the Universal Orchestrator service (see [Start the Universal Orchestrator Service on page 80](#)).
8. In the Keyfactor Command Management Portal, re-approve the orchestrator. The orchestrator will update to a status of new (if it had been approved previously) upon receiving updated capabilities. See *Orchestrator Management* in the *Keyfactor Command Reference Guide* for information on approving orchestrators.

Contact your Keyfactor representative for more information about custom-built solutions or to obtain access to the NuGet packages required for development of Universal Orchestrator extensions.

2.2.5.4 Configuring Script-Based Certificate Store Jobs

The Keyfactor Universal Orchestrator supports the option to implement custom-built certificate store jobs using one or more scripts (PowerShell or Bash) rather than a full extension (see [Installing Custom-Built Extensions on page 67](#)). To implement custom-built certificate store jobs in this way, you need to create your scripts that will execute the certificate store actions (e.g. inventory, add certificates, remove certificates) and a manifest.json file to reference the jobs and install them on the orchestrator. Optionally, each certificate store action script can call a prescript and/or a postscript to perform actions before or after the main action.



Note: The scripting method of running custom-built certificate store jobs cannot be used to run other types of custom jobs. These are supported only with the use of a custom extension (see [Installing Custom-Built Extensions on page 67](#)). However, both certificate store jobs and custom jobs support the use of prescripts and postscripts (see [Orchestrator Job Overview on page 4](#)).

To configure a set of custom-built certificate store scripts:

1. On the Universal Orchestrator server, locate the scripts directory within the install directory. By default, this is:

Windows: C:\Program Files\Keyfactor\Keyfactor Orchestrator\Scripts

Linux: /opt/keyfactor/orchestrator/Scripts

2. Under the scripts directory, create a new directory with an appropriate name for your custom-built certificate store job set (e.g. MyStore). This name matches the name of the job referenced in the manifest.json file.

3. Place the scripts created for your custom-built certificate store job set in the new directory. Supported script file names are:
 - Add (e.g. Add.ps1 or Add.sh)
A management job to add a certificate to the certificate store.
 - Create (e.g. Create.ps1 or Create.sh)
A management job to create the certificate store if it does not already exist.
 - Discovery (e.g. Discovery.ps1 or Discovery.sh)
A discovery job.
 - Inventory (e.g. Inventory.ps1 or Inventory.sh)
An inventory job.
 - Reenrollment (e.g. Reenrollment.ps1 or Reenrollment.sh)
A reenrollment job.
 - Remove (e.g. Remove.ps1 or Remove.sh)
A management job to remove a certificate from the certificate store.
4. In order to use a Bash script with orchestrators installed on Linux, you must first register the Bash script driver in the appsettings.json file. This file is found in the configuration directory. Edit the file and add the following below the existing AppSettings configuration section in the file (before the final closing bracket):

```
"extensions": {  
  "Keyfactor.Orchestrators.ScriptDrivers.IScriptDriver": {  
    "RegisteredScriptDriver": {  
      "assemblypath": "Keyfactor.Orchestrators.BashDriver.dll",  
      "TypeFullName": "Keyfactor.Orchestrators.ScriptDrivers.BashDriver"  
    }  
  }  
}
```

5. On the Universal Orchestrator server, locate the JobExtensionDrivers directory within the extensions directory under the install directory. By default, this is:

Windows: C:\Program Files\Keyfactor\Keyfactor Orchestrator\extensions\JobExtensionDrivers

Linux: /opt/keyfactor/orchestrator/extensions/JobExtensionDrivers
6. In the JobExtensionDrivers directory, create a file called manifest.json or open the existing one. There should be only one manifest.json file no matter how many script directories you create.
7. Using a text editor, edit the manifest.json file and configure it appropriately for your custom-built certificate store job set. Some things to keep in mind are:

- The opening and closing lines of the file must match those shown in red here:

```
{
  "extensions": {
    "Keyfactor.Orchestrators.Extensions.IOrchestratorJobExtension":
  {
    "CertStores.MyStore.Inventory": {
      "assemblypath": "Keyfactor.Orchestrators.JobExtensionDrivers.dll",
      "TypeFullName": "Keyfactor.Orches-
trators.JobExtensionDrivers.InventoryJobExtensionDriver"
    }
  }
}
```

- Each customized section of the file starts with a certificate store reference (e.g. `CertStores.MyStore.Inventory`). Certificate stores jobs (beginning *CertStores*) correspond to certificate store types created with the Keyfactor API *POST /CertificateStoreType* method (see *POST Certificate Store Types* in the *Keyfactor API Reference Guide*) or in the Keyfactor Command Management Portal (see *Certificate Store Type Operations: Adding or Editing a Certificate Store Type* in the *Keyfactor Command Reference Guide*). For example, a custom certificate store type with a *Capability* of *MyStore* configured to do inventory, management and discovery, would have three separate sections in the file as *CertStores.MyStore.Inventory*, *CertStores.MyStore.Management*, and *CertStores.MyStore.Discovery*. The capability reference (e.g. *MyStore*) must also match the name you give to the directory where you place your scripts. An inventory section is required.
- The *assemblypath* referenced in each section points to the DLL in the extensions directory of the Job Extensions Driver extension. This built-in extension is used to run custom-built certificate store jobs as scripts. This value will be the same for all entries in the file.
- The *TypeFullName* referenced in each section corresponds to the name of the type that resides inside of the DLL listed for the assembly path—the Job Extensions Driver extension in this case. This value will be the same for all entries in the file.
- Each section may optionally have a *PreScript* reference, which points to an additional script file on the orchestrator machine that will run before the main job for the section executes.



Tip: If either your PreScript or PostScript fails, this will cause the entire job to fail. You can use this to your advantage if you'd like to fail the job under certain conditions by doing a *Write-Error* on Windows or *exit <error code>* on Linux.



Note: The prescript and postscript functionality of the Keyfactor Universal Orchestrator has been replaced by other functionality in Keyfactor Command such as that provided by Keyfactor Command workflows (see *Workflow Definitions* in *Keyfactor Command Reference Guide*). As a result, prescript and postscript functionality has been deprecated and will be removed from a future release.

- Each section may optionally have a *PostScript* reference, which points to an additional script file on the orchestrator machine that will run after the main job for the section executes.
- Custom-built certificate store jobs support up to four job types—Inventory, Management, Discovery, and Reenrollment. Each one of these job types should have a separate section in the file.

```
{
  "extensions": {
    "Keyfactor.Orchestrators.Extensions.IOrchestratorJobExtension": {
      "CertStores.MyStore.Inventory": {
        "assemblypath": "Keyfactor.Orchestrators.JobExtensionDrivers.dll",
        "TypeFullName": "Keyfactor.Orches-
trators.JobExtensionDrivers.InventoryJobExtensionDriver"
      },
      "CertStores.MyStore.Management": {
        "assemblypath": "Keyfactor.Orchestrators.JobExtensionDrivers.dll",
        "TypeFullName": "Keyfactor.Orches-
trators.JobExtensionDrivers.InventoryJobExtensionDriver"
        "PreScript": "C:\\Program Files\\Keyfactor\\Keyfactor
Orchestrator\\scripts\\MyStore\\MyStoreManagementPreScript.ps1",
        "PostScript": "C:\\Program Files\\Keyfactor\\Keyfactor Orches-
trator\\scripts\\MyStore\\MyStoreManagementPostScript.ps1"
      },
      "CertStores.MyStore.Discovery": {
        "assemblypath": "Keyfactor.Orchestrators.JobExtensionDrivers.dll",
        "TypeFullName": "Keyfactor.Orches-
trators.JobExtensionDrivers.InventoryJobExtensionDriver"
      },
      "CertStores.MyStore.Reenrollment": {
        "assemblypath": "Keyfactor.Orchestrators.JobExtensionDrivers.dll",
        "TypeFullName": "Keyfactor.Orches-
trators.JobExtensionDrivers.InventoryJobExtensionDriver"
      }
    }
  }
}
```

8. Restart the Universal Orchestrator service (see [Start the Universal Orchestrator Service on page 80](#)).
9. In the Keyfactor Command Management Portal, re-approve the orchestrator. The orchestrator will update to a status of new (if it had been approved previously) upon receiving updated capabilities. See *Orchestrator Management* in the *Keyfactor Command Reference Guide* for information on approving orchestrators.

Contact your Keyfactor representative for more information about custom solutions or for assistance creating custom scripts.

2.2.5.5 Configure Logging for the Universal Orchestrator

Keyfactor Universal Orchestrator provides extensive logging for visibility and troubleshooting. For more information about troubleshooting, see [Troubleshooting on page 130](#).

By default, the Keyfactor Universal Orchestrator places its log files in the logs directory under the installed directory, generates logs at the INFO logging level and stores logs for two days before deleting them. If you wish to change these defaults, follow the directions below for your installation type.

Windows Installations

1. On the Windows server where you wish to adjust logging, open a text editor (e.g. Notepad) using the “Run as administrator” option.
2. In the text editor, browse to open the Nlog.config file for the Universal Orchestrator. The file is located in the configuration directory within the install directory, which is the following directory by default:

C:\Program Files\Keyfactor\Keyfactor Orchestrator\configuration

3. Your Nlog.config file may have a slightly different layout than shown here, but it will contain the five fields highlighted in [Figure 14: Universal Orchestrator on Windows NLog.config File](#). The fields you may wish to edit are:

- `variable name="logDirectory" value="logs/"`

The path to the log file location.



Important: If you choose to change the path for storage of the log files, you will need to create the new directory (e.g. D:\KeyfactorLogs) and grant the Universal Orchestrator service account under which the Keyfactor Orchestrator Service is running full control permissions on this directory.

- `fileName="${logDirectory}/Log.txt"`

The path and file name of the active orchestrator log file, referencing the logDirectory variable.

- `archiveFileName="${logDirectory}/Log_Archive_{#}.txt"`

The path and file name of previous days' orchestrator log files, referencing the logDirectory variable. The orchestrator rotates log files daily and names the previous files using this naming convention.

- `maxArchiveFiles="2"`

The number of archive files to retain before deletion.

- `name="*" minlevel="Info" writeTo="logfile"`

The level of log detail that should be generated and output to the log file. The default INFO level logs error and some informational data but at a minimal level to avoid generating large log files. For troubleshooting, it may be desirable to set the logging level to DEBUG or TRACE. Available log levels (in order of increasing verbosity) are:

- OFF—No logging
- FATAL—Log severe errors that cause early termination
- ERROR—Log severe errors and other runtime errors or unexpected conditions that may not cause early termination
- WARN—Log errors and use of deprecated APIs, poor use of APIs, “almost” errors, and other runtime situations that are undesirable or unexpected but not necessarily “wrong”
- INFO—Log all of the above plus runtime events (startup/shutdown)
- DEBUG—Log all of the above plus detailed information on the flow through the system
- TRACE—Maximum log information—this option can generate VERY large log files


```

<variable name="logDirectory" value="logs"/>
<targets>
  <target name="buffered_wrapper" xsi:type="BufferingWrapper" slidingTimeout="true" bufferSize="500" flushTimeout="500">
    <target xsi:type="File" name="logfile" fileName="${logDirectory}/Log.txt" layout="${longdate} ${logger} [{level}] - ${message}"
      archiveFileName="${logDirectory}/Log_Archive_{#}.txt" archiveEvery="Day" archiveNumbering="Rolling" maxArchiveFiles="2" archiveAboveSize="2147483648"/>
    </target>
    <target xsi:type="OutputDebugString" name="String" layout="${longdate} ${logger}::${message}"/>
    <target xsi:type="Debugger" name="debugger" layout="${longdate} ${logger}::${message}"/>
    <target xsi:type="Console" name="console" layout="${logger} ${message}"/>
    <target xsi:type="EventLog" name="eventLog" source="Keyfactor Orchestrator"
      eventId="${event-properties:item=eventID}" category="${event-properties:item=categoryID}" layout="${event-properties:item=message}" />
  </targets>
  <rules>
    <logger name="*-EVENT" minlevel="Info" writeTo="eventLog" final="true" />
    <logger name="*" minlevel="Info" writeTo="console" />
    <logger name="*" minlevel="Info" writeTo="logfile">
      <filters>
        <when condition="contains('${logger}', 'Quartz') and level <= LogLevel.Warn" action="IgnoreFinal" />
        <when condition="starts-with('${logger}', 'Microsoft.Hosting.Lifetime') and level >= LogLevel.Info" action="LogFinal" />
        <when condition="starts-with('${logger}', 'Microsoft.Azure.SignalR') and level >= LogLevel.Debug" action="LogFinal" />
        <when condition="starts-with('${logger}', 'Microsoft.AspNetCore') action="Ignore" />
        <when condition="starts-with('${logger}', 'Microsoft') and level <= LogLevel.Warn" action="Ignore" />
      </filters>
    </logger>
  </rules>

```

Figure 14: Universal Orchestrator on Windows NLog.config File

Linux Installations

1. On the orchestrator machine where you wish to adjust logging, open a command shell and change to the directory in which the orchestrator is installed. By default this is /opt/keyfactor/orchestrator.
2. In the command shell in the directory in which the orchestrator is installed, change to the configuration directory.
3. Using a text editor, open the nlog.config file in the configuration directory. Your nlog.config file may have a slightly different layout than shown here, but it will contain the five fields highlighted in the below figure. The fields you may wish to edit are:

- `variable name="logDirectory" value="logs/"`

The path to the log file location.



Important: If you choose to change the path for storage of the log files, you will need to create the new directory (e.g. /opt/kyflogs) and grant the Universal Orchestrator service account under which the keyfactororchestrator-default service is running full control permissions on this directory.

- `fileName="${logDirectory}/Log.txt"`

The path and file name of the active orchestrator log file, referencing the logDirectory variable.

- `archiveFileName="${logDirectory}/Log_Archive_{#}.txt"`

The path and file name of previous days' orchestrator log files, referencing the logDirectory variable. The orchestrator rotates log files daily and names the previous files using this naming convention.

- `maxArchiveFiles="2"`

The number of archive files to retain before deletion.

- `name="*" minlevel="Info" writeTo="logfile"`

The level of log detail that should be generated and output to the log file. The default INFO level logs error and some informational data but at a minimal level to avoid generating large log files. For troubleshooting, it may be desirable to set the logging level to DEBUG or TRACE. Available log levels (in order of increasing verbosity) are:

- OFF—No logging
- FATAL—Log severe errors that cause early termination
- ERROR—Log severe errors and other runtime errors or unexpected conditions that may not cause early termination
- WARN—Log errors and use of deprecated APIs, poor use of APIs, “almost” errors, and other runtime situations that are undesirable or unexpected but not necessarily “wrong”
- INFO—Log all of the above plus runtime events (startup/shutdown)
- DEBUG—Log all of the above plus detailed information on the flow through the system
- TRACE—Maximum log information—this option can generate VERY large log files

```
<variable name="logDirectory" value="logs"/>
<targets>
  <target name="buffered_wrapper" xsi:type="BufferingWrapper" slidingTimeout="true" bufferSize="500" flushTimeout="500">
    <target xsi:type="File" name="logfile" fileName="${logDirectory}/Log.txt" layout="${longdate} ${logger} [{level}] - ${message}"
      archiveFileName="${logDirectory}/Log_Archive_{#}.txt" archiveEvery="Day" archiveNumbering="Rolling" maxArchiveFiles="2" archiveAboveSize="2147483648"/>
    </target>
  </target>
  <target xsi:type="OutputDebugString" name="String" layout="${longdate} ${logger}::${message}"/>
  <target xsi:type="Debugger" name="debugger" layout="${longdate} ${logger}::${message}"/>
  <target xsi:type="Console" name="console" layout="${logger} {message}"/>
  <target xsi:type="EventLog" name="eventLog" source="Keyfactor Orchestrator"
    eventId="{event-properties:item=eventID}" category="{event-properties:item=categoryID}" layout="{event-properties:item=message}" />
</targets>
<rules>
  <logger name="*-EVENT" minlevel="Info" writeTo="eventLog" final="true" />
  <logger name="*" minlevel="Info" writeTo="console" />
  <logger name="*" minlevel="Info" writeTo="logfile" />
  <filters>
    <when condition="contains('${logger}', 'Quartz') and level <= LogLevel.Warn" action="IgnoreFinal" />
    <when condition="starts-with('${logger}', 'Microsoft.Hosting.Lifetime') and level >= LogLevel.Info" action="LogFinal" />
    <when condition="starts-with('${logger}', 'Microsoft.Azure.SignalR') and level >= LogLevel.Debug" action="LogFinal" />
    <when condition="starts-with('${logger}', 'Microsoft.AspNetCore') and level <= LogLevel.Warn" action="Ignore" />
    <when condition="starts-with('${logger}', 'Microsoft') and level <= LogLevel.Warn" action="Ignore" />
  </filters>
</rules>
```

Figure 15: Universal Orchestrator on Linux NLog.config File

2.2.5.6 Start the Universal Orchestrator Service

The Keyfactor Universal Orchestrator service runs on the orchestrator server and controls orchestrator communications with the Keyfactor Command server. During the configuration process you set the service account under which the orchestrator service will run. The service should start automatically at the conclusion of the installation. To check to see if it's running and start it if necessary, follow the directions below for your installation type.

Windows Installations

The service on Windows is added with a display name of Keyfactor Orchestrator Service (Default) by default.

1. On the Universal Orchestrator server, open the Services MMC.
2. In the Services MMC confirm that the Keyfactor Orchestrator Service is set to a Startup Type of Automatic (if desired). If the service is not running, click the green arrow to start it.

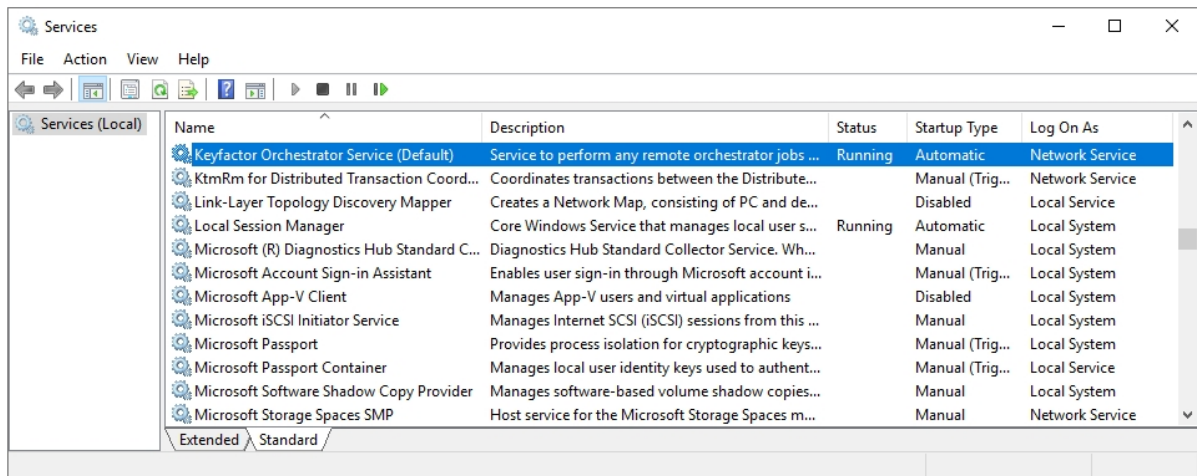


Figure 16: Universal Orchestrator Service



Note: Your service will have a name other than (*Default*) following *Keyfactor Orchestrator Service* if you opted to use the *ServiceSuffix* installation parameter.

Linux Installations

The service on Linux is added as `keyfactor-orchestrator-default` by default, so when referencing it in startup commands, it should be referenced by this name, including case. For example:

```
systemctl start [stop] [restart] [status] keyfactor-orchestrator-default.service
```



Note: Your service will have a name other than *default* following *keyfactor-orchestrator-* if you opted to use the *service-suffix* installation parameter.

2.2.5.7 Change Service Account Passwords

The process for changing the passwords for the service accounts used by the Keyfactor Universal Orchestrator varies for the different service accounts (see [Create Service Accounts for the Universal Orchestrator on page 11](#)) and based on the type of authentication used for the service account used to connect to Keyfactor Command.



Important: Keyfactor highly recommends that you use strong passwords for any accounts or certificates related to Keyfactor Command and associated products, especially when these have elevated or administrative access. A strong password has at least 12 characters (more is better) and multiple character classes (lowercase letters, uppercase letters, numeral, and symbols). Ideally, each password would be randomly generated. Avoid password re-use.

Universal Orchestrator Service Account

The password for the service account that's used to run the Universal Orchestrator service on the orchestrator server can be changed through standard operating system methods.

On a Linux server, this would be, for example, the command line `passwd` command executed for the service account running the orchestrator service (by default `keyfactor-orchestrator`). So, this command on a Linux server might be:

```
sudo passwd keyfactor-orchestrator
```

On a Windows server, if you've opted to run the Universal Orchestrator service as a custom service account rather than *Network Service*, the password would need to be changed in Active Directory or the local user store and in the Services MMC.

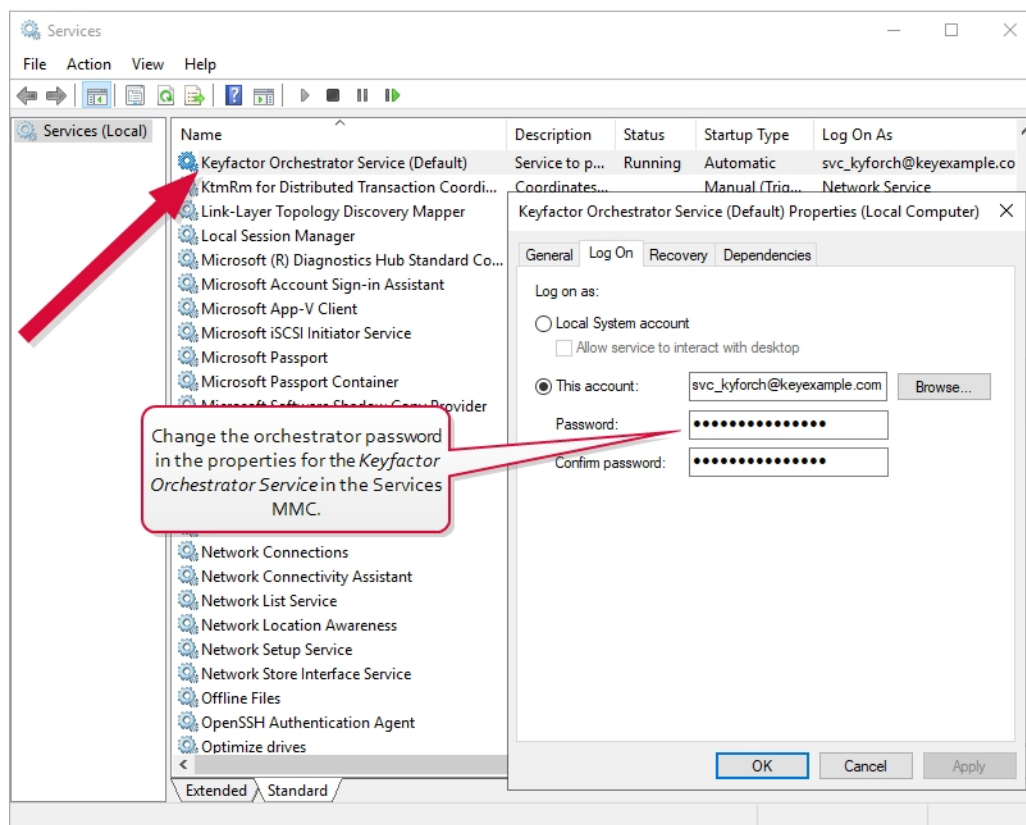


Figure 17: Change Service Account Password in Services MMC

Keyfactor Command Connect Service Account with Basic Authentication

For both Windows and Linux servers, the password change for the service account that's used to make the connection to Keyfactor Command when Basic authentication is used follows this process:

1. Change the password for the service account in Active Directory.
2. On the Windows or Linux server, open a command window. For Windows, this should be a PowerShell window open using the "Run as Administrator" option. Change to the directory in which the orchestrator is installed and locate the change_secrets script. By default, this is:

```
Windows: C:\Program Files\Keyfactor\Keyfactor Orchestrator\change_secrets.ps1
Linux: /opt/keyfactor/orchestrator/change_secrets.sh
```

3. For Linux only, use the chmod command to make the change_secrets.sh script file executable. The file ships in a non-executable state to avoid accidental execution. For example:

```
sudo chmod +x change_secrets.sh
```

4. For Windows only, in the PowerShell window, run the following command to populate a variable with the password for the service account:

```
$credKeyfactor = Get-Credential
```

Enter the appropriate username and password when prompted (the service account that the orchestrator uses to connect to Keyfactor Command). Usernames should be given in DOMAIN\username format.

Or, to avoid being prompted for credentials:

```
$keyfactorUser = "DOMAIN\mykeyfactorconnectusername"
$keyfactorPassword = "MySecurePassword"
$secKeyfactorPassword = ConvertTo-SecureString $keyfactorPassword -AsPlainText -
Force
$credKeyfactor = New-Object System.Management.Automation.PSCredential
($keyfactorUser, $secKeyfactorPassword)
```

5. Run the password change script on the Universal Orchestrator server using the following parameters:

-WebCredential (Windows)

This is the credential object of the service account that the orchestrator uses to communicate with Keyfactor Command that you created as per [Create Service Accounts for the Universal Orchestrator on page 11](#). It is provided as a PSCredential object.

For Basic authentication password change operations, this parameter is **required**.

This parameter cannot be used in conjunction with the *ClientSecret* or *ClientAuthPassword* parameter.

--username (Linux)


The service account that the orchestrator uses to communicate with Keyfactor Command created as per [Create Service Accounts for the Universal Orchestrator on page 11](#). It may be entered either as `username@domain` (e.g. `svc_kyforch@keyexample.com`) or `DOMAIN\username` (e.g. `KEYEXAMPLE\svc_kyforch`).

For Basic authentication password change operations, this parameter is **required**.


This parameter cannot be used in conjunction with the *client-secret* or *client-auth-password* parameter.

--password (Linux)

The password for the service account that the orchestrator uses to communicate with Keyfactor Command specified with the *username* parameter.

 **Important:** The password for the service account the orchestrator uses to communicate with Keyfactor Command is stored in clear text in the `orchestratorsecrets.json` file in the configuration directory under the installation directory for the orchestrator. By default, this file is granted read/write permissions for the orchestrator service account running the service on the Linux machine (*keyfactor-orchestrator* by default) and no permissions for any other users. Access to this file should be strictly controlled.

This parameter is **required** if the *username* parameter is specified.

 **Tip:** If you prefer to avoid providing the password at the command line (and storing it in command history), use an input file instead as follows:

- Create a file that contains just your password. For example:

```
vi my_password_file
```
- When using the password parameter, reference the file. For example:

```
--password $(cat my_password_file)
```
- Delete the password file after the install is complete. For example:

```
rm my_password_file
```

-SecretsPath (Windows) or --secrets-path (Linux)

The full path and file name of the `orchestratorsecrets.json` file that stores the secret information. This file is found in the configuration directory under the installation directory for

the Universal Orchestrator, which is by default:

```
Windows: C:\Program Files\Keyfactor\Keyfactor
Orchestrator\configuration\orchestratorsecrets.json
Linux: /opt/keyfactor/orchestrator/configuration/orchestratorsecrets.json
```

The location and file name for this file cannot be changed from the default. The parameter is provided to allow for installations in non-standard locations or multiple locations on the same server.

This parameter is **required**.

Windows example using basic authentication:

```
$keyfactorUser = "KEYEXAMPLE\svc_kyforch"
$keyfactorPassword = "MySecurePassword123!"
$secKeyfactorPassword = ConvertTo-SecureString $keyfactorPassword -AsPlainText -Force
$credKeyfactor = New-Object System.Management.Automation.PSCredential ($keyfactorUser, $secKey-
factorPassword)

.\change_secrets.ps1 -WebCredential $credKeyfactor -SecretsPath "C:\Program Files\Keyfactor\Keyfactor
Orchestrator\configuration\orchestratorsecrets.json"

Saved secrets to 'C:\Program Files\Keyfactor\Keyfactor Orches-
trator\configuration\orchestratorsecrets.json'
Restarting service KeyfactorOrchestrator-Default
```

Linux example using basic authentication:

```
vi password_file_new

sudo ./change_secrets.sh --username svc_kyforch@keyexample.com --password $(cat password_file_new) --
secrets-path /opt/keyfactor/orchestrator/configuration/orchestratorsecrets.json

Saving secrets to '/opt/keyfactor/orchestrator/configuration/orchestratorsecrets.json'
Restarting service keyfactor-orchestrator-default
```

Keyfactor Command Connect Service Account with Token Authentication

For both Windows and Linux servers, the secret change for the client that's used to make the connection to Keyfactor Command when token authentication is used follows this process:

1. Change the secret for the client in your identity provider (see [Create Service Accounts for the Universal Orchestrator on page 11](#)).

2. On the Windows or Linux server, open a command window. For Windows, this should be a PowerShell window open using the “Run as Administrator” option. Change to the directory in which the orchestrator is installed and locate the change_secrets script. By default, this is:

Windows: C:\Program Files\Keyfactor\Keyfactor Orchestrator\change_secrets.ps1
Linux: /opt/keyfactor/orchestrator/change_secrets.sh

3. For Linux only, use the chmod command to make the change_secrets.sh script file executable. The file ships in a non-executable state to avoid accidental execution. For example:

```
sudo chmod +x change_secrets.sh
```

4. Run the password change script on the Universal Orchestrator server using the following parameters:

-ClientSecret (Windows)

This is the secret of the identity provider client used to authenticate the session with Keyfactor Command (see [Create Service Accounts for the Universal Orchestrator on page 11](#)).

For token authentication password change operations, this parameter is **required**.

This parameter cannot be used in conjunction with the *WebCredential* or *ClientAuthPassword* parameter.

--client-secret (Linux)

This is the secret of the identity provider client used to authenticate the session with Keyfactor Command (see [Create Service Accounts for the Universal Orchestrator on page 11](#)).

For token authentication password change operations, this parameter is **required**.

This parameter cannot be used in conjunction with the *username* and *password* or *client-auth-password* parameters.



Tip: If you prefer to avoid providing the secret at the command line (and storing it in command history), use an input file instead as follows:

- a. Create a file that contains just your password. For example:

```
vi my_secret_file
```

- b. When using the password parameter, reference the file. For example:

```
--client-secret $(cat my_secret_file)
```




c. Delete the password file after the install is complete. For example:

```
rm my_secret_file
```

`-SecretsPath` (Windows) or `--secrets-path` (Linux)

The full path and file name of the or the `orchestratorsecrets.json` file that stores the secret information. This file is found in the configuration directory under the installation directory for the Universal Orchestrator, which is by default:

```
Windows: C:\Program Files\Keyfactor\Keyfactor
Orchestrator\configuration\orchestratorsecrets.json
Linux: /opt/keyfactor/orchestrator/configuration/orchestratorsecrets.json
```

The location and file name for this file cannot be changed from the default. The parameter is provided to allow for installations in non-standard locations or multiple locations on the same server.

This parameter is **required**.

