User Guide

This guide shows you how to install Amster, and how to integrate with ForgeRock Access Management. Read the Release Notes before you get started.

Install Amster

Install and start using Amster.

Connect to AM

Connect Amster to an AM instance.

Export AM Configuration

Export AM configuration data using Amster.

Import AM Configuration

Import partial and full AM configuration data using Amster

ForgeRock® Identity Platform serves as the basis for our simple and comprehensive Identity and Access Management solution. We help our customers deepen their relationships with their customers, and improve the productivity and connectivity of their employees and partners. For more information about ForgeRock and about the platform, see https://www.forgerock.com.

What Is Amster?

Amster is a command-line interface built upon the ForgeRock Access Management REST interface. Use Amster in DevOps processes, such as continuous integration, command-line installations, and scripted cloud deployments.

Amster provides the following features:

- **Remote, Scripted Deployments**. Script AM deployments by using the Groovy scripting support within Amster.

  For more information, see Scripting and Install AM with Amster.

- **AM Configuration Import and Export**. Amster can export all the configuration related to an AM instance, and import it back to the same, or a different instance.

  Note that Amster only manages configuration data. User information in data stores is not imported or exported, or modified in any way.

  For more information, see Export Configuration Data and Import Configuration Data.

- **Configuration Stored in JSON**. Amster exports configuration to a hierarchy of JSON format text files on the local filesystem.

  Global defaults and configuration are exported to the `global` folder, and the configuration for realms is exported into subfolders of the `realms` folder.

  The following is a simplified example of an exported hierarchy, including the top-level `root` realm:

```
|-- global
|   |-- ActiveDirectoryModule.json
|   |-- GlobalScripts
|   |   |-- 157298c0-7d31-4059-a95b-eeb08473b7e5.json
|   |   `-- 36863ffb-40ec-48b9-94b1-9a99f71cc3b5.json
|   |-- HotpModule.json
|   |-- Realms
|   |   `-- root.json
|   |-- Servers
|   |   `-- 01
|   |       |-- CtsDataStoreProperties.json
|   |       |-- SessionProperties.json
|   |       `-- 01.json
|   `-- Session.json
`-- realms
    `-- root
        |-- AmsterModule
        |   `-- amster.json
        |-- AuthenticationChains
        |   |-- amsterService.json
        |   `-- myScriptedChain.json
        |-- DataStoreModule
        |   `-- datastore.json
        |-- ScriptedModule
        |   `-- myScriptedAuthModule.json
        `-- Scripts
```

```
            |-- 9de3eb62-f131-4fac-a294-7bd170fd4acb.json
            `-- c827d2b4-3608-4693-868e-bbcf86bd87c7.json
```

Store these files in a version control system to manage and maintain AM configurations.

For a list of the available entities, see the Entity Reference.

- **Encryption of Sensitive Data**. Amster can encrypt exported passwords and sensitive data in the configuration files that are stored on disk. Only a correctly configured AM instance with the required transport key installed is able to decrypt and import the values.

  For more information, see Create Transport Keys to Export Configuration Data.

Install Amster

Prerequisites

Amster is a standalone client that does not require any other component from the ForgeRock Identity Platform to run. See the following list of prerequisites for installation:

- Amster requires a Java developer environment. Check the output of the **`java -version`** command to make sure your version is supported. For information on supported versions, see Before you install.
- The `JAVA_HOME` environment variable must be set.

Install Process

The ForgeRock BackStage website hosts downloadable versions of Amster. For each release of AM you can download Amster as a `.zip` file.

After you download the `.zip` file, create a new directory for Amster and unzip the `.zip` file. For example:

```
$ mkdir /path/to/amster_7.1.4
$ unzip ~/Downloads/Amster-7.1.4.zip -d amster_7.1.4
```

▼ File and Directory Reference

The following files and directories are extracted:

bcprov-jdk15on-1.55.jar
: Third-party cryptography library, by Bouncy Castle.

bcpkix-jdk15on-1.55.jar
: Third-party cryptography library, by Bouncy Castle.

amster
: The **amster** command.

README.md
: Amster readme file, with quick-start information.

LICENSE
: ForgeRock's Amster terms of license.

amster-7.1.4.jar
: The main Amster Java library.

/legal-notices
: Directory containing legal notices relating to the Amster distribution.

/samples
: Directory containing sample scripts for export, import, and others. For more information about these files, see Amster Sample Scripts.

First Steps

Once Amster is extracted, run the **amster** command to start the client:

```
$ cd /path/to/amster
$ ./amster

Amster OpenAM Shell (version build build, JVM: version)
Type ':help' or ':h' for help
-------------------------------------------------------------------------
-------------
am>
```

The version of Amster is included in the first line of output, as well as the version of the running JDK.

NOTE

If the **amster** command fails to load, make sure the JAVA_HOME environment variable is set, and that your JDK version is supported. For information on supported versions, see Before you install.

To exit the client, run the **:exit** or **:q** commands:

```
am> :exit
$
```

To get a list of the commands available to the client, run the **:help** command:

```
am> :help

For information about Groovy, visit:
   http://groovy-lang.org

Available commands:
   connect         (am  ) Connect to an OpenAM instance
   create          (c   ) Create an OpenAM entity
   read            (r   ) Read an OpenAM entity
   update          (u   ) Update an OpenAM entity
   delete          (d   ) Delete an OpenAM entity
   query           (q   ) Query an OpenAM entity
   action          (a   ) Perform action an OpenAM entity
   import-config   (i   ) Import configuration into OpenAM
   export-config   (x   ) Export configuration from OpenAM
   replace         (rep ) Replace all matching text
   install-openam  (inst) Install OpenAM
   :help           (:h  ) Display this help message
   ?               (:?  ) Alias to: :help
   :exit           (:x  ) Exit the shell
   :quit           (:q  ) Alias to: :exit
   :load           (:l  ) Load a file or URL into the buffer
   .               (:.  ) Alias to: :load

For help on a specific command type:
   :help command
```

To show help information available for a particular command, run **:help *command***. For example:

```
am> :help connect

usage: connect [options] <baseurl>
Options:

  -i, --interactive
        If specified you will be prompted for credentials.
Defaults to private
        key authentication.
```

```
  -k, --private-key
        Path to a private key file or directory containing one of
amster_rsa,
        id_rsa or id_ecdsa. Defaults to {USER_HOME}/.ssh.

  -t, --connection-timeout
        The default timeout is 10 seconds. If specified, this
parameter sets
        the timeout in seconds.

Connect to the OpenAM instance at the given URL.
Example:

  connect -i https://am.example.com/openam

  connect -i -t 30 https://am.example.com/openam
```

TIP

When a command does not proceed as expected, it can sometimes be helpful to start the **amster** command in debug mode and try again. To activate debug mode, start the **amster** command using the `-d` flag. For example:

```
$ ./amster -d
Listening for transport dt_socket at address: 6006
DEBUG [org.codehaus.groovy.tools.shell.BufferManager] Created new
buffer with index: 0
DEBUG [org.codehaus.groovy.tools.shell.BufferManager] Buffers reset
DEBUG [org.codehaus.groovy.tools.shell.Parser] Using parser flavor:
rigid
…
```

While in debug mode, the **amster** command output shows additional information, such as connection handshakes and Groovy calls.

Connect to AM

Amster can connect to an AM instance using interactive login or using RSA or ECDSA key files, either over HTTP or HTTPS protocols. If you use self-signed certificates for AM, you must either:

- Import the certificates into the JVM `cacerts` keystore on the Amster client.

- Run the **amster** command specifying the truststore containing the certificates and its type. For example:

```
$ ./amster \
 -D javax.net.ssl.trustStore=/path/to/keystore.jceks \
 -D javax.net.ssl.trustStoreType=jceks
```

Interactive Login Connections

To establish an interactive connection with AM, Amster makes use of the default authentication chain for administrator users configured in the AM instance. To locate this property, log in to AM and navigate to Realms > Top Level Realm > Authentication > Settings > Core.

The `ldapService` authentication chain, configured by default after AM installation, requires a valid user in AM. Log in as an administrative user, for example `amadmin`, to perform operations such as export and import of the configuration.

Connect with Interactive Login

This procedure assumes the use of the `ldapService` chain. Perform the following steps to connect to a local or remote AM instance using interactive login:

1. Start the Amster command-line interface.
2. Run the **connect** command with the `--interactive` or the `-i` options:

```
am> connect --interactive
https://openam.example.com:8443/openam
```

TIP

When using the **amster** command to import or export a significant amount of data, the default timeout of 10 seconds may be insufficient.

