

# Installation

## NOTE

Product names changed when ForgeRock became part of Ping Identity. Learn more about the name changes from [New names for ForgeRock products](#)<sup>↗</sup>.

This guide describes how to install Java Agent.

## Example installation for this guide

Unless otherwise stated, the examples in this guide assume the following installation:

- Java Agent installed on `https://agent.example.com:443/app`.
- AM installed on `https://am.example.com:8443/am`.
- Work in the top-level realm `/`.

If you use a different configuration, substitute in the procedures accordingly.

## Prepare for installation

### Before you install

Consider the following points before you install:

- Install AM and Java Agent in different containers.
- Install the container before you install the agent.
- Install only one Java Agent for each container.
- Install a supported version of the Java runtime environment, as described in [Java requirements](#). Set the `JAVA_HOME` environment variable accordingly. The agent installer requires Java.

```
$ echo $JAVA_HOME  
/path/to/java
```

- For environments with load balancers or reverse proxies, consider the communication between the agent and AM servers, and between the agent and the client. Configure both AM and the environment **before** you install the agent. Learn more in [Configure load balancers and reverse proxies](#).

## Download and unzip Java Agent

Go to the [BackStage download site](#)<sup>[7]</sup> and download an agent based on your architecture, and operating system requirements. Verify the checksum of the downloaded file against the checksum posted on the download page.

Unzip the file in the directory where you plan to store the agent configuration and log files. The following directories are extracted:

Directory	Description
bin	The <b>agentadmin</b> installation and configuration program. Learn more from <a href="#">agentadmin command</a> .
config	Configuration templates used by the <b>agentadmin</b> command during installation
data	Not used
etc	Configuration templates used during installation
installer-logs	Location of log files written during installation
legal-notice	Licensing information including third-party licenses
lib	Shared libraries used by the agent
locale	Property files used by the installation program
README	README file containing platform and install information for the agent

## Preinstallation tasks

1. Create a text file for the agent password, and protect it. For example, use commands similar to these, but use a strong password and store it in a secure place:

Unix	Windows
------	---------

```
$ cat > /secure-directory/pwd.txt
password
CTRL+D

$ chmod 400 /secure-directory/pwd.txt
```

#### TIP

Although the agent accepts any password length and content, you are strongly encouraged to generate secure passwords. This can be achieved in various ways, for example using a password manager or by using the command line tool `agentadmin --key`.

2. (Optional) Create a signing key for pre-authentication cookies and POST data preservation cookies. The key must be at least 64 characters long, but preferably 80.

- a. Create the key with the `agentadmin --key` command:

Unix

Windows

```
$ agentadmin --key 80
ZRY...xX0
```

- b. Write the key to a file:

Unix

Windows

```
$ cat > /secure-directory/signing-key.txt
ZRY...xX0
CTRL+D

$ chmod 400 /secure-directory/signing-key.txt
```

3. In AM, add an agent profile, as described in [Create agent profiles](#):

The examples in this guide use an agent profile in the top-level realm, with the following values:

- **Agent ID:** java-agent
- **Agent URL:** `https://agent.example.com:443/app`
- **Server URL:** `https://am.example.com:8443/am`

- **Password:** password
4. In AM, add a policy set and policy, to protect resources with the agent, as described in [Policies](#) in AM's *Authorization guide*.

The examples in this guide use a policy set and policy in the top-level realm, with the following values:

- Policy set:
  - **Name:** PEP
  - **Resource Types:** URL
- Policy:
  - **Name:** PEP-policy
  - **Resource Type:** URL
  - **Resource pattern:** \*/\*\*/\*/\*
  - **Resource value:** \*/\*\*/\*/\*
  - **Actions tab:** Allow HTTP GET and POST
  - **Subjects tab:** All Authenticated Users.

When you create your own policy set instead of using the default policy set, `iPlanetAMWebAgentService`, update the following properties in the agent profile:

- [Policy Set Map](#)
  - [Policy Evaluation Realm Map](#)
5. When you exchange **signed** OpenID Connect JWTs between AM and the agent, set up a new key and secret as described in [Configure communication with AM servers](#). Do not use the default `test` key pair in a real environment.

## Configure communication with AM servers

AM communicates authentication and authorization information to Java Agent by using OpenID Connect (OIDC) JSON web tokens (JWT). To secure the JSON payload, AM and the agent support JWT signing with the RS256 algorithm. For more information, refer to [RFC 7518](#).

AM uses an HMAC signing key to protect requested ACR claims values between sending the user to the authentication endpoint, and returning from successful authentication.

By default, AM uses a demo key and an autogenerated secret for these purposes. For production environments, perform the steps in the following procedure to create new key aliases and configure them in AM.

### *Configure AM secret labels for the agents' OAuth 2.0 provider*

By default, AM is configured to:

- Sign JWTs with the secret mapped to the `am.global.services.oauth2.oidc.agent.idtoken.signing` secret label. The label defaults to the `rsajwt signingkey` key alias provided in AM's JCEKS keystore.
- Sign claims with the secret mapped to the `am.services.oauth2.jwt.authenticity.signing` secret label. The label defaults to the `hmac signingtest` key alias available in AM's JCEKS keystore.

For more information about secret stores, refer to [Secret stores](#) in AM's *Security guide*.

1. Create the following aliases in one of the secret stores configured in AM, for example, the default JCEKS keystore:
  - RSA key pair
  - HMAC secret
2. In the AM admin UI, select **Configure** > **Secret Stores** > *Keystore Secret Store Name* > **Mappings**, and configure the following secret labels:
  - The new RSA key alias in the `am.global.services.oauth2.oidc.agent.idtoken.signing` secret label.
  - The new HMAC secret in the `am.services.oauth2.jwt.authenticity.signing` secret label.

You might already have a secret configured for this secret label, because it is also used for signing certain OpenID Connect ID tokens and remote consent requests. For more information, refer to [Secret label default mappings](#) in AM's *Security guide*.

3. Save your changes.

## Create agent profiles

Use Java Agent profiles to connect to and communicate with Advanced Identity Cloud or AM.

### NOTE

You can find additional details about creating an agent profile in Advanced Identity Cloud in [Create an agent profile in Advanced Identity Cloud](#).

### *Create an agent profile for a single agent instance*

This section describes how to create an agent profile in the AM admin UI. Alternatively, create agent profiles by using the `/realm-config/agents/WebAgent/{id}` endpoint in the REST API. Learn more in [API Explorer](#) in AM's *REST guide*.

1. In the AM admin UI, select **Realms** > *Realm Name* > **Applications** > **Agents** > **Java**, and add an agent using the following hints:

#### **Agent ID**

The ID of the agent profile. This ID resembles a username and is used during the agent installation. For example, MyAgent .

##### **TIP**

When AM is not available, the related error message contains the agent profile name. Consider this in your choice of agent profile name.

#### **Agent URL**

The URL where the agent resides. Learn more in [Example installation for this guide](#).

When the agent is in [remote configuration mode](#), the Agent URL is used to populate the agent profile for services, such as notifications.

#### **Server URL**

The full URL to an authorization server, such as AM. Learn more in [Example installation for this guide](#).

If the authorization server is deployed in a site configuration (behind a load balancer), enter the site URL. When the agent is in [remote configuration mode](#), the Server URL is used to populate the agent profile for login, logout, naming, and cross-domain SSO.

#### **Password**

The password the agent uses to authenticate to an authorization server, such as AM. Use this password when installing an agent.

##### **TIP**

Although the agent accepts any password length and content, you are strongly encouraged to generate secure passwords. This can be achieved in various ways, for example using a password manager or by using the command line tool `agentadmin --key`.

2. (Optional - From AM 7.5) Use AM's secret service to manage the agent profile password. If AM finds a matching secret in a secret store, it uses that secret instead of the agent password configured in Step 1.

- a. In the agent profile page, set a label for the agent password in **Secret Label Identifier**.

AM uses the identifier to generate a secret label for the agent.

The secret label has the format

`am.application.agents.identifier.secret` , where *identifier* is the **Secret**

## Label Identifier.

The **Secret Label Identifier** can only contain characters a-z , A-Z , 0-9 , and periods ( . ). It can't start or end with a period.





- b. Select **Secret Stores** and configure a secret store.
- c. Map the label to the secret. Learn more in AM's [mapping](#).

Note the following points for using AM's secret service:

- Set a **Secret Label Identifier** that clearly identifies the agent.
- If you update the **Secret Label Identifier**:
  - If no other agent shares that secret mapping, AM updates any corresponding secret mapping for the previous identifier.
  - If another agent shares that secret mapping, AM creates a new secret mapping for the updated identifier and copies its aliases from the previously shared secret mapping.
- If you delete the **Secret Label Identifier**, AM deletes any corresponding secret mapping for the previous identifier, provided no other agent shares that secret mapping.
- When you rotate a secret, update the corresponding mapping.

## Create an agent profile group and inherit settings

Use agent profile groups to set up multiple agents that inherit settings from the group.

1. In the AM admin UI, select **Realms** > *Realm Name* > **Applications** > **Agents** > **Java**.
2. In the **Group** tab, add a group. Use the URL to the AM server in which to store the profile.
3. Edit the group configuration as necessary, and save the configuration.
4. Select **Realms** > *Realm Name* > **Applications** > **Agents** > **Java**, and select an agent you created previously.
5. In the **Global** tab, select **Group**, and add the agent to the group you created previously. The  icon appears next to some properties.
6. For each property where  appears, toggle the icon to set inheritance:
  - : Do not inherit the value from the group.
  - : Inherit the value from the group.