Windows example using token authentication:

```
.\change_secrets.ps1 -ClientSecret "aLru2IvZYJh0kFmHa36xs2xTLSp4ya" -SecretsPath "C:\Program
Files\Keyfactor\Keyfactor Orchestrator\configuration\orchestratorsecrets.json"
```

```
Saved secrets to 'C:\Program Files\Keyfactor\Keyfactor Orches-
trator\configuration\orchestratorsecrets.json'
Restarting service KeyfactorOrchestrator-Default
```

Linux example using token authentication:

```
vi my_new_secret

sudo change_secrets.sh --client-secret $(cat my_new_secret) --secrets-path /opt/key-
factor/orchestrator/configuration/orchestratorsecrets.json

Saving secrets to '/opt/keyfactor/orchestrator/configuration/orchestratorsecrets.json'
Restarting service keyfactor-orchestrator-default

rm my_new_secret
```

Universal Orchestrator Running in a Container

When you're running the Universal Orchestrator in a container, the method for password changes is different and applies only to the Keyfactor Command connect service account. In this scenario, the

approach is to tear down the container and stand up a new one. As long as the new container connects to Keyfactor Command with the same set of capabilities, the same service account username, and the same orchestrator name (either the container hostname or the ORCHESTRATOR_NAME parameter), the orchestrator will continue to operate seamlessly and will not need re-approval.

To change the Keyfactor Command connect service account password for a container:

1. Change the service account's Active Directory password or change the secret for the client in your identity provider (see [Create Service Accounts for the Universal Orchestrator on page 11](#)).
2. Use an appropriate command to tear down the current orchestrator container (being sure you know its configuration). For example:

```
docker compose [-f myconfig.yaml] down
```

Or:

```
kubect1 delete -f myconfig.yaml]
```

3. Stand up a new container referencing the same set of capabilities, the same service account username, and the same orchestrator name (either the container hostname or the ORCHESTRATOR_NAME parameter) if you want the orchestrator to continue seamlessly without requiring re-approval (see [Install the Universal Orchestrator in a Linux Container on page 49](#)).

2.2.5.8 Register a Client Certificate Renewal Extension

The Keyfactor Universal Orchestrator supports automated renewal of the certificate used for client certificate authentication. It does this using a custom extension point interface on the orchestrator that can be implemented by the end user. When the client certificate used for authentication by the orchestrator is approaching expiration (within 180 days of expiration by default), the extension generates a CSR with a private key and submits the CSR to Keyfactor Command for enrollment. When Keyfactor Command returns the certificate to the orchestrator, it is paired with the private key and installed for use as the client certificate for authentication. The extension both supplies the information for the CSR and holds a dictionary of client parameters (see [Build a Client Certificate Renewal Extension on page 94](#)).

To register a client authentication certificate renewal extension:

1. Create the extension DLL (see [Build a Client Certificate Renewal Extension on page 94](#)).
2. On the Universal Orchestrator server, locate the extensions folder under the install directory for the orchestrator. By default, this is:

```
Windows: C:\Program Files\Keyfactor\Keyfactor Orchestrator\extensions
Linux: /opt/keyfactor/orchestrator/extensions
```

3. Under the extensions directory, create a new directory for your extension (e.g. CertRotation).

4. Place your DLL in the new CertRotation directory.
5. Create a manifest.json file in the CertRotation directory with the following contents:

```
{
  "extensions": {
    "Keyfactor.Orchestrators.Extensions.IOrchestratorRegistrationUpdater": {
      "RegisteredRegistrationUpdater": {
        "assemblypath": "RegistrationUpdater.dll",
        "TypeFullName": "Custom.Registration.Updaters.CustomRegistrationUpdater",
        "config": {
          "DnsSan": "orchestrator_name.keyexample.com",
          "Subject": "CN=Client Certificate Authentication",
          "DataCenter": "WestCoast",
          "ForceRenewal": "False"
        }
      }
    }
  }
}
```

Only the values shown in **red** above should be modified from what is shown in this example:

- The *assemblypath* is the name of your DLL.
- The example **Custom.Registration.Updaters** portion of the *TypeFullName* must match the *namespace* in your code. The example **CustomRegistrationUpdater** portion of the *TypeFullName* must match the *class* in your code.
- The config section is only needed if you wish to pass configuration values such as a standard DNS SAN or certificate subject into the extension. Those shown here are examples that match the sample code (see [Build a Client Certificate Renewal Extension on page 94](#)).

The certificate renewal process occurs as follows:

1. When each registration or session renewal of the orchestrator service occurs, the orchestrator, in conjunction with underlying Keyfactor Command functionality, checks the expiration date of the client authentication certificate and compares that with the defined client certificate warning period (180 days) and expiry period (30 days) in Keyfactor Command to determine whether a new certificate is needed.



Note: If the certificate is in the warning period, operations will continue while a new certificate is requested. If the certificate is in the expiry period or already expired, the orchestrator will not be allowed to register a new session when the existing session expires or the orchestrator service is restarted.



Orchestrator log messages indicating that a certificate is in the warning period look similar to the following:

```
2021-09-17 12:45:59.7927 Keyfactor.Orchestrators.JobEngine.SessionClient [Warn] - Remote
CMS call 'https://keyfactor.keyexample.com/KeyfactorAgents/Session/Register' returned:
Agent certificate is approaching expiration and should be renewed. (A0100007)
```

Orchestrator log messages indicating that a certificate is in the expiry period look similar to the following:

```
2021-09-09 17:27:37.5367 Keyfactor.Orchestrators.JobEngine.SessionClient [Error] - Remote
CMS call 'https://keyfactor.keyexample.com/KeyfactorAgents/Session/Register' returned:
Agent certificate is approaching expiration and must be renewed. (A0100008)
2021-09-09 17:27:37.5642 Keyfactor.Orchestrators.JobEngine.SessionJobExecutor [Error] -
Error in SessionManager: Unable to register session.
    at Keyfactor.Orchestrators.JobEngine.SessionClient.RegisterAsync(IEnumerable`1 capab-
ilities, CancellationToken cancellationToken)
    at Keyfactor.Orchestrators.JobEngine.SessionJobExecutor.Execute(IJobExecutionContext
context)

Error: A0100008
Agent certificate is approaching expiration and must be renewed.
    at Keyfactor.Orchestrators.JobEngine.SessionClient.RegisterAsync(IEnumerable`1 capab-
ilities, CancellationToken cancellationToken)
```



Tip: The length of the warning period and expiry period are defined in Keyfactor Command and are not user-configurable values. Contact support@keyfactor.com if you need to modify these values.



Tip: The orchestrator can be forced into the warning or expiry state before it reaches these based on certificate lifetime using the *POST /Agent-s/SetAuthCertificateReenrollment* method in the Keyfactor API or the *Request Renewal* button on the Orchestrator Management page of the Keyfactor Command Management Portal. A status of *Request* (1) is the equivalent of the warning period and a status of *Require* (2) is the equivalent of the expiry period.

- When either the warning period or expiry period is identified, the Keyfactor Universal Orchestrator will pass a value of *true* to the *GetCSRInfo* method (*newOrchestratorCertRequestedByPlatform* in the sample extension—see [Build a Client Certificate Renewal Extension on page 94](#)). The extension generates a private key and a CSR using CSR information (e.g. subject, key size) provided or generated by the extension (depending on the extension

design), returns the CSR to the orchestrator, which submits it to Keyfactor Command for certificate enrollment.

If the certificate is within the warning period but not within the expiry period, orchestrator activity will be allowed to continue as usual. If the certificate is within the expiry period, a new session will not be granted when the orchestrator requests a new session and orchestrator activity will not be allowed to continue until the orchestrator acquires a new certificate. If the certificate has expired, the certificate rotation cannot take place since the orchestrator cannot authenticate to Keyfactor Command to complete the rotation.



Tip: If a certificate has expired or some other certificate problem is causing the orchestrator not to be able to acquire a session, the orchestrator can be reset using either the Reset button on the Orchestrator Management page in the Keyfactor Command Management Portal or the `POST /Agents/{id}/Reset` method in the Keyfactor API. This removes the certificate history and allows the orchestrator to register for a session with the certificate currently configured in the `appsettings.json` file under the configuration directory. You will need to re-approve the orchestrator if you reset it.

3. In Keyfactor Command, a certificate is issued based on:

- The `OrchestratorConstants.CertificateAttributes.CERTIFICATE_AUTHORITY` and `OrchestratorConstants.CertificateAttributes.CERTIFICATE_TEMPLATE` values defined in the custom registration handler enroll function.
- If no values are supplied in the custom registration handler enroll function, the certificate authority and template defined by the *Certificate Authority For Submitted CSRs* and *Template For Submitted CSRs* application settings in the Keyfactor Command Management Portal.

Application Settings[?]

Application Settings define operational parameters for the system.

Console Auditing Enrollment **Agents** API SSH Workflow

General

● Hover over the label to get more information on the setting.

Heartbeat Interval (minutes)

5

Session Length (minutes)

1380

Registration Check Interval (minutes)

30

Registration Handler Timeout (seconds)

5

Job Failures and Warnings Age Out (days)

7

Template For Submitted CSRs

Corp Keyfactor Agent Auth

Certificate Authority For Submitted CSRs

corpca01.keyexample.com/CorplssuingCA1

Revoke old Client Auth Certificate

☒ True ☐ False

Number of times a job will retry before reporting failure

5

Send Entropy during on device key generation (ODKG/Reenrollment)

☐ True ☒ False

Notification Alert Interval (minutes)

10

Notification Alert Email Recipients

Notification Alert Email Recipients

Authentication

F5

SSL

SAVE UNDO ALL

Figure 18: Application Settings for Client Certificate Renewal

- Once the certificate is issued, it is returned to the orchestrator and married with the private key. If certificate authentication is configured using a certificate stored in the local computer or Universal Orchestrator service account user's personal store (Windows only), the orchestrator updates the `appsettings.json` file with the thumbprint of the new certificate. The thumbprint is stored in the `AuthCertThumbprint` value in the `appsettings.json` file (see [Change Service Account Passwords on page 81](#)). If certificate authentication is configured using a PKCS12 file stored in the file system, a PKCS12 file is generated and replaces the original PKCS12 file. The randomly generated password for the PKCS12 file is updated in the `orchestratorsecrets.json` file.



Note: If certificate authentication is configured using a certificate stored in the local computer personal store on Windows, when the new certificate is generated, it will be placed in the service account user's personal store, not the local computer personal store. This is true if the service is running as a domain account and if the service is running as the default *Network Service*.

5. With the orchestrator's next session registration or heartbeat, it will begin using the new certificate.



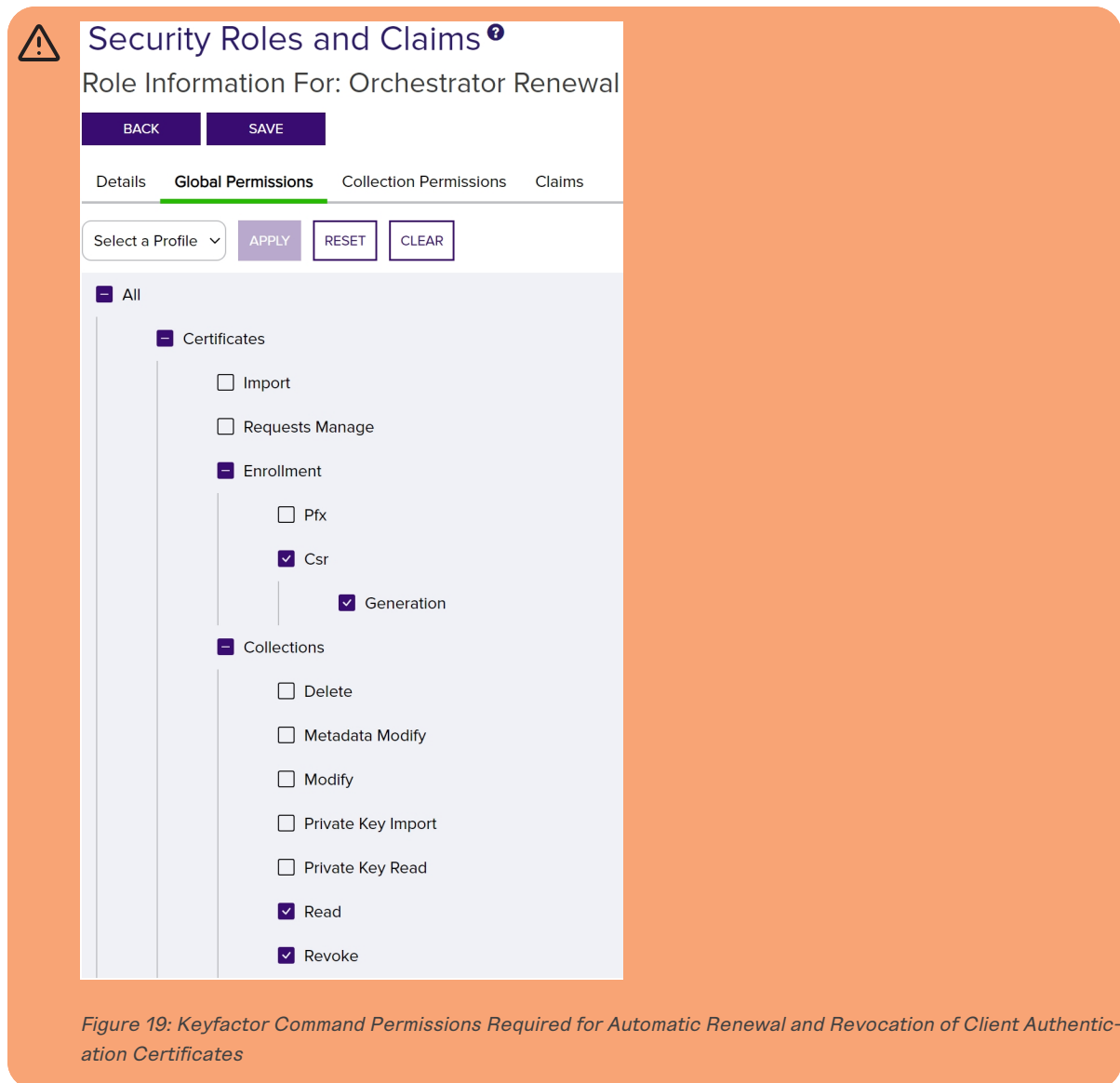
Tip: If you manually configured the orchestrator to renew the certificate using the *POST /Agents/SetAuthCertificateReenrollment* method in the Keyfactor API or the *Request Renewal* button on the Orchestrator Management page, that flag will be cleared when the orchestrator successfully registers for a session or completes a heartbeat using the new certificate.

Once a new session is established with the new certificate, the stored legacy thumbprint for the replaced certificate will be removed from the database. This occurs whether or not you opt to automatically revoke the old certificate and occurs before the certificate revocation takes place. The legacy thumbprint is not cleared on a heartbeat with the new certificate.

6. If you've opted to enable the *Revoke old Client Auth Certificate* application setting (see [Figure 18: Application Settings for Client Certificate Renewal](#)) in Keyfactor Command, the previous certificate for the orchestrator will be revoked automatically by Keyfactor Command once the orchestrator has made a successful registered for a new session with the new certificate.



Important: The service account under which the Universal Orchestrator is operating must have permissions to enroll for certificates from the CA and have at least the *Enroll CSR* role permission for *Certificate Enrollment* and the *Read* role permission for *Certificates* in Keyfactor Command. If you've opted to enable automated revocation of the old certificate, the service account must also have permissions to revoke certificates on the CA and have at least the *Revoke* role permission for *Certificates* in Keyfactor Command.



Build a Client Certificate Renewal Extension

The functionality to renew the certificate used by the Keyfactor Universal Orchestrator for authentication is available via an extension point interface provided by Keyfactor. To implement a custom extension, you will need to obtain the *Keyfactor.Orchestrators.IOrchestratorRegistrationUpdater* nuget package from Keyfactor. Contact your Client Success Manager or support@keyfactor.com for assistance with that.

To build a client certificate renewal extension:

1. Create a project for the extension in your favorite integrated development environment (e.g. Visual Studio).

2. Import the *Keyfactor.Orchestrators.IOrchestratorRegistrationUpdater* nuget package into the project.
3. Consult the sample code to help you design your extension. A sample extension for the client authentication registration updater interface is provided on the Keyfactor GitHub:

<https://keyfactor.github.io/>

4. Build an assembly file (DLL file) containing the extension.
5. Follow the instructions for registering the extension (see [Register a Client Certificate Renewal Extension on page 88](#)).

2.3 Java Agent

The Keyfactor Java Agent allows organizations to run discovery jobs to locate Java keystores on Windows and Linux systems and PEM certificate stores on Linux systems, inventory the certificates found in them, and push new certificates out to them.

The system requirements for the Java Agent on Windows are:

- 64-bit versions of Windows 8.1, Windows 10, and Windows Server 2019
- 64-bit versions of Oracle Java or OpenJDK 8, 11 or 13
- The WiX Toolset (<http://wixtoolset.org/>) for users wishing to build an MSI



Note: The path to the WiX executables needs to be added to the System PATH (e.g. C:\Program Files (x86)\WiX Toolset v3.11\bin) to support this.

The system requirements for the Java Agent on Linux are:

- Red Hat 6 or greater, CentOS 6 or greater, or Ubuntu 14 or greater
- 64-bit versions of Oracle Java or OpenJDK 8, 11 or 13
- JSVC on SysV style (init.d) systems



Important: The Keyfactor Java Agent will be deprecated in a future version of Keyfactor Command. Customers are encouraged to begin planning a migration to the Keyfactor Universal Orchestrator with the Remote File custom extension publicly available at:

<https://github.com/Keyfactor/remote-file-orchestrator>

For more information, see *Installing Custom-Built Extensions* in the *Keyfactor Orchestrators Installation and Configuration Guide*.

2.3.1 Preparing for the Java Agent

This section describes the steps that need to be taken prior to a Java Agent installation to install the prerequisites, create the required supporting components, and gather the necessary information to complete the Java Agent installation and configuration process.

2.3.1.1 Create Service Accounts for the Java Agent

The Java Agent makes use of up to two service accounts to allow it to communicate with the Keyfactor Command server. These two service accounts work together to transfer information from the Java Agent to the Keyfactor Command server. The two service accounts can be thought of as players on two sides of a fence, with the service account for the Java Agent lobbing information over the fence to the service account on the Keyfactor Command server side to catch and hand to the Keyfactor Command server:

- **Java Agent Side**
On the Java Agent side of the fence, you may use either a local account or an Active Directory service account.

Windows

For domain-joined Windows machines, an Active Directory service account is typically used. For non-domain-joined Windows machines, you may use a local account created on the Windows machine as the service account instead of a domain account.

The service account under which the Keyfactor Java Agent service runs on Windows must be granted permissions to “Log on as a service” through local security policy. This step is generally done automatically as part of the installation scripts, but may need to be completed by hand in certain environments or on certain operating systems. The service account needs sufficient permissions to allow it to discover and inventory Java keystores and PEM certificate stores as applicable (read permissions on the appropriate files and directories) and update the stores if desired (write permissions on the files and directories in which the files are stored).



Important: During the installation process, you enter the Java agent service identity username and password interactively in a PowerShell window to configure the service account. PowerShell will not support the following characters in the service account password when used in this interface:

" \$

If you need to support these characters in the password, you can re-enter the username and password in the Services MMC after receiving an error in the PowerShell interface.

Linux

For the purposes of this documentation it is assumed that Linux machines will be non-domain joined and will use a local account to run the Java Agent.

For Linux systems, Keyfactor recommends running the service as an account other than root.

- **Keyfactor Command Server Side**
On the Keyfactor Command server side of the fence, an Active Directory service account in the primary Keyfactor Command server forest is used. This can be the same service account used for other Keyfactor Command server services. This service account appears in the Management Portal Orchestrator Management grid as the *Identity* for the Java Agent.

If the Java Agent is installed on a domain-joined machine in the same forest as the Keyfactor Command server, the same Active Directory service account may be used on both sides of the fence.

The service accounts need to be created prior to installation of the Java Agent software, and the person installing the Java Agent software needs to know the domain, username and password of each service account.



Important: Keyfactor highly recommends that you use strong passwords for any accounts or certificates related to Keyfactor Command and associated products, especially when these have elevated or administrative access. A strong password has at least 12 characters (more is better) and multiple character classes (lowercase letters, uppercase letters, numeral, and symbols). Ideally, each password would be randomly generated. Avoid password re-use.

2.3.1.2 Create a Group for Java Agent Auto-Registration (Optional)

Keyfactor Command can use an Active Directory group to support auto-registration of Java Agents. Auto-registration is an optional feature that allows you to define the conditions under which a Java Agent can automatically be approved for operation with the Keyfactor Command server without administrator input, if desired. This is useful in environments hosting a large number of agents. There are six Java Agent auto-registration roles that share the same AD group:

Java Keystore Discovery

Auto-register the Java Agent to allow it to run discovery tasks to locate Java keystores.

Java Keystore Inventory Reporting

Auto-register the Java Agent to allow it to inventory certificates in Java keystores.

Java Keystore Management

Auto-register the Java Agent to allow it to manage (add/remove) certificates in Java keystores.

PEM Cert Store Management Jobs

Auto-register the Java Agent to allow it to manage (add/remove) certificates in PEM certificate stores.

PEM Server Configuration Directive Parser

Auto-register the Java Agent to allow it to run discovery tasks to locate PEM certificate stores.

PEM Server Inventory

Auto-register the Java Agent to allow it to inventory certificates in PEM certificate stores.

The same Active Directory group must be used for all roles. All auto-registration settings must be populated if any are to be used even if all features are not planned for use. For example, if you plan to use PEM but not Java keystore functionality, you still need to populate the Java keystore auto-registration settings to enable auto-registration for the Java Agent to function correctly.



Note: If all your agents will be connecting to Keyfactor Command as the same service account, you can directly add that user in the auto-registration configuration and skip using a group, if desired.

Although you can choose to enable auto-registration without user validation, allowing any agent to register regardless of the user account under which the agent is running, user validation with either an Active Directory group or a specific Active Directory user is the more secure option.

2.3.1.3 Configure Certificate Root Trust for the Java Agent

Keyfactor recommends using HTTPS to secure the channel between each Java Agent and the Keyfactor Command server(s). This requires an SSL certificate configured in IIS on the Keyfactor Command server(s). This certificate can either be a publicly-rooted certificate (e.g. from Symantec, Entrust, etc.), or one issued from a private certificate authority (CA). If your Keyfactor Command server is using a publicly rooted certificate, the Java Agent machine may already trust the certificate root for this certificate. However, if you have opted to use an internally-generated certificate, your Java Agent server may not trust this certificate. In order to use HTTPS for communications between the Java Agent and the Keyfactor Command server with a certificate generated from a private CA, you will need to import the certificate chain for the certificate into a Java CA certificate store on the Java Agent server. This can be done automatically as part of the installation process. You will need to have the root certificate available as a PEM-encoded format file when you run the installation script.

2.3.1.4 Create Environment Variables for the Java Agent on Windows

The Keyfactor Java Agent determines the location of the current installed Java version on Windows by checking the Windows system environment variables Path and JAVA_HOME. Depending on how your version of Java was installed, these environment variables may or may not be set.

To check and set the environment variables for the Java Agent install:

1. Identify your Java base directory (e.g. C:\Program Files\Java\jdk-13.0.2). This directory typically contains the versioning and release files. Copy this path to a text file for easy access.

2. Identify the location of the Java virtual machine library (e.g. C:\Program Files\Java\jdk-13.0.2\bin\server\jvm.dll), and copy the path to the text file created in the previous step.
3. Identify the location of the main Java executable (e.g. C:\Program Files\Java\jdk-13.0.2\bin\java.exe), and copy the path to the text file.
4. As a user with local administrator permissions, use the search function to search for *environment* and select the option to edit the *system environment variables* from the search results.

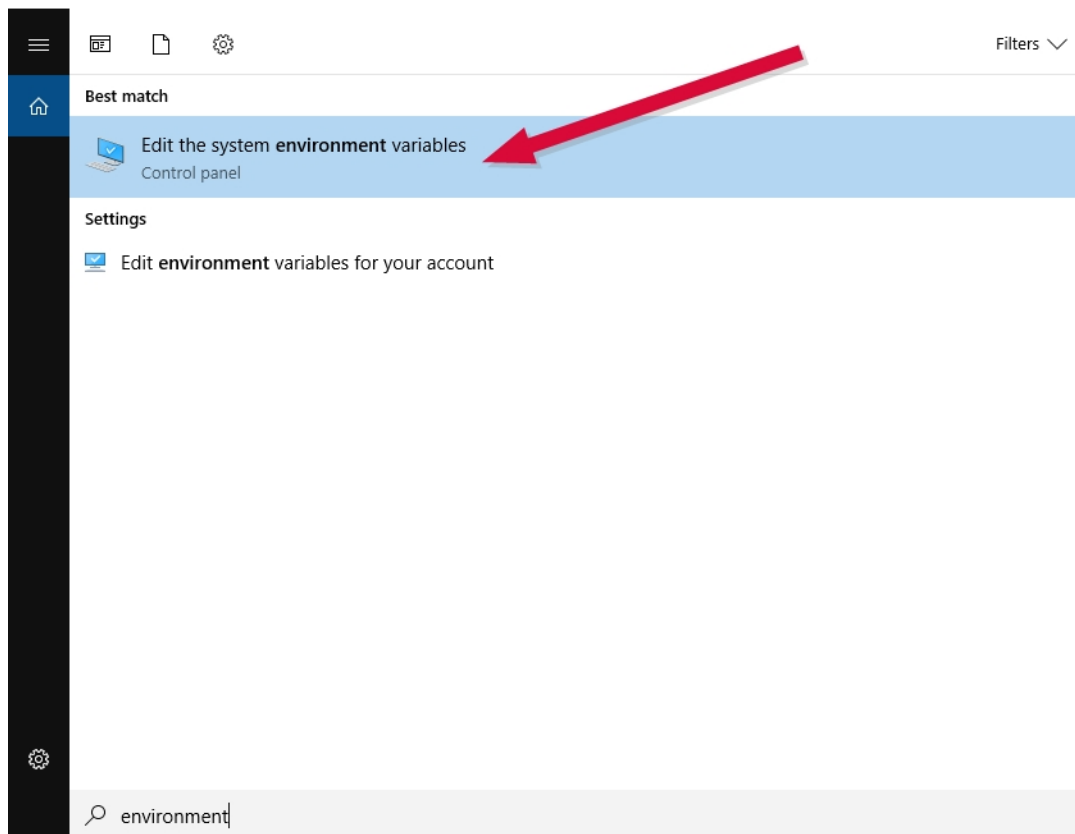


Figure 20: Search for System Environment Variables

5. In the System Properties dialog on the Advanced tab, click *Environment Variables*.
6. In the Environment Variables dialog in the *System variables* section at the bottom, scroll down to locate the *Path* variable, highlight it and click **Edit**.

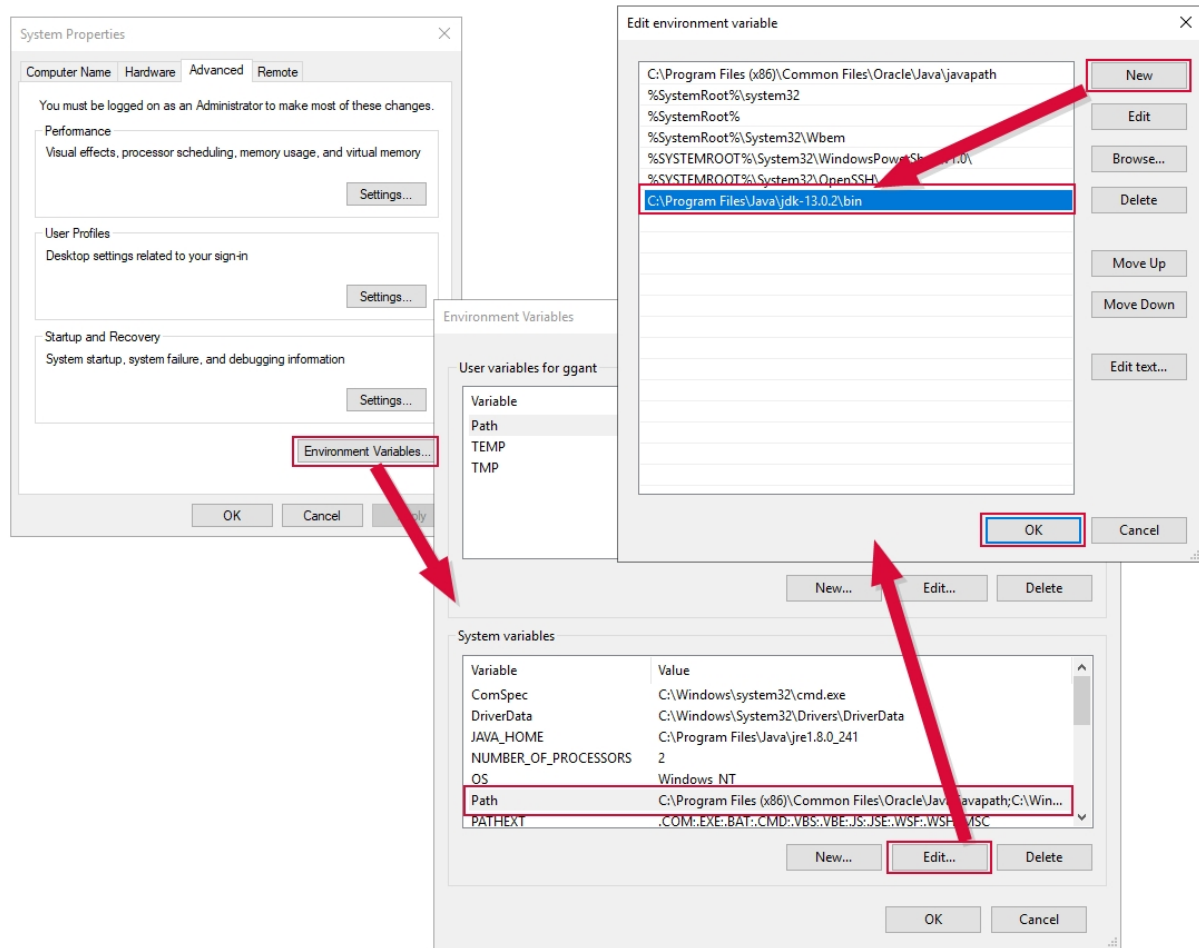


Figure 21: Edit the System Path Environment Variable to Add the Path to Java

7. In the Edit Environment Variable dialog, click **New**. On the newly added line, paste the path to the main Java executable (e.g. C:\Program Files\Java\jdk-13.0.2\bin) that you saved earlier (do not include the java.exe part) and click **OK**.
8. If it doesn't exist already among the *System variables*, create the JAVA_HOME environment variable—click **New** below the *System variables* box and, in the New System Variable dialog, type **JAVA_HOME** in the *Variable name* field and paste the path to the Java base directory in the *Variable value* field. If the field exists already but with a value that is not correct for the version of Java you wish to use, click **Edit** and update the *Variable value* field with the appropriate Java base directory.