To increase the default timeout, add the **--connection-timeout** ***seconds*** option. For example:

```
am> connect --connection-timeout 45 \
 --interactive https://openam.example.com:8443/openam
```

3. Specify username and password to authenticate to AM:

```
Sign in to OpenAM
User Name: amadmin
Password: *************
amster openam.example.com:8443>
```

Private Key Connections

Amster can connect to an AM instance by using a private key pair and an authentication module and a chain in AM. The private key must be available to the Amster client, and the AM instance must trust the client IP address and have the public key in its `authorized_keys` file. Successful connections create an `amAdmin` session in AM.

An installation, or an upgrade of AM creates the following infrastructure for Amster:

- The `Forgerock Amster` authentication module in the Top Level Realm. The module is enabled by default in new AM installations and disabled by default when upgrading an existing AM.

  For information on how to configure this module, see Amster Authentication Module Properties in the *Authentication and Single Sign-On Guide*.
- The `amsterService` authentication chain in the Top Level Realm. Changing or removing this chain may result into not being able to connect with Amster.
- The following RSA key pair files, in PKCS#1 PEM format:

  Default Private Keypair Files

File Name	Description
`/path/to/openam/security/keys/amster/authorized_keys`	Holds the public keys of trusted Amster clients. AM check incoming Amster connections against these trusted keys. By default, contains a copy of the public key of a generated key pair that Amster can use. If this file exists in the configuration directory before a new installation is performed, the file is not overwritten; the contents of the newly-created `amster_rsa.pub` file are appended to it instead.
`/path/to/openam/security/keys/amster/amster_rsa`	Contains the private key of a generated key pair that Amster can use.

File Name	Description
`/path/to/openam/security/keys/amster/amster_rsa.pub`	Contains the public key of a generated key pair that Amster can use.

Connecting Locally with Default Private Key Files

An Amster installation local to a new AM instance can connect without further configuration.

Connect Locally With the Default Key Pair

Perform the following steps to connect to a local AM instance using the default key pair:

1. Start the Amster command-line interface.
2. Run the **`connect`** command with the `--private-key`, or the `-k` options:

```
am> connect --private-key
/path/to/openam/security/keys/amster/amster_rsa \
 https://openam.example.com:8443/openam
amster openam.example.com:8443>
```

Connecting to a Remote AM Instance

To connect to a remote AM instance, create a private key pair for Amster, and append the contents of the public key to the `authorized_keys` file of the instance.

Create and Configure a Private Key Pair

To create a new key pair and append the public key to the AM instance perform the following steps:

1. Login to the Amster server.
2. Create a directory for the keys, for example, `$HOME/.ssh`.
3. Run the **`ssh-keygen`** command to generate a key pair without passphrase. You can create RSA or ECDSA key pairs:
   - To create an RSA key pair, run the **`ssh-keygen`** command with the `-t rsa` option:

```
$ ssh-keygen -t rsa -N "" -f $HOME/.ssh/id_rsa -b 2048
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:
78:ca:43:bc:0a:84:b0:ab:ac:40:96:49:48:84:80:63
root@amster_server
```

- To create a ECDSA keypair, run the **ssh-keygen** command with the `-t ecdsa` option. You can create key pairs of 256, 384, or 521 curve sizes.

  For example:

```
$ ssh-keygen -t ecdsa -N "" -f $HOME/.ssh/id_ecdsa -b 521
Generating public/private ecdsa key pair.
Your identification has been saved in id_ecdsa.
Your public key has been saved in id_ecdsa.pub.
The key fingerprint is:
6b:b9:75:cb:42:07:91:25:a7:bf:d6:d0:bc:6f:5a:d7
root@amster_server
```

  NOTE

  AM requires the private key to be in PKCS#1 PEM format. Recent versions of the OpenSSH **ssh-keygen** tool creates keys in its own format, which AM cannot process.

  If your generated private key (`id_ras`) begins with `-----BEGIN OPENSSH PRIVATE KEY-----` , you will need to recreate your keypair in PKCS#1 PEM format.

  Append the `-m pem` option to the **ssh-keygen** commands above to create a new pair in the supported PKCS#1 PEM format.

  For example:

```
$ ssh-keygen -m pem -t rsa -N "" -f $HOME/.ssh/id_rsa -b 2048
```

The commands generate two files, `id_rsa.pub` or `id_ecdsa.pub` containing the public key, and `id_rsa` or `id_ecdsa` containing the private key.

4. Append the contents of the `id_rsa.pub` or `id_ecdsa.pub` files into the `authorized_keys` file in your AM instance(s); for example, into `/path/to/openam/security/keys/amster/authorized_keys` .

5. Start the Amster command-line interface.
6. To connect to AM using a specific private key file, run the **`connect`** command with the `--private-key`, or the `-k` options, specifying the path to the private key file. For example:

```
am> connect --private-key $HOME/.ssh/id_rsa \
 https://openam.example.com:8443/openam
amster openam.example.com:8443>
```

Create Transport Keys to Export Configuration Data

To import and export encrypted password values in the configuration files you must generate a *transport key*, and install it in the keystore of each AM instance that you will be transporting passwords between.

The transport key must be stored in the default AM keystore, located at `/path/to/openam/security/keystores/keystore.jceks`, and should have a key alias of `sms.transport.key`.

The presence of the transport key causes additional fields containing encrypted password values to appear in the exported configuration files. These additional fields have a `-encrypted` suffix, as shown below:

```
{"authenticatorPushDeviceSettingsEncryptionKeystorePassword":null
,
 "authenticatorPushDeviceSettingsEncryptionKeystorePassword-
encrypted":"encrypted-pwd"}
```

Encrypted password fields will only be added to REST calls made by administrative users, such as `amAdmin`.

Performance of an AM instance with a transport key present will be significantly impacted. You **MUST** delete the transport key when imports and exports have been completed.

Without a transport key present, all password fields are set to `null` in the exported configuration files, regardless of their actual value in the configuration.

Generate and Install a New Transport Key

Use the **keytool** command to generate the transport key by performing the following steps:

1. Run the **keytool** command, specifying the location of the .storepass file as the password to use for the keystore, and the location of the .keypass file as the password to use for the key aliases:

```
$ keytool -genseckey -alias "sms.transport.key" -keyalg
AES -keysize 128 \
  -storetype jceks -keystore
"/path/to/openam/security/keystores/keystore.jceks" \
  -storepass:file
"/path/to/openam/security/secrets/default/.storepass" \
  -keypass:file
"/path/to/openam/security/secrets/default/.keypass"
```

2. You must restart AM for the transport key change to take effect.

   The instance will now be able to include encrypted passwords in the exported configuration files.

To decrypt and import configuration files that contain encrypted passwords, you must install the same transport key used to encrypt it into the keystore of the target AM instance.

Duplicate and Install a Transport Key

Use the **keytool** command to export the transport key from the source instance keystore, and then install the result on the target instance keystore, by performing the following steps:

1. On the source instance, export the transport key into a keystore that can be transported to another instance by executing the following **keytool** command:

```
$ keytool -importkeystore -srcstoretype jceks -srcalias
"sms.transport.key" \
  -deststoretype jceks -destalias "sms.transport.key" \
  -srckeystore
"/path/to/openam/security/keystores/keystore.jceks" \
  -destkeystore
"/path/to/openam/security/keystores/transport_keystore.jce
ks" \
  -srckeypass:file
```

```
"/path/to/openam/security/secrets/default/.keypass" \
  -srcstorepass:file
"/path/to/openam/security/secrets/default/.storepass" \
  -destkeypass "myTransp0rtK3yP4ssword" \
  -deststorepass "myTransp0rtK3yP4ssword"
```

This command exports the transport key to a temporary keystore file `/path/to/openam/security/keystores/transport_keystore.jceks`, and set a store and key password of `myTransp0rtK3yP4ssword`. You need to use these temporary passwords when importing to the target instance.

2. Move the keystore file created in the previous step, in this example `transport_keystore.jceks`, to the filesystem of the target server.
3. On the target server, import the transport key into the AM keystore by executing the following **`keytool`** command:

```
$ keytool -importkeystore -srcstoretype jceks -srcalias
"sms.transport.key" \
  -deststoretype jceks -destalias "sms.transport.key" \
  -srckeystore
"/path/to/openam/security/keystores/transport_keystore.jce
ks" \
  -destkeystore
"/path/to/openam/security/keystores/keystore.jceks" \
  -srckeypass "myTransp0rtK3yP4ssword" \
  -srcstorepass "myTransp0rtK3yP4ssword" \
  -destkeypass:file
"/path/to/openam/security/secrets/default/.keypass" \
  -deststorepass:file
"/path/to/openam/security/secrets/default/.storepass"
```

This command imports the transport key from the temporary keystore file `/path/to/openam/security/keystores/transport_keystore.jceks` into the AM keystore, and set the transport key password to match the password used by the target keystore.