## Authenticate agents to the identity provider

### Authenticate agents to Advanced Identity Cloud

IMPORTANT

Java Agent is automatically authenticated to Advanced Identity Cloud by a non-configurable authentication module. Authentication chains and modules are deprecated in Advanced Identity Cloud and replaced by journeys.

You can now authenticate Java Agent to Advanced Identity Cloud with a journey. The procedure is currently optional, but will be required when authentication chains and modules are removed in a future release of Advanced Identity Cloud.

For more information, refer to Advanced Identity Cloud's [Journeys](#).

This section describes how to create a journey to authenticate Java Agent to Advanced Identity Cloud. The journey has the following requirements:

- It must be called **Agent**
- Its nodes must pass the agent credentials to the Agent Data Store Decision node.

When you define a journey in Advanced Identity Cloud, that same journey is used for all instances of PingGateway, Java Agent, and Web Agent. Consider this point if you change the journey configuration.

1. Log in to the Advanced Identity Cloud admin UI as an administrator.
2. Click **Journeys > New Journey**.
3. Add a journey with the following information and click **Create journey**:
  - **Name**: Agent
  - **Identity Object**: The user or device to authenticate.
  - (Optional) **Description**: Authenticate an agent to Advanced Identity Cloud

The journey designer is displayed, with the **Start** entry point connected to the **Failure** exit point, and a **Success** node.

4. Using the **Filter nodes** bar, find and then drag the following nodes from the **Components** panel into the designer area:
  - Zero Page Login Collector node to check whether the agent credentials are provided in the incoming authentication request and use their values in the following nodes.

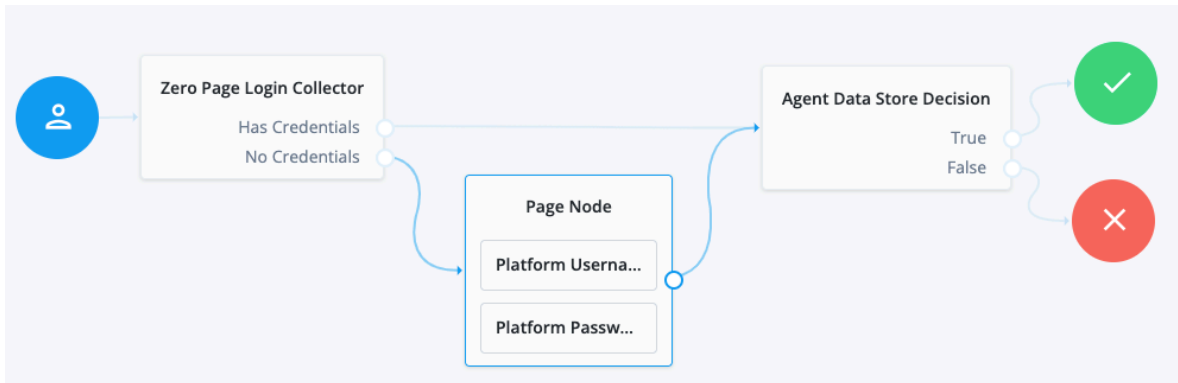
This node is required for compatibility with Java agent and Web agent.

- Page node to collect the agent credentials if they are not provided in the incoming authentication request and use their values in the following nodes.
- Agent Data Store Decision node to verify that the agent credentials match the registered Java Agent agent profile.

#### IMPORTANT

Many nodes can be configured in the panel on the right side of the page. Unless otherwise stated, do not configure the nodes and use only the default values.

5. Drag the following nodes from the **Components** panel into the Page node:
  - Platform Username node to prompt the user to enter their username.
  - Platform Password node to prompt the user to enter their password.
6. Connect the nodes as follows and save the journey:



## Authenticate agents to AM

#### IMPORTANT

##### From AM 7.3

When AM 7.3 is installed with a default configuration, as described in [Evaluation](#), Java Agent is automatically authenticated to AM by an authentication tree. Otherwise, Java Agent is authenticated to AM by an AM authentication module.

Authentication chains and modules were deprecated in AM 7. When they are removed in a future release of AM, it will be necessary to configure an appropriate authentication tree when you are not using the default configuration.

For more information, refer to AM's [Authentication Nodes and Trees](#).

This section describes how to create an authentication tree to authenticate Java Agent to AM. The tree has the following requirements:

- It must be called Agent
- Its nodes must pass the agent credentials to the Agent Data Store Decision node.

When you define a tree in AM, that same tree is used for all instances of PingGateway, Java Agent, and Web Agent. Consider this point if you change the tree configuration.




1. On the **Realms** page of the AM admin UI, choose the realm in which to create the authentication tree.

2. On the **Realm Overview** page, click **Authentication > Trees > Create tree**.

3. Create a tree named **Agent** .

The authentication tree designer is displayed, with the **Start** entry point connected to the **Failure** exit point, and a **Success** node.

The authentication tree designer provides the following features on the toolbar:

Button	Usage
	Lay out and align nodes according to the order they are connected.
	Toggle the designer window between normal and full-screen layout.
	Remove the selected node. Note that the <b>Start</b> entry point cannot be deleted.

4. Using the **Filter** bar, find and then drag the following nodes from the **Components** panel into the designer area:

- Zero Page Login Collector node to check whether the agent credentials are provided in the incoming authentication request and use their values in the following nodes.

This node is required for compatibility with Java agent and Web agent.

- Page node to collect the agent credentials if they are not provided in the incoming authentication request and use their values in the following nodes.
- Agent Data Store Decision node to verify that the agent credentials match the registered Java Agent profile.

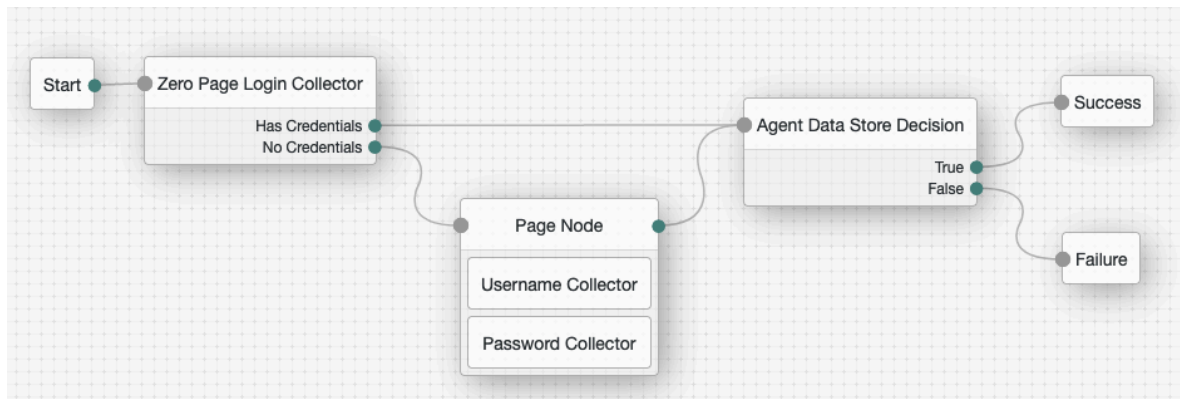
#### IMPORTANT

Many nodes can be configured in the panel on the right side of the page. Unless otherwise stated, do not configure the nodes and use only the default values.

5. Drag the following nodes from the **Components** panel into the **Page** node:

- **Username Collector** node, to prompt the user to enter their username
- **Password Collector** node, to prompt the user to enter their password

6. Connect the nodes as follows and save the tree:



## Check agents can connect to the identity provider

You can authenticate as the agent you created a profile for in Advanced Identity Cloud or AM to check the agent can connect successfully. A successful connection proves the agent can connect to Advanced Identity Cloud or AM, their credentials are correct, and a valid agent profile exists.

Authenticate as follows:

```
$ curl \
--request POST \
--header "X-OpenAM-Username: agent-id" \ ①
--header "X-OpenAM-Password: password" \ ②
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=2.1" \
'https://am.example.com:8443/am/json/realms/root/realms/alpha/auth
enticate?auth-service' ③
```

- ① Replace *agent-id* with the ID of the agent profile you created.
- ② Replace *password* with the agent password.
- ③ Replace *auth-service* with either  
 authIndexType=module&authIndexValue=Application or  
 authIndexType=service&authIndexValue=Agent depending on whether you  
authenticate using the default non-configurable authentication module or a journey  
 called Agent .

If authentication is successful, the response includes the `tokenId` that corresponds to the agent session and the URL to which the agent would normally be redirected. For example:

```
{
  "tokenId": "AQIC5wM...TU3OQ*",
  "successUrl": "/am/console",
```

```
"realm": "/alpha"  
}
```

## Create agent administrators for a realm

To create agent profiles when installing Java Agent, you need the credentials of an AM user who can read and write agent profiles.

This section describes how to create an agent administrator for a specific realm. Use this procedure to reduce the scope given to users who create agent profiles.

1. In the AM admin UI, select **Realms** > *Realm Name* > **Identities**.
2. In the **Groups** tab, add a group for agent administrators.
3. In the **Privileges** tab, enable **Log Read** and **Log Write**.
4. Return to **Realms** > *Realm Name* > **Identities**, add agent administrator identities.
5. For each identity, select the **Groups** tab, add the user to agent profile administrator group.
6. Provide each system administrator who installs agents with their agent administrator credentials.