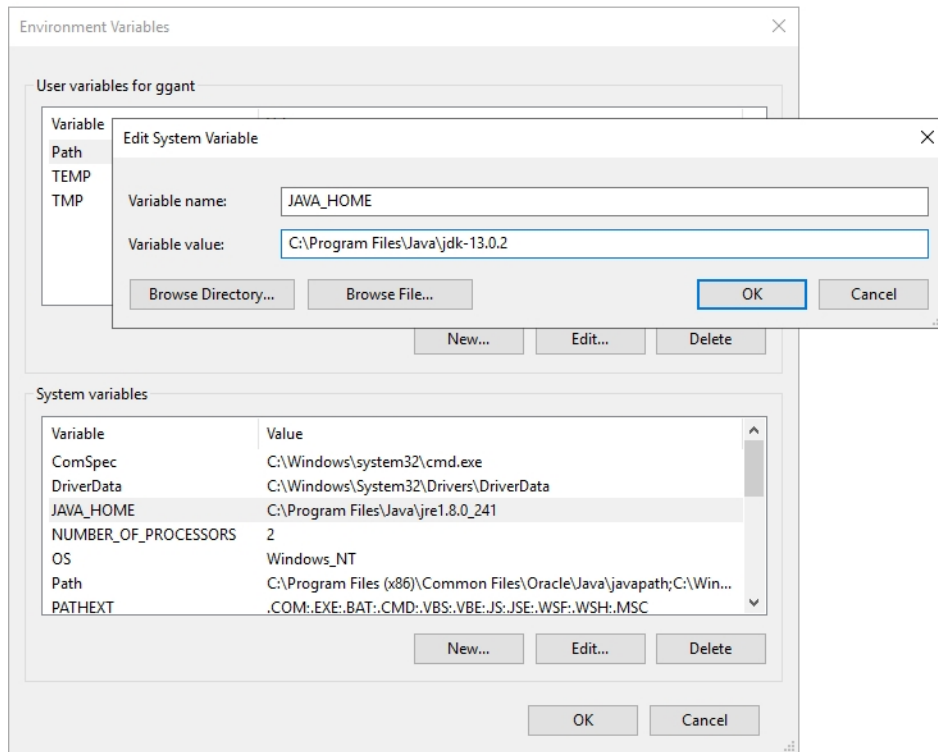


Figure 22: Add JAVA_HOME System Environment Variable



Important: When you run the install.ps1 script, you may be prompted to input the absolute path to the Java Virtual Machine library. From the text file in which you saved paths, take the path to the Java Virtual Machine library (e.g. C:\Program Files\Java\jdk-13.0.2\bin\server-jvm.dll) and input that string to complete the install.

2.3.2 Install the Java Agent on Windows

The Keyfactor Java Agent installation script offers the option to install the Java agent directly or use the installation script to build an msi package that you can then use to install the Java agent on multiple machines.



Note: If you have a previously installed version of the Keyfactor Java Agent on this server, you need to uninstall it (see [Uninstall the Java Agent on page 117](#)) before installing a new version.

To begin the Java agent installation on Windows, unzip the installation files and place them in a temporary working directory.

1. On the Windows machine on which you wish to install the Java agent or build the package, open a PowerShell window using the “Run as administrator” option and change to the temporary

directory where you placed the installation files.

2. In the PowerShell window, run the cms-java-agent-installer.bat file to begin the installation. You will be prompted to answer several questions:

Username the Java Agent will connect as

This is the service account on the Keyfactor Command server side of the fence you created as per [Create Service Accounts for the Java Agent on page 96](#). It should be entered in the format DOMAIN\username.

Password for the account that the Java Agent will connect as

This is the password for the service account on the Keyfactor Command server side of the fence.

Hostname or address for the Keyfactor Command Agents server

This is the FQDN or IP address of the Keyfactor Command server running the Keyfactor Command Agent Services role, which is installed as part of the Keyfactor Command Services role. If you installed all the Keyfactor Command server roles together, this is the FQDN or IP address of your Keyfactor Command server.

If you choose to use SSL to connect to the Keyfactor Command server, you'll need to enter a hostname at this prompt that is found in the SSL certificate.

If you're using a non-standard port for IIS on your Keyfactor Command server, enter that here as part of your hostname or IP address (e.g. keyfactor.keyexample.com:444).

Virtual directory for the Keyfactor Command Agents service URL

Press Enter to accept the default of KeyfactorAgents. Only enter an alternate virtual directory if your Keyfactor Command server was configured with an alternate virtual directory for the Keyfactor Command Agents service.

Connect to Keyfactor using SSL?

Press Enter to accept the default of Yes or enter No. The following instructions assume that you answered Yes.

To connect to Keyfactor Command, the Java Agent needs to trust the SSL certificate presented by the Keyfactor Command Agents server

If your Keyfactor Command server is using a publicly rooted certificate, the server most likely already trusts the certificate issuer, and you can press Enter here.

If the certificate on the Keyfactor Command server was internally generated, you will need to enter the full path and file name pointing to a file on the local server containing the PEM-encoded root certificate for the certificate authority chain that issued the certificate (see [Configure Certificate Root Trust for the Java Agent on page 98](#)).

The root certificate will be saved in a Java keystore file called `trust.jks` located in the Java agent's install directory (`C:\Program Files\Keyfactor\Keyfactor Java Agent` by default). The default keystore password is *changeit*. Please contact Keyfactor technical support for assistance in changing the default password, if desired.

This question will not appear if you answered no to the question about using SSL.

Verify Keyfactor Command connectivity?

Press Enter to accept the default of “Yes”. The Java agent will attempt to connect to the Keyfactor Command server using the credentials provided to confirm that the server name, agent's URL, root trust, and provided credentials are valid. Enter “No” to skip this validation if you don't have connectivity to the Keyfactor Command server at the time of installation.



Tip: If the installer terminates after this question without an error or with an error writing the `trust.jks` file, it can be an indication that the path to the root certificate you provided in the previous question was incorrect in some way (e.g. the path is not valid, the root certificate doesn't match the certificate on the Keyfactor Command server, etc.)

Please specify the installation format

The options at this prompt are “local” or “msi”. If you press enter to accept “local”, the Java agent will be installed locally. If you enter “msi”, the batch file will generate an msi after all the questions have been answered. You can use this to install the Java agent on other Windows systems with the installation questions already answered. The subsequent questions differ depending on the answer given to this question. The following instructions include both **local** and **msi** questions. You will not see all of these questions.

If you select “msi”, the Java agent will not be installed locally.

Path to the desired directory for installation (**Local**)

Press Enter to accept the default installation directory of `C:\Program Files\Keyfactor\Keyfactor Java Agent` or enter an alternate path if desired. This question does not appear when generating an msi.

Local user account the agent should run as \ User account on the target machine that the agent should run as (**Local****MSI**)

Press Enter to accept the default of the local SYSTEM account for local installs (this is not an option when generating an msi) or enter a specific user account—the service account for the

Java agent side of the fence you created as per [Create Service Accounts for the Java Agent on page 96](#). Domain user accounts should be entered in the format DOMAIN\username. You do not need to enter the password for this user at this time. The username is entered at this time to allow permissions to be configured appropriately.

Hostname the agent will connect to Keyfactor as (Local)

Press Enter to accept the default of the local machine's hostname as determined by a reverse DNS lookup or, failing that, the value of the local environment variable for the computer name. If desired, you can enter an alternative value to use as the hostname. This is the identifier for the server on which you are installing the Java agent. This identifier can be in the form of a host-name or FQDN, but you can use another unique identifier, if desired. This identifier appears in the Keyfactor Command Management Portal on the orchestrators page. This question does not appear when generating an msi.



Tip: When installing from an msi, you can specify a custom hostname by using the AGENTNAME parameter. In order to use this option, you must install the msi from the command line. For example:

```
msiexec /i C:\temp\cms-java-agent.msi AGENTNAME=jvagt38.keyexample.com
```



Note: If the agent machine has a non-private address, you will most likely need to use this option.

Directory where the agent logs should be placed (Local)

Press Enter to accept the default log directory of C:\CMS\logs or enter an alternate path if desired. This question does not appear when generating an msi.

Number of log files that should be kept (Local\MSI)

Press Enter to accept the default of 7 log files or enter an alternate number if desired. Older files are automatically deleted once more files than this have been generated.

Maximum size of each log file (Local\MSI)

Press Enter to accept the default log file size of 3 MB or enter an alternate value if desired.

Register AnyAgent components with the Keyfactor Java Agent? (Local)

Press Enter to accept the default value and begin the installation. If you would like to install one or more Any Agent implementations, enter yes. In this case, you'll be presented with a list of custom certificate store types for which to provide an implementation. After choosing each one, you'll need to enter the path to the .jar file that implements the certificate store type. That .jar file will be copied to the installation directory, under the libs folder. You'll need to manually copy any other dependent .jar files to that location as well. Note that this option is only available when

the Java agent is installed locally. This question does not appear when generating an msi.

3. After answering the AnyAgent components question, the installation begins. Review the output to be sure that no errors have occurred and then press any key to return to the PowerShell prompt.

```
Welcome to the Keyfactor Java Agent Installer.
This installer will collect all information necessary to configure the Java Agent for use with your Keyfactor Command instance. You will be given the option to apply this configuration to the local machine, or to use the configuration data to construct a Linux RPM package or Windows MSI installer for distribution within your environment.
Please enter the following information:

Username the Java Agent will connect as (format "DOMAIN\user"):
KEYEXAMPLE\svc_kyfjava
Password for the account that the Java Agent will connect as:
Re-enter password:
Hostname or address for the Keyfactor Command Agents server (e.g. "server1.corp.local" or "192.168.0.100"):
keyfactor.keyexample.com
Virtual directory for the Keyfactor Command Agents service URL (default: KeyfactorAgents):
Connect to Keyfactor using SSL? (default: Yes):
To connect to Keyfactor Command, the Java Agent needs to trust the SSL certificate presented by the Keyfactor Command Agents server.
Please enter a local path to a CA certificate that can be used to trust the SSL certificate that will be presented.
You can find this certificate by navigating to the Keyfactor Command Management Portal in a secure browser session and viewing the server certificate chain.
c:\temp\CorpRoot.cer
Verify Keyfactor Command connectivity? (default: Yes):
Verifying connection...
Please specify the installation format. Enter "local" or "msi". (default: "local"):
Path to the desired directory for installation. Directory must not already exist. (default: C:\Program Files\Keyfactor\Keyfactor Java Agent ):
Local user account the agent should run as (default : SYSTEM):
Hostname the agent will connect to Keyfactor as:
jvagt54.keyexample.com
Directory where the agent logs should be placed (default: C:\CMS\logs\):
NOTE: Logging configuration, including additional options, can be adjusted through the "log4j2.xml" file within the agent "config" directory.
Number of log files that should be kept (default: 7):
Maximum size of each log file (default: "3 MB"):
Register AnyAgent components with the Keyfactor Java Agent? (Default: No):
Building Java Agent installer...
Encrypting credentials
Generating config files
Copying files
Option to be replaced: $javaOptionsArray += "-Dcms.agentMachine=jvagt54.keyexample.com"
Creating SSL certificate trust store
Install completed
You can use the scripts included in the installation to set up the Java agent as a service on your platform. Additional configuration may be necessary for the service to run automatically on machine startup.
```

Figure 23: Keyfactor Java Agent Local Installation on Windows

4. In the PowerShell window, change to the install directory within the directory in which you installed the Java agent. If you installed in the default install directory, this path is:

C:\Program Files\Keyfactor\Keyfactor Java Agent\install

5. In the PowerShell window, run the install.ps1 PowerShell script. Unless you selected SYSTEM as the user the agent should run as, you will be prompted to enter the username (DOMAIN\user-name format) and password of the account that will run the Keyfactor Java Agent service on the local machine. This is the service account for the Java agent side of the fence you created as per [Create Service Accounts for the Java Agent on page 96](#). Press Enter without entering any data to run the service under the local system credentials.



Note: The install.ps1 may fail with an error similar to the following on older versions of Windows:

Method invocation failed because [System.Object[]] doesn't contain a method named 'Replace'.

If this occurs, you need to manually grant the service account under which the Keyfactor Java Agent service will run the local "Log on as a service" permission and then run the install.ps1 script again.



Tip: If you choose the “msi” option rather than the “local” option, the MSI file will be generated in the directory in which you executed the batch file.

2.3.3 Install the Java Agent on Linux

The Java Agent installation script offers the option to install the Java Agent directly or use the installation script to build an RPM package that you can then use to install the Java Agent on multiple machines.



Note: If you have a previously installed version of the Keyfactor Java Agent on this server, you need to uninstall it (see [Uninstall the Java Agent on page 117](#)) before installing a new version.

To begin the Java Agent installation on Linux, unzip the installation files and place them in a temporary working directory.

1. On the Linux machine on which you wish to install the Java Agent or build the package, at a command shell change to the temporary directory where you placed the installation files.
2. Use the `chmod` command to make the `cms-java-agent-Installer.sh` script executable. The file ships in a non-executable state to avoid accidental execution. For example:

```
sudo chmod +x cms-java-agent-installer.sh
```

3. In the command shell, run the `cms-java-agent-Installer.sh` script *as root* to begin the installation. You will be prompted to answer several questions:

Username the Java Agent will connect as

This is the service account on the Keyfactor Command server side of the fence you created as per [Create Service Accounts for the Java Agent on page 96](#). It should be entered in the format `DOMAIN\username`.

Password for the account that the Java Agent will connect as

This is the password for the service account on the Keyfactor Command server side of the fence.

Hostname or address for the Keyfactor Command Agents server

This is the FQDN or IP address of the Keyfactor Command server running the Keyfactor Command Agent Services role, which is installed as part of the Keyfactor Command Services role. If you installed all the Keyfactor Command server roles together, this is the FQDN or IP address of your Keyfactor Command server.

If you choose to use SSL to connect to the Keyfactor Command server, you'll need to enter a hostname at this prompt that is found in the SSL certificate.

If you're using a non-standard port for IIS on your Keyfactor Command server, enter that here as part of your hostname or IP address (e.g. `keyfactor.keyexample.com:444`).

Virtual directory for the Keyfactor Command Agents service URL

Press Enter to accept the default of KeyfactorAgents. Only enter an alternate virtual directory if your Keyfactor Command server was configured with an alternate virtual directory for the Keyfactor Command Agents service.

Connect to Keyfactor using SSL?

Press Enter to accept the default of Yes or enter No. The following instructions assume that you answered Yes.

To connect to Keyfactor Command, the Java Agent needs to trust the SSL certificate presented by the Keyfactor Command Agents server...

If your Keyfactor Command server is using a publicly rooted certificate, the server most likely already trusts the certificate issuer, and you can press Enter here.

If the certificate on the Keyfactor Command server was internally generated, you will need to enter the full path and file name pointing to a file on the local server containing the PEM-encoded root certificate for the certificate authority chain that issued the certificate (see [Configure Certificate Root Trust for the Java Agent on page 98](#)).

The root certificate will be saved in a Java keystore file called `trust.jks` located in the Java Agent's install directory (`/opt/keyfactor-java-agent` by default). The default keystore password is *changeit*. Please contact Keyfactor technical support for assistance in changing the default password, if desired.

This question will not appear if you answered no to the question about using SSL.

Verify Keyfactor Command connectivity?

Press Enter to accept the default of "Yes". The Java Agent will attempt to connect to the Keyfactor Command server using the credentials provided to confirm that the server name, agents URL, root trust, and provided credentials are valid. Enter "No" to skip this validation if you don't have connectivity to the Keyfactor Command server at the time of installation.



Tip: If the installer terminates after this question without an error or with an error writing the `trust.jks` file, it can be an indication that the path to the root certificate you provided in the previous question was incorrect in some way (e.g. the path is not valid, the root certificate doesn't match the certificate on the Keyfactor Command server, etc.)

Please specify the installation format

The options at this prompt are “local” or “rpm”. If you press enter to accept “local”, the Java Agent will be installed locally. If you enter “rpm”, the script will generate an rpm after all of the questions have been answered. You can use this to install the Java Agent on other Linux systems with the installation questions already answered. The subsequent questions differ depending on the answer given to this question. The following instructions include both **local** and **rpm** questions. You will not see all of these questions.

If you select “rpm”, the Java agent will not be installed locally.

Path to the desired directory for installation (**Local**)

Press Enter to accept the default installation directory of /opt/keyfactor-java-agent or enter an alternate path if desired. This question does not appear when generating an rpm.

Local user account the agent should run as \ User account on the target machine that the agent should run as (**Local****RPM**)

This is the service account for the Java Agent side of the fence you created as per [Create Service Accounts for the Java Agent on page 96](#). It should be entered as just the user name. Entry of the password for this service account is not required. The username is entered at this time to allow permissions to be configured appropriately.

Hostname the agent will connect to Keyfactor as (**Local**)

Press Enter to accept the default of the local machine’s hostname as determined by a reverse DNS lookup or, failing that, the value of the local environment variable for the computer name. If desired, you can enter an alternative value to use as the hostname. This is the identifier for the server on which you are installing the Java agent. This identifier can be in the form of a host-name or FQDN, but you can use another unique identifier, if desired. This identifier appears in the Keyfactor Command Management Portal on the orchestrators page. This question does not appear when generating an rpm.

Full path to the desired buildroot directory for RPM package staging. Directory must not exist. (**RPM**)

Press Enter to accept the default path of /temp under the current directory or enter an alternate path if desired. This is a temporary location the build process will use while the package is being created. This is not the directory where the final RPM file will be placed. This question does not appear when installing locally.



Note: Ensure the path does not contain spaces. Any space in the java agent path causes issues when building an rpm.



Tip: The RPM file will be generated in a subdirectory (rpmbuild/RPMS) of the home directory of the user running the cms-java-agent-Installer.sh script. If you run the script as root, this will be root's home directory, so you may choose to run the script as a non-root user if you plan to create an RPM.

Path the RPM will install to on the target machine (RPM)

Press Enter to accept the default installation directory of /opt/keyfactor-java-agent or enter an alternate path if desired. This question does not appear when installing locally.

Architecture of the RPM target machine (RPM)

Press Enter to accept the default as determined by the machine on which the RPM is being generated or enter an alternate architecture if desired. A separate RPM needs to be generated with each required machine architecture. This question does not appear when installing locally.

Directory where the agent logs should be placed (Local\RPM)

Press Enter to accept the default log directory of /opt/cms-java-agent/logs or enter an alternate path if desired.

Number of log files that should be kept (Local\RPM)

Press Enter to accept the default of 7 log files or enter an alternate number if desired. Older files are automatically deleted once more files than this have been generated.

Maximum size of each log file (Local\RPM)

Press Enter to accept the default log file size of 3 MB or enter an alternate value if desired.

Register AnyAgent components with the Keyfactor Java Agent? (Local)

Press Enter to accept the default value and begin the installation. If you would like to install one or more Any Agent implementations, enter yes. In this case, you'll be presented with a list of custom certificate store types for which to provide an implementation. After choosing each one, you'll need to enter the path to the .jar file that implements the certificate store type. That .jar file will be copied to the installation directory, under the libs folder. You'll need to manually copy any other dependent .jar files to that location as well. Enter "Done" when you've finished listing agent implementations. Note that this option is only available when the JavaAgent is installed locally.

4. After answering the log file size question, the installation begins. Review the output to be sure that no errors have occurred.


```

Welcome to the Keyfactor Java Agent Installer.
This installer will collect all information necessary to configure the Java Agent for use with your Keyfactor Command instance. You will be given the option to apply this configuration to the local machine, or to use the configuration data to construct a Linux RPM package or Windows MSI installer for distribution within your environment.
Please enter the following information:

Username the Java Agent will connect as (format "DOMAIN\user"):
KEYEXAMPLE\svc_kyfjava
Password for the account that the Java Agent will connect as:
Re-enter password:
Hostname or address for the Keyfactor Command Agents server (e.g. "server1.corp.local" or "192.168.0.100"):
keyfactor.keyexample.com
Virtual directory for the Keyfactor Command Agents service URL (default: KeyfactorAgents):

Connect to Keyfactor using SSL? (default: Yes):

To connect to Keyfactor Command, the Java Agent needs to trust the SSL certificate presented by the Keyfactor Command Agents server.
Please enter a local path to a CA certificate that can be used to trust the SSL certificate that will be presented.
You can find this certificate by navigating to the Keyfactor Command Management Portal in a secure browser session and viewing the server certificate chain.
/tmp/CorpRoot.crt
Verify Keyfactor Command connectivity? (default: Yes):

Verifying connection...

Please specify the installation format. Enter "local" or "rpm". (default: "local"):

Path to the desired directory for installation. Directory must not already exist. (default: /opt/keyfactor-java-agent ):

Local user account the agent should run as:
kyfuser

Hostname the agent will connect to Keyfactor as:
jvagt162.keyexample.com
Directory where the agent logs should be placed (default: /opt/keyfactor-java-agent/logs/):

NOTE: Logging configuration, including additional options, can be adjusted through the "log4j2.xml" file within the agent "config" directory.
Number of log files that should be kept (default: 7):

Maximum size of each log file (default: "3 MB"):

Register AnyAgent components with the Keyfactor Java Agent? (Default: No):

Building Java Agent installer...
Encrypting credentials
Generating config files
Copying files
Creating SSL certificate trust store
Install completed
You can use the scripts included in the installation to set up the Java agent as a service on your platform. Additional configuration may be necessary for the service to run automatically on machine startup.

```

Figure 24: Keyfactor Java Agent Local Installation on Linux

5. Keyfactor provides scripts that can be used to configure the Keyfactor Java Agent to start automatically. These can be used on systems using startups based on SysV style (init.d) or systemd. Other startup systems will need to be configured manually. If your machine has neither of these startup systems, you will not be able to use these scripts to configure the Keyfactor Java Agent to start automatically. The appropriate startup script to use depends on whether you are doing a local install or installing from a previously generated RPM file.

Local Install

- a. In the command shell, change to the directory in which you installed the Java Agent. The default install directory is:

```
/opt/keyfactor-java-agent
```

- b. Select the appropriate installation script for your startup system. The two available scripts for local installs are:

```
install-init-service.sh
install-systemd-service.sh
```




Tip: The scripts with `-with-configured-hostname` in their names (e.g. `install-systemd-service-with-configured-hostname.sh`) are for use with installations from RPM packages and should not be used for local installs.

- c. Use the `chmod` command to make the desired script executable. The file ships in a non-executable state to avoid accidental execution. For example:

```
sudo chmod +x install-systemd-service.sh
```

- d. Run the appropriate shell script as root. This will add the `keyfactor-java-agent` as a service, which you can then stop and start using the standard service stop and start commands. For example:

```
service keyfactor-java-agent restart
systemctl restart keyfactor-java-agent.service
```

Install from RPM

- a. Locate the RPM file on the machine on which it was generated and copy it to the machine on which you wish to install the Java agent.



Tip: The RPM file is generated in a subdirectory (`rpmbuild/RPMS`) of the home directory of the user running the `cms-java-agent-Installer.sh` script. If you run the script as root, this will be root's home directory.

- b. Execute the RPM as root. For example:

```
sudo rpm -ivh keyfactor-java-agent-8.6.0-1.i686.rpm
```

- c. In the command shell, change to the directory in which you installed the Java Agent. The default install directory is:

```
/opt/cms-java-agent
```

- d. There are four possible installation scripts for installation from RPM packages:

```
install-init-service.sh
install-init-service-with-configured-hostname.sh
install-systemd-service.sh
install-systemd-service-with-configured-hostname.sh
```

Select the appropriate installation script type for your startup system (`init` or `systemd`). The versions of the scripts that contain a reference to *with-configured-hostname* in the file name allow you to enter a custom agent name (see [Hostname the agent will connect to](#)

[Keyfactor as \(Local\) on page 108](#)). The versions without this reference will use the system hostname as the agent name.

- e. Use the `chmod` command to make the desired script executable. The file ships in a non-executable state to avoid accidental execution. For example:

```
sudo chmod +x install-systemd-service-with-configured-hostname.sh
```

- f. Run the appropriate shell script as root. You will be prompted to answer questions specific to the machine on which the Java agent is being installed—the hostname or other identifier for the machine (see [Hostname the agent will connect to Keyfactor as \(Local\) on page 108](#)) if you used a *with-configured-hostname* script and the username and password for the service account that will connect the agent to Keyfactor Command (see [Username the Java Agent will connect as on page 106](#)).
- g. Change the ownership on the file containing the startup credentials to the local user that the agent will run as. This file is found in the *config* directory under the installed directory and is called *install.creds*. For example:

```
sudo chown kyfuser /opt/cms-java-agent/config/install.creds
```

- h. The install shell script adds the `keyfactor-java-agent` as a service, which you can then stop and start using the standard service stop and start commands. You may need to restart the service after changing the ownership on the credentials file. For example:

```
service keyfactor-java-agent restart
systemctl restart keyfactor-java-agent.service
```



Tip: If desired, you can pass the responses to the questions the installer asks in from a file. For example, for a full install (not working from an RPM file you previously created), create a file that contains values something like this:

```
KEYEXAMPLE\svc_kyfjava
MyVerySecurePassword
MyVerySecurePassword <- The installer requires entry of the password twice
keyfactor.keyexample.com
KeyfactorAgents
Yes
/tmp/CorpRoot.crt
Yes
local
/opt/keyfactor-java-agent
kyfuser
jvagt162.keyexample.com
```



```
/opt/keyfactor-java-agent/logs  
7  
"3 MB"  
No
```

Note that the values needed in your input file will vary depending on how you answer some of the questions. For example, the first Yes shown above will go in response to the question of whether to use SSL for the connection to Keyfactor Command. If you answer No here, you will not receive the question about needing a root certificate, and so the path to a root certificate shown after this will not correctly match the next question. The script will fail.

Place the file in the same directory as the install script. Then, execute the install script like this:

```
sudo ./cms-java-agent-installer.sh < myinputfile.txt
```

2.3.4 Optional Configuration

Once the installation scripts are complete, the Java Agent should be running and ready to communicate with the Keyfactor Command server.



Important: Java Agent tasks will not run until you complete the Java Agent configuration by making the appropriate configuration changes in the Keyfactor Command Management Portal. See *Orchestrators* in the *Keyfactor Command Reference Guide* for instructions on approving the Java Agent in the Keyfactor Command Management Portal on the *Orchestrators->Auto-Registration* and *Orchestrators->Management* pages, and on configuring certificate stores on the *Certificate Management->Certificate Stores* page (see *Certificate Store Operations: Adding or Modifying a Certificate Store* and *Certificate Store Discovery* in the *Keyfactor Command Reference Guide*).

2.3.4.1 Configure Logging for the Java Agent

By default, the Java Agent places its log files in the C:\CMS\logs directory on Windows and the /opt/keyfactor-java-agent/logs directory on Linux, generates logs at the *Info* logging level and stores seven 3 MB logs before deleting them (how long this will be will depend on the logging level and the volume of usage the Java Agent is receiving).

If you wish to change these defaults after the installation is complete on Windows:

1. On the Java Agent machine where you wish to adjust logging, open a text editor (e.g. Notepad) using the “Run as administrator” option.

2. In the text editor, browse to open the log4j2.xml file in the config directory under the directory in which you installed the Java Agent. By default, the file is located in the following directory:

C:\Program Files\Keyfactor\Keyfactor Java Agent\config

3. Your log4j2.xml file may have a slightly different layout than shown here, but it will contain the four fields highlighted in the below figure. The fields you may wish to edit are:

fileName="C:\CMS\logs\CMS-Java.txt"

The path and file name of the active Java Agent log file.



Important: If you choose to change the path for storage of the log files, you will need to create the new directory (e.g. D:\KeyfactorLogs) and grant the service account under which the Keyfactor Java Agent service is running full control permissions on this directory.

size="3 MB"

The maximum file size of each log file. After a log file reaches the maximum size, it is rotated to an archive file name and a new log file is generated.

max="7"

The number of archive files to retain before deletion.

level="info"

The level of log detail that should be generated. The default info level logs error and some informational data but at a minimal level to avoid generating large log files. For troubleshooting, it may be desirable to set the logging level to debug or trace. Available log levels (in order of increasing verbosity) are:

- OFF – No logging
- FATAL – Log severe errors that cause early termination
- ERROR – Log severe errors and other runtime errors or unexpected conditions that may not cause early termination
- WARN – Log errors and use of deprecated APIs, poor use of APIs, almost errors, and other runtime situations that are undesirable or unexpected but not necessarily wrong
- INFO – Log all of the above plus runtime events (startup/shutdown)
- DEBUG – Log all of the above plus detailed information on the flow through the system
- TRACE – Maximum log information—this option can generate VERY large log files

```

<RollingFile name="logfile" fileName="C:\CMS\logs\CMS-Java.txt" filePattern="CMS-Java-%i.txt">
  <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n" />
  <MarkerFilter marker="EVENT" onMatch="DENY" onMismatch="NEUTRAL" />
  <SizeBasedTriggeringPolicy size="3 MB"/>
  <DefaultRolloverStrategy max="7"/>
</RollingFile>
Section of file removed in graphic for simplicity.
<Loggers>
  <Root level="info">
    <AppenderRef ref="console" />
    <AppenderRef ref="logfile" />
  </Root>
</Loggers>

```

Figure 25: Configure Logging for Keyfactor Java Agent on Windows

If you wish to change these defaults after the installation is complete on Linux:

1. On the Java Agent machine where you wish to adjust logging, open a command shell and change to the directory in which the Java Agent is installed. By default this is /opt/keyfactor-java-agent.
2. In the command shell in the directory in which the Java Agent is installed, change to the config directory.
3. Using a text editor, open the log4j2.xml file in the config directory. Your log4j2.xml file may have a slightly different layout than shown here, but it will contain the four fields highlighted in the below figure. The fields you may wish to edit are:

```
fileName="/opt/keyfactor-java-agent/logs/CMS-Java.txt"
```

The path and file name of the active Java Agent log file.



Important: If you choose to change the path for storage of the log files, you will need to create the new directory (e.g. /opt/javalogs) and grant the service account under which the Keyfactor Java Agent service is running full control permissions on this directory.

```
size="3 MB"
```

The maximum file size of each log file. After a log file reaches the maximum size, it is rotated to an archive file name and a new log file is generated.

```
max="7"
```

The number of archive files to retain before deletion.

```
level="info"
```

The level of log detail that should be generated. The default INFO level logs error and some informational data but at a minimal level to avoid generating large log files. For troubleshooting, it may be desirable to set the logging level to DEBUG or TRACE. Available log levels (in order of increasing verbosity) are:

- OFF – No logging
- FATAL – Log severe errors that cause early termination
- ERROR – Log severe errors and other runtime errors or unexpected conditions that may not cause early termination
- WARN – Log errors and use of deprecated APIs, poor use of APIs, almost errors, and other runtime situations that are undesirable or unexpected but not necessarily wrong
- INFO – Log all of the above plus runtime events (startup/shutdown)
- DEBUG – Log all of the above plus detailed information on the flow through the system
- TRACE – Maximum log information—this option can generate VERY large log files

```
<RollingFile name="logfile" fileName="/opt/cms-java-agent/logs/CMS-Java.txt" filePattern="CMS-Java-%i.txt">
  <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n" />
  <MarkerFilter marker="EVENT" onMatch="DENY" onMismatch="NEUTRAL" />
  <SizeBasedTriggeringPolicy size="3 MB"/>
  <DefaultRolloverStrategy max="7"/>
</RollingFile>
Section of file removed in graphic for simplicity.
<Root level="info">
  <AppenderRef ref="console" />
  <AppenderRef ref="logfile" />
</Root>
```

Figure 26: Configure Logging for Keyfactor Java Agent on Linux

2.3.4.2 Start the Keyfactor Java Agent Service

The Keyfactor Java Agent service runs on the Java Agent machine and controls discovery, inventory and certificate store update tasks. During the Java Agent configuration process you set the service account under which the Keyfactor Java Agent service will run. The service should start automatically at the conclusion of the installation scripts.

To check to see if the Keyfactor Java Agent service is running and start it if necessary on Windows:

1. On a Windows Java Agent server, open the Services MMC.
2. In the Services MMC confirm that the Keyfactor Java Agent service is set to a Startup Type of Automatic (if desired). If the service is not running, click the green arrow to start it.