4. You must restart the target AM instance for the transport key change to take effect.

The target instance will now be able to correctly decrypt passwords stored in the imported configuration files.

IMPORTANT

IMPORTANT

The presence of the transport key includes encrypted passwords in requests made by an administrative user, causing significant performance degradation. You **MUST** delete the transport key when imports and exports have been completed.

Delete a Transport Key

1. Run the following **`keytool`** command:

```
$ keytool -delete -alias "sms.transport.key" -storetype
jceks \
  -storepass:file
"/path/to/openam/security/secrets/default/.storepass" \
  -keystore
"/path/to/openam/security/keystores/keystore.jceks"
```

2. You must restart the target AM instance for the transport key change to take effect.

   The target instance will no longer include encrypted passwords, nor be able to correctly decrypt passwords stored in configuration files.

Export Configuration Data

Amster can export configuration data from an AM instance. Export configuration data by using the **`export-config`** command.

The exported configuration data is written to a number of JSON-formatted files. The files are arranged in a hierarchy of global and realm configuration data.

To export encrypted password values in the configuration files you must generate and install a *transport key*. See Create Transport Keys to Export Configuration Data.

Usage:

```
am> export-config --path Path [options]
```

--path Path
The path into which exported configuration files are placed.

Existing files will be overwritten if they exist. The path is created if it does not exist.

Options:

--realms Realm […]
Space-separated list of realms from which to export. Specify the full path of each realm to export. Use a single forward-slash (`/`) to represent the top-level root realm.

Example: `/ /subRealm/subSubRealm`

Default: all

--realmEntities Entity […]
Space-separated list of realm-based entities to export.

Use a space character in single-quotes (' ') to specify that no realm-based entities should be exported.

For a list of the available entities, see the Entity Reference.

Default: all

--globalEntities Entity […]
Space-separated list of global entities to export.

Use a space character in single-quotes (' ') to specify that no global entities should be exported.

For a list of the available entities, see the Entity Reference.

Default: all

--failOnError [true|false]
If specified, the export process halts if an error occurs.

Default: `false`

--listPasswords [true|false]
If specified, the export process creates a listing of entities that contain password data. The listing is stored in a file in the root of the specified export directory.

Default: `false`

Examples

Before trying the following examples, start the Amster command-line interface, and connect to the AM instance from which to export data.

For information on connecting to AM instances, see Connect to AM.

Export Entire Configuration

This example exports all configuration data, and will fail immediately if an error occurs.

```
am> export-config --path /tmp/myExportedConfigFiles --failOnError
true
Export completed successfully
```

Export Selected Configuration

This example exports the configuration for the `DataStoreModule`, `Scripts`, and `OAuth2Provider` entities in a subrealm of the root realm named `mySubRealm`.

Configuration data for global entities is not exported.

```
am> export-config --path /tmp/myExportedConfigFiles --realms
'/mySubRealm' \
  --realmEntities 'DataStoreModule Scripts OAuth2Provider' --
globalEntities ' '
Export completed successfully
```

Import Configuration Data

Amster can import configuration data to an AM instance. Import configuration data by using the **`import-config`** command.

CAUTION

A successful import overwrites any configuration that already exists in the target AM instance.

Before importing configuration data to an AM instance, consider the following points:

- You must connect to the AM instance where you will import the configuration data after starting the Amster command-line interface. For information, see Connect to AM.
- You must ensure that the configuration data you are trying to import is compatible with the version of AM you have deployed.

  For example, do not try to import configuration data exported from an AM 5 instance into an AM 6.5 instance.
- AM validates that external data stores are configured and available when creating connections to them, including when importing using Amster.

You must ensure that each external datastore configured in the source instance has an equivalent datastore available and running before importing Amster configuration into the destination AM instance.

- When importing a full set of configuration data from an instance of AM, specify the `--clean` option to remove configuration settings from the target instance.

  The `--clean` option removes the following items from the target AM instance:

  - Realms, other than the Top Level Realm.
  - Authentication chains and modules.
  - Server and site settings, other than the current server.
  - Services.
  - Secret ID Mappings and secret definitions.
  - Scripts.
  - Audit settings.
  - Policies, policy sets and resource types.
  - Identity store configuration.
  - Agents, and agent groups.

  IMPORTANT

  Do not use the `--clean` option if you are only importing a partial Amster export.

  The target AM instance may not have the settings required to start up and operate if you do not replace the deleted settings by importing a complete set of configuration.

- By default, Amster configures the value of the `com.iplanet.am.lbcookie.value` property as the value of the server ID.

  To override the default during import, prefix the new value with `override-server-id:` in the configuration files. For example:

  ```
  "com.iplanet.am.lbcookie.value" : "override-server-
  id:myLBCookieValue"
  ```

- To import encrypted password values in the configuration files you must install the *transport key* used to encrypt the data. For more information, see Create Transport Keys to Export Configuration Data.
- You must ensure that any special characters in names and passwords in Amster shell variables are escaped as required by the Groovy language.

For example, the dollar `$` character is a special character in Groovy. The following are two possible ways of escaping the `$` character:

```
variable.name="/pa$$word/"
variable.name='pa\\$\\$word'
```

Note that you cannot use variables, such as `${varname}`, or configuration expressions, such as `&{varname}` if you convert a double-quoted string into a single-quoted string.

Refer to the Groovy documentation for more information on escaping special characters in strings.

Usage:

```
am> import-config --path Path [options]
```

--path Path

The path containing configuration files to import.

Specify a directory to import from all correctly-formatted JSON files within that directory and recurse through each sub-directory, or specify an individual JSON file.

Options:

--failOnError [true|false]

If specified, the import process halts if an error occurs.

Default: `false`

--clean [true|false]

If specified, all configuration data is removed from the target AM instance before the import is performed.

Only set this option to `true` when importing a **full** set of configuration files into a new AM instance. Otherwise, the target instance may not function correctly.

Default: `false`

Example

Before trying the following examples, start the Amster command-line interface, and connect to the AM instance where you will import the configuration data.

For information on connecting to AM instances, see Connect to AM.

This example cleans all configuration from the target AM instance before importing a full set of configuration data, but will not halt the import if an error occurs.

```
am> import-config --path /tmp/myExportedConfigFiles --clean true
--failOnError false
Cleaning global settings
Deleting JSON: Global JSON Handler
Deleting Scripting: 9de3eb62-f131-4fac-a294-7bd170fd4acb
Deleting Scripting: 7e3d7067-d50f-4674-8c76-a3e13a810c33
Deleting Scripting: c827d2b4-3608-4693-868e-bbcf86bd87c7
Global settings cleaned
Importing directory /tmp/myExportedConfigFiles
…
Import completed successfully
```

Install AM with Amster

Amster can configure a deployed AM as a single, stand-alone instance, or as an instance that is part of a site.

Amster can configure AM to use an embedded DS server as the configuration and identity store, for evaluation purposes.

For production environments, you can specify an external configuration store. Configuring AM to use an external configuration store also requires an external identity store, which will use the specified configuration store by default, unless otherwise specified.

Install AM configuration with Amster by using the **`install-openam`** command:

Usage:

```
am> install-openam \
 --serverUrl protocol://FQDN:port/URI \
 --adminPwd amAdmin_password \
   [options]
```

`--adminPwd amAdmin-password`

Specifies the password of the `amAdmin` user. If the `--cfgStoreDirMgrPwd` option is not specified, this value is also the password of the configuration store's directory manager user.

The password must be at least 8 characters in length.

--serverUrl protocol://FQDN:port/URI

Specifies the protocol, URL, port, and deployment URI of the AM instance. For example, `https://openam.example.com:8443/openam`.

Options:

[options]

Specifies optional parameters to configure properties such as the cookie domain, ports and passwords for the configuration store, and others.

For more information about the possible options, run the **`:help install-openam`** command, or see install-openam - Install Access Management.

Examples

Before trying the following examples, make sure the AM instance is deployed and running but not yet configured. For more information, see the ForgeRock Access Management Installation Guide.

For more information about options available to the **`install-openam`** command, see install-openam - Install Access Management.

TIP

Amster also supports scripting the installation process. For more information, see Scripting.

Installing AM Instances for Evaluation (Embedded Configuration Store)

The following examples show you how to install AM for evaluation, demo, or test purposes. The embedded configuration store is not supported for production environments, and instances configured with it cannot be part of a site.