When installing the agent with the `--custom-install` option, the system administrator can choose the option to create the profile during installation, and then provide the agent administrator username and the path to a read-only file containing the agent administrator password.

## Install Java Agent

---

### Install Tomcat Java Agent

Before you install, make sure that all Tomcat scripts are present in the `$CATALINA_HOME/bin` directory. The Tomcat Windows executable installer does not include the scripts. If the scripts are not present in your installation, copy the contents of the `bin` directory from a `.zip` download of Tomcat of the same version as the one you installed.

#### *Install Tomcat Java Agent interactively*

1. Review the information in [Before you install](#), and perform the steps in [Preinstallation tasks](#).
2. Shut down the Tomcat server where you plan to install the agent.
3. Make sure AM is running.

4. Run **agentadmin --install** to install the agent:

```
$ /path/to/java_agents/tomcat_agent/bin/agentadmin --install
```

5. When prompted, enter information for your deployment.

**TIP**

To cancel the installation at any time, press **CTRL+C**.

- a. Enter the complete path to the Tomcat configuration folder:

```
...
[ ? : Help, ! : Exit ]
Enter the Tomcat Server Config Directory Path
[/opt/apache-tomcat/conf]: /path/to/apache-tomcat/conf
```

- a. Enter the AM URL:

```
...
[ ? : Help, < : Back, ! : Exit ]
AM server URL: https://am.example.com:8443/am
```

To load balance connections between the agent and an AM site, enter the URL of the load balancer in front of the AM site.

If a reverse proxy is configured between AM and the agent, enter the proxy URL. For more information, refer to [Configure an Apache HTTP Server as a reverse proxy](#).

- b. Enter the `$CATALINA_HOME` environment variable, specifying the path to the root of the Tomcat server:

```
...
[ ? : Help, < : Back, ! : Exit ]
Enter the $CATALINA_HOME environment variable:
/path/to/apache-tomcat
```

- c. Enter the agent URL:

```
...
[ ? : Help, < : Back, ! : Exit ]
Agent URL: https://agent.example.com:443/app
```

- d. Enter the name of the agent profile created in AM:

```
...
[ ? : Help, < : Back, ! : Exit ]
Enter the Agent Profile name: java-agent
```

e. Enter the AM realm containing the agent profile. Realms are case-sensitive.

```
...
[ ? : Help, < : Back, ! : Exit, ^ : Accept Empty value ]
Enter the Agent Profile realm [/]:
```

f. Enter the path to the password file you created during pre-installation:

```
...
[ ? : Help, < : Back, ! : Exit ]
Enter the path to the password file: /secure-  
directory/pwd.txt
```

g. Enter the path to a file containing the agent pre-authentication cookie signing value:

```
...
[ ? : Help, < : Back, ! : Exit ]
Enter the path to the signing file:
```

Provide a path to a file containing a randomly generated key that is at least 64 characters long but preferably about 80 characters. For help to create signing a key, refer to [Create a cookie signing key](#).

For information about how the agent uses pre-authentication cookies, refer to the *Authentication* section of [Request flow](#).

To disable cookie signing, press return without providing a value.

**TIP**

Cookie signing is a CPU-intensive process that renders cookies more tamper-proof. Weigh the potential increase in security against the potential loss in performance.

6. Review a summary of your responses and select how to continue:

```
...
Verify your settings above and decide from the choices below.
1. Continue with Installation
2. Back to the last interaction
3. Start Over
```

```
4. Exit
Please make your selection [1]: 1
...
```

After successful installation, the installer adds the agent configuration to the Tomcat configuration, and sets up configuration and log directories for the agent.

7. Test the installation by browsing to a resource that the agent protects. AM redirects you to authenticate. After authentication, AM redirects you back to the requested resource.

### *Install Tomcat Java Agent silently*

Use the **agentadmin --useResponse** command for silent installation. For information about the option, refer to [agentadmin command](#).

The following example uses a response file containing the same configuration as in Install Tomcat Java Agent interactively.

1. Review the information in [Before you install](#), and perform the steps in [Preinstallation tasks](#).
2. Shut down the Tomcat server where you plan to install the agent.
3. Make sure AM is running.
4. Create a response file with the following content, at `/path/to/response-file`:

```
# Response File
CONFIG_DIR= /path/to/apache-tomcat/conf
AM_SERVER_URL= https://am.example.com:8443/am
CATALINA_HOME= /path/to/apache-tomcat
AGENT_URL= https://agent.example.com:443/app
AGENT_PROFILE_NAME= java-agent
AGENT_PROFILE_REALM= /
AGENT_PASSWORD_FILE= /secure-directory/pwd.txt
AGENT_SIGNING_FILE= /secure-directory/signing-key.txt
```

5. Run the **agentadmin** command with the `--useResponse` option:

```
$ agentadmin --install --useResponse /path/to/response-file
```

### *Install in a subrealm*

Other installation examples install the agent in the top-level realm. To install the agent in a subrealm during interactive or silent installation, use the subrealm during the installation or in the response file. For example, instead of:

```
AGENT_PROFILE_REALM = /
```

specify:

```
AGENT_PROFILE_REALM = /myrealm
```

Even though the agent is installed in a subrealm, the default login redirect requires users to log into the top-level realm. For information about how to change the login, refer to [Use the request domain to redirect login to a different realm](#).

## Install JBoss Java Agent

The examples in this section assume that you are using JBoss, but the procedures are the same for WildFly. Agent binaries for JBoss and WildFly are the same.

### *Install JBoss Java Agent interactively*

1. Review the information in [Before you install](#), and perform the steps in [Preinstallation tasks](#).
2. Shut down the JBoss server where you plan to install the agent.
3. Make sure AM is running.
4. Run **agentadmin --install** to install the agent:

```
$ /path/to/java_agents/jboss_agent/bin/agentadmin --install
```

5. Enter the absolute path to the JBoss installation directory:

```
...  
[ ? : Help, ! : Exit ]  
Enter the path to the JBoss installation: /path/to/jboss
```

6. Enter the name of the deployment mode for the JBoss installation:

- **standalone** : Manage a single JBoss instance

In standalone mode, the agent installer uses an auto-deployment feature provided by the JBoss deployment scanner so that you do not have to deploy the `agentapp.war` manually.

- **domain** : Manage multiple server instances from a single control point.

In this mode, at the end of the procedure, you must manually deploy the `java_agents/jboss_agent/etc/agentapp.war` file to JBoss.

7. Enter the name of the profile to use in `standalone` or `domain` mode:

- `standalone` : Default.
- `full` : Supports Java EE 6 Full Profile, and subsystems that are not required for high-availability.
- `ha` : Enables all default subsystems, and adds the clustering capabilities.
- `full-ha` : Enables all default subsystems, including those required for high-availability, and adds clustering capabilities.

8. Choose whether to deploy the agent as a global JBoss module.

```
...
[ ? : Help, < : Back, ! : Exit ]
Install agent as global module? [true]: true
```

To include specific modules for a web application, enter `false`, and complete the additional steps at the end of this procedure.

9. Enter the AM URL, including the deployment URI:

```
...
[ ? : Help, < : Back, ! : Exit ]
AM server URL: https://am.example.com:8443/am
```

To load balance connections between the agent and an AM site, enter the URL of the load balancer in front of the AM site.

If a reverse proxy is configured between AM and the agent, enter the proxy URL. For more information, refer to [Configure an Apache HTTP Server as a reverse proxy](#).

10. Enter the agent URL:

```
...
[ ? : Help, < : Back, ! : Exit ]
Agent URL: https://agent.example.com:443/app
```

11. Enter the agent profile name created in AM as part of the pre-installation procedure:

```
...
[ ? : Help, < : Back, ! : Exit ]
Enter the Agent Profile name: JBossAgent
```

12. Enter the realm in which the specified agent profile exists.

Press  to accept the default value of `/` for the top-level realm. If you specify the () : Accept Empty value option, the top-level realm is used.

```
...
[ ? : Help, < : Back, ! : Exit, ^ : Accept Empty value ]
Enter the Agent Profile realm [/]:
```

13. Enter the path to the password file you created as part of the pre-installation procedure:

```
...
[ ? : Help, < : Back, ! : Exit ]
Enter the path to the password file: /secure-directory/pwd.txt
```

- a. Enter the path to a file containing the agent pre-authentication cookie signing value:

```
...
[ ? : Help, < : Back, ! : Exit ]
Enter the path to the signing file:
```

Provide a path to a file containing a randomly generated key that is at least 64 characters long but preferably about 80 characters. For help to create signing a key, refer to [Create a cookie signing key](#).

For information about how the agent uses pre-authentication cookies, refer to the *Authentication* section of [Request flow](#).

To disable cookie signing, press return without providing a value.

**TIP**

Cookie signing is a CPU-intensive process that renders cookies more tamper-proof. Weigh the potential increase in security against the potential loss in performance.

14. Review a summary of your responses and select how to continue:

```
...
Verify your settings above and decide from the choices below.
1. Continue with Installation
2. Back to the last interaction
3. Start Over
4. Exit
Please make your selection [1]: 1
...
```

After successful completion, the installer updates the JBoss configuration, adds the agent web application under `JBOSS_HOME/server/standalone/deployments`, and sets up configuration and log directories for the agent.