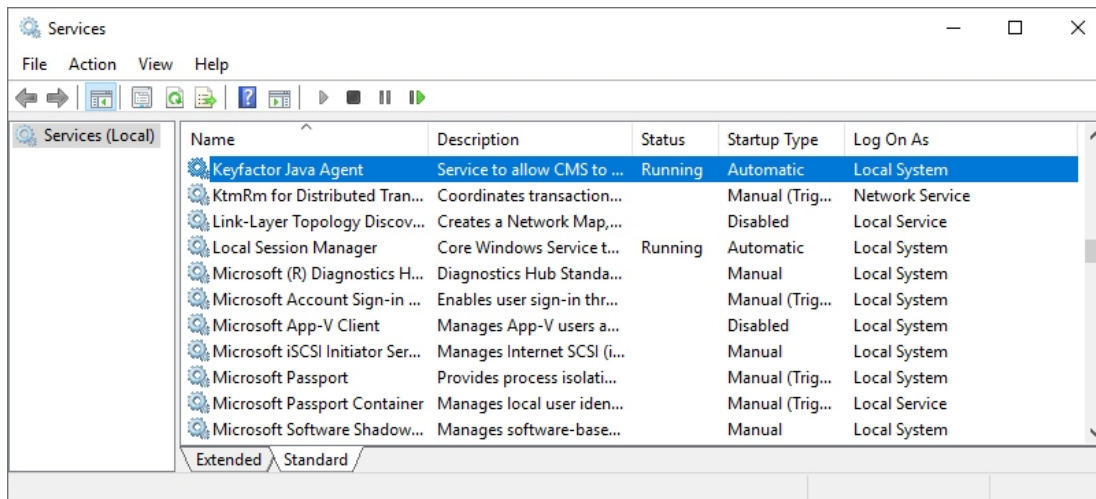


Figure 27: Keyfactor Java Agent Service on Windows

Service startup and shutdown procedures vary by Linux implementation and version depending on the startup system. The service on Linux is added as `keyfactor-java-agent`, so when referencing it in startup commands, it should be referenced by this name, including case. For example:

```
service keyfactor-java-agent start [stop] [restart]
systemctl start [stop] [restart] [status] keyfactor-java-agent.service
```

Once you have finished the Java keystore and PEM certificate store inventory configuration using the Keyfactor Command Management Portal and have imported certificates from the stores, you can use the Certificate Search feature in the Keyfactor Command Management Portal to review the certificate store certificates. See *Certificate Search and Collections* in the *Keyfactor Command Reference Guide* for information on using the Certificate Search feature.

2.3.4.3 Uninstall the Java Agent

To uninstall the Java Agent on Linux.

1. On the Linux machine on which the Java Agent is installed, run the command to stop the service.

```
sudo systemctl stop keyfactor-java-agent.service
```

2. Run the command to remove the service

```
sudo systemctl disable keyfactor-java-agent.service
```

3. After steps 1 & 2 are executed, it is safe to manually remove the Java Agent folder (default location is `/opt/keyfactor-java-agent/`).

2.4 Bash Orchestrator

SSH supports a wide variety of authentication mechanisms. Often, enterprises fall back to simple username and password at least some of the time due to the complexities of key management for key-based authentication. Without key management, SSH keys tend to multiply, and you can quickly lose track of who has access to what where. The Keyfactor Bash Orchestrator is designed to allow organizations to inventory and manage secure shell (SSH) keys across the enterprise.

Important: SSH Key Management licensing is required to use any of the functionality outlined in the Keyfactor Bash Orchestrator documentation. Contact support@keyfactor.com for assistance with obtaining the proper licenses.

The orchestrator runs on Linux servers and can be operated in two possible modes:

- The orchestrator is used in *inventory only* mode to perform discovery of SSH public keys and associated Linux user accounts across multiple configured targets. When used in *inventory and publish policy* mode, the orchestrator:
 - Scans the `authorized_keys` files of all current users on each configured server.

Note: OpenSSH maintains a file for each user that contains the public keys authorized to connect via SSH. By default, this file is named `authorized_keys`. In this document, we refer to this file as *authorized_keys*, however in your environment, this file may have a different name. The file name used in a given environment is defined in the `AuthorizedKeysFile` setting in the OpenSSH `sshd_config` file.

- Aggregates all public key data per Linux user logon.
- Reports aggregate key and logon data back to Keyfactor Command.

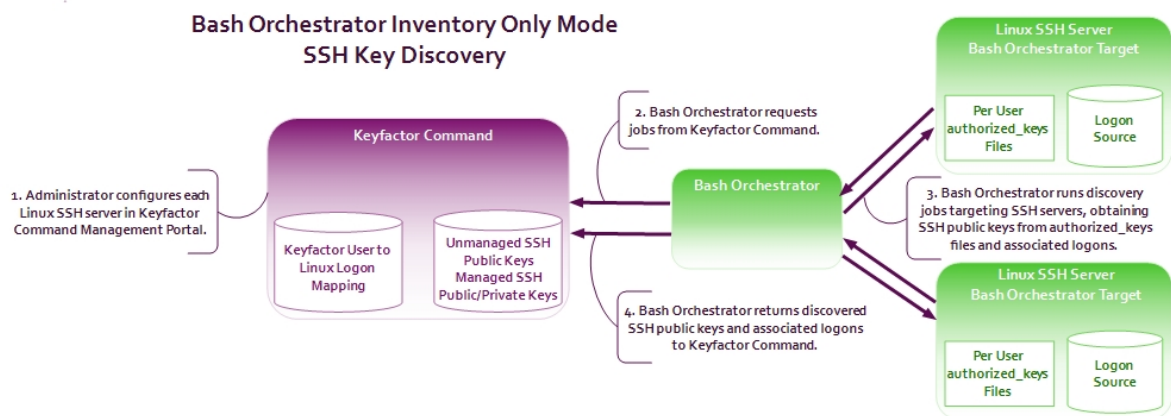


Figure 28: SSH Key Discovery Flow

- When operated in *inventory and publish policy* mode, the orchestrator can be used to add SSH public keys and Linux user accounts on targets and remove rogue keys that appear without authorization. [Figure 29: SSH User Key Management Flow](#) shows the flow from a user requesting

a new key pair to the public key being placed on a target server to allow the user to connect to the server via SSH. The flow is similar for requesting a key pair for a service, though the request is made by an administrator through a different interface in the Keyfactor Command Management Portal. When used in *inventory and publish policy* mode, policies are published to the orchestrator from the Keyfactor Command server following this flow:

- The Keyfactor Command server determines what content needs to go into the `authorized_keys` files for each logon on each target server. Content includes keys and associated comments aggregated from all servers where that key was found. For example, if a given public key exists on three different servers for the same user but in the original `authorized_keys` files the key is associated with a different comment on each server, when Keyfactor Command publishes the key down to the servers, it will be published with an aggregated comment string (all three comments together in each `authorized_keys` file).
- Aggregate logon and key information pushed down to each orchestrator target.
- Orchestrator determines where to place key information, builds the file, and overwrites the existing file with the new one. The process is done in this way to enforce policy and prevent rogue keys from being placed in `authorized_keys` files.
- Orchestrator informs Keyfactor Command of the success or failure of each machine logon combination.

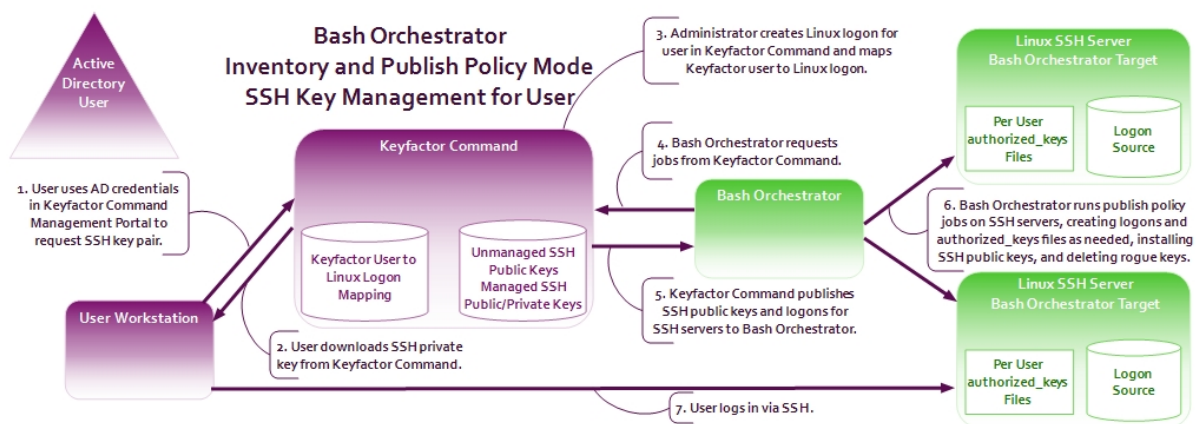


Figure 29: SSH User Key Management Flow

2.4.1 Preparing for the Keyfactor Bash Orchestrator

This section describes the steps that need to be taken prior to a Keyfactor Bash Orchestrator installation to install the prerequisites, create the required supporting components, and gather the necessary information to complete the orchestrator installation and configuration process.

2.4.1.1 System Requirements

The Keyfactor Bash Orchestrator is supported on the following operating systems:

- Oracle Linux 7 or higher
- Red Hat Enterprise 7 or higher
- Ubuntu 16 or higher

The minimum specs are:

- 2GB RAM
- 1 2GHz Processor
- 20GB disk space



Note: As more servers are added to be orchestrated by the orchestrator, increasing those specs will improve performance.

The following applications are required:

systemd

The orchestrator requires a Linux server that uses the systemd service manager. You can use the following command to test whether a system is running systemd:

```
ps -p 1
```

bash

The orchestrator can only be installed on a Linux server that is running bash version 4.3 or higher. You can use the following command to check the bash version of a server:

```
bash --version
```

For systems running an older version of bash, you may be able to successfully operate these as control targets (see [Install Remote Control Targets on page 127](#)).



Note: The default version of bash on Red Hat Enterprise 7 is 4.2. If you're using this platform and your bash version has not already been updated, this will need to be done.

curl

The orchestrator can only be installed on a Linux server that has curl installed. You can use the following command to check the curl version of a server:

```
curl --version
```

This is a requirement for orchestrators only; curl does not need to be installed on control targets (see [Install Remote Control Targets on page 127](#)).

2.4.1.2 Create a Service Account for the Keyfactor Bash Orchestrator

The Keyfactor Bash Orchestrator uses a service account in the Active Directory domain where the Keyfactor Command server resides to allow it to communicate with Keyfactor Command. This can be the same service account used for other Keyfactor Command server services. This service account appears in the Management Portal as the *Identity* on the Orchestrator Management grid for the Keyfactor Bash Orchestrator.

The service account needs to be created prior to installation of the Keyfactor Bash Orchestrator software, and the person installing the Keyfactor Bash Orchestrator software needs to know the domain, username and password of the service account.



Important: Keyfactor highly recommends that you use strong passwords for any accounts or certificates related to Keyfactor Command and associated products, especially when these have elevated or administrative access. A strong password has at least 12 characters (more is better) and multiple character classes (lowercase letters, uppercase letters, numeral, and symbols). Ideally, each password would be randomly generated. Avoid password re-use.

During installation of the orchestrator, a local Linux user account is created automatically as an identity under which the orchestrator service will operate. This allows the orchestrator to run as a non-root user. On servers on which you install the orchestrator directly, the following Linux user account is created:

```
keyfactor-bash
```

On servers configured as remote control targets, the following Linux user account is created:

```
keyfactor-bash-orchestrator-svc
```

These users are granted access to read `authorized_keys` files for inventory purposes and to update `authorized_keys` files when the orchestrator is operating in *inventory* and *publish policy* mode using `sudo`. On install, modifications are made to the `sudo` configuration with the addition of a file in the `/etc/sudoer.d` directory granting the orchestrator user select `sudo` rights. The commands the service account user may be granted the right to use via `sudo` include:

```
adduser, awk, cat, chmod, chown, flock, gpasswd, ls, mkdir, restorecon, rm, sed, tee, test, touch, usermod
```

2.4.1.3 Create a Group for Auto-Registration (Optional)

Keyfactor Command can use an Active Directory group to support auto-registration of Keyfactor Bash Orchestrator. Auto-registration is an optional feature that allows you to define the conditions under which a Keyfactor Bash Orchestrator can automatically be approved for operation with the Keyfactor Command server without administrator input, if desired. This is useful in environments hosting a large number of orchestrators or if you wish to automatically add orchestrators to server

groups and add them as servers in the Management Portal as you install them. The auto-registration role used by the Keyfactor Bash Orchestrator is called *Secure Shell Management*.

Add the service account or service accounts under which the orchestrators will communicate with Keyfactor Command to this group.



Note: If all your orchestrators will be connecting to Keyfactor Command as the same service account, you can directly add that user in the auto-registration configuration and skip using a group, if desired.

Although you can choose to enable auto-registration without user validation, allowing any orchestrator to register regardless of the user account under which the orchestrator is running, user validation with either an Active Directory group or a specific Active Directory user is the more secure option.

2.4.1.4 Certificate Root Trust for the Keyfactor Bash Orchestrator

Keyfactor recommends using HTTPS to secure the channel between each Keyfactor Bash Orchestrator and the Keyfactor Command server(s). This requires an SSL certificate configured in IIS on the Keyfactor Command server(s). This certificate can either be a publicly-rooted certificate (e.g. from Symantec, Entrust, etc.), or one issued from a private certificate authority (CA). If your Keyfactor Command server is using a publicly rooted certificate, the orchestrator machine may already trust the certificate root for this certificate. However, if you have opted to use an internally-generated certificate, your orchestrator server may not trust this certificate. In order to use HTTPS for communications between the orchestrator and the Keyfactor Command server with a certificate generated from a private CA, you will need to import the certificate chain for the certificate into the orchestrator's root certificate store. This can be done automatically as part of the installation process. You will need to have the root certificate available as a PEM-encoded format file when you run the installation script.

2.4.2 Install the Keyfactor Bash Orchestrator

To begin the Keyfactor Bash Orchestrator installation, place the installation files in a temporary working directory on the Linux server and:

1. On the Linux machine on which you wish to install the main orchestrator, in a command shell change to the temporary directory where you placed the installation files.
2. Use the `chmod` command to make the following script files executable. The files ship in a non-executable state to avoid accidental execution.
 - [yourpath]/heartbeat.sh
 - [yourpath]/static-analysis.sh
 - [yourpath]/syncjob.sh
 - [yourpath]/Service/keyfactor-bash-orchestrator.sh
 - [yourpath]/Service/systemd/configure-systemd.sh

- [yourpath]/Service/systemd/stop.sh
- [yourpath]/Installation/install.sh
- [yourpath]/Installation/remoteinstall.sh
- [yourpath]/Installation/uninstall.sh

For example, this command will add the executable flag to every file with a .sh extension in the /tmp/BashOrchestrator directory and all its sub-directories:

```
sudo find /tmp/BashOrchestrator -type f -iname "*.sh" -exec chmod +x {} \;
```

3. In the command shell, run the Installation/install.sh script as root using the following syntax to begin the installation:

-n, --username <service account name>

This is the service account that the orchestrator uses to communicate with Keyfactor Command that you created as per [Create a Service Account for the Keyfactor Bash Orchestrator on page 121](#). It should be entered in the format username@domain (e.g. svc_sshorch@keyexample.com). This parameter is required.

-u, --url <Keyfactor Command agents URL>

This is the URL to the Agent Services endpoint on the Keyfactor Command server running the Keyfactor Command Agent Services role, which is installed as part of the Keyfactor Command Services role. If you installed all the Keyfactor Command server roles together, this is the URL for your Keyfactor Command server with /KeyfactorAgents after the server's IP or FQDN (e.g. https://keyfactor.keyexample.com/KeyfactorAgents). If you choose to use SSL to connect to the Keyfactor Command server, you'll need to enter a URL that contains a hostname that is found in the SSL certificate. This parameter is required.



Tip: If your Keyfactor Command server was configured with an alternate virtual directory for the Keyfactor Command Agents Services endpoint, you will need to enter that in the URL rather than /KeyfactorAgents.

-p, --password <your-secure-password>

This is the password for the orchestrator service account. If you leave this parameter out, you will be prompted to enter this password.

-s, --ssl

Specifying this parameter causes the orchestrator to use SSL for communications with Keyfactor Command. Leave out this parameter if you prefer not to use SSL. This parameter does not take any arguments.

-t, --trusted-root </path/root-filename>

If your Keyfactor Command server is using a publicly rooted certificate, you do not need to use this option. If the certificate on the Keyfactor Command server was internally generated, you will need to use this option to specify the full path and file name of the file containing the root certificate for the certificate authority that issued the certificate (e.g. -t /tmp/myroot.crt). See [Certificate Root Trust for the Keyfactor Bash Orchestrator on page 122](#).

-i, --server-group-id <GUID of existing SSH server group>

If desired, you may specify this parameter to automatically add the server to an existing server group in Keyfactor Command. The server group must be specified by group ID (e.g. -i 74a9afcc-087d-423a-a331-06686427fdd9). You can find a server group's ID by editing the server group record in the Keyfactor Command Management Portal. This function is only supported if you have enabled auto-registration for SSH (see [Create a Service Account for the Keyfactor Bash Orchestrator on page 121](#)).

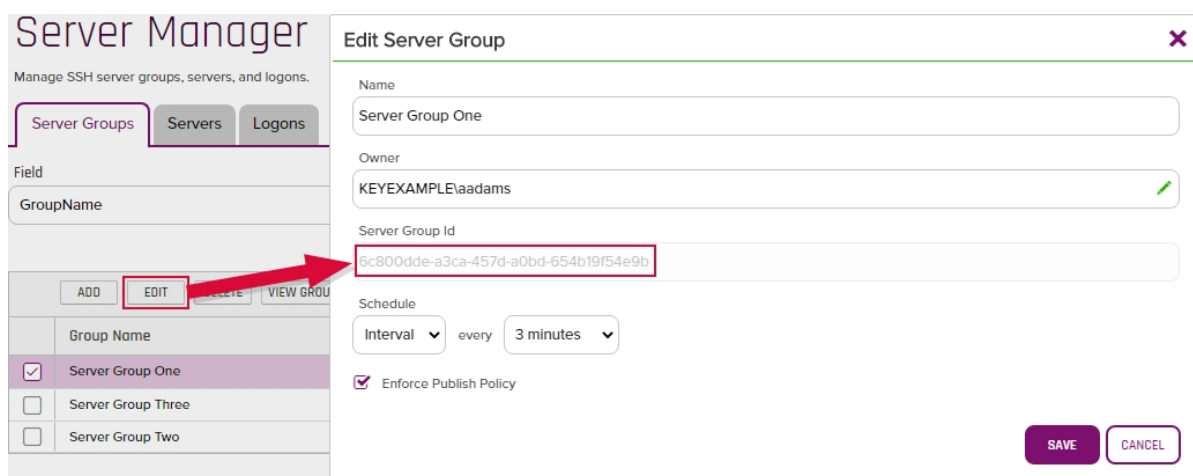


Figure 30: Find the Server Group ID

-c, --client-machine-name <client name>

Specifying this parameter allows you to override the client name the orchestrator would by default use to register itself in Keyfactor Command. By default, the orchestrator uses the results from a hostname lookup for the server's name. See the example output below where the name passed into Keyfactor Command (appsrvr80-SSH.ubuntu.keyexample.com) differs from the name used in the SSH key comment for the local Linux user (appsrvr80.keyexample.com).

-d, --use-sssd

This is required. You must explicitly specify whether or not you want to enable the orchestrator to use SSSD for user lookups.

- To enable SSSD set either:

```
-d true
```

```
--use-sssd true
```

- To disable SSSD set either:

```
-d false
```

```
--use-sssd false
```

When enabled, the orchestrator will check both the local user store and the SSSD user store (e.g. Active Directory) on requests to create logons and distribute key information, allowing keys to be managed both for local users and for domain users. When enabled, user logon must be created in Keyfactor Command with the username as it appears in SSSD (see *SSH-SSSD Case Sensitivity Flag* and *Adding Logons* in the *Keyfactor Command Reference Guide*).

If you're using SSSD, you must be using SSSD on any remote servers the orchestrator will manage. Additionally, the *LogonHomeDirectories* setting is expected to be consistent on all remote servers.

Domain users can be managed with or without preexisting home directories.

-l, --logon-home-directories </homedirectoryroot>

Specifying this parameter allows you to set the base path for home directories of SSH users. This is referenced both when new logons are created, as requested through Keyfactor Command, and when doing discovery for existing logons and keys. If you don't specify a value, the default of */home* is used.

The value set for the Keyfactor Bash Orchestrator *login-home-directories* needs to match the value set for the path in the *override_homedir* or *fallback_homedir* SSSD configuration. For example, if *fallback_homedir* = */home/my/dir/path/%u@%d*, *login-home-directories* needs to be set to */home/my/dir/path*. All SSSD logons to be discovered by or created with the Keyfactor Bash Orchestrator must have a home directory in this directory, not a subdirectory of this directory. For example, given the previously referenced directory, the path */home/my/dir/path/myusername@keyexample.com* would be valid but */home/my/dir/path/anotherdirlevel/myusername@keyexample.com* would not be valid. Home directories are created automatically when logons are created.



Important: Any remotely controlled targets (see [Install Remote Control Targets on page 127](#)) of a server using SSSD logons with the Keyfactor Bash Orchestrator must also be configured for SSSD logons and must have the same configuration value for *fallback_homedir* or *override_homedir*.

The output from the command should look similar to the following, given the example command shown.

```
sudo ./install.sh -u https://keyfactor.keyexample.com/KeyfactorAgents -n svc_
sshorch@keyexample.com -s -t /tmp/MyRoot.crt -i 74a9afcc-087d-423a-a331-06686427fdd9 -c
appsrvr80-SSH.ubuntu.keyexample.com -d false

Service Account Password:
Creating orchestrator installation directory...
Creating file structure...
Generating public/private rsa key pair.
Your identification has been saved in id_rsa.
Your public key has been saved in id_rsa.pub.
The key fingerprint is:
SHA256:APQcjxNSzPFF0Dg+cFlteJjHb2CoIAK7/ysAkUkKk7s keyfactor-bash@appsrvr80.keyexample.com
The key's randomart image is:
+---[RSA 2048]-----+
|=* .++=..B+B      |
|Bo. .==*=.* 0     |
|oo . .*=oo = o    |
|o.      o+      o  |
|o.      S.      .  |
|E.                |
| ..                |
| ..                |
| .o.                |
+----[SHA256]-----+
Creating orchestrator log file...
Creating Session Cache File...
Adding uninstall script to installation directory...
Installing Keyfactor Bash Orchestrator...
Creating credential file...
Creating job schedule table...
Adding root certificate to local ca store...
Updating certificates in /etc/ssl/certs...
0 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...

done.
done.
Creating Keyfactor SSH Daemon...
Creating service unit file...
Setting file ownership...
Ensuring service account 'keyfactor-bash' has necessary permissions...
```



```
Creating remote setup script...
Starting Keyfactor Bash Orchestrator...
```

4. Review the output from the installation to confirm that no errors have occurred.

The script creates a directory, `/opt/keyfactor-bash-orchestrator`, and places the orchestrator files in this directory. Log files are found in `/opt/keyfactor-bash-orchestrator/logs`, though this is configurable (see [Configure Logging for the Keyfactor Bash Orchestrator on page 129](#)).

The orchestrator service, named `keyfactor-bash-orchestrator.service`, should be automatically started at the conclusion of the install and configured to restart on reboot.



Tip: Once the installation of the orchestrator and any targets for it to control is complete, you need to use the Keyfactor Command Management Portal to approve the orchestrator (if you don't have auto-registration for Keyfactor Bash Orchestrators enabled) and configure SSH server groups and servers as per *Server Manager* in the *Keyfactor Command Reference Guide*. SSH server records are automatically created for the main bash orchestrator if you enable auto-registration for bash orchestrators and use the `-i` switch when registering the bash orchestrator. They are not automatically created for remote targets.

2.4.3 Install Remote Control Targets

After you complete the installation of at least one Keyfactor Bash Orchestrator, you can configure other Linux servers in the environment as control targets for this orchestrator. This is done by running a script on the target servers that installs the SSH public key matching the orchestrator's private key on the target server, along with making a few configuration changes. This allows the orchestrator service on the orchestrator (the local Linux user `keyfactor-bash`) to communicate with the targets using secured SSH.



Important: Any remotely controlled targets of a server using SSSD logons with the Keyfactor Bash Orchestrator must also be configured for SSSD logons and must have the same configuration value for `fallback_homedir` or `override_homedir`.

To configure orchestrator targets:

1. On the orchestrator machine, locate the `remoteinstall.sh` script in the `/opt/keyfactor-bash-orchestrator` directory. Do not use the `remoteinstall-template.sh` script found in the source material under Installation. This script has not been modified to contain the specific public key of your orchestrator.



Tip: A copy of the configured `remoteinstall.sh` script may also be found in the directory from which you executed the installation of the Keyfactor Bash Orchestrator.

2. Copy the customized `remoteinstall.sh` script to the orchestrator target that you wish to configure and place it in a temporary working directory.
3. On the Linux machine you wish to control with the orchestrator, in a command shell change to the temporary directory where you placed the `remoteinstall.sh` script.
4. Use the `chmod` command to make the script file executable. The file ships in a non-executable state to avoid accidental execution. For example:

```
sudo ./chmod +x remoteinstall.sh
```

5. In the command shell, run the `remoteinstall.sh` script as root with no parameters. There is no output from the command when it completes successfully.

```
sudo ./remoteinstall.sh
```

The script creates a directory, `/opt/keyfactor-bach-orchestrator-client`, and places the public key of the orchestrator Linux service account user in an `authorized_keys` file within it. It also creates a local service account user (see [Create a Service Account for the Keyfactor Bash Orchestrator on page 121](#)) and grants this user ownership on this file to allow the orchestrator server service account to perform tasks on the target.

Log messages are written to the standard Linux syslog. The location of these will vary depending on the system OS.



Tip: Once the installation of the orchestrator and any targets for it to control is complete, you need to use the Keyfactor Command Management Portal to approve the orchestrator (if you don't have auto-registration for Keyfactor Bash Orchestrators enabled) and configure SSH server groups and servers as per *Server Manager* in the *Keyfactor Command Reference Guide*. SSH server records are not automatically created for remote targets, even if you enable auto-registration for bash orchestrators and use the `-i` switch when registering the bash orchestrator that will control your targets.

2.4.4 Optional Configuration

Once the installation is complete, the Keyfactor Bash Orchestrator should be running and ready to communicate with the Keyfactor Command server. The initial installation allows the orchestrator to scan itself to do discovery of SSH keys and then management of SSH keys if the server is configured for management in Keyfactor Command. At this point, you may wish to configure one or more orchestrator target servers for the orchestrator to additionally control (see [Install Remote Control Targets on the previous page](#)).



Important: Orchestrator tasks will not run until you complete the orchestrator configuration by making the appropriate configuration changes in the Keyfactor Command Management Portal. See *Orchestrators* in the *Keyfactor Command Reference Guide* for instructions on approving the orchestrator in the Keyfactor Command Management Portal on the *Orchestrators->Management* pages and on configuring SSH server groups and servers on the *SSH-*



>Server Managers page (see *SSH Server Managers* in the *Keyfactor Command Reference Guide*).

2.4.4.1 Configure Logging for the Keyfactor Bash Orchestrator

By default, the Keyfactor Bash Orchestrator places its log files in the `/opt/keyfactor-bash-orchestrator/logs` directory, generates logs at non-debug level, rotates the logs when they reach 50 MB, and retains 10 archive logs before deletion.

If you wish to change these defaults after the installation is complete:

1. On the orchestrator machine where you wish to adjust logging, open a command shell and change to the directory in which the orchestrator is installed. By default this is `/opt/keyfactor-bash-orchestrator`.
2. In the command shell in the directory in which the orchestrator is installed, change to the Configuration directory.
3. Using a text editor, open the `orchestrator_config` file in the Configuration directory. Your `orchestrator_config` file may have a slightly different layout than shown here, but it will contain the three fields highlighted in the below figure. The fields you may wish to edit are:

- `logFile=/opt/keyfactor-bash-orchestrator/logs/bash-orchestrator-log.txt`

The path and file name of the active orchestrator log file.



Important: If you choose to change the path for storage of the log files, you will need to create the new directory (e.g. `/opt/sshorchlogs`) and grant the Linux service account under which the orchestrator service is running (see [Create a Service Account for the Keyfactor Bash Orchestrator on page 121](#)) full control permissions on this directory.

- `logFileSize=50000000`
The maximum file size of each log file. After a log file reaches the maximum size, it is rotated to an archive file name and a new log file is generated. The default is 50000000 (50 MB).
- `logFilesToKeep=10`
The number of archive files to retain before deletion.
- `debugLogEnabled=false`
The level of log detail that should be generated. The default of *false* logs error and some informational data but at a minimal level to avoid generating large log files. For troubleshooting, it may be desirable to set the debug level to *true*.

```
logFile=/opt/keyfactor-bash-orchestrator/logs/bash-orchestrator-log.txt
logFileSize=50000000
logFilesToKeep=10
keyfactorAgentServer=https://keyfactor.keyexample.com/KeyfactorAgents
serverGroupId=71804a75-78de-42fb-bb7e-96b123742f0d
debugLogEnabled=false
clientMachineName=appsrvr79.keyexample.com
```

Figure 31: Configure Logging for the Keyfactor Bash Orchestrator



Tip: Log messages for remote control targets are written to the standard Linux syslog. The location of these will vary depending on the system OS. Log messages for the orchestrator's communication with the remote control targets are included in the primary orchestrator log (described above). It can be helpful to look in both places when troubleshooting an issue with a remote control target.

2.4.4.2 Start the Keyfactor Bash Orchestrator Service

The keyfactor-bash-orchestrator service runs on the Keyfactor Bash Orchestrator machine and controls SSH public key discovery and management tasks for the orchestrator machine itself and target servers it controls. The service should start automatically at the conclusion of the installation.

The service on Linux is added as keyfactor-bash-orchestrator, so when referencing it in startup commands, it should be referenced by this name, including case. For example:

```
systemctl start [stop] [restart] [status] keyfactor-bash-orchestrator.service
```

Once you have finished the SSH server group and server configuration using the Keyfactor Command Management Portal and have completed a scan of the configured servers, you can view discovered keys and logons in the Keyfactor Command Management Portal and then begin using the management features. See *SSH* in the *Keyfactor Command Reference Guide* for information on using the SSH features.

2.5 Troubleshooting

The following error conditions and general troubleshooting tips may be helpful in resolving issues with the Keyfactor orchestrators. Generally speaking, issues are often related to trusts of root and intermediate certificates, firewall challenges, or insufficient permissions for the service account running the orchestrator service.

Validate Management Portal Configuration

Things to check in the Management Portal include:

- Is the last seen time for the orchestrator on the Orchestrator Management page in the Management Portal within the last few minutes (see *Orchestrator Management* in the *Keyfactor Command Reference Guide*)? Most orchestrators send a heartbeat to Keyfactor Command every 5 minutes, so this date should at most be 5 minutes out of date if the orchestrator is operating

correctly.



Tip: Orchestrator control targets for the Keyfactor Bash Orchestrator do not appear on the Orchestrator Management page, so for a remote server that's not operating as expected, this would be the orchestrator that is controlling the target.

Orchestrator Management [?]

Orchestrators are used to perform tasks directly on computers and communicate information back to Keyfactor. These tasks may include synchronizing certificates and templates from remote configuration tenant CAs or non-domain-joined CAs, reporting inventory of Java Keystores, installing certificates into Java Keystores, and requesting certificates on Macintosh clients.