Example 1

This example installs a single AM instance with the default values:

```
am> install-openam \
 --serverUrl https://openam.example.com:8443/openam \
 --adminPwd forgerock \
 --acceptLicense
timestamp: Checking license acceptance…
timestamp: License terms accepted.
timestamp: Checking configuration directory /tomcat/openam.
timestamp: …Success.
timestamp: Extracting OpenDJ, please wait…
timestamp: Complete
```

```
timestamp: Running OpenDJ setup
timestamp: Setup command: --cli --adminConnectorPort 4444
--baseDN dc=openam,dc=forgerock,dc=org --rootUserDN uid=admin
--ldapPort 50389 --skipPortCheck --rootUserPassword xxxxxxx --
jmxPort 1689
--no-prompt --doNotStart --hostname openam.example.com --
noPropertiesFile
--backendType je
%0AConfiguring+Directory+Server+…..+Done.
0A%0ATo+see+basic+server+configuration+status+and+configuration%2
C+you+can+launch%
0A%2Ftomcat_b%2Fopenam%2Fopends%2Fbin%2Fstatus%0A%0Atimestamp: …
Success.
timestamp: …Success
timestamp: Installing OpenAM configuration store in
/tomcat/openam/opends
…
timestamp: Configuring server instance.
timestamp: …Done
timestamp: Creating demo user.
timestamp: …Done
timestamp: Setting up monitoring authentication file.
Configuration complete!
```

Notes:

- If only the required parameters are supplied, Amster installs AM like the web configurator does when using the defaults.
- This example installs a single AM instance with an embedded configuration store.
- When installing AM locally to Amster, Amster stores AM's configuration in the home directory of the user that is running the **`amster`** command. For example, for the `tomcat` user, the configuration is stored in `/path/to/tomcat_home/openam`.

  To modify this behavior, use the `--cfgDir` option.
- If the default ports for the configuration store are already in use, the installer uses the next available free ports.
- The `demo` user is created in the embedded store.

Example 2

This example installs a single AM instance and specifies the configuration directory:

```
am> install-openam \
 --serverUrl https://openam.example.com:8443/openam \
```

```
 --adminPwd forgerock \
 --acceptLicense \
 --cfgDir /tomcat/openam2
timestamp: Checking license acceptance…
timestamp: License terms accepted.
timestamp: Checking configuration directory /tomcat/openam2.
timestamp: …Success.
timestamp: Extracting OpenDJ, please wait…
timestamp: Complete
timestamp: Running OpenDJ setup
timestamp: Setup command: --cli --adminConnectorPort 4444
--baseDN dc=openam,dc=forgerock,dc=org --rootUserDN uid=admin
--ldapPort 389 --skipPortCheck --rootUserPassword xxxxxxx --
jmxPort 1689
--no-prompt --doNotStart --hostname openam.example.com --
noPropertiesFile
--backendType je
%0AConfiguring+Directory+Server+…..+Done.
…
timestamp: …Done
timestamp: Setting up monitoring authentication file.
Configuration complete!
```

Notes:

- This example installs a single AM instance with an embedded configuration store.
- Amster will create the directory specified in the `--cfgDir` option.
- The `demo` user is created in the embedded store.

Installing AM Instances (External Configuration Store)

Installing AM with an external configuration store requires manual configuration of the directory server. This is also true when specifying a separate identity store.

Note that you cannot install AM with an external configuration store that already contains configuration data, unless you are adding an instance to the existing site.

Example 1

This example installs AM with an external configuration store, which is also used as the identity store.

Before running the **`amster`** command, create a truststore for AM, and prepare the external configuration store, as detailed in the ForgeRock Access Management Installation Guide.

```
am> install-openam \
 --serverUrl https://openam.example.com:8443/openam \
 --adminPwd forgerock \
 --acceptLicense \
 --cfgStoreDirMgrPwd mypassword \
 --cfgStore dirServer \
 --cfgStoreHost opendj.example.com \
 --cfgStoreAdminPort 4444 \
 --cfgStorePort 1636 \
 --cfgStoreRootSuffix dc=example,dc=com
 --cfgStoreSsl SSL
timestamp: Checking license acceptance…
timestamp: License terms accepted.
timestamp: Checking configuration directory
/Users/forgerock/openam.
timestamp: …Success.
timestamp: Tag swapping schema files.
timestamp: …Success.
timestamp: Loading Schema odsee_config_schema.ldif
timestamp: …Success.
…
timestamp: Loading Schema
/Users/forgerock/openam/opendj_pushdevices.ldif
timestamp: …Success.
timestamp: Installing new plugins…
timestamp: Plugin installation complete.
timestamp: Setting up monitoring authentication file.
Configuration complete!
```

Notes:

This example installs the configuration store, and the identity store in the `opendj.example.com` host. Both the configuration and identity data use the same DS instance.

- The DS instance in the example requires secure connections; therefore, the **`amster`** command specifies the `1636` port for LDAPS, and `SSL`.

  For different options, see install-openam - Install Access Management.

  Failure to create a truststore and configuring it in the container where AM runs will cause the install process to fail.

- When installing AM locally to Amster, Amster stores AM's configuration in the home directory of the user that is running the **`amster`** command. For example, for the `tomcat` user, the configuration is stored in `/path/to/tomcat_home/openam`.

To modify this behavior, use the `--cfgDir` option.

- If there is any problem setting up the configuration store, the installation process will exit with an error, and navigating to AM will open the configuration page.
- The `demo` user is not created in the identity store.

Example 2

This example installs AM with external configuration and identity stores.

Before running the **`amster`** command, create a truststore for AM, and prepare the external configuration and identity stores, as detailed in the ForgeRock Access Management Installation Guide.

```
am> install-openam \
 --serverUrl https://openam.example.com:8443/openam \
 --adminPwd forgerock \
 --acceptLicense \
 --cfgStoreDirMgrPwd mypassword \
 --cfgStore dirServer \
 --cfgStoreHost opendj.example.com \
 --cfgStoreAdminPort 4444 \
 --cfgStorePort 1636 \
 --cfgStoreSsl SSL \
 --cfgStoreRootSuffix dc=example,dc=com \
 --userStoreDirMgrPwd mypassword2 \
 --userStoreHost ldap.example.com \
 --userStoreType LDAPv3ForOpenDS \
 --userStorePort 1636 \
 --userStoreSsl SSL \
 --userStoreRootSuffix dc=example,dc=com
timestamp: Checking license acceptance…
timestamp: License terms accepted.
timestamp: Checking configuration directory
/Users/forgerock/openam.
timestamp: …Success.
timestamp: Tag swapping schema files.
timestamp: …Success.
timestamp: Loading Schema odsee_config_schema.ldif
timestamp: …Success.
…
timestamp: Loading Schema
/Users/forgerock/openam/opendj_pushdevices.ldif
timestamp: …Success.
timestamp: Installing new plugins…
timestamp: Plugin installation complete.
```

```
timestamp: Setting up monitoring authentication file.
Configuration complete!
```

Notes:

- When installing AM locally to Amster, Amster stores AM's configuration in the home directory of the user that is running the **`amster`** command. For example, for the `tomcat` user, the configuration is stored in `/path/to/tomcat_home/openam`.

  To modify this behavior, use the `--cfgDir` option.

- The DS instances in the example require secure connections; therefore, the **`amster`** command specifies the `1636` port for both stores, and `SSL`.

  For different options, see install-openam - Install Access Management.

  Failure to create a truststore and configuring it in the container where AM runs will cause the install process to fail.

- If there is any problem setting up the configuration store, the installation process will exit with an error, and navigating to the AM will open the configuration page.
- The `demo` user is not created in the identity store.

Example 3

This example installs two AM instances within a site that use an external configuration store, which is also used as the identity store.

Before running the **`amster`** command, create a truststore for each AM instance, and prepare the external configuration store, as detailed in the ForgeRock Access Management Installation Guide. Both instances will share the configuration and identity store and, therefore, their truststores should contain the same certificates.

First instance:

```
am> install-openam \
 --serverUrl https://openam1.example.com:8443/openam \
 --adminPwd forgerock \
 --acceptLicense \
 --cookieDomain example.com \
 --lbSiteName TestSite01 \
 --cfgDir /tomcat/openam1 \
 --lbPrimaryUrl http://site.example.com:80/openam \
 --cfgStore dirServer \
 --cfgStoreHost opendj.example.com \
 --cfgStoreAdminPort 3444 \
 --cfgStorePort 1636 \
 --cfgStoreSsl SSL \
```

```
 --cfgStoreRootSuffix dc=examplecfg1,dc=com \
 --cfgStoreDirMgr uid=admin \
 --cfgStoreDirMgrPwd mySecretPassword
timestamp: Checking license acceptance…
timestamp: License terms accepted.
timestamp: Checking configuration directory /tomcat/openam1.
timestamp: …Success.
…
timestamp: …Success.
timestamp: Loading Schema odsee_config_schema.ldif
timestamp: …Success.
…
timestamp: …Success.
timestamp: Installing new plugins…
timestamp: Plugin installation complete.
timestamp: Setting up monitoring authentication file.
Configuration complete!
```

Notes:

- Amster will create the directory specified in the `--cfgDir` option.
- Since an identity store is not specified, Amster reuses the configuration store for identities.
- The DS instance in the example requires secure connections; therefore, the **`amster`** command specifies the `1636` port to connect to the store, and `SSL` .