15. Follow these steps if you responded `false` to the question `Deploy the policy agent as a global JBoss module during the installation`:

- a. Add the following line to the web application file `/path/to/protected/app/META-INF/MANIFEST.MF`:

```
Dependencies: org.forgerock.openam.agent
```

- b. Create a file at `/path/to/protected/app/WEB-INF/jboss-deployment-structure.xml` with the following content:

```
<?xml version="1.0"?>
  <jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.2"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <deployment>
      <dependencies>
        <module name="org.forgerock.openam.agent" >
          <imports>
            <include path="META-INF"/>
            <include path="org"/>
          </imports>
        </module>
      </dependencies>
    </deployment>
  </jboss-deployment-structure>
```

16. If you chose `domain` as the deployment mode, manually deploy the `java_agents/jboss_agent/etc/agentapp.war` file to JBoss.
17. Test the installation by browsing to a resource that the agent protects. AM redirects you to authenticate. After authentication, AM redirects you back to the requested resource.

### *Install JBoss Java Agent Silently*

To install the Java Agent silently, create a response file containing the installation parameters, and then provide it to the **agentadmin** command.

The following is an example response file to install the agent when JBoss is configured in `standalone` mode:

```
# Agent User Response File
HOME_DIR= /path/to/jboss
INSTANCE_NAME= standalone
GLOBAL_MODULE= true
INSTALL_PROFILE_NAME=
AM_SERVER_URL= https://am.example.com:8443/am
AGENT_URL= https://agent.example.com:443/app
AGENT_PROFILE_NAME= JBossAgent
AGENT_PROFILE_REALM= /
AGENT_PASSWORD_FILE= /secure-directory/pwd.txt
AGENT_SIGNING_FILE= /secure-directory/signing-key.txt
```

The `INSTALL_PROFILE_NAME` variable is used only when the `INSTANCE_NAME` is set to `domain`. It specifies the name of the JBoss domain profile.

To load balance connections between the agent and an AM site, set `AM_SERVER_URL` to the URL of the load balancer in front of the AM site.

If a reverse proxy is configured between AM and the agent, set `AM_SERVER_URL` to the proxy URL. For more information, refer to [Configure an Apache HTTP Server as a reverse proxy](#).

1. Review the information in [Before you install](#), and perform the steps in [Preinstallation tasks](#).
2. Make sure that the response file for the installation is ready, or create a response file, for example:

```
$ agentadmin --install --saveResponse response-file
```

3. Shut down the JBoss server where you plan to install the agent.
4. Make sure AM is running.
5. Run the **agentadmin** command with the `--useResponse` option:

```
$ agentadmin --install --useResponse /path/to/response-file
```

6. If you configured the `GLOBAL_MODULE` variable as `false` in the response file, add the following line to the `META-INF/MANIFEST.MF` file of the web application:

```
Dependencies: org.forgerock.openam.agent
```

7. If you configured the `INSTANCE_NAME` variable as `domain` in the response file, manually deploy the `java_agents/jboss_agent/etc/agentapp.war` file to JBoss.

## Install in a subrealm

Other installation examples install the agent in the top-level realm. To install the agent in a subrealm during interactive or silent installation, use the subrealm during the installation or in the response file. For example, instead of:

```
AGENT_PROFILE_REALM = /
```

specify:

```
AGENT_PROFILE_REALM = /myrealm
```

Even though the agent is installed in a subrealm, the default login redirect requires users to log into the top-level realm. For information about how to change the login, refer to [Use the request domain to redirect login to a different realm](#).

## Install Jetty Java Agent

Consider the following points before you install:

- For installation on Jetty 12, you can use Javax EE8, Jakarta EE9, or Jakarta EE10. Make sure you use the correct agent (Javax or Jakarta) for your environment.
- Command-line examples in this chapter show Jetty accessed remotely. If you have issues accessing Jetty remotely, consider changing the filter settings in the deployment descriptor file, `/path/to/jetty/webapps/test/WEB-INF/web.xml`, as shown in the following example:

```
<filter>
<filter-name>TestFilter</filter-name>
<filter-class>com.acme.TestFilter</filter-class>
<init-param>
  <param-name>remote</param-name>
  <param-value>true</param-value> <!-- default: false -->
</init-param>
</filter>
```

## Jetty configuration

The Jetty compliance mode is configured by default to prevent path traversal vulnerabilities. Consider the impact on security before you change [org.eclipse.jetty.http.UriCompliance](#) to a less safe value such as `UNSAFE` or `RFC3986`.

## *Install Jetty Java Agent interactively*

1. Review the information in [Before you install](#), and perform the steps in [Preinstallation tasks](#).
2. Shut down the Jetty server where you plan to install the agent.
3. Make sure AM is running.
4. Run **agentadmin --install** to install the agent:

```
$ /path/to/java_agents/jetty_agent/bin/agentadmin --install
```

5. Enter the absolute path to the root of the Jetty installation:

```
...
[ ? : Help, ! : Exit ]
Enter the Jetty home directory [/opt/jetty]:
/path/to/jetty/home
```

This is the same as the JETTY\_HOME environment variable for Jetty.

6. Enter the absolute path to the Jetty configuration directory:

```
...
[ ? : Help, &lt; : Back, ! : Exit ]
Enter the absolute path of the Jetty etc directory:
/path/to/jetty/etc
```

7. Enter the absolute path to the Jetty base directory:

```
...
[ ? : Help, < : Back, ! : Exit ]
Enter the Jetty base directory [/usr/local/jetty]:
/path/to/jetty/base
```

This is the same as the JETTY\_BASE environment variable for Jetty.

This path may be the same as the one specified as the root of the Jetty installation.

8. Enter the AM URL, including the deployment URI:

```
...
[ ? : Help, < : Back, ! : Exit ]
AM server URL: https://am.example.com:8443/am
```

To load balance connections between the agent and an AM site, enter the URL of the load balancer in front of the AM site.

If a reverse proxy is configured between AM and the agent, enter the proxy URL. For more information, refer to [Configure an Apache HTTP Server as a reverse proxy](#).

9. Enter the agent URL:

```
...
[ ? : Help, < : Back, ! : Exit ]
Agent URL: https://agent.example.com:443/app
```

10. Enter the agent profile name created in AM as part of the pre-installation procedure:

```
...
[ ? : Help, &lt; : Back, ! : Exit ]
Enter the Agent Profile name: JettyAgent
```

11. Enter the realm in which the specified agent profile exists.

Press  to accept the default value of / for the top-level realm. If you specify the () : Accept Empty value option, the top-level realm is used.

```
...
[ ? : Help, < : Back, ! : Exit, ^ : Accept Empty value ]
Enter the Agent Profile realm [/]:
```

12. Enter the path to the password file you created as part of the pre-installation procedure:

```
...
[ ? : Help, < : Back, ! : Exit ]
Enter the path to the password file: /secure-directory/pwd.txt
```

a. Enter the path to a file containing the agent pre-authentication cookie signing value:

```
...
[ ? : Help, < : Back, ! : Exit ]
Enter the path to the signing file:
```

Provide a path to a file containing a randomly generated key that is at least 64 characters long but preferably about 80 characters. For help to create signing a key, refer to [Create a cookie signing key](#).

For information about how the agent uses pre-authentication cookies, refer to the *Authentication* section of [Request flow](#).

To disable cookie signing, press return without providing a value.

**TIP**

Cookie signing is a CPU-intensive process that renders cookies more tamper-proof. Weigh the potential increase in security against the potential loss in performance.

13. Review a summary of your responses and select how to continue:

```
...
Verify your settings above and decide from the choices below.
1. Continue with Installation
2. Back to the last interaction
3. Start Over
4. Exit
Please make your selection [1]: 1
...
```

After successful completion, the installer updates Jetty's `start.jar` to refer to the agent, sets up the agent web application, and sets up configuration and log directories for the agent.

14. Test the installation by browsing to a resource that the agent protects. AM redirects you to authenticate. After authentication, AM redirects you back to the requested resource.

### *Install Jetty Java Agent silently*

To install the Java Agent silently, create a response file containing the installation parameters, and then provide it to the **agentadmin** command. The following is an example response file:

```
# Agent User Response File
CONFIG_DIR= /path/to/jetty/etc
JETTY_HOME= /path/to/jetty/home
JETTY_BASE= /path/to/jetty/base
AM_SERVER_URL= https://am.example.com:8443/am
AGENT_URL= https://agent.example.com:443/app
AGENT_PROFILE_NAME= JettyAgent
AGENT_PROFILE_REALM= /
AGENT_PASSWORD_FILE= /secure-directory/pwd.txt
AGENT_SIGNING_FILE= /secure-directory/signing-key.txt
```

To load balance connections between the agent and an AM site, set `AM_SERVER_URL` to the URL of the load balancer in front of the AM site.

If a reverse proxy is configured between AM and the agent, set `AM_SERVER_URL` to the proxy URL. For more information, refer to [Configure an Apache HTTP Server as a reverse proxy](#).

1. Review the information in [Before you install](#), and perform the steps in [Preinstallation tasks](#).
2. Shut down the Jetty server where you plan to install the agent.
3. Make sure that AM is running.
4. Run the **agentadmin** command with the `--useResponse` option:

```
$ agentadmin --install --useResponse /path/to/response-file
```

### *Install in a subrealm*

Other installation examples install the agent in the top-level realm. To install the agent in a subrealm during interactive or silent installation, use the subrealm during the installation or in the response file. For example, instead of:

```
AGENT_PROFILE_REALM = /
```

specify:

```
AGENT_PROFILE_REALM = /myrealm
```

Even though the agent is installed in a subrealm, the default login redirect requires users to log into the top-level realm. For information about how to change the login, refer to [Use the request domain to redirect login to a different realm](#).