Field: Comparison: Value:

☐ Include Disapproved

	Client Machine	Identity	Platform	Version	Status	Last Seen	Capabilities	Cert Store Type ...	Orchestrator Blu...
<input type="checkbox"/>	webservr85-12.keyexample.com	service-account-universal-orchestrator	NET	11.0.0.0	New	9/21/2023, 7:50:39 PM	WinCert, CitrixAdc, ...	IISU	
<input type="checkbox"/>	appsrvr76-13.keyexample.com	service-account-universal-orchestrator	NET	11.0.0.0	Approved	9/21/2023, 7:46:32 PM	F5-SL-REST, CitrixA...	CitrixAdc, F5-SL-RE...	
<input type="checkbox"/>	appsrvr185	service-account-universal-orchestrator	NET	11.0.0.0	Approved	9/13/2023, 8:23:33 PM	SSL		

Figure 32: Orchestrator Management for a Keyfactor Bash Orchestrator

- Has the orchestrator been approved on the Orchestrator Management page in the Management Portal (see *Orchestrator Management* in the *Keyfactor Command Reference Guide*)?
- Is there a sync schedule set to run frequently for the orchestrator (SSH), remote control target (SSH), or certificate store? Sync schedules for certificates stores are automatically disabled if inventory jobs are failing.
- For the Keyfactor Bash Orchestrator:
 - Has the server record for the orchestrator or remote control target been created under SSH Server Manager on the Servers tab in the Management Portal (see *SSH Servers* in the *Keyfactor Command Reference Guide*)?

Server Manager [?]

Manage SSH server groups, servers, and logons.

Server Groups **Servers** Logons Users

Field: Comparison: Value:

ADD	EDIT	Hostname	Owner	Group Name	Orchestrator	Management Status	Sync Schedule
		appsrvr162.keyexample.com	keyexample\jsmith	Server Group Three	appsrvr163-SSH.rhel.keyexample.com	Inventory and Publish Policy	Every 6 minutes
		appsrvr163	keyexample\jsmith	Server Group Two	appsrvr163-SSH.rhel.keyexample.com	Inventory and Publish Policy	Every 3 minutes
		appsrvr79.keyexample.com	keyexample\jsmith	Server Group Three	appsrvr163-SSH.rhel.keyexample.com	Inventory and Publish Policy	Every 6 minutes
		appsrvr80.keyexample.com	keyexample\jsmith	Server Group Two	appsrvr163-SSH.rhel.keyexample.com	Inventory and Publish Policy	Every 3 minutes

Figure 33: Orchestrator Management for a Keyfactor Bash Orchestrator

- Does the server record for the orchestrator or remote control target in the Management Portal have the correct hostname or IP address? If the name or IP address is incorrect, sync jobs will fail.

- Is the server record for the remote control target in the Management Portal associated with the correct orchestrator? If the control target is associated with the wrong orchestrator, you may be looking at the wrong log files (see [Debug Logging and Error Messages below](#)) for troubleshooting information.

Debug Logging and Error Messages

It is often helpful to enable debug logging on the orchestrator. For information on configuring this, see the specific orchestrator chapters.

Once the logging is set at debug or trace level, it can be helpful to watch the logs live while activity is going on. On Linux, you can do this with *tail* (or a similar tool) to watch the log in real time. For example:

```
tail -f /opt/keyfactor-bash-orchestrator/logs/keyfactorbash-orchestrator-log.txt
```

```
tail -f /opt/keyfactor/orchestrator/logs/Log.txt
```

On Windows, there are also tools with tail-like functionality. Notepad++, for example, has this functionality built in.

Some messages in the KeyfactorUniversal Orchestrator log include a correlation ID that helps to identify log messages that originated from the same request. The correlation ID is a randomly generated GUID that often appears just after the date in the log entry (**B0C4946E-DB3B-4404-8080-79AFF260DE4E** in the following example) and is the same for all log messages for the given request until the request completes.

```
2023-09-15 18:19:00.3780 B0C4946E-DB3B-4404-8080-79AFF260DE4E 230398 Keyfactor.Orches-
trators.JobExecutors.OrchestratorJobExecutor [Debug] - Running job extension for job with Id
'b0c4946e-db3b-4404-8080-79aff260de4e'
2023-09-15 18:19:00.3780 B0C4946E-DB3B-4404-8080-79AFF260DE4E 230398 Keyfactor.Ex-
tensions.Orchestrator.WindowsCertStore.WinCert.Inventory [Trace] - Entered 'ProcessJob' method.
2023-09-15 18:19:00.3780 B0C4946E-DB3B-4404-8080-79AFF260DE4E 230398 Keyfactor.Ex-
tensions.Orchestrator.WindowsCertStore.WinCert.Inventory [Trace] - {"JobCan-
celled":-
false,"ServerError":null,"JobHistoryID":230398,"RequestStatus":1,"ServerUserName":"keyexample\\svc_
kyforch","ServerPassword":"*****","JobConfigurationProperties":{"spnwithport":false,"WinRm
Protocol":"https","WinRm Port":"5986","ServerUsername":"keyexample\\svc_kyforch","Server-
UseSsl":true,"sniflag":0},"UseSSL":true,"JobTypeID":"00000000-0000-0000-0000-
000000000000","JobID":"b0c4946e-db3b-4404-8080-79aff260de4e","Cap-
ability":"CertStores.WinCert.Inventory","LastInventory":[],"CertificateStoreDetails":
{"ClientMachine":"web-
srvr93.keyexample.com","StorePath":"My","StorePassword":"*****","Type":117}}
2023-09-15 18:19:00.3780 B0C4946E-DB3B-4404-8080-79AFF260DE4E 230398
Keyfactor.Extensions.Orchestrator.WindowsCertStore.WinCert.Inventory [Trace] - Establishing runtime on client machine: websrvr93.keyexample.com
2023-09-15 18:19:00.3780 B0C4946E-DB3B-4404-8080-79AFF260DE4E 230398
```

```

Keyfactor.Extensions.Orchestrator.WindowsCertStore.PsHelper [Trace] - Entered 'GetClientPsRunspace'
method.
2023-09-15 18:19:00.3780 B0C4946E-DB3B-4404-8080-79AFF260DE4E 230398 Keyfactor.Ex-
tensions.Orchestrator.WindowsCertStore.PsHelper [Trace] - Creating remote session at: https://web-
srvr93.keyexample.com:5986/wsman
2023-09-15 18:19:00.3780 B0C4946E-DB3B-4404-8080-79AFF260DE4E 230398
Keyfactor.Extensions.Orchestrator.WindowsCertStore.PsHelper [Trace] - Credentials Specified
[Messages removed for clarity]
2023-09-15 18:19:00.7389 B0C4946E-DB3B-4404-8080-79AFF260DE4E 230398 Keyfactor.Ex-
tensions.Orchestrator.WindowsCertStore.WinCert.Inventory [Trace] - Connecting to remote server websr-
vr93.keyexample.com failed with the following error message : acquiring creds with username only
failed No credentials were supplied, or the credentials were unavailable or inaccessible SPNEGO
cannot find mechanisms to negotiate For more information, see the about_Remote_Troubleshooting Help
topic.
    at System.Management.Automation.Runspaces.AsyncResult.EndInvoke()
    at System.Management.Automation.Runspaces.Internal.RunspacePoolInternal.EndOpen(IAsyncResult asyn-
cResult)
    at System.Management.Automation.Runspaces.Internal.RemoteRunspacePoolInternal.Open()
    at System.Management.Automation.Runspaces.RunspacePool.Open()
    at System.Management.Automation.RemoteRunspace.Open()
    at Keyfactor.Extensions.Orchestrator.WindowsCertStore.WinCert.Inventory.PerformInventory(Invent-
oryJobConfiguration config, SubmitInventoryUpdate submitInventory)

2023-09-15 18:19:00.7389 B0C4946E-DB3B-4404-8080-79AFF260DE4E 230398 Keyfactor.Ex-
tensions.Orchestrator.WindowsCertStore.WinCert.Inventory [Warn] - Inventory job failed for Site 'My'
on server 'websrvr93.keyexample.com' with error: 'Connecting to remote server websr-
vr93.keyexample.com failed with the following error message : acquiring creds with username only
failed No credentials were supplied, or the credentials were unavailable or inaccessible SPNEGO
cannot find mechanisms to negotiate For more information, see the about_Remote_Troubleshooting Help
topic.
    at System.Management.Automation.Runspaces.AsyncResult.EndInvoke()
    at System.Management.Automation.Runspaces.Internal.RunspacePoolInternal.EndOpen(IAsyncResult asyn-
cResult)
    at System.Management.Automation.Runspaces.Internal.RemoteRunspacePoolInternal.Open()
    at System.Management.Automation.Runspaces.RunspacePool.Open()
    at System.Management.Automation.RemoteRunspace.Open()
    at Keyfactor.Extensions.Orchestrator.WindowsCertStore.WinCert.Inventory.PerformInventory(Invent-
oryJobConfiguration config, SubmitInventoryUpdate submitInventory)

2023-09-15 18:19:00.7389 B0C4946E-DB3B-4404-8080-79AFF260DE4E 230398 Keyfactor.Orches-
trators.JobExecutors.OrchestratorJobExecutor [Debug] - Finished running job extension for job with
Id 'b0c4946e-db3b-4404-8080-79aff260de4e'

```

Some messages to look for include:

- This message (or similar—text varies slight from orchestrator to orchestrator) indicates that the orchestrator has not yet been approved in the Keyfactor Command Management Portal:

```
2021-07-29 09:01:28.5957 Keyfactor.Orchestrators.JobEngine.SessionJobExecutor [Info] - Agent has not yet been registered with CMS. Trying again in 30 minutes.
```

After approving the orchestrator in the Management Portal, you can restart the orchestrator service to avoid waiting 30 minutes for the next automated retry.

- Some log message spell out the problem pretty clearly. For example, this message from the Java Agent log:

```
2021-07-29 09:00:02.437 [Scheduler_Worker-1] ERROR com.css_security.cms.JksUtilities - Keystore /opt/apps/myapp.jks loaded as type JKS but the provided password is incorrect
```

In this case, the certificate store configuration in the Management Portal is not using the correct password for the store.

- This series of messages in the Java Agent log indicates that the stored credentials file for the Java Agent is no longer useable:

```
2021-07-01 11:24:59.292 [Scheduler_Worker-1] ERROR com.css_security.cms.apache.http.HttpClientFactory - Given final block not properly padded. Such issues can arise if a bad key is used during decryption.
2021-07-01 11:24:59.313 [Scheduler_Worker-1] ERROR com.css_security.cms.apache.http.HttpClientFactory - Could not decrypt credentials file at config/install.creds
2021-07-01 11:24:59.313 [Scheduler_Worker-1] INFO com.css_security.cms.apache.http.HttpClientFactory - Your machine key may have changed. Reencrypt credentials using local machine key.
2021-07-01 11:24:59.313 [Scheduler_Worker-1] INFO com.css_security.cms.apache.http.HttpClientFactory - Generate new credentials by running included cms-credential-encryptor utility
2021-07-01 11:24:59.313 [Scheduler_Worker-1] INFO com.css_security.cms.apache.http.HttpClientFactory - Try 1. Trying again in 30 seconds
```

The credentials file can be recreated to return the Java Agent to functionality (see Appendix A—Generate New Credentials for the Java Agent).

- This series of messages indicates that the Keyfactor Command server is unreachable:

```
2021-07-29 11:59:02.1003 Keyfactor.Orchestrators.JobEngine.SessionClient [Error] - Unable to heartbeat:
```



```

2021-07-29 11:59:02.1003 Keyfactor.Orchestrators.JobEngine.SessionClient [Trace] - Leaving CMSSessionClient.Heartbeat
2021-07-29 11:59:02.1006 Keyfactor.Orchestrators.JobEngine.SessionJobExecutor [Debug] - Heartbeat success: Unreachable
2021-07-29 11:59:02.1006 Keyfactor.Orchestrators.JobEngine.SessionJobExecutor [Warn] - Heartbeat endpoint unreachable. Trying again later

```

This could indicate a network or firewall issue.

- A series of messages similar to this for the Universal Orchestrator can indicate a problem retrieving the CRL for the certificate used to secure the Keyfactor Command server if you've chosen to connect to Keyfactor Command over SSL:

```

2022-09-14 11:15:06.1830 Keyfactor.Orchestrators.JobEngine.SessionJobExecutor [Error] - Error in SessionManager: Unable to register session.
The SSL connection could not be established, see inner exception.
The remote certificate is invalid because of errors in the certificate chain: RevocationStatusUnknown, OfflineRevocation

```

Confirm that the CRLs for the CA that issued the certificate and the remaining CAs in the chain are valid. Confirm that they are available in a location that is accessible to the orchestrator server (e.g. a location other than LDAP if the orchestrator is installed on a server not joined to a domain in the forest where they were issued). If you're using delta CRLs and hosting them on an IIS website using the default CRL suffix as a naming convention (+), be sure to enable double escaping in IIS to allow the orchestrator to retrieve the CRL files containing a plus in the file name.

- Messages that look like errors during SSL scanning are common as attempts are made to connect to TLS endpoints and connections fail or are refused. This is part of the process of testing whether an SSL endpoint is responding and then whether there is a certificate there. Most of these message exist at TRACE level, so monitoring at DEBUG rather than TRACE level will eliminate these messages if they become overwhelming. For example:

```

2022-09-12 10:56:32.3948 EE033BD9-421A-44CA-89BC-10C86949B506 166937 Tls13Probe [Trace] - Endpoint 192.168.216.87:443 returned status 'ExceptionDownloading' with exception 'System.ArgumentException': The specified nonce is not a valid size for this algorithm. (Parameter 'nonce')
2022-09-12 10:56:39.0567 EE033BD9-421A-44CA-89BC-10C86949B506 166937 Tls13Probe [Trace] - Endpoint 192.168.216.158:443 returned status 'ConnectionRefused' with exception 'System.Net.Sockets.SocketException': An existing connection was forcibly closed by the remote host.
2022-09-12 10:57:23.4727 EE033BD9-421A-44CA-89BC-10C86949B506 166937 Tls13Probe [Trace] - Connection to 192.168.216.87:443 failed
2022-09-12 10:57:24.3345 EE033BD9-421A-44CA-89BC-10C86949B506 166937 a [Trace] - Endpoint 192.168.216.211:443 returned status 'ExceptionDownloading' with exception

```

```
'Keyfactor.Orchestrators.SSL.Pipeline.Exceptions.ConnectionGoneException': Read zero bytes on a blocking read
2022-09-12 10:57:57.9505 EE033BD9-421A-44CA-89BC-10C86949B506 166937 b [Trace] - Endpoint 192.168.216.96:443 returned status 'SslRefused' with exception 'Keyfactor.Orchestrators.SSL.Pipeline.Exceptions.TlsAlertException': Got TLS alert during TLS handshake: Alert level 2, Alert description 70
```

Heartbeat

You should see a heartbeat message similar to the following in the log every 5 minutes:

- Keyfactor Universal Orchestrator on Windows:

```
2023-09-12 11:01:16.4598 Keyfactor.Orchestrators.JobEngine.SessionJobExecutor [Debug] - Existing session found. Heartbeating...
```

- Keyfactor Bash Orchestrator:

```
Tue Aug 11 18:06:02 UTC 2023 [Debug]: Performing orchestrator heartbeat...
```

- Keyfactor Java Agent on Linux:

```
2023-07-30 00:52:11.662 [Scheduler_Worker-1] DEBUG com.css_security.cms.agents.jobs.SessionManager - Existing session found. Heartbeating...
```

This is the orchestrator checking in with the Keyfactor Command server to see if there are any jobs. If this message is missing, it could indicate that the heartbeat service is not running.

If you're running the Keyfactor Bash Orchestrator, you can see the heartbeat service as a separate entity. Execute this command on the orchestrator in the command shell as root:

```
systemctl status keyfactor-bash-orchestrator.service
```

Output from this command should look something like that shown in [Figure 34: Status for the Keyfactor Bash Orchestrator Service](#). If you don't see heartbeat.sh in the output, the heartbeat service is not running.

Successful Inventory and Policy Publishing

In this snippet you see a successful inventory showing keys found for the Linux users ginag and svc_greenchicken and a logon found for the Linux user zadams with no key found. You see that the server is configured in *inventory and publish policy mode*, since after performing the inventory the server went through the steps of publishing logons and keys. Details about these are not written to the log.

```
Tue Aug 11 18:07:45 UTC 2020 [Debug]: Sending request to 'https://key-
factor.keyexample.com/KeyfactorAgents/SshSync/1/Configure' with payload '{"SessionToken":
"5451f7aa-4fd5-4bf5-a563-2e4f7bd3ed3f", "JobId": "b835bde8-8174-447a-b351-810e582148c0"}'
Tue Aug 11 18:07:45 UTC 2020 [Debug]: Configure Response for job with id 'b835bde8-8174-447a-b351-
810e582148c0': {"Host-
name": "appsrvr79.keyexample.com", "InventoryCompleteEndpoint": "/SshSync/1/InventoryComplete",
"Port": 22, "AuditId": 7642, "JobCancelled": false, "Result": {"Status": 1, "Error": null}}
Tue Aug 11 18:07:46 UTC 2020 [Debug]: Using sshd_config file '/etc/ssh/sshd_config' on server
'appsrvr79.keyexample.com' for job with id 'b835bde8-8174-447a-b351-810e582148c0'
Tue Aug 11 18:07:46 UTC 2020 [Info]: Beginning local inventory job on server 'appsr-
vr79.keyexample.com' for job with id 'b835bde8-8174-447a-b351-810e582148c0'
Tue Aug 11 18:07:49 UTC 2020 [Debug]: Sending request to 'https://key-
factor.keyexample.com/KeyfactorAgents/SshSync/1/InventoryComplete' with payload '{"Status": 2, "Res-
ults": [{
"user": "ginag",
"lastlogon": "",
"keys": [ "ssh-rsa AAAAB3NzaC1yc2EAAAAD[truncated for display purposes]9M5v16f Gina G. Gant" ]
},{
"user": "zadams",
"lastlogon": "",
"keys": []
},{
"user": "svc_greenchicken",
"lastlogon": "",
"keys": [ "ssh-rsa AAAAB3NzaC1yc2EAAAAD[truncated for display purposes]vicWhZ0d John W. Smith" ]
}], "SessionToken": "5451f7aa-4fd5-4bf5-a563-2e4f7bd3ed3f", "JobId": "b835bde8-8174-447a-b351-
810e582148c0"}'
Tue Aug 11 18:07:49 UTC 2020 [Debug]: Inventory Complete Response for job with id 'b835bde8-8174-
447a-b351-810e582148c0' on server 'appsrvr79.keyexample.com': {"SshDesiredState": [{"User-
name": "ginag", "Keys": ["ssh-rsa AAAAB3NzaC1yc2EAAAAD[truncated for display purposes]9M5v16f Gina G.
Gant"]}, {"Username": "zadams", "Keys": []}, {"Username": "svc_greenchicken", "Keys": ["ssh-rsa AAAAB3Nz-
aC1yc2EAAAAD[truncated for display purposes]vicWhZ0d John W. Smith"]}], "Result": {"Status": 1, "Er-
ror": null}}
Tue Aug 11 18:07:49 UTC 2020 [Info]: Enforcing publish policy on server 'appsrvr79.keyexample.com'
for job with id 'b835bde8-8174-447a-b351-810e582148c0'
Tue Aug 11 18:07:52 UTC 2020 [Info]: Publishing logons on local server 'appsrvr79.keyexample.com'
```

```
for job with id 'b835bde8-8174-447a-b351-810e582148c0'
Tue Aug 11 18:07:52 UTC 2020 [Info]: Published logons successfully on server 'appsr-
vr79.keyexample.com' for job 'b835bde8-8174-447a-b351-810e582148c0'
Tue Aug 11 18:07:52 UTC 2020 [Info]: Publishing keys on local server 'appsrvr79.keyexample.com' for
job with id 'b835bde8-8174-447a-b351-810e582148c0'
Tue Aug 11 18:07:54 UTC 2020 [Info]: Published keys successfully on server 'appsr-
vr79.keyexample.com' for job 'b835bde8-8174-447a-b351-810e582148c0'
Tue Aug 11 18:07:54 UTC 2020 [Debug]: Sending request to 'https://key-
factor.keyexample.com/KeyfactorAgents/SshSync/1/Complete' with payload '{"SessionToken": "5451f7aa-
4fd5-4bf5-a563-2e4f7bd3ed3f", "JobId": "b835bde8-8174-447a-b351-810e582148c0", "Status": 2}'
Tue Aug 11 18:07:54 UTC 2020 [Info]: Execution of 'b835bde8-8174-447a-b351-810e582148c0' on server
'appsrvr79.keyexample.com' complete.
```

Validate Service Account Logon

During installation of the orchestrator, a local Linux user account should be created automatically as an identity under which the orchestrator service will operate. This allows the orchestrator to run as a non-root user. On servers on which you install the orchestrator directly, the following Linux user account is created:

```
keyfactor-bash
```

On servers configured as remote control targets, the following Linux user account is created:

```
keyfactor-bash-orchestrator-svc
```

You can validate that the user has been created and has the correct configuration by reviewing the `/etc/passwd` file.

In a command shell, output the content of the `/etc/passwd` file to the screen:

```
cat /etc/passwd
```

In the output from this command, look for the entry for the `keyfactor-bash` or `keyfactor-bash-orchestrator-svc` user. It will look similar to one of these:

```
keyfactor-bash:x:978:976:./home/keyfactor-bash:/bin/bash
keyfactor-bash-orchestrator-svc:x:112:65534:./opt/keyfactor-bash-orchestrator-
client:/bin/bash
```

On the remote control target server, you should find an entry in the `sshd_config` file that directs the service account logon over to the install path for the client to find the `authorized_keys` file for the service account user, like so:

```
Match User keyfactor-bash-orchestrator-svc
AuthorizedKeysFile /opt/keyfactor-bash-orchestrator-client/authorized_keys
```

On both the orchestrator and remote control target servers, you should find a file in the `/etc/sudoer.d` directory named for the service name of the orchestrator or remote control target user (`keyfactor-bash` or `keyfactor-bash-orchestrator-svc`) and containing a list of commands the orchestrator is allowed to execute as root. For example:

```
keyfactor-bash appsrvr79.keyexample.com = (root) NOPASSWD: /bin/ls
keyfactor-bash appsrvr79.keyexample.com = (root) NOPASSWD: /bin/cat
keyfactor-bash appsrvr79.keyexample.com = (root) NOPASSWD: /usr/bin/test
keyfactor-bash appsrvr79.keyexample.com = (root) NOPASSWD: /bin/rm
keyfactor-bash appsrvr79.keyexample.com = (root) NOPASSWD: /usr/bin/tee
keyfactor-bash appsrvr79.keyexample.com = (root) NOPASSWD: /bin/touch
keyfactor-bash appsrvr79.keyexample.com = (root) NOPASSWD: /bin/chmod
keyfactor-bash appsrvr79.keyexample.com = (root) NOPASSWD: /bin/chown
keyfactor-bash appsrvr79.keyexample.com = (root) NOPASSWD: /usr/bin/gpasswd
keyfactor-bash appsrvr79.keyexample.com = (root) NOPASSWD: /usr/sbin/usermod
keyfactor-bash appsrvr79.keyexample.com = (root) NOPASSWD: /bin/sed
keyfactor-bash appsrvr79.keyexample.com = (root) NOPASSWD: /usr/bin/flock
keyfactor-bash appsrvr79.keyexample.com = (root) NOPASSWD: /bin/mkdir
keyfactor-bash appsrvr79.keyexample.com = (root) NOPASSWD: /usr/sbin/adduser
```

Validate Remote Control Target Public Key

The orchestrator connects to the remote control targets it is managing using SSH with a public key pair. On the orchestrator, the key pair is stored in the `.ssh` directory under the directory where the orchestrator is installed. By default, this is:

```
/opt/keyfactor-bash-orchestrator/.ssh
```

Both the private key (`id_rsa`) and public key (`id_rsa.pub`) are found here.

In a command shell, output the content of the public key file to the screen:

```
cat id_rsa.pub
```

On the remote control target, the public key of the key pair is stored in the `authorized_keys` file for the remote control target service account, which is found in the remote control install path. By default, this is:

```
/opt/keyfactor-bash-orchestrator-client
```

In a command shell, output the content of the `authorized_keys` file to the screen:

```
cat authorized_keys
```

Compare the public key string from the remote control target `authorized_keys` file to the public key string from the orchestrator `id_rsa.pub` file. They should match exactly. If they do not match, the remote control target is not using the correct public key, which will cause connection attempts made to it from the orchestrator to fail.



Tip: You should also see in the `.ssh` directory on the orchestrator a file named by hostname (e.g. `appsvr80.keyexample.com`) for each of the remote control targets managed by the orchestrator. These contain a list of known, trusted host key stores. If this file has not been created for your remote control target, connectivity to the target is failing at a very fundamental level (before the stage of a public key mismatch). See [Firewall Ports on page 137](#).

Keyfactor Bash Orchestrator Log Messages

If the orchestrator is managing more than one server (remote control targets), it can be difficult to interpret the logs, because the orchestrator operates in a multi-threaded manner and log messages for jobs with different servers will be mixed together. Find a message related to the job you're interested in and look for the ID for that job. Then look for all other messages referencing that ID.

Look for error messages in the log. These should appear with the word *Error* in brackets just after the date like so:

```
Tue Aug 11 19:14:33 UTC 2020 [Error]: Error occurred during job with id 'b835bde8-8174-447a-b351-810e582148c0' on server 'appsvr79.keyexample.com': An error occurred attempting to configure the job 'b835bde8-8174-447a-b351-810e582148c0'
```

This particular message doesn't tell you very much except that this job was unable to complete for some reason. If you look at the debug messages that appear immediately before and after the error message, they may provide more information.

This message indicates that the orchestrator was unable to make an SSH connection to the remote control target named in the message:

```
Mon Aug 10 23:36:10 UTC 2020 [Error]: Error occurred during job with id '3f04f552-05fd-4c90-b3b1-edec70878bb' on server 'appsvr80.ubuntu.keyexample.com': Unable to connect to 'appsvr80.ubuntu.keyexample.com' on port '22' via SSH
```

This could happen for a number of reasons. Perhaps the hostname configured for the remote target is incorrect. Perhaps the public key on the remote target is incorrect. It can be helpful in this case to check the Linux syslog on the orchestrator for more context on the message. For example, this set of messages from the Linux syslog reveals that the public key on the target is invalid in some fashion:

```
Aug 11 13:03:04 appsvr158 keyfactor-bash[29417]: Testing 'keyfactor-bash-orchestrator-svc' on
server 'appsvr80.keyexample.com' via SSH for job with id 'eeabd541-b9d2-46d2-a215-9cb99fed4adc'...
Aug 11 13:03:04 appsvr158 keyfactor-bash-orchestrator.sh[932]: keyfactor-bash-orchestrator-
svc@appsvr80.keyexample.com: Permission denied (publickey).
Aug 11 13:03:30 appsvr158 keyfactor-bash[29486]: Error occurred during job with id 'eeabd541-b9d2-
46d2-a215-9cb99fed4adc' on server 'appsvr80.keyexample.com': Unable to connect to 'appsvr-
vr80.keyexample.com' on port '22' via SSH
```

For information on troubleshooting public key issues with remote control targets, see [Validate Remote Control Target Public Key on page 140](#). For more information on troubleshooting remote control target issues in general, see [Remote Control Target Logs below](#). For information on what successful inventory and publish policy log messages look like, see [Successful Inventory and Policy Publishing on page 138](#).

Remote Control Target Logs

Unlike on the orchestrator itself, where you can enable debug logging to see a more detailed picture of what's going on when the orchestrator attempt to connect or run a job, on a remote control target, the only logs available are the SSH logs showing attempts by the orchestrator to make a remote connection into the target and then the commands the orchestrator runs from an SSH perspective. These logs are found in the Linux system log where SSH logs are consolidated. The name and location of this will vary by operating system, but it is often found in /var/log by default (*auth.log* or *secure* is common). A large number of entries are generated in the log on a successful connection for inventory or inventory and policy publishing, so it can be difficult to interpret the logs.

In these logs you can check to see if the orchestrator is successfully making an SSH connection. If it isn't, you may see some messages that will help determine why it isn't. If it's successfully making the initial connection but then failing further along in the process, this log may also help reveal that. Perhaps one of the commands that the service account needs to run isn't in the expected path, for example.

When the orchestrator first connects to the remote control target, the log entries on the target will look something like:

```
Aug 11 17:36:51 appsvr80 sshd[95543]: Accepted publickey for keyfactor-bash-orchestrator-svc from
10.4.3.158 port 47778 ssh2: RSA SHA256:u5zNB4UEoPNcax5p4fBbkkWaoiWq6AcEkA65XdzUkM4
Aug 11 17:36:51 appsvr80 sshd[95543]: pam_unix(sshd:session): session opened for user keyfactor-
bash-orchestrator-svc by (uid=0)
Aug 11 17:36:51 appsvr80 systemd-logind[656]: New session 13019 of user keyfactor-bash-orches-
trator-svc.
Aug 11 17:36:51 appsvr80 systemd: pam_unix(systemd-user:session): session opened for user
keyfactor-bash-orchestrator-svc by (uid=0)
```



```
Aug 11 17:36:51 appsrvr80 sudo: keyfactor-bash-orchestrator-svc : TTY=unknown ; PWD=/opt/keyfactor-bash-orchestrator-client ; USER=root ; COMMAND=/bin/cat /etc/ssh/sshd_config
```

An inventory of an `authorized_keys` file for a user will appear as a series of entries, something like:

```
Aug 11 18:11:28 appsrvr164 sudo: keyfactor-bash-orchestrator-svc : TTY=unknown ; PWD=-  
=/opt/keyfactor-bash-orchestrator-client ; USER=root ; COMMAND=/bin/test -f /home/j-  
smith/.ssh/authorized_keys  
Aug 11 18:11:28 appsrvr164 sudo: keyfactor-bash-orchestrator-svc : TTY=unknown ; PWD=-  
=/opt/keyfactor-bash-orchestrator-client ; USER=root ; COMMAND=/bin/ls -l /home/j-  
smith/.ssh/authorized_keys  
Aug 11 18:11:28 appsrvr164 sudo: keyfactor-bash-orchestrator-svc : TTY=unknown ; PWD=-  
=/opt/keyfactor-bash-orchestrator-client ; USER=root ; COMMAND=/bin/cat /home/j-  
smith/.ssh/authorized_keys
```

Removal of a rogue key on a remote control target under management (in *inventory and publish policy* mode) will appear as a series of entries where the `authorized_keys` file is removed, recreated and repopulated with any valid keys (none in this case), like:

```
Aug 12 09:01:24 appsrvr80 sudo: keyfactor-bash-orchestrator-svc : TTY=unknown ; PWD=/opt/keyfactor-bash-orchestrator-client ; USER=root ; COMMAND=/usr/bin/test -f /home/jsmith/.ssh/authorized_keys  
Aug 12 09:01:24 appsrvr80 sudo: keyfactor-bash-orchestrator-svc : TTY=unknown ; PWD=/opt/keyfactor-bash-orchestrator-client ; USER=root ; COMMAND=/bin/rm /home/jsmith/.ssh/authorized_keys  
Aug 12 09:01:25 appsrvr80 sudo: keyfactor-bash-orchestrator-svc : TTY=unknown ; PWD=/opt/keyfactor-bash-orchestrator-client ; USER=root ; COMMAND=/usr/bin/test -f /home/jsmith/.ssh/authorized_keys  
Aug 12 09:01:25 appsrvr80 sudo: keyfactor-bash-orchestrator-svc : TTY=unknown ; PWD=/opt/keyfactor-bash-orchestrator-client ; USER=root ; COMMAND=/usr/bin/test -d /home/jsmith/.ssh  
Aug 12 09:01:25 appsrvr80 sudo: keyfactor-bash-orchestrator-svc : TTY=unknown ; PWD=/opt/keyfactor-bash-orchestrator-client ; USER=root ; COMMAND=/usr/bin/touch /home/jsmith/.ssh/authorized_keys  
Aug 12 09:01:25 appsrvr80 sudo: keyfactor-bash-orchestrator-svc : TTY=unknown ; PWD=/opt/keyfactor-bash-orchestrator-client ; USER=root ; COMMAND=/bin/chmod 640 /home/jsmith/.ssh/authorized_keys  
Aug 12 09:01:25 appsrvr80 sudo: keyfactor-bash-orchestrator-svc : TTY=unknown ; PWD=/opt/keyfactor-bash-orchestrator-client ; USER=root ; COMMAND=/bin/chown jsmith: /home/jsmith/.ssh/authorized_keys  
Aug 12 09:01:25 appsrvr80 sudo: keyfactor-bash-orchestrator-svc : TTY=unknown ; PWD=/opt/keyfactor-bash-orchestrator-client ; USER=root ; COMMAND=/usr/bin/flock /home/jsmith/.ssh/authorized_keys  
echo  
Aug 12 09:01:25 appsrvr80 sudo: keyfactor-bash-orchestrator-svc : TTY=unknown ; PWD=/opt/keyfactor-bash-orchestrator-client ; USER=root ; COMMAND=/usr/bin/tee -a /home/jsmith/.ssh/authorized_keys
```

General Errors

Below are some possible errors you might encounter and some suggested troubleshooting tips or solutions.