  For different options, see install-openam - Install Access Management.

  Failure to create a truststore and configuring it in the container where AM runs will cause the install process to fail.
- Amster will create a site with the name specified in the `--lbSiteName` option, which can be accessed using the URL specified in the `--lbPrimaryUrl` option.
- The cookie domain is specified in the `--cookieDomain` . If not specified, Amster sets the cookie domain to the URL of the AM instance, which is not optimal when having multiple instances in a site.

Second instance:

```
am> install-openam \
 --serverUrl https://openam2.example.com:8443/openam \
 --adminPwd forgerock \
 --acceptLicense \
 --cookieDomain example.com \
 --lbSiteName TestSite01 \
```

```
 --cfgDir /tomcat/openam2 \
 --lbPrimaryUrl http://site.example.com:80/openam \
 --cfgStore dirServer\
 --cfgStoreHost opendj.example.com \
 --cfgStoreAdminPort 3444 \
 --cfgStorePort 1636 \
 --cfgStoreSsl SSL \
 --cfgStoreRootSuffix dc=examplecfg1,dc=com \
 --cfgStoreDirMgr uid=admin \
 --cfgStoreDirMgrPwd mySecretPassword \
 --pwdEncKey MneLwkkOokJx58znp7QyvGmiawmc2vl4
timestamp: Checking license acceptance…
timestamp: License terms accepted.
timestamp: Checking configuration directory /tomcat/openam2.
timestamp: …Success.
timestamp:Reinitializing system properties.
timestamp:…Done
timestamp:Configuring server instance.
timestamp: …Done
timestamp: Installing new plugins…
timestamp: Plugin installation complete.
timestamp: Setting up monitoring authentication file.
Configuration complete!
```

IMPORTANT

To complete the installation of the second instance, follow the steps in Post-Installation Steps for Site Deployments.

Notes:

- This example installs an AM instance with as part of the `TestSite01` site. Note that the configuration store details are the same as those used for the first server, since they are sharing the same DS instance.
- The password specified in the `--adminPwd` option must be the same password used across the site.
- Since an identity store is not specified, Amster reuses the configuration store for identities.
- Amster will create the directory specified in the `--cfgDir` option.
- The DS instance in the example requires secure connections; therefore, the **`amster`** command specifies the `1636` port to connect to the store, and `SSL` .

  For different options, see install-openam - Install Access Management.

Failure to create a truststore and configuring it in the container where AM runs will cause the install process to fail.

- The cookie domain is specified in the `--cookieDomain` option. The cookie domain must be the same as the one used when installing the first instance, in this case, `example.com`. When unspecified, Amster sets the cookie domain to the URL of the AM instance, which is not optimal when having multiple instances in a site.

  Failure to set this option correctly may result in login failure to the new instance.

- The `--pwdEncKey` specifies the encryption key used by the servers already in the site. To locate the encryption key value, navigate to Deployment > Servers > Server Name > Security > Encryption.

  Failure to set this option to the appropriate value will cause the original encryption key to be overwritten, which will render the site unable to read the configuration and identity stores.

Post-Installation Steps for Site Deployments

Keystore and secret store infrastructure is shared across all the AM instances in the site. This is so that every instance in the site can encrypt, decrypt, and verify messages, JWTs, and others, with the same keys.

The install process creates the required keystores and secret stores on the first instance in the site only. You must configure the keystore and secret store infrastructure in the rest of the instances manually.

> The following steps assume you have an AM site composed of one or more instances, and that you installed a new instance and added it to the site using Amster:
>
> 1. Make the site keystore infrastructure available to the new instance:
>   - Back up the new instance's default keystore and password files in the following locations:
>   - `/path/to/openam/security/keystores/`
>   - `/path/to/openam/security/secrets/default/`
>   - Ensure that the existing keystores in the site are available in the same location to the new instance. This may mean copying the keystores and their password files, mounting a volume, or others.
>   - Ensure that the keystore files configured in the `/path/to/openam/config/boot.json` file are available to the new instance.
> 2. Make the secret store infrastructure in the site available to the new instance:

- Log in to the AM console of an existing instance in the site and navigate to Configure > Secret Stores.
- Review the list of secret stores configured globally, and make sure to provide the relevant stores to the new instance. For example:
  - For keystore-type secret stores, copy the keystores to the same path on the new instance.
  - For filesystem-type secret stores, copy the contents of the directories to the same path, or make the filesystem available on the same mount point on the new instance.
  - For HSM-type stores, ensure the new instance can access it.
  - For secrets configured as environment variables accessible by the container where AM runs, ensure they are also accessible by the container of the new instance.
- Navigate to Realms > *Realm Name* > Secret Stores.
- Review the list of secret stores configured per realm, and make sure to provide the relevant stores to the new instance.

3. Restart the new instance.

   The instance is now configured for the site.

Troubleshoot AM Installations

The following table describes possible errors when installing AM with the `install-openam` command:

Error	Solution
`Invalid Suffix for directory server ds.example.com:1389. No Base Entity dc=incorrectsuffix,dc=com found.`	Review that the suffix you are trying to use exists in DS.
`Cannot connect to Directory Server. Invalid Credentials.`	Review the credentials to connect to DS.
`Cannot connect to Directory Server. Connect Error: Connection refused.`	Review DS's host and/or connection port.
`Unexpected LDAP exception occurred.`	Review DS logs. DS may be stopped, or may have become unreachable during install.

Error	Solution
`Cannot connect to Directory Server. Connect Error: The LDAP connection has failed because an error occurred during the SSL handshake…`	Review that the container where AM will be installed trusts DS's SSL certificates.

Integrate Amster in your Environment

Amster allows you to export the configuration of an AM instance, and customize it ready for import into any other AM instance in your environment. For example, you can export the configuration from the development environment, customize the passwords and keystore values, and import it in your QA or integration environment for testing.

To make this process easier, Amster allows you to configure variables inside the exported configuration files, and also, to script a series of commands and Amster tasks, so they can be easily included in your processes or pipelines.

Use Configuration Expressions

Learn Amster configuration expressions.

Script Amster Commands

Execute scripts with Amster.

Using Configuration Expressions in Exported Configuration Files

Amster supports the use of configuration expressions as the values of configuration properties in the exported configuration files. Amster substitutes the expressions with values obtained from the Amster shell, expression files, environment variables, and others, when importing the configuration files into an AM instance. Property value substitution enables you to achieve the following:

- Define a configuration that is specific to a single instance.

  For example, setting the location of the keystore on a particular host.

- Define a configuration whose parameters vary between different environments.

  For example, the URLs and passwords for test, development, and production environments.

- Disable certain capabilities on specific AM instances.

  For example, you might want to disable a particular instance from sending notifications to agents.

Property value substitution uses expressions to introduce variables into the server configuration. You set expressions as the values of configuration properties. The effective property values can be evaluated in a number of ways.

Working With Expressions

Expressions share their syntax and underlying implementation with other ForgeRock Identity Platform components. Expressions have the following characteristics:

- To distinguish them from static values, configuration expressions are preceded by an ampersand (`&`) and enclosed in braces (`{}`). Use the dot (`.`) character as a separator character for the *expression token*. For example, `&{smtp.port}` .
- You can use a default value in a configuration expression by including it after a vertical bar (`|`) character following the token.

  For example, the following SMTP port expression sets the default value of the SMTP port to 1349: `&{smtp.port|1349}` .
- A configuration property can include a mix of static values and expressions.

  For example, suppose `hostname` is set to `openam` . Then `&{hostname}.example.com` evaluates to `openam.example.com` .

  You can also use expressions in conjunction with Unix environment variables once these are made available to the Amster shell. For more information, see Scripting.
- You can define *nested* properties (that is, a property definition within another property definition).

  For example, suppose `listen.port` is set to `&{port.prefix}389` , and `port.prefix` is set to `2` . Then `&{listen.port}` evaluates to `2389` .
- To use an expression in the key of a key:value pair, use the special `AMSTER` marker, rather than the dollar sign (`$`).