## Install WebLogic Java Agent

### *Install WebLogic Java Agent interactively*

1. Review the information in [Before you install](#), and perform the steps in [Preinstallation tasks](#).
2. Shut down the WebLogic server where you plan to install the agent.
3. Make sure AM is running.
4. Run **agentadmin --install** to install the agent:

```
$ /path/to/java_agents/weblogic_agent/bin/agentadmin --install
```

5. Enter the path to the `startWebLogic.sh` file of the WebLogic domain where you want to install the agent:

```
...
[ ? : Help, ! : Exit ]
Enter the Startup script location
[/usr/local/bean/user_projects/domains/base_domain/startWebLogic.sh]:
/path/to/Oracle_Home/user_projects/domains/base_domain/startWebLogic.sh
```

6. Enter the path to the WebLogic installation directory:

```
...
[ ? : Help, < : Back, ! : Exit ]
Enter the WebLogic home directory
[/usr/local/bean/wlserver_10.0]:
/path/to/weblogic
```

7. Enter the AM URL, including the deployment URI:

```
...
[ ? : Help, < : Back, ! : Exit ]
AM server URL: https://am.example.com:8443/am
```

To load balance connections between the agent and an AM site, enter the URL of the load balancer in front of the AM site.

If a reverse proxy is configured between AM and the agent, enter the proxy URL. For more information, refer to [Configure an Apache HTTP Server as a reverse proxy](#).

8. Enter the agent URL:

```
...
[ ? : Help, < : Back, ! : Exit ]
Agent URL: https://agent.example.com:443/app
```

9. Enter the agent profile name created in AM as part of the pre-installation procedure:

```
...
[ ? : Help, < : Back, ! : Exit ]
```

Enter the Agent Profile name: **WebLogicAgent**

10. Enter the realm in which the specified agent profile exists.

Press **ENTER** to accept the default value of / for the top-level realm. If you specify the (**^**) : Accept Empty value option, the top-level realm is used.

```
...
[ ? : Help, < : Back, ! : Exit, ^ : Accept Empty value ]
Enter the Agent Profile realm [/]:
```

11. Enter the path to the password file you created as part of the pre-installation procedure:

```
...
[ ? : Help, < : Back, ! : Exit ]
Enter the path to the password file: /secure-directory/pwd.txt
```

a. Enter the path to a file containing the agent pre-authentication cookie signing value:

```
...
[ ? : Help, < : Back, ! : Exit ]
Enter the path to the signing file:
```

Provide a path to a file containing a randomly generated key that is at least 64 characters long but preferably about 80 characters. For help to create signing a key, refer to [Create a cookie signing key](#).

For information about how the agent uses pre-authentication cookies, refer to the *Authentication* section of [Request flow](#).

To disable cookie signing, press return without providing a value.

**TIP**

Cookie signing is a CPU-intensive process that renders cookies more tamper-proof. Weigh the potential increase in security against the potential loss in performance.

12. Review a summary of your responses and select how to continue:

```
$ /path/to/java_agents/weblogic_agent/bin/agentadmin --install
```

```
...
Verify your settings above and decide from the choices below.
```

```
1. Continue with Installation
2. Back to the last interaction
3. Start Over
4. Exit
Please make your selection [1]: 1
...
```

13. Source the agent in one of the following ways:

- Manually source the file containing the agent environment settings for WebLogic before starting the container.

```
$ . /path/to/setAgentEnv_AdminServer.sh
```

- Add the `setAgentEnv_AdminServer.sh` line to the shown location [path] in the `startWebLogic.sh` script. Note that the file can be overwritten:

```
$ cat /path/to/startWebLogic.sh
...
# Any changes to this script may be lost when adding
extensions to this
# configuration.
DOMAIN_HOME="/opt/Oracle/Middleware/user_projects/domains/
base_domain"
. /path/to/setAgentEnv_AdminServer.sh
${DOMAIN_HOME}/bin/startWebLogic.sh $*
```

If the sourcing is not set properly, the following message appears:

```
<Error> <HTTP> <cent.example.com>
<AdminServer> <[STANDBY] ExecuteThread: '5' for queue:
weblogic.kernel.
Default (self-tuning)'> <<WLS Kernel>>
<BEA-101165> <Could not load user defined filter in
web.xml:
ServletContext@1761850405[app:agentapp module:agentapp.war
path:null
spec-version:null]
com.sun.identity.agents.filter.AmAgentFilter.
java.lang.ClassNotFoundException:
com.sun.identity.agents.filter.AmAgentFilter
```

14. Start the WebLogic server.

15. Deploy the `/path/to/java_agents/weblogic_agent/etc/agentapp.war` agent web application in WebLogic.

16. Test the installation by browsing to a resource that the agent protects. AM redirects you to authenticate. After authentication, AM redirects you back to the requested resource.

### *Install WebLogic Java Agent silently*

To install the Java Agent silently, create a response file containing the installation parameters, and then provide it to the **agentadmin** command. The following is an example response file:

```
# Agent User Response File
STARTUP_SCRIPT=
/path/to/Oracle_Home/user_projects/domains/base_domain/startWebLogic.sh
SERVER_NAME= AdminServer
WEBLOGIC_HOME_DIR= /path/to/weblogic
AM_SERVER_URL= https://am.example.com:8443/am
AGENT_URL= https://agent.example.com:443/app
AGENT_PROFILE_NAME= WebLogicAgent
AGENT_PROFILE_REALM= /
AGENT_PASSWORD_FILE= /secure-directory/pwd.txt
AGENT_SIGNING_FILE= /secure-directory/signing-key.txt
```

To load balance connections between the agent and an AM site, set **AM\_SERVER\_URL** to the URL of the load balancer in front of the AM site.

If a reverse proxy is configured between AM and the agent, set **AM\_SERVER\_URL** to the proxy URL. For more information, refer to [Configure an Apache HTTP Server as a reverse proxy](#).

1. Review the information in [Before you install](#), and perform the steps in [Preinstallation tasks](#).
2. Make sure that the response file for the installation is ready, or create a response file, for example:

```
$ agentadmin --install --saveResponse response-file
```

3. Shut down the WebLogic server where you plan to install the agent.
4. Make sure AM is running.
5. Run the **agentadmin** command with the **--useResponse** option:

```
$ agentadmin --install --useResponse /path/to/response-file
```

6. Source the agent in one of the following ways:

- Manually source the file containing the agent environment settings for WebLogic before starting the container.

```
$ . /path/to/setAgentEnv_AdminServer.sh
```

- Add the `setAgentEnv_AdminServer.sh` line to the shown location [path] in the `startWebLogic.sh` script. Note that the file can be overwritten:

```
$ cat /path/to/startWebLogic.sh
...
# Any changes to this script may be lost when adding
extensions to this
# configuration.
DOMAIN_HOME="/opt/Oracle/Middleware/user_projects/domains/
base_domain"
. /path/to/setAgentEnv_AdminServer.sh
${DOMAIN_HOME}/bin/startWebLogic.sh $*
```

If the sourcing is not set properly, the following message appears:

```
<Error> <HTTP> <cent.example.com>
<AdminServer> <[STANDBY] ExecuteThread: '5' for queue:
weblogic.kernel.
Default (self-tuning)'\> <<WLS Kernel>>
<BEA-101165> <Could not load user defined filter in
web.xml:
ServletContext@1761850405[app:agentapp module:agentapp.war
path:null
spec-version:null]
com.sun.identity.agents.filter.AmAgentFilter.
java.lang.ClassNotFoundException:
com.sun.identity.agents.filter.AmAgentFilter
```

7. Start the WebLogic Server.
8. Deploy the `/path/to/java_agents/weblogic_agent/etc/agentapp.war` agent web application in WebLogic.

### *Install WebLogic Java Agent in multi-server domains*

In many WebLogic domains, the administration server provides a central point for controlling and managing the configuration of the managed servers that host protected web applications.

If WebLogic-managed servers run on different hosts, you must create separate agent profiles and perform separate installations for each so that AM can send notifications to the appropriate addresses.

### *Install WebLogic Java Agent on administration and managed servers*

1. If servers are on different hosts, create agent profiles for each server where you plan to install the agent. For more information, refer to [Installing the WebLogic Java Agent](#).
2. Prepare your protected web applications by adding the agent filter configuration as described in [Configure the agent filter for a web application](#).
3. Use the **agentadmin** command to install the agent either interactively, or silently on each server in the domain:
  - For interactive installation, follow the instructions in [To install the WebLogic Java Agent](#).
  - For silent installation, follow the instructions in [Installing the WebLogic Java Agent silently](#).
4. On each managed server in the domain, update the classpath to include agent .jar files.

In WebLogic Node Manager console, navigate to Environment > Servers > *server* > **Server Start** > **Class Path**, and then edit the classpath as in the following example, but all on a single line:

```
/path/to/java_agents/weblogic_agent/lib/agent.jar:  
/path/to/java_agents/weblogic_agent/lib/opensocclientsdk.jar:  
...  
/path/to/java_agents/weblogic_agent/locale:  
/path/to/java_agents/weblogic_agent/Agent_001/config:  
$CLASSPATH
```

Replace the paths in the example with the actual paths for your domain.