Unable to connect to the remote server

Here is an example of some very similar errors you might see when trying to connect to a target machine to inventory a certificate store or execute a management or discovery job on a certificate store:

```
Error: Unable to connect to the remote server - No connection could be made because
the target machine actively refused it 192.196.12.12:443 (80131500)
```

```
Error: Unable to complete the inventory operation. One or more errors occurred.
An error occurred while sending the request.
Unable to connect to the remote server
A connection attempt failed because the connected party did not properly respond after
a period of time, or established connection failed because connected host has failed
to respond 192.168.12.12:443 (80131500)
```

```
Error: Unable to connect to the remote server (80131509)
```

```
Error occurred during job with id 'b5e93ae6-df3b-4b36-9640-b41146db6d36' on server
'appsrvr13.keyexample.com': Unable to connect to 'appsrvr13.keyexample.com' on port
'22' via SSH
```

Messages of this type are generally the result of the target server being inaccessible. This might happen if the server was turned off or in maintenance mode. Perhaps there is a network problem routing to that server. If the certificate store has never worked in Keyfactor Command, perhaps there is a typo in the server name configuration.

Request Entity Too Large

You may encounter this error when doing an inventory of an IIS certificate store:

```
Error: Response status code does not indicate success: 413 (Request Entity Too Large).
(80131500)
```

This is an indication that the certificate store you are inventorying contains more certificates (or more precisely, the certificates add up to a total number of bytes greater) than IIS on the Keyfactor Command server is configured to accept. To resolve this, adjust the values on the IIS server that control the upload limits. For example, the *maxAllowedContentLength*. See *SSL Network Operations: Monitoring Network Scan Jobs with View Scan Details* in the *Keyfactor Command Reference Guide* on fine tuning SSL monitoring for more information.

IIS Error 403.16

You may receive a 403.16 error while trying to authenticate an orchestrator to Keyfactor Command using certificate authentication. On the face of it, this error indicates that the chain for the certificate you're using to authenticate is not trusted by the Keyfactor Command server. First,

check to be sure that your certificate is trusted by the Keyfactor Command server. But if your certificate is fully trusted and you're still getting this error, what then?

This error can indicate that the trusted root store on the Keyfactor Command server contains a certificate that is not a root certificate (for example, an intermediate certificate is accidentally in the root store). To check this, open the Local Computer certificates MMC on the Keyfactor Command server, drill down to Certificates under the Trusted Root Certificate Authorities and scan for any certificates where the *Issued To* does not match the *Issued By*. Remove any certificates you find like this.

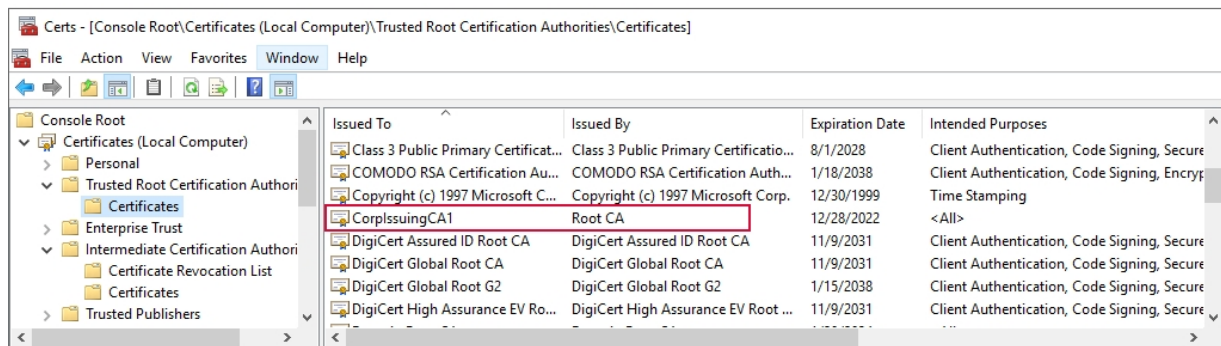


Figure 35: Certificate Incorrectly in the Trusted Root Certificate Store

Error: An attempt was made to load a program with an incorrect format.

If you receive an error similar to the following:

```
Could not load file or assembly 'Keyfactor.CAClient.Microsoft.DCOM, Version=2.1.1.0, Culture=neutral, PublicKeyToken=0ed89d330114ab09'. An attempt was made to load a program with an incorrect format.
```

This may indicate that the Keyfactor Universal Orchestrator was installed without the Microsoft Visual C++ Redistributable x64 required to manage certificates from remote Microsoft CAs (see [System Requirements on page 7](#)).

Error: The remote certificate is invalid because of errors in the certificate chain

If you receive an error similar to the following (some portions of message removed for clarity):

```
2023-02-15 11:54:27.6600 Keyfactor.Orchestrators.JobEngine.SessionJobExecutor [Error] - Error in SessionManager: Unable to register session.
```

```
The SSL connection could not be established, see inner exception.
```

```
The remote certificate is invalid because of errors in the certificate chain: RevocationStatusUnknown, OfflineRevocation
```

This may indicate that the Keyfactor Universal Orchestrator cannot access the CRL(s) for the SSL certificate used to secure the Keyfactor Command server (see [System Requirements on page 7](#)).

To check this:

1. Enable at least debug level logging (see [Configure Logging for the Universal Orchestrator on page 77](#)).
2. Either wait for the orchestrator to attempt to register again, or restart the orchestrator service (see [Start the Universal Orchestrator Service on page 80](#)) to force an immediate attempt to register.
3. Look in the logs for a log message similar to the following (referencing your Keyfactor Command server name):

```
2023-02-15 12:08:14.6076 Keyfactor.Orchestrators.Core.Http.KeyfactorHttpClient  
[Debug] - Sending request to  
'https://keyfactor.keyexample.com/KeyfactorAgents/Session/Register'
```

4. Visit the referenced URL (`https://keyfactor.keyexample.com/KeyfactorAgents/Session/Register`) in a browser on the orchestrator server. This should give you a response of:

```
The requested resource does not support http method 'GET'.
```

5. In the browser, view details for the certificate (the exact method for this will vary depending on the browser) and check the *CRL Distribution Points* field in the certificate.

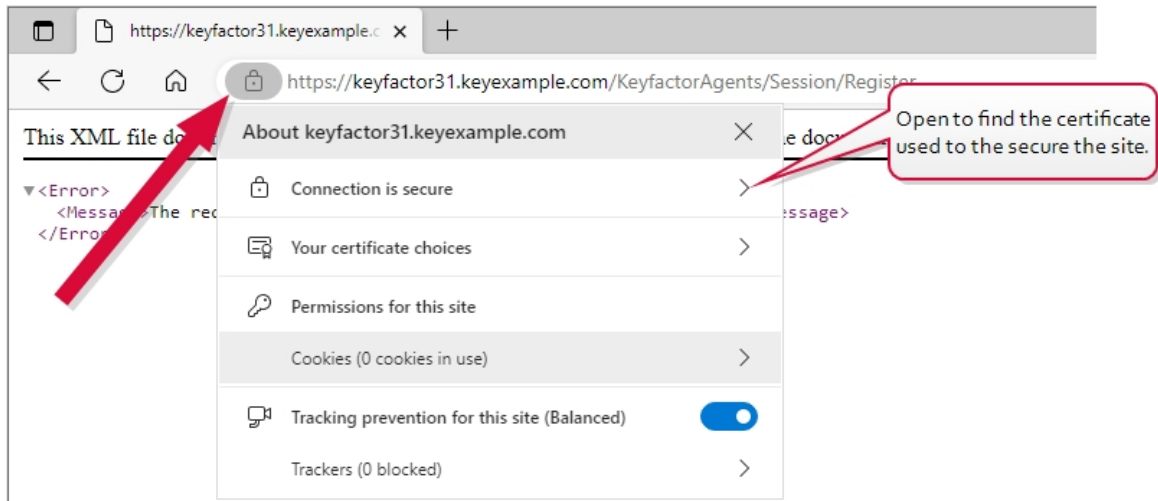


Figure 36: Find the Certificate for the Keyfactor Command Web Site

6. In the same browser on the orchestrator server, attempt to browse to the URL for the CRL (assuming it's a URL).
7. If the CRL downloads without error, then likely CRL access is not the issue. Open the CRL and check the *Next update* date to see if it's in the past (indicating the CRL is out of date).

Note: CRL checks are done on port 80 since the CRL lookup is part of the validation of the server's SSL certificate. This means the CRLs need to be available at an http URL. The CRL file that is retrieved is signed by the CA, so although the network communication is not encrypted when retrieving it, the data that is being validated can't be tampered with (because it is signed).

Tip: The Keyfactor Universal Orchestrator can be installed without checking the CRL of the Keyfactor Command if desired. Use the `-NoRevocationCheck` option for the Windows orchestrator (see [-NoRevocationCheck on page 33](#)), the `--no-revocation-check` option for the Linux orchestrator (see [--no-revocation-check on page 46](#)), or the `AppSettings__CheckServerCertificateRevocation` option for the orchestrator in Linux containers (see [Table 1: Linux Container Parameters](#)).

Remote Management Helpful Tools

The following tips are useful for servers being remotely managed using PowerShell remoting and WinRM.

- Test the connection from the orchestrator server to the remotely managed Windows server:

```
Test-netConnection -ComputerName <target> -Port 5986 or Test-netConnection -
ComputerName <target> -Port 5985
```

- Test PS Session from the orchestrator server to the remotely managed server:

```
Enter-PSSession -ComputerName <target>
```

- On the remotely managed server, check what's available:

```
winrm enumerate winrm/config/listener
```

- Enable secure winrm:

```
winrm quickconfig -transport:https
```

- Check the secure winrm port certificate:

```
gci -path cert:\localmachine\my | ft -property thumbprint,subject,NotBefore,NotAfter
```

2.6 Appendices

- [Appendix - Generate New Credentials for the Java Agent below](#)
- [Appendix - Set up the Universal Orchestrator to Use Client Certificate Authentication via a Reverse Proxy: Citrix ADC on page 150](#)
- [Appendix - Set up the Universal Orchestrator to Use Client Certificate Authentication with Certificates Stored in Active Directory on page 162](#)
- [Appendix - Set up the Universal Orchestrator to Use a Forwarding Proxy on page 176](#)

2.6.1 Appendix - Generate New Credentials for the Java Agent

Under some circumstances, you may find it necessary to generate new credentials for the Java Agent. This can happen, for example, if you make a change to the hostname of the machine on which the Java Agent is running. The credentials file stores the username and password for the service account user that allows the Java Agent to communicate with Keyfactor Command—the identity for the agent (see [Create Service Accounts for the Java Agent on page 96](#))—encrypted with the hostname to prevent the file from being used on machines other than the machine on which the agent has been installed.

Log messages that indicate a new credentials file is needed look similar to the following:

```
2020-10-02 15:21:43.307 [Scheduler_Worker-1] ERROR com.css_security.cms.apache.http.HttpClientFactory
- Given final block not properly padded. Such issues can arise if a bad key is used during decryption.
2020-10-02 15:21:43.307 [Scheduler_Worker-1] ERROR com.css_security.cms.apache.http.HttpClientFactory
- Could not decrypt credentials file at config\install.creds
2020-10-02 15:21:43.526 [Scheduler_Worker-1] INFO com.css_security.cms.apache.http.HttpClientFactory
- Your machine key may have changed. Reencrypt credentials using local machine key.
```

```
2020-10-02 15:21:43.541 [Scheduler_Worker-1] INFO com.css_security.cms.apache.http.HttpClientFactory
- Generate new credentials by running included cms-credential-encryptor utility
```

To generate a new credentials file on Windows:

1. Open a command prompt using the “Run as administrator” option.
2. Change directories to the directory in which the Java Agent is installed. By default, this is:
`C:\Program Files\Keyfactor\Keyfactor Java Agent`
3. Type the following command to generate a new credentials file in the current directory:
`java -jar CSS.CMS.CredentialEncryptor.jar encode-basic install.creds`
4. Locate the existing credentials file in the config directory under the installed directory. By default, this is:
`C:\Program Files\Keyfactor\Keyfactor Java Agent\config`
5. Delete or name off the existing install.creds file in the config directory and copy the new install.creds file from the base install directory to the config directory.
6. Restart the Java Agent service (see [Start the Keyfactor Java Agent Service on page 116](#)).
7. Review the log messages to confirm that credential errors are no longer occurring (see [Configure Logging for the Java Agent on page 113](#)).

To generate a new credentials file on Linux:

1. Open a command shell.
2. Change directories to the directory in which the Java Agent is installed. By default, this is:
`/opt/keyfactor-java-agent`
3. As a user with rights to write to the current directory (or use sudo), type the following command to generate a new credentials file in the current directory:
`java -jar CSS.CMS.CredentialEncryptor.jar encode-basic install.creds`
4. Locate the existing credentials file in the config directory under the installed directory. By default, this is:
`/opt/keyfactor-java-agent/config`
5. Delete or name off the existing install.creds file in the config directory and copy the new install.creds file from the base install directory to the config directory.
6. Restart the Java Agent service (see [Start the Keyfactor Java Agent Service on page 116](#)).
7. Review the log messages to confirm that credential errors are no longer occurring (see [Configure Logging for the Java Agent on page 113](#)).

2.6.2 Appendix – Set up the Universal Orchestrator to Use Client Certificate Authentication via a Reverse Proxy: Citrix ADC

The Keyfactor Universal Orchestrator can be configured to support TLS termination at a reverse proxy or network edge device such as a Citrix ADC (a.k.a. NetScaler) or F5. The orchestrator supports using either basic authentication or client certificate authentication between the orchestrator and the Keyfactor Command orchestrator endpoint. When a client certificate is used for the segment between the orchestrator and the reverse proxy, the reverse proxy authenticates the orchestrator with the provided client certificate and then sends the certificate on to Keyfactor Command as an added request header to authenticate the orchestrator to Keyfactor Command with the original certificate. The orchestrator is authenticated and authorized to make the connection to Keyfactor Command in one of two ways:

- A username and password with appropriate permissions within Keyfactor Command are stored on the reverse proxy and presented to Keyfactor Command as part of the request. Basic authentication is used to authenticate the reverse proxy to IIS on the Keyfactor Command server. The same credentials provide authorization for the orchestrator in Keyfactor Command. The original certificate from the orchestrator, provided in a request header, authenticates the orchestrator to the Keyfactor Command orchestrator endpoint.
- A username and password with appropriate permissions within Keyfactor Command are stored in IIS on the Keyfactor Command. In this scenario, a second client certificate residing on the reverse proxy is used to authenticate the reverse proxy to IIS on the Keyfactor Command server. The basic authentication credentials provide authorization for the orchestrator in Keyfactor Command and the original client certificate from the request header provides authentication. The basic authentication credentials are stored locally and do not need to travel over the network. The original certificate from the orchestrator, provided in a request header, authenticates the orchestrator to the Keyfactor Command orchestrator endpoint.

The following instructions cover one method of configuring a Citrix ADC device to support these.



Tip: The following provides instructions for using the Citrix ADC GUI interface to create the appropriate configuration. The same configuration could be accomplished using the command line interface.

Complete the following steps and then configure the orchestrator to enable client certificate authentication as per the installation instructions (see [--client-auth-certificate \(Client Certificate Authentication\) on page 43](#) or [Install the Universal Orchestrator on Windows on page 25](#)).

Define Rewrite Actions in Citrix

Create the following two rewrite actions.



Tip: If you're using a second client certificate to authenticate the proxy to Keyfactor Command, you only need to create the first of these actions.

Capture the client certificate from the orchestrator:

1. In the Citrix ADC GUI, browse to *AppExpert > Rewrite > Actions*.
2. On the Rewrite Actions page, click **Add**.
3. On the Create Rewrite Action page, enter a **Name** for the action that will take the certificate received from the orchestrator and convert it to PEM format (e.g. CaptureClientCert).
4. Give the action a *Type* of *INSERT_HTTP_HEADER*.
5. Give the action a *Header Name* (e.g. NS-ClientCert). Be sure to make a note of this header name. You will need it later when you configure certificate authentication for the orchestrator.
6. Enter an *Expression* to convert the client authentication certificate to PEM format:
`CLIENT.SSL.CLIENT_CERT.TO_PEM`
7. Enter *Comments* if desired and click **OK** to save the action.

Store basic authentication credentials to authenticate the proxy to IIS on the Keyfactor Command server and provide authorization information:

1. Click **Add** to add another action.
2. On the Create Rewrite Action page, enter a **Name** for the action that will send the basic authentication credentials for the orchestrator to Keyfactor Command (e.g. SendServiceCreds).
3. Give the action a *Type* of *INSERT_HTTP_HEADER*.
4. Give the action a *Header Name* of *Authorization*.
5. Enter an *Expression* to send Base64-encoded basic authentication credentials to the Keyfactor Command server (where *service@keyexample.com* and *MySecurePassword* are the correct service name and password for your environment):
`"Basic "+("service@keyexample.com"+" ":"MySecurePassword").B64ENCODEM`
6. Enter *Comments* if desired and click **OK** to save the action.

Define Rewrite Policies in Citrix

Create the following two rewrite policies.



Tip: If you're using a second client certificate to authenticate the proxy to Keyfactor Command, you only need to create the first of these policies.

Put the client certificate from the orchestrator in the header:

1. In the Citrix ADC GUI, browse to *AppExpert > Rewrite > Policies*.
2. On the Rewrite Policies page, click **Add**.

3. On the Create Rewrite Policy page, enter a **Name** for the policy that will confirm that a certificate has been received from the orchestrator and run the action to convert it to PEM format (e.g. NS-GetCert).
4. Give the policy the **Action** you created in the previous section to capture the client authentication certificate (e.g. CaptureClientCert).
5. Define a **Log Action** if desired.
6. Set the **Undefined-Result Action** to *-Global-undefined-result-action-*.
7. Enter an *Expression* to validate that the client authentication certificate has been received from the orchestrator:

```
CLIENT.SSL.CLIENT_CERT.EXISTS
```

8. Enter *Comments* if desired and click **OK** to save the policy.

Send the basic authentication credentials to the Keyfactor Command server:

1. Click **Add** to add another policy.
2. On the Create Rewrite Policy page, enter a **Name** for the policy that will send the basic authentication credentials for the orchestrator to the Keyfactor Command server (e.g. NS-SendCreds).
3. Give the policy the **Action** you created in the previous section to send the basic authentication credentials (e.g. SendServiceCreds).
4. Define a **Log Action** if desired.
5. Set the **Undefined-Result Action** to *-Global-undefined-result-action-*.
6. Enter an *Expression* to confirm that the authorization header does not already exist in the request header:

```
HTTP.REQ.HEADER("Authorization").EXISTS.NOT
```

7. Enter *Comments* if desired and click **OK** to save the policy.

Define a Responder Policy in Citrix

Create the following responder policy.

Validate that the client certificate presented by the orchestrator was issued by the specified issuing CA:

1. In the Citrix ADC GUI, browse to *AppExpert > Responder > Policies*.
2. On the Responder Policies page, click **Add**.

3. On the Create Responder Policy page, enter a **Name** for the policy that will validate that the certificate received from the orchestrator was issued by the correct CA (e.g. NS-Validatelssuer).
4. Select an **Action** of *Reset*.
5. Define a **Log Action** if desired.
6. Do not configure an **AppFlow Action**.
7. Set the **Undefined-Result Action** to *-Global-undefined-result-action-*.
8. Enter an *Expression* to confirm that the certificate received from the orchestrator was issued from the correct issuing CA (where *CorpIssuingCA* is the logical name of your CA):

```
CLIENT.SSL.CLIENT_CERT.ISSUER.CONTAINS("CorpIssuingCA").NOT
```



Tip: Connections from the orchestrator will fail if the client authentication certificate was issued by any CA other than the one configured here. You can use AND logic to add more than one CA. For example:

```
CLIENT.SSL.CLIENT_CERT.ISSUER.CONTAINS("CorpIssuingCA1").NOT &&  
CLIENT.SSL.CLIENT_CERT.ISSUER.CONTAINS("CorpIssuingCA2").NOT
```

With this expression, certificates issued from either one of these CAs would be accepted.

9. Enter *Comments* if desired and click **OK** to save the policy.

Update the Virtual Server in Citrix



Important: Once you modify the virtual server to require certificates for authentication, many other Keyfactor Command transactions will no longer function if they are sharing the same virtual server. Be sure that you are using a separate virtual server for incoming requests to /KeyfactorAgents on the Keyfactor Command server versus other types of requests. The following instructions refer to setting all policies on a single load balancing virtual server, but your configuration may include multiple virtual servers of other types, which may require slight modifications to these instructions.

Modify the configuration for your load balancing virtual server that is used for Keyfactor Command KeyfactorAgent requests as follows.

Configure the Citrix device to authenticate the orchestrator using its client certificate:

1. In the Citrix ADC GUI, browse to *Traffic Management > Load Balancing > Virtual Servers*.
2. On the Virtual Servers page, select your virtual server and click **Edit**.
3. In the SSL Parameters section, click to edit, check the **Client Authentication** box, and set the **Client Certificate** dropdown to **Mandatory**.

Associate the two rewrite policies.



Tip: If you're using a second client certificate to authenticate the proxy to Keyfactor Command, you only need to associate the first of these policies.

Configure the policy to include the certificate in the header:

1. On the Virtual Servers page, under Advanced Settings expand Policies.
2. In the Policies section, click the plus to add a new policy.
3. On the Choose Type page, select **Choose Policy***Rewrite* and **Choose Type***Request* and click **Continue**.
4. On the Choose Type page, click **Add Binding**.
5. On the Policy Binding page, click the **Select Policy** field and on the Rewrite Policies page select the radio button for the rewrite policy you created to capture the client authentication certificate (e.g. NS-GetCert). Click **Select** to save the selection.
6. On the Policy Binding page, set a **Priority** of 110.
7. Set **Goto Expression** to *Next*.
8. Set **Invoke LabelType** to *None*.
9. Click **Bind** to save the binding.

Configure the policy to send the basic authentication credentials to the Keyfactor Command server:

1. On the Choose Type page for Rewrite Request, click **Add Binding**.
2. On the Policy Binding page, click the **Select Policy** field and on the Rewrite Policies page select the radio button for the rewrite policy you created to send the service account credentials to the Keyfactor Command server (e.g. NS-SendCreds). Click **Select** to save the selection.
3. On the Policy Binding page, set a **Priority** of 120.
4. Set **Goto Expression** to *Next*.
5. Set **Invoke LabelType** to *None*.
6. Click **Bind** to save the binding.
7. Click **Close** to return to the virtual server settings page.

Associate the responder policy:

1. On the Virtual Servers page, in the Policies section, click the plus to add a new policy.
2. On the Choose Type page, select **Choose Policy Responder** and **Choose Type Request** and click **Continue**.
3. On the Choose Type page, click **Add Binding**.
4. On the Policy Binding page, click the **Select Policy** field and on the Responder Policies page select the radio button for the responder policy you created to validate the issuer of the client authentication certificate (e.g. NS-ValidateIssuer). Click **Select** to save the selection.
5. On the Policy Binding page, set a **Priority** of 100.
6. Set **Goto Expression** to *END*.
7. Set **Invoke LabelType** to *None*.
8. Click **Bind** to save the binding.
9. Click **Close** to return to the virtual server settings page.

Configure Keyfactor Command for Client Certificate Authentication

Once you have all the components configured on Citrix, you're ready to configure Keyfactor Command to enable client certificate authentication for the orchestrators. Once you do this, all orchestrators connecting to this instance of Keyfactor Command will be required to provide a certificate to authenticate. If you have some orchestrators deployed that do not support certificate authentication (e.g. Java agents), you will need to design a solution with multiple Keyfactor Command servers to support multiple authentication types. Contact your Keyfactor representative for assistance with this.

To configure Keyfactor Command to require client certificate authentication for orchestrators:

1. On the Keyfactor Command server, open the Keyfactor Configuration Wizard.
2. In the Certificate Authentication section of the Orchestrators tab, check the **Enabled** box.
3. In the **Certificate Authentication HTTP Header** field, enter the *Header Name* you gave to the rewrite action you created to capture the certificate from the orchestrator (e.g. NS-ClientCert). Keyfactor Command uses the certificate supplied in this header to identify the orchestrator attempting to authenticate.
4. In the **Certificate Authentication Username** and **Certificate Authentication Password** fields, enter the credentials for an Active Directory service account for the orchestrator(s).



Tip: The service account entered here does not need to match the service account entered on the Citrix device to authenticate the orchestrator.

5. Click **Verify Configuration** and **Apply Configuration**.

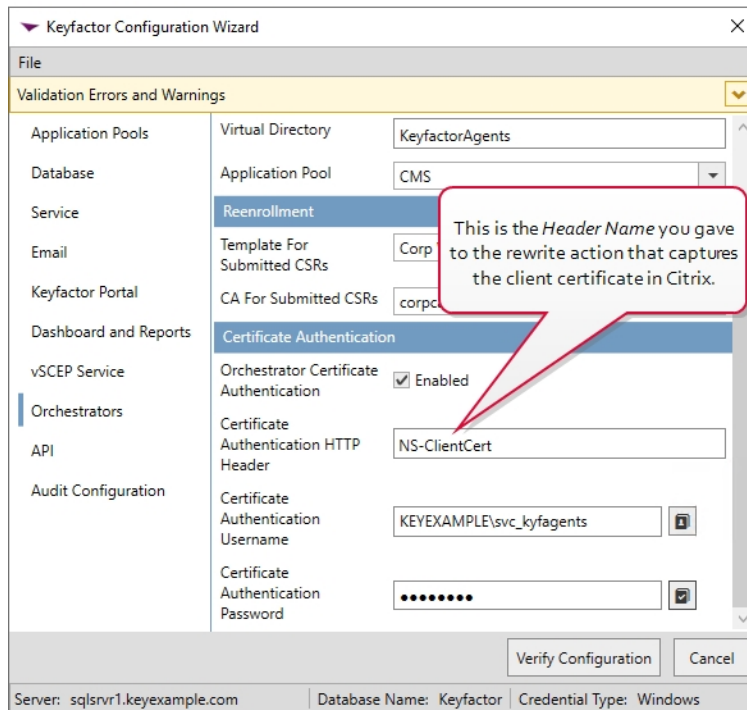


Figure 37: Configure Keyfactor Command for Client Certificate Authentication

Configure IIS to Provide Credentials When a Second Client Certificate is Used to Authenticate the Proxy

If you have opted to configure the Citrix ADC device to use a client certificate to authenticate from the device to the Keyfactor Command server instead of submitting basic authentication credentials from the device, you will need to configure IIS on the Keyfactor Command server to recognize the client certificate for authentication and then use basic authentication credentials on the Keyfactor Command server to provide authorization to Keyfactor Command. In addition, you will need to configure Keyfactor Command to force it to use the client certificate from the orchestrator stored in the header to authenticate the orchestrator, not the client certificate presented by the proxy in the second hop of the transaction.

Install the Required Windows Module

On your Keyfactor Command server, install the following additional module:

- *IIS Client Certificate Mapping Authentication* (rather than *Client Certificate Mapping Authentication*)



Tip: It's fine to install both *IIS Client Certificate Mapping Authentication* and *Client Certificate Mapping Authentication*, but the former is what's needed for this solution.

If you have more than one Keyfactor Command server with separated roles, this only needs to be installed on the server accepting traffic to the /KeyfactorAgents web application.

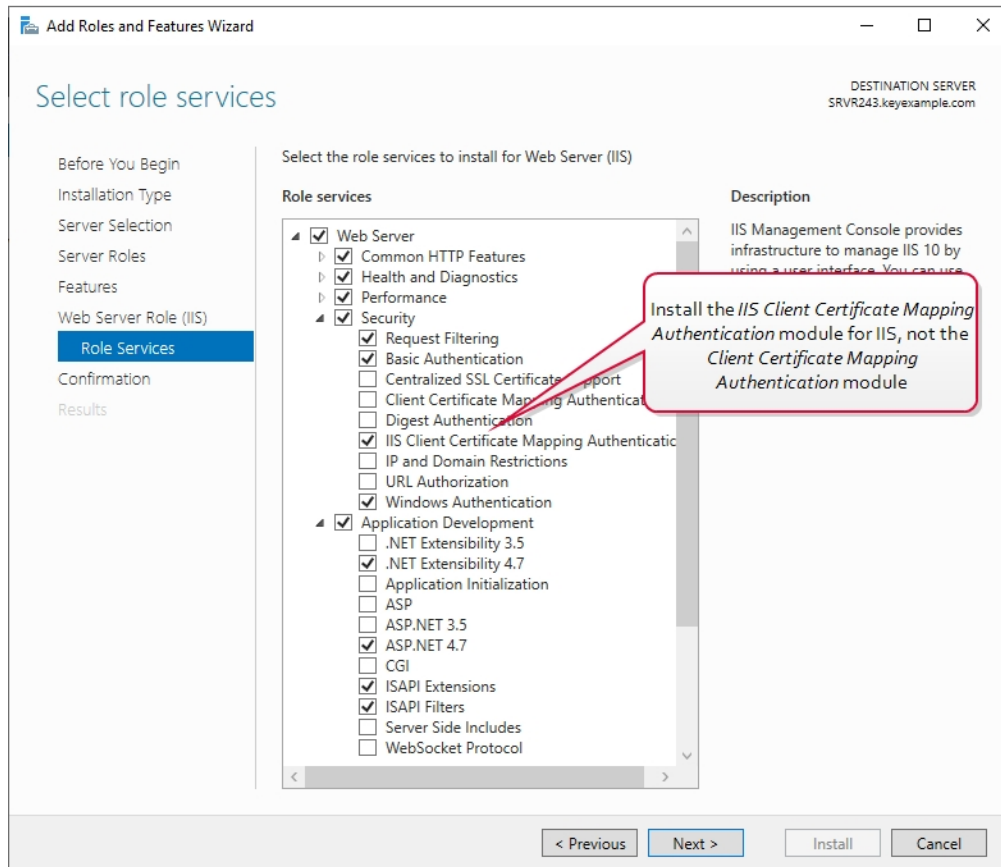


Figure 38: IIS Module for Client Certificate Authentication

The PowerShell command to install the appropriate module is :

```
Add-WindowsFeature Web-Cert-Auth
```

Configure Certificate Authentication and SSL Settings in IIS

Make the following changes in the IIS Management console on the Keyfactor Command server:

1. In the IIS Management console, highlight the server name on the left and open Authentication. Make sure *Anonymous Authentication* is the only enabled method.