  For example, to update the fully-qualified domain name mappings, you could use syntax such as the following:

  `"com.sun.identity.server.fqdnMap[AMSTER`*`{realm.dns.alias}`*`]" : "`*`${realm.dns.alias}`*`"`

Amster defines the following expressions by default:

&{amster.import.dir}
: This expression is resolved in the following ways:

  - As the directory containing the configuration files being imported into AM when using the **`import-config --path`** ***`directory`*** command.
  - As the parent directory of the configuration file being imported into AM when using the **`import-config --path`** ***`file`*** command.

&{amster.import.url}
: This expression is resolved in the same way as `&{amster.import.dir}`, but in URL format. For example, `file://path/to/directory`.

Expression Evaluation and Order of Precedence

You must define expression values before importing the configuration into AM. When evaluated, an expression must return the appropriate type for the configuration property. For example, the `smtp.port` property takes an integer. If you set the property using an expression, the result of the evaluated expression must be an integer.

If the type is wrong, AM may fail to start after a configuration import, with unexpected errors. For more information about data type coercion, see Transforming Data Types.

Amster can obtain expressions from the following sources.

Environment variables

You set an environment variable in your operating system shell. For example, `export SMTP_PORT=1342`.

The environment variable name must be composed of uppercase characters and underscores. The name maps to the expression token as follows:

- Uppercase characters are converted into lower case characters.
- Underscores (`_`) are replaced with dot (`.`) characters.

In other words, the value of `SMTP_PORT` replaces `&{smtp.port}` in the AM configuration files.

Java system properties

You set a Java system property to hold the value when you call the **`amster`** command with parameters. For example:

```
"-Dsmtp.port=3306"
```

Amster shell variables

You set Amster shell variables to hold values.

To define expressions as Amster shell variables, remove the dot (`.`) character, and use the standard camel case notation for naming variables in Groovy.

For example, the `&{smtp.port}` expression would be defined as:

```
am> smtpPort = "1342"
===> 1342
```

In the configuration file, however, you still define the expression as `&{smtp.port}`.

TIP

For more information about how to convert Java system properties and environment variables into Amster shell variables, see Using Variables in Amster Scripts.

Expression files

You set a key in a `.json` or `.properties` file to hold the value.

Keys in `.properties` files must match expression tokens exactly. In other words, the value of the `smtp.port` key replaces `&{smtp.port}` in the server configuration.

The following is an example properties expression file:

```
smtp.port=1342
smtp.user=Greg
stateless.tokens.enabled=true
```

JSON expression files can contain nested objects.

JSON field names map to expression tokens as follows:

- The JSON path name matches the expression token.
- The `.` character serves as the JSON path separator character.

The following is an example of a JSON expression file:

```
{
   "smtp" : {
      "port" : 1342,
      "user" : "Greg"
```

```
    },
    "stateless" : {
       "tokens" : {
          "enabled" : "true"
       }
    },
    "blacklist" : {
       "java" : {
          "classes" : [
             "java.lang.Class",
             "java.security.AccessController",
             "java.lang.reflect.*"
          ]
       }
    }
}
```

You can set multiple configuration files to store your properties. For example, you could have a file to store authentication tree values and another file to store policy-related values.

Note the following constraints when using expression files:

- Amster scans the files in the provided directory in a non-deterministic order.
- Amster reads all files with `.json` and `.properties` extensions.
- Amster does not have a predictable order of precedence for handling multiple configuration tokens with the same name. You are responsible for ensuring name uniqueness of configuration tokens across multiple expression files.

To provide expression files to Amster, use the **`envconfig`** command followed by the full path to a directory or a file. For example:

```
am> envconfig /path/to/expressionfiles/
```

Expression evaluation

Expressions are evaluated and replaced with the expected values when importing a configuration into an AM instance. Attempting to import AM JSON configuration files containing expressions that are not defined causes an error message similar to the following:

```
amster openam.example.com:8443> import-config \
 --path /tmp/myExportedConfigFiles/realms/root/EmailService.json
Importing file
```

```
/tmp/myExportedConfigFiles/realms/root/EmailService.json
-----------------------------------------------------------------
----
IMPORT ERRORS
-----------------------------------------------------------------
----
Failed to import
/tmp/myExportedConfigFiles/realms/root/EmailService.json  :
Can't substitute source vale: unresolved variables
([email.service.port]) or cycles detected ([])
```

The following list reflects the order of precedence:

- Environment variables override default expressions, Amster variables, Java system properties and settings in expression files.
- Amster variables override Java system properties, tokens found in expression files, and default expressions.
- Java system properties override tokens found in expression files and the default expression tokens.

Transforming Data Types

By default, when configuration tokens are resolved, the result is always a string. However, when evaluated, an expression must return the appropriate type for the configuration property. For example, the `smtp.port` property takes an integer. If you set the property using an expression, the result of the evaluated expression must be an integer. If the type is wrong, AM may fail to start after a configuration import, with unexpected errors.

You can transform the output type of the evaluated token to match the type that is required by the property. Amster can coerce expressions to resolve as the following types:

- `$int` . Coerce to an integer.
- `$number` . Coerce to integers, doubles, longs, and floats.
- `$bool` . Coerce to a boolean. For example, `true` .
- `$array` . Coerce to a JSON array. For example, `["a","b","c"]` .
- `$list` . Coerce to a JSON list. For example, `1,2,3` .
- `$object` . Coerce to a JSON object. For example, `{"a","b"}` .
- `$base64-encode` and `$base64-decode` . Encode or decode the string to or from Base64.

To convert the value of a property into a different type in the exported configuration file, specify the new type as follows:

```
"port" : {
    "$int" : "&{smtp.port}"
}
```

You can also replace configuration values with the contents of a file by using the `$inline` coercion function. You can specify the path to the file, or replace it with an expression. For example:

```
{
 "message" : {
    "$inline" : "&{config.path}/emailcontent.txt"
 },
 "message2" : {
    "$inline" : "/path/to/emailcontent.txt"
 }
}
```

Consider the following points when using the `$inline` coercion function:

- When the file contains a script, such as an authentication script, you must transform its value to Base-64. For example:

  ```
  "script": {
    "$base64:encode": {
      "$inline": "/path/to/scripted-decision.groovy"
    }
  }
  ```

- When the file contains a value of a type that is different from the configuration value type, you must transform its value. For example:

  ```
  "port": {
    "$int": {
        "$inline": "myfile.txt"
    }
  }
  ```

TIP

Recognizing the type of a particular configuration property in the JSON files may not always be straightforward. When in doubt, try the following approaches:

- Check the property in the AM console.
- Configure the property in the AM console, then export the configuration with Amster for an example.

Expression Example Files

This section contains an example expression file, and some excerpts of exported configuration files with expressions inserted in them.

Expression File

```
{
  "env" : {
     "name" : "DEV"
  },
  "oauth" : {
     "authmodule" : {
        "issuer" : "dev-oathissuer.example.com",
        "authlevel" : 2,
        "checksum" : "true"
     },
     "devprof" : {
        "kstore" : {
           "path" : "&{product.install.dir}/&
{env.name}/keystore.jceks",
           "type" : "JCEKS",
           "encpas" :
"AAAAA0FFUwIQ1WDDMsxGoZMiRHhDQ+ywUfTMdGtYqEsvZZLV9W8ygfHi/5kBWjMp
yg=="
        }
     }
  }
}
```

As well as configuration details, such as hostnames and ports, passwords and secrets are likely to differ between AM instances.

The previous example demonstrates an expression file tailored for the development environment. Note how the `&{devprof.kstore.encpas}` expression holds the value of the encrypted keystore password for the OAuth device profile keystore configuration.

For security reasons, Amster only exports passwords in configuration files if the transport key exists in AM's keystore.

If your environments have different passwords, you could manage passwords using expressions as follows:

1. Configure AM with the desired password values for each of your environments.
2. Export the configurations *using the same transport key*.

Using this technique ensures that the passwords for all the environments are properly encrypted. You can safely create expression files by environment with the appropriate values.