5. Restart the managed servers.

## Post-installation tasks

---

### Review directories for configuration, logs, and POST data


Each agent instance has a numbered directory, starting with Agent\_001 for the first instance. The following directories are created under

/path/to/java\_agents/*agent\_type*/Agent\_*n*:

- `config` : Learn more from [Agent configuration](#).
- `logs` : During agent startup, the location of the logs is based on the container which is being used. For example, bootstrap logs for Tomcat agents are written to `catalina.out` . The following log directories are created:
  - `logs/audit/` : Operational audit log directory, used only if remote logging to AM is disabled.
  - `logs/debug/` : The directory where the agent writes debug log files after startup.
- `pdp` : The directory to store POST data. The directory is created on installation, but used only when [Enable POST Data Preservation](#) and [POST Data Preservation in Files or Cache](#) are `true` .

## Configure the agent filter for a web application

After installation, configure an *agent filter* to intercept inbound client requests and give them access to resources. The agent filter class is `com.sun.identity.agents.filter.AmAgentFilter` . The agent filter gives access based on the value of [Agent Filter Mode Map](#).

Configure the agent filter in the web application's `web.xml` file. For information about configuration options, refer to the documentation for your web application. For example, refer to Oracle's [Developing Web Applications for WebLogic Server](#) .

Configure the agent filter first, before configuring other filters in `web.xml` . If several web applications run in the same container, configure an agent filter for each web application.

The following example protects every resource in the web application where it is configured:

```
<filter>
  <filter-name>Agent</filter-name>
  <display-name>AM Agent</display-name>
  <description>AM Agent Filter</description>
  <filter-
class>com.sun.identity.agents.filter.AmAgentFilter</filter-class>
</filter>
<filter-mapping>
  <filter-name>Agent</filter-name>
  <url-pattern>/*</url-pattern>
  <dispatcher>REQUEST</dispatcher>
  <dispatcher>INCLUDE</dispatcher>
  <dispatcher>FORWARD</dispatcher>
```

```
<dispatcher>ERROR</dispatcher>
</filter-mapping>
```

The following example protects an application that processes requests asynchronously:

```
<filter>
  <filter-name>Agent</filter-name>
  <display-name>AM Agent</display-name>
  <description>AM Agent Filter</description>
  <filter-
class>com.sun.identity.agents.filter.AmAgentFilter</filter-class>
  <async-supported>true</async-supported>
</filter>
```

## Configure the agent filter mode

By default, the agent filter uses the filter mode `URL_POLICY`. After installation, you can change the filter mode with the property [Agent Filter Mode Map](#), or in the AM admin UI:

1. In the AM admin UI, go to **Realms# > *Realm Name* > Applications > Agents > Java**, and select your Java Agent.
2. On the **Global** tab, select **Agent Filter Mode Map**, and set the filter mode as follows:
  - To use `URL_POLICY` for all web applications in the web container, do not change the setting; this is the default.
  - To use `SSO_ONLY` for the `BankApp` web application, set these values:
3. (Optional) In **Agent Filter Mode**, override the global mode for a specific context path:
  - **Key:** `BankApp`.
  - **Value:** Enter the mode name, for example `URL_POLICY`.
4. Click **Add**, and save your changes.

## Configure strategy for AM downtime

By default, if AM becomes unavailable at runtime, for example, due to network errors, the agent responds as follows:

- Matches incoming requests against not-enforced rules
- Resolves unmatched requests against the policy cache
- Returns HTTP 503 for requests that don't match cached results

The cache expires after the time defined by Policy Cache TTL.

To change the agent response to AM downtime, configure Strategy when AM unavailable.

## Secure connections

---

### Secure communication between the agent and AM

After installation, consider securing communication between the agent and AM.

1. Configure AM to send cookies only when the communication channel is secure:
  - a. In the AM admin UI, select **Realms** > *Realm Name* > **Applications** > **Agents** > **Java** > *Agent Name* > **SSO**.
  - b. Enable Transmit Cookies Securely.
2. Import a CA certificate in the JDK truststore, usually at `$JAVA_HOME/jre/lib/security/cacerts`. The certificate should be the one configured for HTTPS connections in the AM container, or signed with the same CA root certificate. For example:

```
$ keytool \  
-import \  
-trustcacerts \  
-alias agentcert \  
-file /path/to/cacert.pem \  
-keystore $JAVA_HOME/jre/lib/security/cacerts
```

Make sure that all containers where AM is installed trust the certificate stored in the JDK truststore, and that the JDK trusts the certificates stored on the containers where AM is installed.

3. Add the following properties to the `AgentBootstrap.properties` file:
  - `javax.net.ssl.trustStore`, to specify the full path to the JDK truststore.
  - `javax.net.ssl.trustStorePassword`, to specify the password of the truststore.

For example:

```
javax.net.ssl.trustStore=/Library/Java/JavaVirtualMachines  
/jdk1.8.0_101.jdk/Contents/Home/jre/lib/security/cacerts  
javax.net.ssl.trustStorePassword=changeit
```

For backward-compatibility, you can also provide the truststore and the password to the agent by specifying them as Java properties in the container's start-up sequence. For example, add them to Tomcat's `$CATALINA_OPTS` variable instead of specifying them in the `AgentBootstrap.properties` file:

```
$ export CATALINA_OPTS="$CATALINA_OPTS \  
-  
Djavax.net.ssl.trustStore=$JAVA_HOME/jre/lib/security/cac  
erts \  
-Djavax.net.ssl.trustStorePassword=changeit"
```

4. Restart the agent.

## Integrate with Bouncy Castle FIPS provider

This section gives an example of how to use the Bouncy Castle FIPS provider. For more information, refer to [JAVA FIPS RESOURCES](#)<sup>☞</sup>. The example uses Tomcat Java Agent but you can adapt it for other agent types.

Perform this procedure before installing the agent and starting the container.

1. Download the latest version of Bouncy Castle FIPS library from [JAVA FIPS RESOURCES](#)<sup>☞</sup>. This example uses `bc-fips-1.0.2.3.jar`.
2. Copy the `.jar` file to the agent library:
  - a. Using the `.amAgentLocator` file, find the directory in which the agent is installed. In this example, the agent is installed in `/path/to/java_agents/tomcat_agent`:

**Unix**

**Windows**

```
$ cd /path/to/tomcat  
$ cat .amAgentLocator; echo  
  
/path/to/java_agents/tomcat_agent
```

- b. Copy `bc-fips-1.0.2.3.jar` to the `lib` subdirectory:

**Unix**

**Windows**

```
$ cd /path/to/java_agents/tomcat_agent/lib  
$ cp /tmp/bc-fips-1.0.2.3.jar  
/path/to/java_agents/tomcat_agent/lib
```

3. Set up the security providers to use Bouncy Castle:

- a. Locate the `java.security` file for your Java instance. For example, in Java 17 and Ubuntu the file is `/etc/java-17-openjdk/security/java.security`.
- b. Edit the file to place the `org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider` line at the top of the list:

```
security.provider.1=org.bouncycastle.jcajce.provider.Bounc
yCastleFipsProvider
security.provider.2=SUN
security.provider.3=SunRsaSign
security.provider.4=SunEC
security.provider.5=SunJSSE
security.provider.6=SunJCE
security.provider.7=SunJGSS
security.provider.8=SunSASL
security.provider.9=XMLDSig
security.provider.10=SunPCSC
security.provider.11=JdkLDAP
security.provider.12=JdkSASL
security.provider.13=SunPKCS11
```

4. In the agent configuration, set `org.bouncycastle.fips.approved_only` to `true` so that only algorithms approved by FIPS can be used:

- a. Locate the `agentadmin` file:

**Unix** **Windows**

```
$ cd /path/to/java_agents/tomcat_agent/bin
```

- b. Change the following line:

```
AGENT_OPTS="$AGENT_OPTS -Dagent.config.dir=$AGENT_HOME"
```

to this line:

```
AGENT_OPTS="$AGENT_OPTS -Dagent.config.dir=$AGENT_HOME -
Dorg.bouncycastle.fips.approved_only=true"
```

5. Configure the Tomcat container to use the BouncyCastle provider. There are many ways to configure the container; this example uses a `setenv.sh` file:

- a. Locate or create a `setenv.sh` file for your Tomcat container. When Tomcat installed in `/path/to/tomcat/`, the file can be `/path/to/tomcat/bin/setenv.sh`.
- b. Add the following line for the `bc-fips-1.0.2.3.jar` classpath:

```
CLASSPATH=/path/to/bc-fips-1.0.2.3.jar
```

- c. Add the following line to run the FIPS module in approved mode:

```
JAVA_OPTS="$JAVA_OPTS -  
Dorg.bouncycastle.fips.approved_only=true"
```

- d. (Optional) Add the following property to the `JAVA_OPTS` to enable logs:

```
-Djava.security.debug=jca
```

6. Install the agent and start the container, as described in [Install Tomcat Java Agent](#).

## Remove Java Agent

---

### Remove Tomcat Java Agent

1. Shut down the server where the agent is installed.
2. Run the **agentadmin** command with the `--listAgents` option list installed agent instances:

```
$ agentadmin --listAgents  
The following agents are configured on this Application  
Server.  
...  
The following are the details for agent Agent_001 :-  
...
```