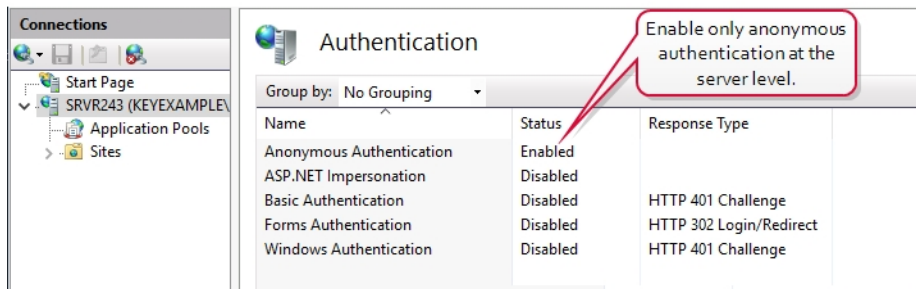


Figure 39: Configure only Anonymous Authentication at the Server Level in IIS

2. In the IIS Management console, drill down into sites and into the Default Web Site (or other web site if your Keyfactor Command instance has been installed in an alternate web site). Under the Default Web Site, locate the KeyfactorAgents application and open Authentication for this. Disable all the authentication methods shown here.

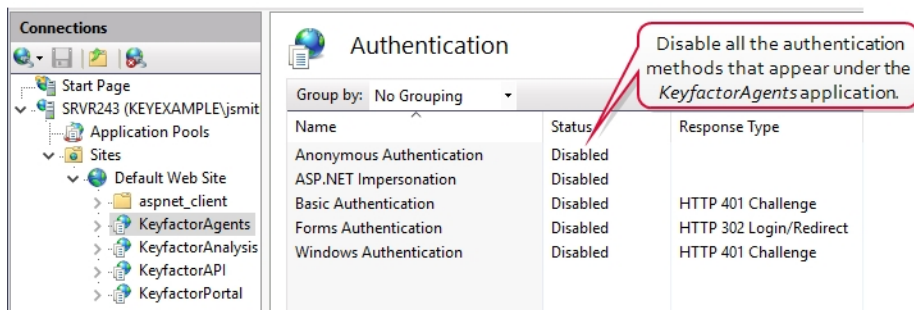




Figure 40: Disable Authentication Methods at the Application Level in IIS

 **Tip:** If your KeyfactorAgents endpoint is running on a standalone server with no other Keyfactor roles, you should also disable all authentication methods at the Default Web Site level as in step two. If your server holds other Keyfactor roles, leave this in the default configuration with Anonymous being the only authentication method enabled as in step one.

3. In the IIS Management console, open SSL Settings for the KeyfactorAgents application. Check the **Require SSL** box and select either **Require** or **Accept** for *Client certificates*.

 **Important:** Only selected **Require** if your are only using orchestrators that support client certificate authentication and plan to configure all of them for certificate authentication.

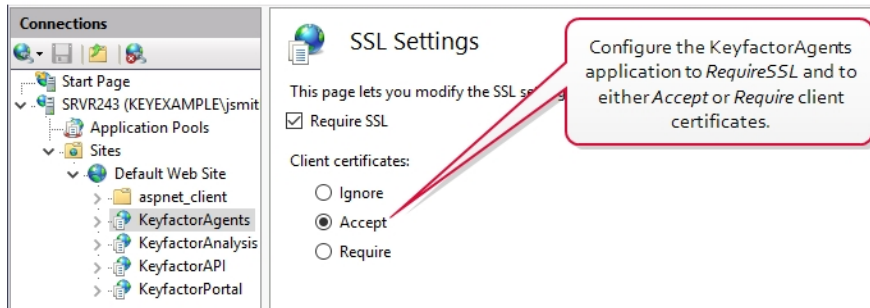



Figure 41: Configure SSL Settings in IIS for Client Certificate Authentication


 **Tip:** If your KeyfactorAgents endpoint is running on a standalone server with no other Keyfactor roles, you may also configure your server to **Require** or **Accept** for *Client certificates* at the Default Web Site level. It is good security practice to check the *Require SSL* box. If your KeyfactorAgents endpoint is running on a server with other Keyfactor roles, you do not need to accept client certificates at this level and should not require them at this level.

Configure Basic Authentication Credentials in IIS

Make the following changes in the IIS Management console on the Keyfactor Command server:

1. In the IIS Management console, drill down to the Default Web Site (or other web site if your Keyfactor Command instance has been installed in an alternate web site). In the Default Web Site, open the Configuration Editor tool.
2. In the Configuration Editor tool at the Default Web Site level, browse to:

system.webServer/security/authentication/iisClientCertificateMappingAuthentication

 **Important:** Don't be tempted to configure this setting only at the application level (KeyfactorAgents) rather than at the Default Web Site level. It will only work if configured at the Default Web Site level and then enabled at the application level.

3. In the configurations for IIS Client Certificate Mapping Authentication, set the *defaultLogoutDomain* to your forest root. Set the *manyToOneCertificateMappingsEnabled* option to *True* and the *oneToOneCertificateMappingEnabled* option to *False*. Click the dots to the right of the *manyToOneMappings* setting to open details for this setting.

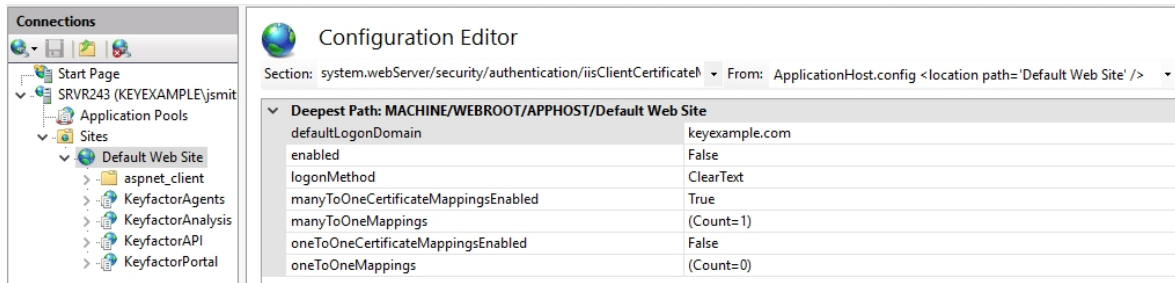


Figure 42: Configure IIS Client Certificate Mapping Authentication for the Default Web Site

- In the Collection Editor for the manyToOneMappings, click **Add** and enter appropriate values for the properties. The service account entered here will be used as the identity in Keyfactor Command of all orchestrators that authenticate via client certificate.

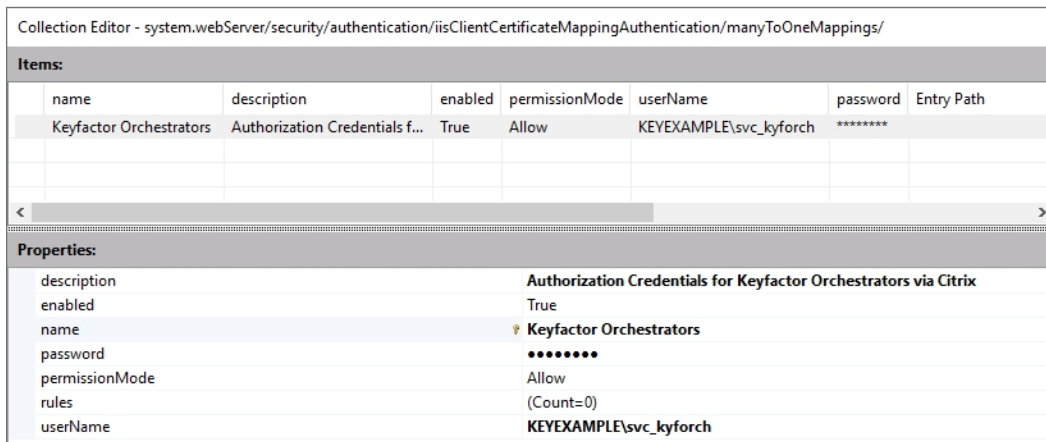


Figure 43: Configure Authorization Credentials for Keyfactor Orchestrators

- In the IIS Management console, drill down into sites and into the Default Web Site (or other web site if your Keyfactor Command instance has been installed in an alternate web site). Under the Default Web Site, locate the KeyfactorAgents application and open the Configuration Editor tool for it.
- In the Configuration Editor tool at the KeyfactorAgents application level, browse to:

system.webServer/security/authentication/iisClientCertificateMappingAuthentication

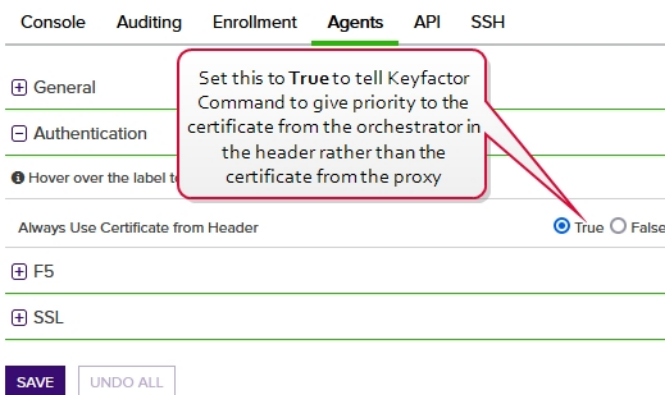
Enable the mapping authentication option at this level. The configuration should have replicated down from the Default Web Site level.

Configure the Keyfactor Command Application Setting to Use the Certificate from the Header

When the orchestrator is configured to use a client certificate to authenticate to a proxy and then the proxy is configured to use a separate client certificate to authenticate to the Keyfactor Command server, authentication to the Keyfactor Command application should be done using the original certificate from the orchestrator, not the certificate inserted in the process at the proxy level. This is done by including the original certificate from the orchestrator in the request header to Keyfactor Command. To assure that Keyfactor Command gives priority to this certificate and not the certificate the proxy uses to authenticate, set the Keyfactor Command authentication application setting *Always Use Certificate from Header* to *True*.

Application Settings


Application Settings define operational parameters for the system.



Console Auditing Enrollment **Agents** API SSH

☒ General

☐ Authentication

 Hover over the label to

Always Use Certificate from Header ☒ True ☐ False

☒ F5

☒ SSL

SAVE UNDO ALL

Figure 44: Configure Application Setting in Keyfactor Command to use the Header Certificate



Tip: In some proxy configurations, the proxy may be unable to negotiate the client certificate handshake with IIS. IIS won't ask directly for a client certificate, and if, during the handshake, the proxy doesn't send one, the client authentication will fail. If this occurs, you may need to enable client certificate negotiation at a lower level below IIS. To do this:

1. On the Keyfactor Command server, open a command prompt using the "Run as administrator" option.
2. Execute the following command to output the current configuration for SSL certificate bindings:

```
netsh http show sslcert
```

Output from this command will look something like this (you may see multiple sections if you have multiple web sites on the server):



SSL Certificate bindings:

```
-----  
IP:port : 0.0.0.0:443  
Certificate Hash : 649dfa6df693583f609af499fe4237f2c1d64224  
Application ID : {4dc3e181-e14b-4a21-b022-59fc669b0914}  
Certificate Store Name : My  
Verify Client Certificate Revocation : Enabled  
Verify Revocation Using Cached Client Certificate Only : Disabled  
Usage Check : Enabled  
Revocation Freshness Time : 0  
URL Retrieval Timeout : 0  
Ctl Identifier : (null)  
Ctl Store Name : (null)  
DS Mapper Usage : Enabled  
Negotiate Client Certificate : Disabled  
Reject Connections : Disabled  
Disable HTTP2 : Not Set  
Disable QUIC : Not Set  
Disable TLS1.2 : Not Set  
Disable TLS1.3 : Not Set  
Disable OCSP Stapling : Not Set  
Disable Legacy TLS Versions : Not Set
```

3. Look at the value for the *Negotiate Client Certificate* setting for the web site on which Keyfactor Command is installed. If the value is *Disabled*, retrieve from the output the values for the *IP:port*, *Certificate Hash*, and *Application ID*.
4. Execute the following commands to remove and re-add the *IP:port* with *Negotiate Client Certificate* enabled (referencing the correct values for *ipport*, *certhash*, and *appid*):

```
netsh http delete sslcert ipport=0.0.0.0:443  
netsh http add sslcert ipport=0.0.0.0:443  
certhash=649dfa6df693583f609af499fe4237f2c1d64224 appid={4dc3e181-e14b-4a21-  
b022-59fc669b0914} clientcertnegotiation=enable
```

5. Execute the *show* command again to confirm that the setting is now shown as enabled.
6. Restart the IIS services (*iisreset*) and try the certificate authentication again.

2.6.3 Appendix - Set up the Universal Orchestrator to Use Client Certificate Authentication with Certificates Stored in Active Directory

The Keyfactor Universal Orchestrator can be configured to support client certificate authentication by acquiring a certificate for the Keyfactor Command connect service account user or machine

account of the orchestrator and storing it in Active Directory and then providing the associated Active Directory credentials to authenticate to Keyfactor Command. This has an advantage over the reverse proxy method (see [Appendix - Set up the Universal Orchestrator to Use Client Certificate Authentication via a Reverse Proxy: Citrix ADC on page 150](#)) in that a username and password do not need to be stored anywhere (other than in Active Directory). This method does have a heavier reliance on Active Directory.

Complete the following steps and then configure the orchestrator to enable client certificate authentication as per the installation instructions (see [-ClientCertificate \(Client Certificate Authentication\) on page 31](#) or [Install the Universal Orchestrator on a Linux Server on page 39](#)).



Tip: Using this method, you do not necessarily need to configure certificate authentication in Keyfactor Command, unlike for the proxy method (see [Appendix - Set up the Universal Orchestrator to Use Client Certificate Authentication via a Reverse Proxy: Citrix ADC on page 150](#)), since the certificate authentication is occurring at the IIS layer before the request reaches Keyfactor Command. You may wish to configure certificate authentication in Keyfactor Command to allow Keyfactor Command to monitor certificate authentication and to support automated certificate renewal (see [Register a Client Certificate Renewal Extension on page 88](#)). If you enable certificate authentication in Keyfactor Command with this method, you will need to provide a value in the *Certificate Authentication HTTP Header* field. This header field is used to pass the certificate contents to Keyfactor Command command in instances when the certificate is not used directly (such as in the reverse proxy scenario). The value is required when configuring certificate authentication in Keyfactor Command, but since for this method you do not need to extract the certificate from the header, the value you set here is unimportant.



Important: If you do opt to enable certificate authentication in Keyfactor Command, be aware that this will force all orchestrators to use certificate authentication when communicating with Keyfactor Command on the configured server.

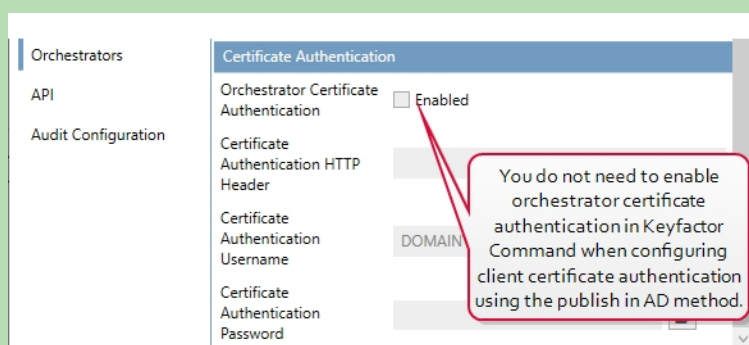


Figure 45: Client Certificate Authentication with AD Storage Does Not Require Certificate Authentication Configuration in Keyfactor Command



Note: The following instructions assume that your Keyfactor Command server is already installed and configured with an SSL certificate that is trusted in your environment. If this is not the case, this will also need to be done.

Install the Required Windows Module

On your Keyfactor Command server, install the following additional module:

- *Client Certificate Mapping Authentication* (rather than *IIS Client Certificate Mapping Authentication*)

If you have more than one Keyfactor Command server with separated roles, this only needs to be installed on the server accepting traffic to the /KeyfactorAgents web application.

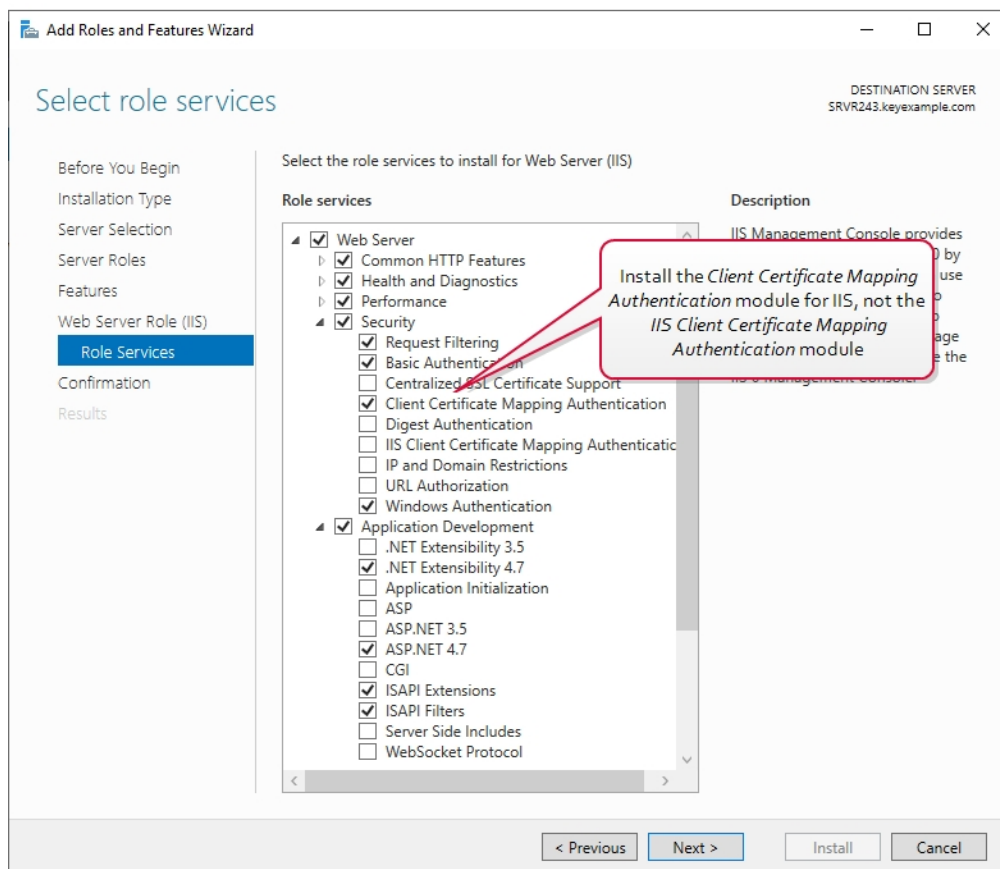


Figure 46: IIS Module for Client Certificate Authentication with AD Storage

The PowerShell command to install the appropriate module is :

```
Add-WindowsFeature Web-Client-Auth
```

Configure Certificate Authentication and SSL Settings in IIS

Make the following changes in the IIS Management console on the Keyfactor Command server:

1. In the IIS Management console, highlight the server name on the left and open Authentication. Change the status of *Active Directory Client Certificate Authentication* to **Enabled**.

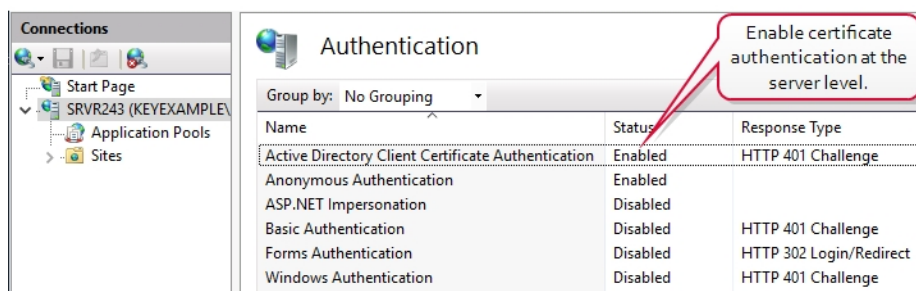


Figure 47: Configure Client Certificate Authentication at the Server Level in IIS

2. In the IIS Management console, drill down into sites and into the Default Web Site (or other web site if your Keyfactor Command instance has been installed in an alternate web site). Under the Default Web Site, locate the KeyfactorAgents application and open Authentication for this. Disable all the authentication methods shown here. The *Active Directory Client Certificate Authentication* method does not appear here.

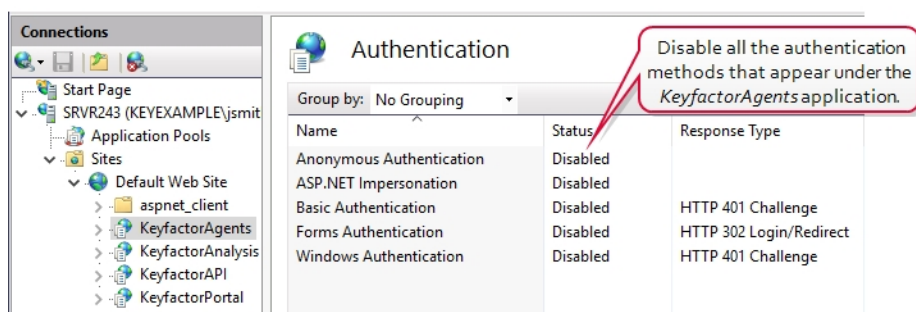


Figure 48: Disable Authentication Methods at the Application Level in IIS



Tip: At the Default Web Site level, the only authentication method that should be enabled is Anonymous. This should not be changed.

3. In the IIS Management console, open SSL Settings for the KeyfactorAgents application. Check the **Require SSL** box and select either **Require** or **Accept** for *Client certificates*.



Important: Only selected **Require** if your are only using orchestrators that support client certificate authentication and plan to configure all of them for certificate authentication.

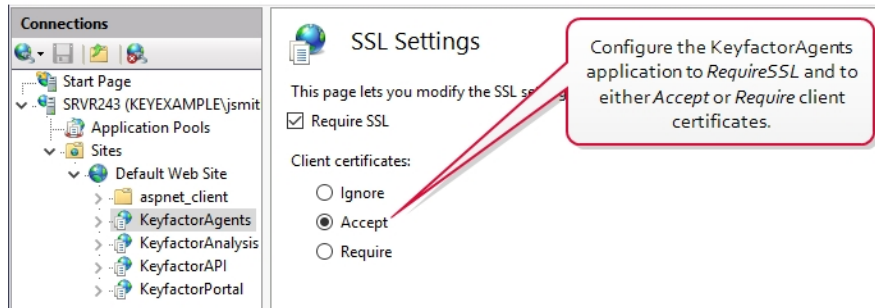


Figure 49: Configure SSL Settings in IIS for Client Certificate Authentication



Tip: At the Default Web Site level, it is good security practice to check the *Require SSL* box, but you do not need to accept client certificates at this level and should not require them at this level.

Create a Certificate Template for Orchestrator Certificates

This method of certificate authentication functions by sending a client certificate from the orchestrator to IIS on the Keyfactor Command server, where IIS does a lookup in Active Directory to determine what Active Directory user is associated with that certificate and then turns around and uses that identity to connect to Keyfactor Command. In order for the certificate to be associated with the Active Directory identity, it must be enrolled using a template that has the *Publish certificate in Active Directory* option enabled.

To create the certificate template that will be used for orchestrator client authentication certificates, start by duplicating a template with a *Computer* subject type. In addition to any standards for your environment, the templates needs:

- The *Publish certificate in Active Directory* box checked.

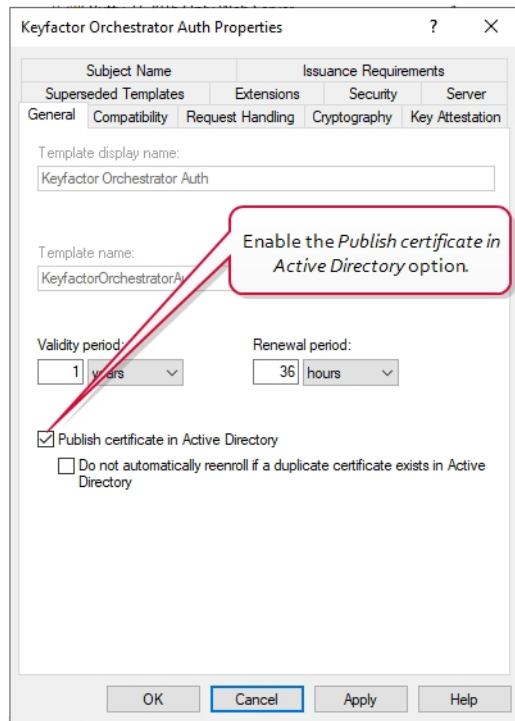


Figure 50: Microsoft Certificate Template General for Client Authentication Certificate

- A key usage that includes Digital Signature.

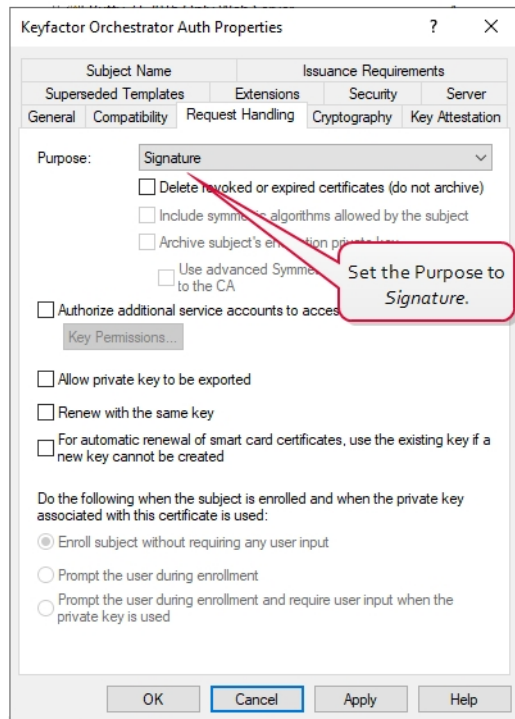


Figure 51: Microsoft Certificate Template Request Handling for Client Authentication Certificate

- An extended key usage (EKU) of Client Authentication.

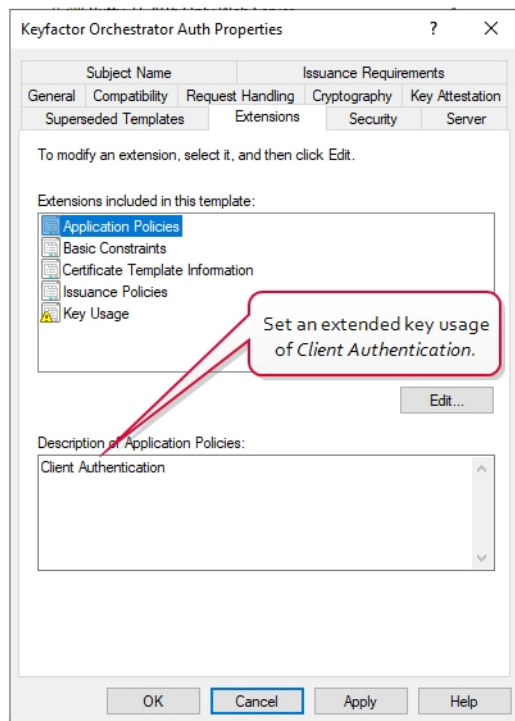


Figure 52: Microsoft Certificate Template Application Policies for Client Authentication Certificate

- Enroll permissions for either the service account that the orchestrator will run as or the machine account for the orchestrator machine (see).

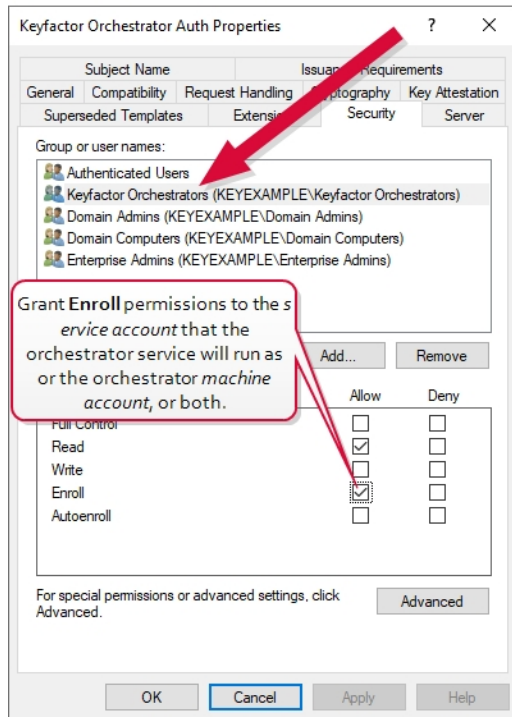


Figure 53: Microsoft Certificate Template Security for Client Authentication Certificate

Enroll for a Client Authentication Certificate

To acquire a certificate for use by the Universal Orchestrator using a Microsoft CA, first create a template using the appropriate configurations as described above and make it available for enrollment on a CA to which the Universal Orchestrator machine has access. If you plan to enroll for the certificate through Keyfactor Command, you will also need to enable the template for enrollment in Keyfactor Command.

You can enroll for a client authentication certificate for the orchestrator in a variety of ways. The certificate needs to be installed in the **local computer** personal store on the Windows server on which the orchestrator is installed. Some possible ways to do this are:

- Use Keyfactor Command to enroll for the certificate using the PFX enrollment method and then import the PFX file on the orchestrator server. If you select this method, you will need to login to the Keyfactor Command Management Portal as the orchestrator service account being used on the Keyfactor Command side of the fence (see [Create Service Accounts for the Universal Orchestrator on page 11](#)) in order to enroll for the certificate in the correct context or use the Keyfactor API to submit a request in a specific user context (see [PFX Enrollment in Keyfactor Command Using a PowerShell Script on the next page](#)). The orchestrator service account will need enroll permissions on the CA, on the template, and in Keyfactor Command.
- Use IIS or the certificates MMC on the orchestrator server to generate a CSR, use the Keyfactor Command CSR enrollment method to enroll for a certificate using the CSR, and then import the CSR on the orchestrator server, marrying it with the private key generated on the

server. If you select this method, you will need to login to the Keyfactor Command Management Portal as the orchestrator service account being used on the Keyfactor Command side of the fence (see [Create Service Accounts for the Universal Orchestrator on page 11](#)) in order to enroll for the certificate in the correct context. The orchestrator service account will need enroll permissions on the CA, on the template, and in Keyfactor Command.

- If there is an existing Universal Orchestrator on the server already running and communicating with Keyfactor Command, use the Keyfactor Command PFX enrollment method and push the certificate out to the certificate store on the orchestrator server using Keyfactor Command. If you select this method, you will need to login to the Keyfactor Command Management Portal as the orchestrator service account being used on the Keyfactor Command side of the fence (see [Create Service Accounts for the Universal Orchestrator on page 11](#)) in order to enroll for the certificate in the correct context or use the Keyfactor API to submit a request in a specific user context (see [PFX Enrollment in Keyfactor Command Using a PowerShell Script below](#)). The orchestrator service account will need enroll permissions on the CA, on the template, and in Keyfactor Command.
- Use the Microsoft MMC on the orchestrator server to enroll for a certificate. If you select this method, the orchestrator will connect to Keyfactor Command using the orchestrator machine account rather than an Active Directory user account. The orchestrator machine account will need enroll permissions on the CA and on the template. This method will only work for servers joined to the same Active Directory forest in which Keyfactor Command is installed.