Configuration File Excerpts

```
{
   "data":{
      "_id":"",
      "defaults":{
         "oathIssuerName":"&{oath.authmodule.issuer}",
         "totpTimeStepsInWindow":2,
         "authenticationLevel":{
            "$int":"&{oath.authmodule.authlevel}"
         }
      },
      "passwordLength":"6",
      "addChecksumToOtpEnabled":{
         "$bool":"&{oath.authmodule.checksum}"
      }
   },
   "data":{
      "_id":"",
      "defaults":{
         "oathAttrName":"oathDeviceProfiles",

"authenticatorOATHDeviceSettingsEncryptionKeystorePrivateKeyPassw
ord":null,

"authenticatorOATHDeviceSettingsEncryptionScheme":"NONE",
         "authenticatorOATHDeviceSettingsEncryptionKeystore":"&
{oath.devprof.kstore.path}",

"authenticatorOATHDeviceSettingsEncryptionKeystoreType":"&
{oath.devprof.kstore.type}",

"authenticatorOATHDeviceSettingsEncryptionKeystorePassword":null,
```

```
"authenticatorPushDeviceSettingsEncryptionKeystorePassword-
encrypted":"&{oath.devprof.kstore.encpas}",

"authenticatorOATHDeviceSettingsEncryptionKeystoreKeyPairAlias":n
ull,
         "authenticatorOATHSkippableName":"oath2faEnabled"
      }
   },
   "data":{
      "_id":"01e1a3c0-038b-4c16-956a-6c9d89328cff",
      "name":"Authentication Tree Decision Node Script &
{env.name}",
      "description":"&{product.install.dir}/&
{env.name}/authdecisionnode_desc.txt",
      "script":{
         "$base64:encode":{
            "$inline":"&{product.install.dir}/&
{env.name}/scripted-decision.groovy"
         }
      }
   }
}
```

Note how the files used by the `$inline` coercion function are stored under a directory that is referenced by the `&{env.name}` expression. For example:

```
"$inline":"&{product.install.dir}/&{env.name}/scripted-
decision.groovy"
```

This is just an example of how you can separate your configuration files by environment.

Scripting

You can create script files containing a series of commands and variable declarations, which can be loaded and executed within Amster.

Start each separate command or variable declaration on a new line. Use the backslash (kbd:[\]) character to represent line continuations.

For example, the following script installs an AM instance, and then exits the Amster command-line interface:

```
install-openam \
 --serverUrl https://openam.example.com:8443/openam \
 --authorizedKey /var/amster/authorized_keys \
 --cookieDomain .example.com \
 --adminPwd forgerock \
 --cfgStoreHost opendj.example.com \
 --cfgStoreDirMgrPwd password  \
 --cfgStoreAdminPort 1389  \
 --cfgStore dirServer \
 --cfgDir /root/openam \
 --userStoreDirMgrPwd password \
 --userStoreHost opendj.example.com \
 --userStoreType LDAPv3ForOpenDS \
 --userStorePort 1389 \
 --userStoreRootSuffix dc=openam,dc=forgerock,dc=org \
 --acceptLicense
 :exit
```

To load and execute the commands within a script, use the `:load` command, as follows:

```
am> :load myScript.amster
```

You can specify more than one script to load. Scripts are loaded and executed in the order they are specified. If a command in a script fails, execution continues with the next command.

You can also invoke scripts by passing them as a parameter to the **`amster`** command.

For example:

```
$ vi samples/myScript.amster
  connect https://openam.example.com:8443/openam -k
/home/forgerock/am/amster_rsa
  :exit
$ ./amster samples/myScript.amster

Amster OpenAM Shell (version build build, JVM: version)
Type ':help' or ':h' for help.
-------------------------------------------------------------------
---------------
am> :load samples/myScript.amster

===> true
```

The Amster shell supports an `eval(String)` function, which evaluates any Amster command expressed as a string. For example, the function is required within looping structures:

```
for (i = 0; i < 4; i++) {
    eval("create DataStoreModule --realm / --body
'{\"_id\":\"myDataStore$i\"}'")
 }
```

You must also use the `eval(String)` function when using Amster commands in conditional structures:

```
dbStatus = databaseName
    ? 'Found'
    : eval("create DataStoreModule --realm / --body
'{\"_id\":\"myDataStore\"}'")
```

TIP

Amster includes a number of sample scripts in the `/path/to/amster/samples` directory.

Check for errors when running in scripts

There is no way to exit with a non-zero status code when an `amster` command produces a result other than success.

To monitor `amster` command errors, send the command output to a file, then search that file for success or failure conditions.

For example:

```
$ ./amster samples/myScript.amster >> myOutputfile.txt
```

Note that error and success messages can change between versions, so any scripts that rely on these messages should be reviewed during upgrades.

Using Variables in Amster Scripts

When scripting Amster tasks, it is often useful to use variables. An example would be storing the AM connection string in a variable, the value of which is swapped among environments.

You can define variables in the Amster Groovy shell directly, or you can import them to the shell if they are defined as Java properties or as operating system environment variables:

Amster shell variables

Define Amster shell variables using the standard camel case notation for naming variables in Groovy. For example:

```
am> **smtpPort = "1342"**
===> 1342
```

You can define maps as shell variables, but Amster commands cannot access the contents of the map directly. Assign key values to Amster shell variables so that commands can use them. For example:

```
am> myMap= [ AM_URL: "https://openam.example.com:8443/openam",
AMSTER_KEY: "/opt/openam/id_rsa" ]
===> [AM_URL:https://openam.example.com:8443/openam,
AMSTER_KEY:/opt/openam/id_rsa]
am> myAM= myMap.AM_URL
===> https://openam.example.com:8443/openam
am> myKey= myMap.AMSTER_KEY
===> /opt/openam/id_rsa
am> connect -k myKey myAM
```

Operating system environment variables

Import environment variables into the Amster shell using Groovy syntax.

The following commands are examples of operations you can perform in a Groovy shell. For more information, refer to the Groovy documentation.

To see all the environment variables available for import from a Unix shell, run the following command:

```
am> System.getenv()
===>
[PATH:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/opt/X11/bin,
         SHELL:/bin/bash, JAVA_HOME:/path/to/jvm, TERM:xterm-
256color,
         USER:ForgeRock, LANG:en_GB.UTF-8,
PWD:/Users/ForgeRock/amster……]
```

To assign the value of an environment variable to an Amster shell variable, run the following command:

```
am> myShell=System.getenv("SHELL")
===> /bin/bash
```

To assign all the environment variables to a map and then list them, run the following commands:

```
am> ENV=System.getenv()
===> [SHELL:/bin/bash, USER:ForgeRock, LANG:en_GB.UTF-8,
PWD:/Users/ForgeRock/amster,
      AMURL:https://openam.example.com:8443/openam,
CFGDIR:/opt/openam……]
am> ENV.each { println it }
SHELL:/bin/bash
USER:ForgeRock
LANG:en_GB.UTF-8
PWD:/Users/ForgeRock/amster
AMURL:https://openam.example.com:8443/openam
CFGDIR:/opt/openam
……
```

Amster commands cannot access the contents of a map directly; you must assign key values to Amster shell variables before commands can use them. For example:

```
am> myAM = ENV.AMURL
===> https://openam.example.com:8443/openam
am> myCfgDir = ENV.CFGDIR
===> /opt/openamam> install-openam --serverURL myAM --adminPWd
forgerock --cfgDir myCfgDir --acceptLicense
```

The following is an example of an Amster script that assigns the value of environment variables to Amster shell variables:

```
myAM = System.getenv("AMURL")
myCfgDir = System.getenv("CFGDIR")
install-openam --serverURL myAM --adminPWd forgerock --cfgDir
myCfgDir --acceptLicense
:exit
```

Java system properties

You can pass environment variables to the Amster shell when executing the **amster** command with the -D parameter.

For example, you could create the following bash script to call the **amster** command:

```
#!/bin/bash
amUrl="https://openam.example.com:8443/openam"
amsterKey="/root/openam/amster_rsa"
configPath="/root/am-config"
./amster export-config.amster -D AM_URL=${amUrl} -D
AMSTER_KEY=${amsterKey} \
-D AM_CONFIG_PATH=${configPath}
```

To see all properties available from the Java runtime, run the following command:

```
am> System.getProperties()
===> [java.runtime.name:Java™ SE Runtime Environment,
AM_URL:https://openam.example.com:8443/openam,
      java.vm.version:_version_, gopherProxySet:false, …,
path.separator::, …]
```

To import the Java variables into the Groovy shell, run the following command:

```
am> amUrl = System.getProperty("AM_URL")
===> https://openam.example.com:8443/openam
am> amsterKey = System.getProperty("AMSTER_KEY")
===> /root/openam/amster_rsa
am> exportPath = System.getProperty("AM_CONFIG_PATH")
===> /root/am-config
```

You can use the variables in an Amster script by importing them in Groovy first. For example:

```
amUrl = System.getProperty("AM_URL")
amsterKey = System.getProperty("AMSTER_KEY")
exportPath = System.getProperty("AM_CONFIG_PATH")

connect amUrl -k amsterKey
export-config --path exportPath --failOnError
:exit
```

To see all the environment variables defined in the Amster shell, run the following command:

```
am> binding.variables.each{ println it.key println it.value }
eval
org.codehaus.groovy.runtime.MethodClosure@3f270e0a
_
/bin/bash
amUrl
https://openam.example.com:8443/openam
smtpPort
1342
amsterKey
/root/openam/amster_rsa
exportPath
/root/am-config
myShell
/bin/bash
===> [eval:org.codehaus.groovy.runtime.MethodClosure@3f270e0a,
_:/bin/bash,
      amUrl:https://openam.example.com:8443/openam,
smtpPort:1342,
      amsterKey:/root/openam/amster_rsa, exportPath:/root/am-
config, myShell:/bin/bash]
```

Amster Usage Examples

In this section, you can find examples of tasks you can do with Amster.