3. Note the configuration information of the agent instance you want to remove.
4. Run the **agentadmin** command with the `--uninstall` option.

```
$ agentadmin --uninstall
```

5. Enter the path of the Tomcat installation directory:

```
Enter the complete path to the directory which is used by
Tomcat Server to
store its configuration Files. This directory uniquely
identifies the
Tomcat Server instance that is secured by this Agent.
[ ? : Help, ! : Exit ]
Enter the Tomcat Server Config Directory Path
[/opt/apache-tomcat/conf]: /path/to/apache-tomcat/conf
```

6. Review a summary of your responses and select how to continue:

```
-----
SUMMARY OF YOUR RESPONSES
-----
...
Verify your settings above and decide from the choices below.
1. Continue with Uninstallation
2. Back to the last interaction
3. Start Over
4. Exit
Please make your selection [1]: **1**
...
```

## Remove JBoss Java Agent

1. Shut down the server where the agent is installed.
2. Run the **agentadmin** command with the **--listAgents** option list installed agent instances:

```
$ agentadmin --listAgents
The following agents are configured on this Application
Server.
...
The following are the details for agent Agent_001 :-
...
```

3. Note the configuration information of the agent instance you want to remove.
4. Run the **agentadmin** command with the **--uninstall** option.

```
$ agentadmin --uninstall
```

5. Enter the path to the JBoss installation directory:

```
Enter the complete path to the home directory of the JBoss
instance.
[ ? : Help, ! : Exit ]
Enter the path to the JBoss installation: /path/to/jboss
```

6. Enter `domain` or `standalone`, for the deployment mode of the JBoss installation to uninstall:

```
Enter the name of the deployment mode of the JBoss
installation that you wish
to use with this agent. Supported values are: domain,
standalone.
[ ? : Help, < : Back, ! : Exit ]
Enter the deployment mode of JBoss [standalone]: standalone
```

7. Review a summary of your responses and select how to continue:

```
-----
SUMMARY OF YOUR RESPONSES
-----
...
Verify your settings above and decide from the choices below.
1. Continue with Uninstallation
2. Back to the last interaction
3. Start Over
4. Exit
Please make your selection [1]: **1**
...
```

## Remove Jetty Java Agent

1. Shut down the server where the agent is installed.
2. Run the **agentadmin** command with the `--listAgents` option list installed agent instances:

```
$ agentadmin --listAgents
The following agents are configured on this Application
Server.
...
The following are the details for agent Agent_001 :-
...
```

3. Note the configuration information of the agent instance you want to remove.

4. Run the **agentadmin** command with the `--uninstall` option.

```
$ agentadmin --uninstall
```

5. Enter the path of the Jetty configuration directory:

```
Enter the complete path to the directory which is used by
Jetty Server to store
its configuration Files. This directory uniquely identifies
the Jetty
Server instance that is secured by this Agent.
[ ? : Help, ! : Exit ]
Enter the Jetty Server Config Directory Path [/opt/jetty/etc]:
/path/to/jetty/etc
```

6. Review a summary of your responses and select how to continue:

```
-----
SUMMARY OF YOUR RESPONSES
-----
...
Verify your settings above and decide from the choices below.
1. Continue with Uninstallation
2. Back to the last interaction
3. Start Over
4. Exit
Please make your selection [1]: 1
...
```

## Remove WebLogic Java Agent

1. Shut down the server where the agent is installed.
2. Run the **agentadmin** command with the `--listAgents` option list installed agent instances:

```
$ agentadmin --listAgents
The following agents are configured on this Application
Server.
...
The following are the details for agent Agent_001 :-
...
```

3. Note the configuration information of the agent instance you want to remove.

4. Run the **agentadmin** command with the **--uninstall** option.

```
$ agentadmin --uninstall
```

5. Enter the path to the **startWebLogic.sh** file of the WebLogic domain where you want to install the agent:

```
Enter the path to the location of the script used to start the
WebLogic domain.
Please ensure that the agent is first installed on the admin
server instance
before installing on any managed server instance.
[ ? : Help, ! : Exit ]
Enter the Startup script location
[/usr/local/boa/user_projects/domains/base_domain/startWebLogi
c.sh]:
/Oracle_Home/user_projects/domains/base_domain/startWebLogic.s
h
```

6. Enter the name of the WebLogic instance:

```
Enter the name of the WebLogic Server instance secured by the
agent.
[ ? : Help, < : Back, ! : Exit ]
Enter the WebLogic Server instance name [AdminServer]:
AdminServer
```

7. Review a summary of your responses and select how to continue:

```
-----
SUMMARY OF YOUR RESPONSES
-----
...
Verify your settings above and decide from the choices below.
1. Continue with Uninstallation
2. Back to the last interaction
3. Start Over
4. Exit
Please make your selection [1]: 1
...
```

## Deploy Java Agent with Docker

The example in this section provides a Dockerfile and instructions to deploy Tomcat Java Agent on Linux to extend and protect an application. Adapt the information for other agent containers and platforms.

Consider the following limitations:

- The Dockerfile doesn't manage logs, so agent logs are lost when the Docker container is killed. Manage logs independently of the Dockerfile in the following ways, according to your environment:
  - Store logs persistently to a volume
  - Store logs to a host machine
  - Tail logs into STDOUT or STDERR so that Docker can collect the data
- The Dockerfile isn't suitable for local configuration mode and doesn't update bootstrap properties. The agent must be configured to operate in the default remote configuration mode. For more information, refer to Location of Agent Configuration Repository.

## Deploy Tomcat Java Agent example

1. In Advanced Identity Cloud or AM, set up an agent profile and policy. For more information, refer to Advanced Identity Cloud's Prepare for installation or AM's Prepare for installation.

This example uses the following configuration:

- AM URL: `https://am.example.com:8443/am`
  - AM realm: `top-level`
  - Agent URL: `https://agent.example.com:443/app`
  - Agent profile name: `java-agent`
  - Agent profile password: `password`
  - Policy set and policy: Allow HTTP GET and POST for all authenticated users.
2. Create a local folder for your application's `web.xml` file, the agent .zip file, the Dockerfile, and the agent profile password—they must be in the same folder. This example uses `/path/to/docker`.
  3. Build a Docker image of your web application. This example uses a sample application called `fr-sample-app:1.0`.
  4. Configure the agent filter in your application's `web.xml` file and save it to `/path/to/docker/web.xml`. For more information, refer to Configure the agent filter for a web application.

This example uses the following `web.xml` file:

```

<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
Application 2.3//EN" "http://java.sun.com/dtd/web-
app_2_3.dtd">
<web-app version="3.0"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
  <filter>
    <filter-name>Agent</filter-name>
    <display-name>AM Agent</display-name>
    <description>AM Agent Filter</description>
    <filter-
class>com.sun.identity.agents.filter.AmAgentFilter</filter-
class>
  </filter>
  <filter-mapping>
    <filter-name>Agent</filter-name>
    <url-pattern>/*</url-pattern>
    <dispatcher>REQUEST</dispatcher>
    <dispatcher>INCLUDE</dispatcher>
    <dispatcher>FORWARD</dispatcher>
    <dispatcher>ERROR</dispatcher>
  </filter-mapping>
  <servlet>
    <servlet-name>ServletInfo</servlet-name>
    <servlet-class>org.forgerock.ServletInfo</servlet-class>
  </servlet>
</web-app>

```

5. Download the agent .zip file to /path/to/docker/. The .zip in this example is tomcat\_agent\_2024.11.zip.
6. Create a file containing the agent profile password. The filename in this example is agent\_secret and the password is password.

```

/path/to/docker$ cat > agent_secret
password
CTRL+D

```

Although the agent accepts any password length and content, you are strongly encouraged to generate secure passwords. This can be achieved in various ways, for example using a password manager or by using the command line tool `agentadmin --key`.