PFX Enrollment in Keyfactor Command Using a PowerShell Script

To enroll for a certificate using the PFX enrollment method in Keyfactor Command, you can either do this in the Keyfactor Command Management Portal while logged in as the orchestrator service account or with a PowerShell script. In either case, the orchestrator service account will need PFX enroll permissions in Keyfactor Command. Below is a sample PowerShell script. Once the PFX file has been generated, import it into the local machine store on the orchestrator server.



Tip: The service account you provide in the PowerShell script is the service account used to provide a connection from the orchestrator to Keyfactor Command. This is not necessarily the same service account that runs the orchestrator service on the orchestrator server. For an orchestrator in a separate forest from Keyfactor Command, this would be a service account in the Keyfactor Command forest, not the orchestrator forest. See [Create Service Accounts for the Universal Orchestrator on page 11](#).

```
#Set variables with the username and password for the orchestrator service account
$orchUsername = 'KEYEXAMPLE\svc_kyforch'
$orchPassword = 'MySecureServiceAccountPassword'
$pair = "$($orchUsername):($orchPassword)"

# Base-64 encode the service account credentials
$encodedCreds = [System.Convert]::ToBase64String([System.Text.Encoding]::ASCII.GetBytes($pair))
```

```

$UTCTime = (Get-Date).ToUniversalTime().ToString("yyyy-MM-ddTHH:mm:ssZ")
$keyfactorServer = 'keyfactor.keyexample.com' # FQDN of the Keyfactor Command server
$caName = 'corpca01.keyexample.com\CorpIssuing01' # CA to use for the enrollment
$templateName = 'KeyfactorOrchestratorAuth' # Template to use for the enrollment
$certSubject = 'Orchestrator Cert Auth' # Using a template that is configured to build from AD will
cause this subject to be replaced
$pfxPassword = 'MySecurePFXPassword' # Password for the resulting PFX file
$outputFile = 'C:\stuff\OrchCertAuth.pfx' # Path and file name for the PFX file to be generated

$basicAuthValue = "Basic $encodedCreds"

$headers = @{
    "Authorization"=$basicAuthValue
    "Accept"="application/json"
    "x-keyfactor-requested-with"="APIClient"
    "x-certificateformat"="PFX"
}

$body = @{
    "Password" = "$pfxPassword"
    "Subject" = "$certSubject"
    "IncludeChain" = "true"
    "CertificateAuthority" = "$caName"
    "Timestamp" = "$UTCTime"
    "Template" = "$templateName"
}

# Output response as a PFX file
$response = Invoke-WebRequest -Uri "https://$keyfactorServer/KeyfactorAPI/Enrollment/PFX" -Method:Post -Headers $headers -ContentType "application/json" -Body ($body|ConvertTo-Json) -ErrorAction:Stop -TimeoutSec 60
$responseContent = $response.Content | ConvertFrom-Json
$bytes = [Convert]::FromBase64String($responseContent.CertificateInformation.Pkcs12Blob)
[IO.File]::WriteAllBytes($outputFile, $bytes)

```

PFX Enrollment and Deployment in Keyfactor Command Using a PowerShell Script

To enroll for a certificate using the PFX enrollment method in Keyfactor Command and deploy it to the orchestrator server using Keyfactor Command, you can either do this in the Keyfactor Command Management Portal while logged in as the orchestrator service account or with a PowerShell script. In either case, the orchestrator service account will need PFX enroll permissions and certificate store management permissions in Keyfactor Command. Below is a sample PowerShell script. This solution is only an option if your orchestrator is already up and running and successfully authenticating to Keyfactor Command using standard authentication (or previously configured

certificate authentication).



Tip: The service account you provide in the PowerShell script is the service account used to provide a connection from the orchestrator to Keyfactor Command. This is not necessarily the same service account that runs the orchestrator service on the orchestrator server. For an orchestrator in a separate forest from Keyfactor Command, this would be a service account in the Keyfactor Command forest, not the orchestrator forest. See [Create Service Accounts for the Universal Orchestrator on page 11](#).

```
#Set variables with the username and password for the orchestrator service account
$orchUsername = 'KEYEXAMPLE\svc_kyforch'
$orchPassword = 'MySecureServiceAccountPassword'
$pair = "$($orchUsername):($orchPassword)"

# Base-64 encode the service account credentials
$encodedCreds = [System.Convert]::ToBase64String([System.Text.Encoding]::ASCII.GetBytes($pair))

$UTCTime = (Get-Date).ToUniversalTime().ToString("yyyy-MM-ddTHH:mm:ssZ")
$keyfactorServer = 'keyfactor.keyexample.com' # FQDN of the Keyfactor Command server
$storeName = 'websrvr38.keyexample.com' # FQDN of the orchestrator server as defined as a certificate store in Keyfactor Command
$caName = 'corpca01.keyexample.com\CorpIssuing01' # CA to use for the enrollment
$templateName = 'KeyfactorOrchestratorAuth' # Template to use for the enrollment
$certSubject = 'Orchestrator Cert Auth' # Using a template that is configured to build from AD will cause this subject to be replaced

$basicAuthValue = "Basic $encodedCreds"

$enrollHeaders = @{
    "Authorization"=$basicAuthValue
    "Accept"="application/json"
    "x-keyfactor-requested-with"="APIClient"
    "x-certificateformat"="Store"
}

$deployHeaders = @{
    "Authorization"=$basicAuthValue
    "Accept"="application/json"
    "x-keyfactor-requested-with"="APIClient"
}

$enrollBody = @{
```

```

    "Subject" = "$certSubject"
    "IncludeChain" = "true"
    "CertificateAuthority" = "$caName"
    "Timestamp" = "$UTCTime"
    "Template" = "$templateName"
}

# Enroll for a certificate using the PFX enrollment method and retrieve the certificate ID from the
response (as part of the content)
$enrollResponse = Invoke-WebRequest -Uri "https://$keyfactorServer/KeyfactorAPI/Enrollment/PFX" -
Method:Post -Headers $enrollHeaders -ContentType "application/json" -Body ($enrollBody|ConvertTo-
Json) -ErrorAction:Stop -TimeoutSec 60
$enrollContent = $enrollResponse.Content | ConvertFrom-Json

# Get the store GUID for the certificate store specified by the client machine name in the query
string with the storeName variable
$storeInfo = Invoke-WebRequest -Uri "https://$key-
factorServer/KeyfactorAPI/CertificateStores?certificateStoreQuery.queryString=ClientMachine%20-
eq%20%22$storeName%22" -Method:Get -Headers $deployHeaders -ContentType "application/json" -
ErrorAction:Stop -TimeoutSec 60
$storeContent = $storeInfo.Content | ConvertFrom-Json
$storeGUID = $storeContent.Id

$deployBody = @{
    "StoreIds" = @( "$storeGUID" )
    "StoreTypes" = @(
        @{
            "StoreTypeId" = 6 # Store type 6 is IIS personal
            "Overwrite" = "false"
        }
    )
    "CertificateId" = $enrollContent.CertificateInformation.KeyfactorId
}

# Deploy certificate to certificate store
Invoke-WebRequest -Uri "https://$keyfactorServer/KeyfactorAPI/Enrollment/PFX/Deploy" -Method:Post -
Headers $deployHeaders -ContentType "application/json" -Body ($deployBody|ConvertTo-Json) -ErrorAc-
tion:Stop -TimeoutSec 60

```

MMC Enrollment

To enroll for a certificate using the MMC:

1. On the Universal Orchestrator machine, do one of following:
 - Using the GUI:
 - a. Open an empty instance of the Microsoft Management Console (MMC).
 - b. Choose **File->Add/Remove Snap-in....**
 - c. In the *Available snap-ins* column, highlight **Certificates** and click **Add**.
 - d. In the Certificates snap-in popup, choose the radio button for Computer account, click **Next**, accept the default of Local computer, and click **Finish**.
 - e. Click **OK** to close the Add or Remove Snap-ins dialog.
 - Using the command line:
 - a. Open a command prompt using the “Run as administrator” option.
 - b. Within the command prompt type the following to open the certificates MMC:
`certlm.msc`
2. Drill down to the Personal folder under **Certificates** for the Local Computer, right-click, and choose **All Tasks->Request New Certificate....**
3. Follow the certificate enrollment wizard, selecting the template you created for orchestrator certificate authentication and providing any required information.

Grant the Service Account Certificate Private Key Permissions

Whichever method you decide to use to acquire the client authentication certificate for the orchestrator, you will need to grant the Universal Orchestrator service account—the account that the orchestrator service is running as on the server—permissions to read the private key of that certificate.



Tip: If the service account is a member of the local administrators group, this step may not be necessary, since the local administrators group is typically granted these permissions automatically.

To grant private key permissions on the certificate using the MMC:

1. On the Universal Orchestrator machine, do one of following:
 - Using the GUI:
 - a. Open an empty instance of the Microsoft Management Console (MMC).
 - b. Choose **File->Add/Remove Snap-in....**
 - c. In the *Available snap-ins* column, highlight **Certificates** and click **Add**.
 - d. In the Certificates snap-in popup, choose the radio button for Computer account,

click **Next**, accept the default of Local computer, and click **Finish**.

- e. Click **OK** to close the Add or Remove Snap-ins dialog.
- Using the command line:
 - a. Open a command prompt using the “Run as administrator” option.
 - b. Within the command prompt type the following to open the certificates MMC:
certlm.msc
2. Drill down to the Personal folder under **Certificates** for the Local Computer to locate the certificate.
3. Highlight the certificate and choose **All Tasks->Manage Private Keys...**
4. In the Permissions for private keys dialog, click **Add**, add the service account under which the Universal Orchestrator is running (created as per [Create Service Accounts for the Universal Orchestrator on page 11](#)), and grant that service account **Read** but not **Full control** permissions. Click **OK** to save.



Tip: If you receive the following error when selecting your certificate in the orchestrator configuration wizard:

The request was aborted: Could not create SSL/TLS secure channel.

- Confirm that the orchestrator server trusts the root and issuing certificates for the SSL certificate on the Keyfactor Command server and the client authentication certificate you are trying to use (see [Configure Certificate Root Trust for the Universal Orchestrator on page 15](#)).
- Confirm that the orchestrator server has access to the CRLs for both the SSL certificate on the Keyfactor Command server and the client authentication certificate you are trying to use and that these CRLs are valid.
- Confirm that you have granted the service account under which the orchestrator service runs private key permissions on the client authentication certificate.

2.6.4 Appendix - Set up the Universal Orchestrator to Use a Forwarding Proxy

Typically with services that use a forwarding proxy, there is a specific proxy configuration done within the application, but the Universal Orchestrator doesn't have such a configuration. Instead, it makes use of an environment variable to retrieve this information on either Windows or Linux.

On Windows, configure a system environment variable of either HTTP_PROXY or HTTPS_PROXY (this is not case sensitive on Windows) pointing to your proxy's URL, including port, then restart the Universal Orchestrator service if the orchestrator is already installed.

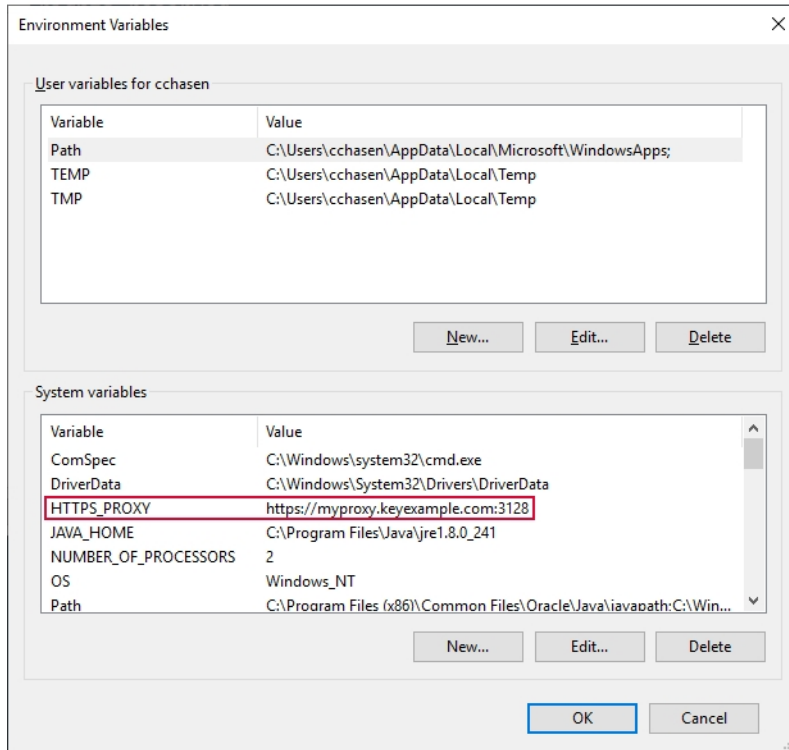


Figure 54: System Environment Variable to Define a Proxy URL for Use by the Universal Orchestrator on Windows

On Linux, there are multiple approaches to setting an environment variable. One method for setting a system-wide environment variable that will be retained after reboot is to add an environment variable statement to the `/etc/environment` file using a command similar to the following (as root):

```
echo https_proxy=https://myproxy.keyexample.com:3128/" >> /etc/environment
```

After setting the environment variable, restart the Universal Orchestrator service if the orchestrator has already been installed.



Note: If you've configured an `HTTPS_PROXY` environment variable because you're using a secure channel to communicate with Keyfactor Command (SSL), you will most likely also need an `HTTP_PROXY` environment variable for the orchestrator to do revocation status (CRL) checking unless you disable revocation status checking.

3.0 Glossary

A

AIA

The authority information access (AIA) is included in a certificate--if configured--and identifies a location from which the chain certificates for that certificate may be retrieved.

AnyAgent

The AnyAgent, one of Keyfactor's suite of orchestrators, is used to allow management of certificates regardless of source or location by allowing customers to implement custom agent functionality via an API.

AnyGateway

The Keyfactor AnyGateway is a generic third party CA gateway framework that allows existing CA gateways and custom CA connections to share the same overall product framework.

API

A set of functions to allow creation of applications. Keyfactor offers the Keyfactor API, which allows third-party software to integrate with the advanced certificate enrollment and management features of Keyfactor Command.

Argument

A parameter or argument is a value that is passed into a function in an application.

Authority Information Access

The authority information access (AIA) is included in a certificate--if configured--and identifies a location from which the chain certificates for that certificate may be retrieved.

B

Bash Orchestrator

The Bash Orchestrator, one of Keyfactor's suite of orchestrators, is used to discover and manage SSH keys across an enterprise.

Blueprint

A snapshot of the certificate stores and scheduled jobs on one orchestrator, which can be used to create matching certificate stores and jobs on another orchestrator with just a few clicks.

C

CA

A certificate authority (CA) is an entity that issues digital certificates. Within Keyfactor Command, a CA may be a Microsoft CA or a Keyfactor gateway to a cloud-based or remote CA.

Certificate Authority

A certificate authority (CA) is an entity that issues digital certificates. Within Keyfactor Command, a CA may be a Microsoft CA or a Keyfactor gateway to a cloud-based or remote CA.

Certificate Revocation List

A Certificate Revocation List (CRL) is a list of digital certificates that have been revoked by the issuing Certificate Authority (CA) before their scheduled expiration date and should no longer be trusted.

Certificate Signing Request

A CSR or certificate signing request is a block of encoded text that is submitted to a CA when enrolling for a certificate. When you generate a CSR within Keyfactor

Command, the matching private key for it is stored in Keyfactor Command in encrypted format and will be married with the certificate once returned from the CA.

CN

A common name (CN) is the component of a distinguished name (DN) that represents the primary name of the object. The value varies depending on the type of object. For a user object, this would be the user's name (e.g. CN=John Smith). For SSL certificates, the CN is typically the fully qualified domain name (FQDN) of the host where the SSL certificate will reside (e.g. server-name.keyexample.com or www.keyexample.com).

Collection

The certificate search function allows you to query the Keyfactor Command database for certificates from any available source based on any criteria of the certificates and save the results as a collection that will be available in other places in the Management Portal (e.g. expiration alerts and certain reports).

Common Name

A common name (CN) is the component of a distinguished name (DN) that represents the primary name of the object. The value varies depending on the type of object. For a user object, this would be the user's name (e.g. CN=John Smith). For SSL certificates, the CN is typically the fully qualified domain name (FQDN) of the host where the SSL certificate will reside (e.g. server-name.keyexample.com or www.keyexample.com).

Configuration Tenant

A grouping of CAs. The Microsoft concept of forests is not used in EJBCA so to

accommodate the new EJBCA functionality, and to avoid confusion, the term forest needed to be renamed. The new name is configuration tenant. For EJBCA, there would be one configuration tenant per EJBCA server install. For Microsoft, there would be one per forest. Note that configuration tenants cannot be mixed, so Microsoft and EJBCA cannot exist on the same configuration tenant.

CRL

A Certificate Revocation List (CRL) is a list of digital certificates that have been revoked by the issuing Certificate Authority (CA) before their scheduled expiration date and should no longer be trusted.

CSR

A CSR or certificate signing request is a block of encoded text that is submitted to a CA when enrolling for a certificate. When you generate a CSR within Keyfactor Command, the matching private key for it is stored in Keyfactor Command in encrypted format and will be married with the certificate once returned from the CA.

D

DER

A DER format certificate file is a DER-encoded binary certificate. It contains a single certificate and does not support storage of private keys. It sometimes has an extension of .der but is often seen with .cer or .crt.

Distinguished Name

A distinguished name (DN) is the name that uniquely identifies an object in a directory. In the context of Keyfactor Command, this directory is generally Active Directory. A DN is made up of attribute=value pairs,

separated by commas. Any of the attributes defined in the directory schema can be used to make up a DN.

DN

A distinguished name (DN) is the name that uniquely identifies an object in a directory. In the context of Keyfactor Command, this directory is generally Active Directory. A DN is made up of attribute=value pairs, separated by commas. Any of the attributes defined in the directory schema can be used to make up a DN.

DNS

The Domain Name System is a service that translates names into IP addresses.

E

ECC

Elliptical curve cryptography (ECC) is a public key encryption technique based on elliptic curve theory that can be used to create faster, smaller, and more efficient cryptographic keys. ECC generates keys through the properties of the elliptic curve equation instead of the traditional method of generation as the product of very large prime numbers.

Endpoint

An endpoint is a URL that enables the API to gain access to resources on a server.

Enrollment

Certificate enrollment refers to the process by which a user requests a digital certificate. The user must submit the request to a certificate authority (CA).

EOBO

A user with an enrollment agent certificate can enroll for a certificate on behalf of another user. This is often used when provisioning technology such as smart cards.

F

Forest

An Active Directory forest (AD forest) is the top most logical container in an Active Directory configuration that contains domains, and objects such as users and computers.

G

Gateway Connector

The Keyfactor Gateway Connector is installed in the customer forest to provide a connection between the on-premise CA and the Azure-hosted, Keyfactor managed Hosted Configuration Portal to provide support for synchronization, enrollment and management of certificates through the Azure-hosted instance of Keyfactor Command for the on-premise CA. It is supported on both Windows and Linux.

H

Host Name

The unique identifier that serves as name of a computer. It is sometimes presented as a fully qualified domain name (e.g. server-name.keyexample.com) and sometimes just as a short name (e.g. servername).

Hosted Config Portal

The Keyfactor Hosted Configuration Portal is used to configure connections between on-premise instances of the Keyfactor

Gateway Connector and on-premise CAs to make them available to Azure-hosted instance of Keyfactor Command. The portal is Azure-hosted and managed by Keyfactor.

Hosted Configuration Portal

The Keyfactor Hosted Configuration Portal is used to configure connections between on-premise instances of the Keyfactor Gateway Connector and on-premise CAs to make them available to Azure-hosted instance of Keyfactor Command. The portal is Azure-hosted and managed by Keyfactor.

Hostname

The unique identifier that serves as name of a computer. It is sometimes presented as a fully qualified domain name (e.g. server-name.keyexample.com) and sometimes just as a short name (e.g. servername).

J

Java Agent

The Java Agent, one of Keyfactor's suite of orchestrators, is used to perform discovery of Java keystores and PEM certificate stores, to inventory discovered stores, and to push certificates out to stores as needed.

Java Keystore

A Java KeyStore (JKS) is a file containing security certificates with matching private keys. They are often used by Java-based applications for authentication and encryption.

JKS

A Java KeyStore (JKS) is a file containing security certificates with matching private keys. They are often used by Java-based

applications for authentication and encryption.

K

Key Length

The key size or key length is the number of bits in a key used by a cryptographic algorithm.

Key Pair

In asymmetric cryptography, public keys are used together in a key pair with a private key. The private key is retained by the key's creator while the public key is widely distributed to any user or target needing to interact with the holder of the private key.

Key Size

The key size or key length is the number of bits in a key used by a cryptographic algorithm.

Key Type

The key type identifies the type of key to create when creating a symmetric or asymmetric key. It references the signing algorithm and often key size (e.g. AES-256, RSA-2048, Ed25519).

Keyfactor CA Management Gateway

The Keyfactor CA Management Gateway is made up of the Keyfactor Gateway Connector, installed in the customer forest to provide a connection to the local CA, and the Azure-hosted and Keyfactor managed Hosted Configuration Portal. The solution is used to provide a connection between a customer's on-premise CA and an Azure-hosted instance of Keyfactor Command for synchronization, enrollment, and management of certificates.

Keyfactor Gateway Connector

The Keyfactor Gateway Connector is installed in the customer forest to provide a connection between the on-premise CA and the Azure-hosted, Keyfactor managed Hosted Configuration Portal to provide support for synchronization, enrollment and management of certificates through the Azure-hosted instance of Keyfactor Command for the on-premise CA. It is supported on both Windows and Linux.

Keyfactor Universal Orchestrator

The Keyfactor Universal Orchestrator, one of Keyfactor's suite of orchestrators, is used to interact with servers and devices for certificate management, run SSL discovery and management tasks, and manage synchronization of certificate authorities in remote forests. With the addition of custom extensions, it can provide certificate management capabilities on a variety of platforms and devices (e.g. Amazon Web Services (AWS) resources, Citrix\NetScaler devices, F5 devices, IIS stores, JKS keystores, PEM stores, and PKCS#12 stores) and execute tasks outside the standard list of certificate management functions. It runs on either Windows or Linux servers or Linux containers.

Keystore

A Java KeyStore (JKS) is a file containing security certificates with matching private keys. They are often used by Java-based applications for authentication and encryption.

L

Logical Name

The logical name of a CA is the common name given to the CA at the time it is created. For Microsoft CAs, this name can

be seen at the top of the Certificate Authority MMC snap-in. It is part of the FQDN\Logical Name string that is used to refer to CAs when using command-line tools and in some Keyfactor Command configuration settings (e.g. `ca2.keyexample.-com\Corp Issuing CA Two`).

M

MAC Agent

The MAC Agent, one of Keyfactor's suite of orchestrators, is used to manage certificates on any keychains on the Mac on which the Keyfactor MAC Agent is installed.

Metadata

Metadata provides information about a piece of data. It is used to summarize basic information about data, which can make working with the data easier. In the context of Keyfactor Command, the certificate metadata feature allows you to create custom metadata fields that allow you to tag certificates with tracking information about certificates.

O

Object Identifier

Object identifiers or OIDs are a standardized system for identifying any object, concept, or "thing" with a globally unambiguous persistent name.

OID

Object identifiers or OIDs are a standardized system for identifying any object, concept, or "thing" with a globally unambiguous persistent name.

Orchestrator

Keyfactor orchestrators perform a variety of functions, including managing certificate stores and SSH key stores.

P

P12

A PFX file (personal information exchange format), also known as a PKCS#12 archive, is a single, password-protected certificate archive that contains both the public and matching private key and, optionally, the certificate chain. It is a common format for Windows servers.

P7B

A PKCS #7 format certificate file is a base64-encoded certificate. Since it's presented in ASCII, you can open it in any text editor. PKCS #7 certificates always begin and end with entries that look something like -----BEGIN CERTIFICATE----- and -----END CERTIFICATE-----. Unlike PEM files, PKCS #7 files can contain only a certificate and its certificate chain but NOT its private key. Extensions of .p7b or .p7c are usually seen on certificate files of this format.

P7C

A PKCS #7 format certificate file is a base64-encoded certificate. Since it's presented in ASCII, you can open it in any text editor. PKCS #7 certificates always begin and end with entries that look something like -----BEGIN CERTIFICATE----- and -----END CERTIFICATE-----. Unlike PEM files, PKCS #7 files can contain only a certificate and its certificate chain but NOT its private key. Extensions of .p7b or .p7c are usually seen on certificate files of this format.

Parameter

A parameter or argument is a value that is passed into a function in an application.

PEM

A PEM format certificate file is a base64-encoded certificate. Since it's presented in ASCII, you can open it in any text editor. PEM certificates always begin and end with entries like -----BEGIN CERTIFICATE----- and -----END CERTIFICATE-----. PEM certificates can contain a single certificate or a full certificate chain and may contain a private key. Usually, extensions of .cer and .crt are certificate files with no private key, .key is a separate private key file, and .pem is both a certificate and private key.

PFX

A PFX file (personal information exchange format), also known as a PKCS#12 archive, is a single, password-protected certificate archive that contains both the public and matching private key and, optionally, the certificate chain. It is a common format for Windows servers.

PKCS #7

A PKCS #7 format certificate file is a base64-encoded certificate. Since it's presented in ASCII, you can open it in any text editor. PKCS #7 certificates always begin and end with entries that look something like -----BEGIN CERTIFICATE----- and -----END CERTIFICATE-----. Unlike PEM files, PKCS #7 files can contain only a certificate and its certificate chain but NOT its private key. Extensions of .p7b or .p7c are usually seen on certificate files of this format.

PKCS#12

A PFX file (personal information exchange format), also known as a PKCS#12 archive,

is a single, password-protected certificate archive that contains both the public and matching private key and, optionally, the certificate chain. It is a common format for Windows servers.

PKI

A public key infrastructure (PKI) is a set of roles, policies, and procedures needed to create, manage, distribute, use, store and revoke digital certificates and manage public-key encryption.

Private Key

Private keys are used in cryptography (symmetric and asymmetric) to encrypt or sign content. In asymmetric cryptography, they are used together in a key pair with a public key. The private or secret key is retained by the key's creator, making it highly secure.

Public Key

In asymmetric cryptography, public keys are used together in a key pair with a private key. The private key is retained by the key's creator while the public key is widely distributed to any user or target needing to interact with the holder of the private key.

Public Key Infrastructure

A public key infrastructure (PKI) is a set of roles, policies, and procedures needed to create, manage, distribute, use, store and revoke digital certificates and manage public-key encryption.

R

Rogue Key

A rogue key, in the context of Keyfactor Command, is an SSH public key that appears in an `authorized_keys` file on a

server managed by the SSH orchestrator without authorization.

Root of Trust

A root of trust (RoT) is a source within a cryptographic system that can always be trusted. It is typically a hardened hardware module. HSMs (hardware security modules) and TPMs (trusted platform modules) are examples of RoTs.

RoT

A root of trust (RoT) is a source within a cryptographic system that can always be trusted. It is typically a hardened hardware module. HSMs (hardware security modules) and TPMs (trusted platform modules) are examples of RoTs.

RPC

Remote procedure call (RPC) allows one program to call a function from a program located on another computer on a network without specifying network details. In the context of Keyfactor Command, RPC errors often indicate Kerberos authentication or delegation issues.

rsyslog

Rsyslog is an open-source software utility used on UNIX and Unix-like computer systems for forwarding log messages in an IP network.

S

SAN

The subject alternative name (SAN) is an extension to the X.509 specification that allows you to specify additional values when enrolling for a digital certificate. A variety of SAN formats are supported, with DNS name being the most common.

server name indication

Server name indication (SNI) is an extension to TLS that provides for including the host-name of the target server in the initial handshake request to allow the server to respond with the correct SSL certificate or allow a proxy to forward the request to the appropriate target.

SMTP

Short for simple mail transfer protocol, SMTP is a protocol for sending email messages between servers.

SNI

Server name indication (SNI) is an extension to TLS that provides for including the host-name of the target server in the initial handshake request to allow the server to respond with the correct SSL certificate or allow a proxy to forward the request to the appropriate target.

SSH

The SSH (secure shell) protocol provides for secure connections between computers. It provides several options for authentication, including public key, and protects the communications with strong encryption.

SSL

TLS (Transport Layer Security) and its predecessor SSL (Secure Sockets Layer) are protocols for establishing authenticated and encrypted links between networked computers.

Subject Alternative Name

The subject alternative name (SAN) is an extension to the X.509 specification that allows you to specify additional values when enrolling for a digital certificate. A variety of

SAN formats are supported, with DNS name being the most common.

T

Template

A certificate template defines the policies and rules that a CA uses when a request for a certificate is received.

TLS

TLS (Transport Layer Security) and its predecessor SSL (Secure Sockets Layer) are protocols for establishing authenticated and encrypted links between networked computers.

Trusted CA

A certificate authority in the forest in which Keyfactor Command is installed or in a forest in a two-way trust with the forest in which Keyfactor Command is installed.

U

Untrusted CA

A certificate authority in a forest in a one-way trust with the forest in which Keyfactor Command is installed or in a forest that is untrusted by the forest in which Keyfactor Command is installed. Non-domain-joined standalone CAs also fall into this category.

W

Web API

A set of functions to allow creation of applications. Keyfactor offers the Keyfactor API, which allows third-party software to integrate with the advanced certificate enrollment and management features of Keyfactor Command.

Windows Orchestrator

The Windows Orchestrator, one of Keyfactor's suite of orchestrators, is used to manage synchronization of certificate authorities in remote forests, run SSL discovery and management tasks, and interact with Windows servers as well as F5 devices, NetScaler devices, Amazon Web Services (AWS) resources, and FTP capable devices, for certificate management. In addition, the AnyAgent capability of the Windows Orchestrator allows it to be extended to create custom certificate store types and management capabilities regardless of source platform or location.

Workflow

A workflow is a series of steps necessary to complete a process. In the context of Keyfactor Command, it refers to the workflow builder, which allows you automate event-driven tasks when a certificate is requested or revoked.

X

x.509

In cryptography, X.509 is a standard defining the format of public key certificates. An X.509 certificate contains a public key and an identity (e.g. a host name or an organization or individual name), and is either signed by a certificate authority or self-signed. When a certificate is signed by a trusted certificate authority it can be used to establish trusted secure communications with the owner of the corresponding private key. It can also be used to verify digitally signed documents and emails.

4.0 Copyright Notice

User guides and related documentation from Keyfactor are subject to the copyright laws of the United States and other countries and are provided under a license agreement that restricts copying, disclosure, and use of such documentation. This documentation may not be disclosed, transferred, modified, or reproduced in any form, including electronic media, or transmitted or made publicly available by any means without the prior written consent of Keyfactor and no authorization is granted to make copies for such purposes.

Information described herein is furnished for general information only, is subject to change without notice, and should not be construed as a warranty or commitment by Keyfactor. Keyfactor assumes no responsibility or liability for any errors or inaccuracies that may appear in this document.

The software described in this document is provided under written license agreement, contains valuable trade secrets and proprietary information, and is protected by the copyright laws of the United States and other countries. It may not be copied or distributed in any form or medium, disclosed to third parties, or used in any manner not provided for in the software licenses agreement except with written prior approval from Keyfactor.