TIP

For Amster examples in Docker and Kubernetes deployments, see the ForgeRock DevOps (ForgeOps) documentation.

Cloning an Access Management Instance

This example shows the high-level steps required to clone an AM instance, from exporting the configuration of the original instance, to installing the new instance and importing the configuration into it.

Follow these steps to clone an AM instance using Amster:

1. Create a transport key in the original AM instance, if one does not exist already. For more information, see Create Transport Keys to Export Configuration Data.
2. Keep the transport key safe by exporting it to another keystore. The key is required to import the configuration into the new AM instance. For more information, see To

Duplicate and Install a Transport Key.

3. Connect to the original AM instance using the **`amster`** command. For more information, see Connect to AM.
4. Export all the configuration of the original AM instance using the **`export-config`** command. For more information, see Export Configuration Data.
5. Take note of the value of the Password Encryption Key field on the original AM, for example, `O6QWwHPO4os+zEz3Nqn/2daAYWyiFE32` .

   To locate it, log in to the original AM instance, and navigate to Deployment > Servers > *Server Name* > Security > Encryption.
6. In the new server, deploy the AM `.war` file in a web container, but do not configure it.
7. Install the new AM instance using the **`install-openam`** command, specifying the original AM password encryption key with the `--pwdEncKey` option. For example:

   ```
   am> install-openam \
    --serverUrl https://openam.example.com:8443/openam \
    --adminPwd forgerock \
    --pwdEncKey O6QWwHPO4os+zEz3Nqn/2daAYWyiFE32 \
    --acceptLicense
   ```

   For more information, see Install AM with Amster.
8. Import the transport key of the original AM instance into the keystore of the new AM instance. For more information, see To Duplicate and Install a Transport Key.
9. Connect to the new AM instance using the **`amster`** command. For more information, see Connect to AM.
10. Import the configuration of the original AM instance using the **`import-config`** command. For more information, see Import Configuration Data.

Amster Sample Scripts

This section covers sample scripts and files found in the `/path/to/amster/samples` directory:

`transport-key.sh`
: Shell script to manage transport keys. You can use it as a template for your own scripts to create, delete, and export the key to another keystore.

  Invoke the script's help for a list of possible actions:

  ```
  $ ./transport-key.sh help
  ```

For more information about the transport key, see Create Transport Keys to Export Configuration Data.

realm.amster
Amster script containing an example of different operations that can be done at realm level, such as creating a data store, displaying its configuration, modifying it, and deleting it.

For more information about writing scripts for Amster, see Scripting.

import-example.amster
Amster script containing an example of the `import-config` command.

For more information about writing scripts for Amster, see Scripting.

export-example.amster
Amster script containing an example of the `export-config` command.

Command-Line Reference

install-openam - Install Access Management

Synopsis

```
install-openam [options]
```

Description

Command to install and setup an AM instance.

The following parameters are required:

--adminPwd amAdmin-password
Specifies the password of the `amAdmin` user. If the `--cfgStoreDirMgrPwd` option is not specified, this value is also the password of the configuration store's directory manager user.

The password must be at least 8 characters in length.

--serverUrl protocol://FQDN:port/URI
Specifies the protocol, URL, port, and deployment URI of the AM instance. For example, `https://openam.example.com:8443/openam`.

The following options are available:

--acceptLicense

Specifies that the user accepts Amster usage terms and conditions.

--authorizedKey path

Specifies the path to an SSH public key file. The content of this file is appended to the `authorized_keys` file of the newly-installed AM instance, allowing users to connect to it with Amster after the install completes.

For more information about connecting to AM with Amster, see Connect to AM.

--cfgDir path

Specifies the configuration directory where AM stores files. It also stores the embedded directory server, when applicable.

Default: `$HOME/openam`

--cfgStore embedded|dirServer

Specifies the type of the configuration data store. Possible values are:

- `embedded` : Amster installs AM with an embedded DS server to act as the configuration, identity, and CTS stores.

  For evaluation deployments only.

- `dirServer` : Amster installs AM on an external DS server to act as the configuration store.

  When you install AM with an external configuration store, you must also use an external identity store. By default, identities are stored in the same directory server instance as the configuration store.

  Default: `embedded`

--cfgStoreAdminPort port

Specifies the administration port number for the configuration store.

Default: `4444`

--cfgStoreDirMgr username

Specifies the distinguished name of the directory manager user for the configuration store.

Default: `uid=admin`

--cfgStoreDirMgrPwd password

Specifies the password of the directory manager user for the configuration store.

Default: If not set, it takes the password defined for the `--adminPwd` option.

--cfgStoreHost FQDN

Specifies the FQDN of the configuration store, for example, `config.example.com`

Default: `localhost`

***--cfgStoreJmxPort** port*
Specifies the Java Management eXtension port number for the configuration store.

Default: `1689`

***--cfgStorePort** port*
Specifies the LDAP or LDAPS port number for the configuration store.

Default: `50636`

***--cfgStoreRootSuffix** DN*
Specifies the root suffix DN for the configuration store.

Default: `dc=openam,dc=forgerock,dc=org`

--cfgStoreSsl [SIMPLE|SSL]
Specifies whether AM should connect to the configuration store over SSL. Possible values are `SIMPLE`, for non-secure connections, and `SSL`, for secure connections.

Default: SSL

***--cookieDomain** domain*
Specifies the name of the trusted DNS domain AM returns to a browser when it grants a session ID to a user.

Default: FQDN used in the `--serverUrl` option

***--installLocale** locale*
Specifies the locale to use during the install process.

Default: `en_US`

***--lbPrimaryUrl** URL*
Specifies the load balancer URL of the site, such as `https://lb.example.com:443/openam`

***--lbSiteName** name*
Specifies the name of the site to create, if any.

***--platformLocale** locale*
Specifies the default locale for the AM installation.

Default: `en_US`

***--pwdEncKey** key*
Specifies the encryption key value that is used to encrypt passwords in the AM instance. For example `O6QWwHPO4os+zEz3Nqn/2daAYWyiFE32`.

If you are installing an AM instance that will be making use of exiting data in a data store, you must provide the same encryption key value originally used to encrypt the passwords in those stores.

To locate the encryption key value in an AM instance, navigate to Deployment > Servers > *Server Name* > Security > Encryption.

If you are installing a new AM instance that will not be using existing data in a data store, you can leave this property empty. AM will generate a random encryption key during installation to encrypt the data that will be added to the data store.

This option is *required* when configuring an AM instance into a site, and must be set to the encryption key configured for the rest of the servers in the site. Failure to set this option to the appropriate value will cause the original encryption key to be overwritten, which will render the site unable to read the configuration, and the user stores.

Default: No value; a random encryption key is generated during installation

***--userStoreDirMgr** username*
Specifies the distinguished name of the directory superuser for the user store, for example, `uid=admin`.

Default: Not set

***--userStoreDirMgrPwd** password*
Specifies the password of the directory manager user for the user store.

Default: Not set

***--userStoreDomainName** FQDN*
Specifies the Active Directory Domain Name, such as `ad.example.com`, when the `--userStoreType` option is set to `LDAPv3ForADDC`.

Default: Not set

***--userStoreHost** FQDN*
Specifies the FQDN of the configuration store, for example, `opendj.example.com`

Default: Not set

***--userStorePort** port*
Specifies the LDAP or LDAPS port number for the configuration store.

Default: Not set

***--userStoreRootSuffix** DN*
Specifies the root suffix DN for the user store.

Default: Not set

--userStoreSsl [SIMPLE|SSL]

Specifies whether AM should connect to the user store over SSL. Possible values are `SIMPLE`, for non-secure connections, and `SSL`, for secure connections.

Default: Not set

--userStoreType type

Specifies the type of user store to use when installing AM with an external configuration store. Possible values for *type* are:

- `LDAPv3ForOpenDS`, for DS stores.
- `LDAPv3ForAD`, for Active Directory with host and port settings.
- `LDAPv3ForADDC`, for Active Directory with domain name setting.
- `LDAPv3ForADAM`, for Active Directory Application Mode.
- `LDAPv3ForODSEE`, for Sun/Oracle DSEE.
- `LDAPv3ForTivoli`, for IBM Tivoli Directory Server.

When using the `LDAPv3ForADDC` store type, set up the `--userStoreDomainName` option to the Active Directory Domain Name, for example `ad.example.com`. Default: Not set