7. Create the following Dockerfile in `/path/to/docker/Dockerfile`. Arguments are provided by the build command.

```
# Application Docker image
ARG BASE_DOCKER_IMAGE
FROM ${BASE_DOCKER_IMAGE}

# Install and unzip the application, required for unpacking
the agent build.
# Not required if the base image is already unzipped.
# For non-Debian Linux distributions, use the appropriate
package manager.
RUN apt-get update && \
    apt-get install unzip --no-install-recommends -y && \
    apt-get clean

# Define the build arguments.
# Arguments without default values must be specified in the
build command.
ARG AGENT_VERSION
ARG AGENT_ZIP_FILE=tomcat_agent_2024.11.zip
ARG APP_NAME
ARG AGENT_HOME=/opt
ARG AM_URL
ARG SERVER_HOME=/usr/local/tomcat
ARG AGENT_URL=http://agent.dummy.url:8080/app
ARG AGENT_REALM=/
ARG AGENT_PROFILE

# Copy the agent .zip file to the Docker directory where the
agent is installed.
COPY ${AGENT_ZIP_FILE} ${AGENT_HOME}/${AGENT_ZIP_FILE}

# Unzip the agent and delete the .zip file
RUN cd ${AGENT_HOME} && \
    unzip ./${AGENT_ZIP_FILE} && \
    rm -rf ./${AGENT_ZIP_FILE}

# Create an agent installation file called install_file
```

```

RUN echo "CONFIG_DIR= ${SERVER_HOME}/conf" >
${AGENT_HOME}/install_file && \
    echo "AM_SERVER_URL= ${AM_URL}" >>
${AGENT_HOME}/install_file && \
    echo "CATALINA_HOME= ${SERVER_HOME}" >>
${AGENT_HOME}/install_file && \
    echo "INSTALL_GLOBAL_WEB_XML= false" >>
${AGENT_HOME}/install_file && \
    echo "AGENT_URL= ${AGENT_URL}" >>
${AGENT_HOME}/install_file && \
    echo "AGENT_PROFILE_NAME= ${AGENT_PROFILE}" >>
${AGENT_HOME}/install_file && \
    echo "AGENT_PROFILE_REALM= ${AGENT_REALM}" >>
${AGENT_HOME}/install_file && \
    echo "AGENT_PASSWORD_FILE= /run/secrets/agent_secret"
>> ${AGENT_HOME}/install_file

# Install the agent and mount the file containing the agent
password
# This command uses silent installation with a provided
install_file
RUN --mount=type=secret,id=agent_secret,required=true \

"${AGENT_HOME}"/java_agents/tomcat_agent/bin/agentadmin \
    --forceInstall \
    --useResponse ${AGENT_HOME}/install_file && \
    rm -rf ${AGENT_HOME}/install_file

# Copy the new web.xml file, which includes agent filter
COPY web.xml ${AGENT_HOME}/

# Replace the original web.xml with the new web.xml file,
which includes agent filter
RUN mkdir /tmp/app && \
    cd /tmp/app/ && \
    mv ${SERVER_HOME}/webapps/${APP_NAME} ./ && \
    jar -xf ./${APP_NAME} && \
    rm -rf ./${APP_NAME} && \
    mv ${AGENT_HOME}/web.xml ./WEB-INF/web.xml && \
    jar -cf ${SERVER_HOME}/webapps/${APP_NAME} * && rm -rf
/tmp/app

```

8. Find values for the following arguments that correspond to your application and environment:

- `BASE_DOCKER_IMAGE` : The name and path to the base image of your application.
- `AGENT_VERSION` : The agent version in the Docker image.
- `AGENT_ZIP_FILE` : Name of the agent .zip file. Default: Derived from `AGENT_VERSION` . Define this property for Jakarta builds.
- `APP_NAME` : Application name including the extension. For example, `app.war`
- `AGENT_HOME` : Docker directory where the agent is installed. Default: `/opt` .
- `AM_URL` : Advanced Identity Cloud or AM server URL including port number.
- `SERVER_HOME` : Path to the Tomcat server configuration. Default: `/usr/local/tomcat` .
- `AGENT_URL` : Agent URL.
- `AGENT_REALM` : Advanced Identity Cloud or AM realm containing the agent profile.
- `AGENT_PROFILE` : Agent profile name. Default `/` .

9. With a Docker daemon running, build the Docker image with the following command, replacing the example values with your own values:

```
/path/to/docker$ docker build --secret id=agent_secret \
--build-arg BASE_DOCKER_IMAGE=fr-sample-app:1.0 \
--build-arg AGENT_VERSION=2024.11 \
--build-arg AGENT_ZIP_FILE=tomcat_agent_2024.11.zip \
--build-arg APP_NAME=app.war \
--build-arg AGENT_HOME=/opt \
--build-arg AM_URL=https://am.example.com:8443/am \
--build-arg AGENT_URL=https://agent.example.com:443/app \
--build-arg SERVER_HOME=/usr/local/tomcat \
--build-arg AGENT_REALM=/ \
--build-arg AGENT_PROFILE=java-agent \
--tag agent-image:2024.11 .

...
=> => writing image sha256:803...ada 0.0s
=> => naming to docker.io/library/java-agent:2023.11
```

10. Run the container:

```
/path/to/docker$ docker run -it --name java-agent -p 8080:8080
agent-image:2023.11

...INFO [main] org.apache.coyote.AbstractProtocol.start
Starting ProtocolHandler...
```

```
...INFO [main] org.apache.catalina.startup.Catalina.start  
Server startup ...
```

11. Access your application through the agent at <https://agent.example.com:443/app>. Access is managed by Advanced Identity Cloud or AM according to the policy configured for the agent profile.

This example displays the Advanced Identity Cloud or AM login page. When you log in as a user, you access the sample application.

## Upgrade and rollback

To upgrade or roll back an agent Docker container to a different agent version:

1. Build a new Docker container with the different agent version, using a tag that corresponds to the version.
2. Replace the image tag in your environment.

## agentadmin command

---

The **agentadmin** command manages Java Agent installation. It requires a Java runtime environment. The command supports the following options:

### **--install**

Installs a new agent instance.

Usage: **agentadmin --install [--useResponse | --saveResponse *file-name*]**

Before installation, shut down the agent container. If a service on an agent URL is responding, the installer stops with an error.

When the command is used without options, the installation process prompts for the following information:

- Information about the container installation.
- The URL of the AM instance. The agent confirms that it can log in to AM by using the profile name and password provided during installation. If unsuccessful, the installation stops with an error.
- The URL of the agent instance. The agent confirms that it can access the host and port of the URL. If the port is busy, it prompts the user to stop the container.
- The agent profile name in AM.
- The AM realm containing the agent profile.

- The path to the file containing the agent password.

#### ***--useResponse***

Run in silent mode by specifying all the responses in the *file-name* file. When this option is used, **agentadmin** runs in non-interactive mode.

#### ***--saveResponse***

Save all the supplied responses in a response file specified by *file-name*.

## **--forceInstall**

Installs a new agent instance, without checking the AM URL or agent URL.

Use this option in deployments with load balancers or reverse proxies, where the URL of the agent and AM can be concealed.

Usage: **agentadmin --forceInstall [--useResponse | --saveResponse *file-name*]**

Before installation, shut down the agent container. If a service on an agent URL is responding, the installer stops with an error.

When the command is used without options, the installation process prompts for the following information:

- Information about the container installation.
- The URL of the AM instance. The agent confirms that it can log in to AM by using the profile name and password provided during installation. If unsuccessful, the installation stops with an error.
- The URL of the agent instance. The agent confirms that it can access the host and port of the URL. If the port is busy, it prompts the user to stop the container.
- The agent profile name in AM.
- The AM realm containing the agent profile.
- The path to the file containing the agent password.

#### ***--useResponse***

Run in silent mode by specifying all the responses in the *file-name* file. When this option is used, **agentadmin** runs in non-interactive mode.

#### ***--saveResponse***

Save all the supplied responses in a response file specified by *file-name*.

## **--custom-install, --custom**

Installs a new agent instance, specifying advanced configuration options.

Usage: **agentadmin --custom-install** [--useResponse | --saveResponse *file-name*]

**--useResponse**

Run in silent mode by specifying all the responses in the *file-name* file. When this option is used, **agentadmin** runs in non-interactive mode.

**--saveResponse**

Save all the supplied responses in a response file specified by *file-name*.

**--uninstall, -r**

Uninstalls an existing agent instance.

Usage: **agentadmin --uninstall** [--useResponse | --saveResponse *file-name*]

**--useResponse**

Run in silent mode by specifying all the responses in the *file-name* file. When this option is used, **agentadmin** runs in non-interactive mode.

**--saveResponse**

Save all the supplied responses in a response file specified by *file-name*.

**--version, -v**

Displays the agent version.

Usage: **agentadmin --version**

**--uninstallAll**

Uninstalls all agent instances.

Usage: **agentadmin --uninstallAll**

**--listAgents, --list, -l**

Displays information about all configured agents.

Usage: **agentadmin --listAgents**

**--agentInfo, --info**

Displays information about the agent corresponding to the specified *agent-id*.

Usage: **agentadmin --agentInfo** *agent-id*

Example: **agentadmin --agentInfo agent\_001**

## **--encrypt**

Encrypts a given string.

Usage: **agentadmin --encrypt *agent-instance password-file***

### ***agent-instance***

Agent instance identifier. The encryption functionality requires the use of agent instance specific encryption key present in its configuration file.

### ***password-file***

File containing a password in clear text to encrypt.

## **--getEncryptKey, --getKey**

Generates an agent encryption key of 40 characters long.

Usage: **agentadmin --getEncryptKey**

## **--key**

Generates an agent encryption key of the specified length. For security, generate keys that are about 80 characters long.

Usage: **agentadmin --key *key-length***

## **--d, -d, --decryptAgent, --decrypt**

Reveals the agent password in clear text, for the agent corresponding to the specified *agent-id*.

Usage: **agentadmin --d [*agent-id*]**

Example: **agentadmin --d Agent\_001**

### ***agent-id***

The agent instance. Default: Agent\_001 .

## **--decryptPassword**

Decrypts the agent password, for the agent corresponding to the specified *agent-id*.

Usage: **agentadmin --decryptPassword *encrypted-password encryption-key***

### ***encrypted-password***

Encrypted agent password.

### *encryption-key*

Key used to encrypt the agent password.

## **--raw-encrypt**

Encrypts the agent password without requiring the agent to be installed first.

Usage: **agentadmin --raw-encrypt --key-file** [**--password-file**] [**--out**]

### *--key-file*

Path and name of the encryption key.

To generate a key, use **agentadmin getEncryptKey** or **agentadmin getKey**.

Required: If the key isn't provided or is too short, an error occurs.

### *--password-file*

File containing a password in clear text to encrypt.

Optional: If not provided, **agentadmin** prompts for the password.

### *--out*

Path and name of the file containing the resulting encrypted password.

Optional: If not provided, the encrypted result is written to the console output.

Was this helpful?  