Core Token Service Guide (CTS)

/ ForgeRock Access Management 7.1

Latest update: 7.1.0

ForgeRock AS.
201 Mission St., Suite 2900
San Francisco, CA 94105, USA
+1 415-599-1100 (US)
www.forgerock.com

Abstract

Guide showing you how to set up a dedicated Core Token Service store for Access Management (AM). ForgeRock Access Management provides intelligent authentication, authorization, federation, and single sign-on functionality.

Table of Contents

Overview

This guide describes how to configure the Core Token Service token store and how to manage the tokens stored within.

Quick Start

Deployment Architecture

Learn about the different ways you can deploy the CTS token store and decide which one is best for your environment.

Configure a Dedicated CTS Store

Discover how to prepare a DS instance to host a dedicated CTS token store, and configure it in AM.

Manage Tokens

Learn how to encrypt and/ or compress tokens, if required.

Configure the CTS Reaper

Configure the how the tokens inside the CTS token store will be pruned after they reach their maximum time-to-live.

About ForgeRock Identity Platform™ Software

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.

Chapter 1

The Core Token Service

AM's Core Token Service (CTS) provides generalized, persistent, and highly available storage for sessions and tokens used by AM. AM uses CTS as the authoritative source for CTS-based sessions and caches these sessions in its memory heap to improve performance.[1]

CTS supports *session high availability*, which lets AM manage a session as long as one of the AM servers in a clustered deployment is available. After a user has successfully authenticated, AM creates a CTS-based session. Any AM instance that is configured to use the same CTS can retrieve the session and allow access to it. The user does not need to log in again unless the entire deployment goes down.[2]

CTS provides storage for the following:

- CTS-based sessions and CTS-based authentications sessions
- Session blacklist (if enabled for client-based sessions)
- Authentication session whitelist (if enabled for client-based sessions)
- SAML v2.0-related data (if enabled for Security Token Service token validation and cancellation)
- OAuth 2.0 and UMA 2.0 CTS-based tokens, and OAuth 2.0 client-based token blacklist
- Push notification data during authentication
- Site-wide notification, such as log out or session termination notifications.

The CTS token store is created at installation time in the external configuration store.

In general, the Core Token Service causes more volatile replication traffic due to the possibility of short-lived entries compared to regular configuration data. To handle the data volatility, ForgeRock recommends that you configure an additional DS server, separate from your configuration store, to isolate session and token information. For more information, see "General Recommendations for CTS Configuration".

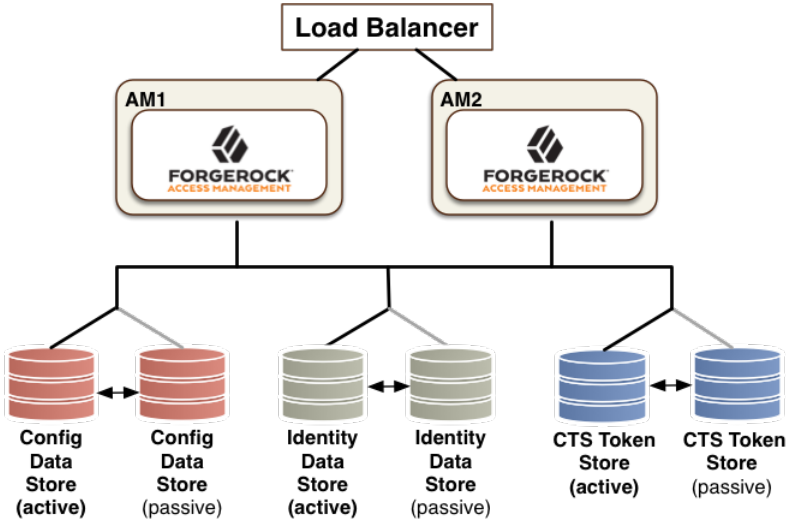
[1] Prior to AM 5, the authoritative source for sessions not stored in the client was the memory heap of AM's web container.
[2] Prior to AM 5, session high availability, formerly referred to as session failover, was optional. Starting with AM 5, session high availability is the default behavior in AM and cannot be disabled.

General Recommendations for CTS Configuration

CTS helps your deployment avoid single points of failure by providing high availability to sessions and tokens stored in its token store. To reduce the impact of any given failure, consider the following recommendations:

- **Configure Separate CTS Stores for High Volumes**. If you require a higher-level performance threshold, you may want to move the CTS token storage to one or more dedicated systems rather than sharing your configuration store, as CTS generally causes much more replication traffic than less volatile configuration data. The CTS token store is the primary source for session tokens and will experience both high read and write activity depending on session usage. Dedicated external CTS stores provide an extra level of control over the amount of global replication that is occurring.
- **Isolate the Different Stores**. CTS entries are large, around 5KB, but are short-lived, whereas configuration data is static and long-lived. User entries are more dynamic than configuration data but much less volatile than CTS data. Therefore, isolating the user, configuration, and CTS data from AM into separate stores allow for different tuning and storage settings per token store type.

Isolate the Datastores



- **Properly Tune Your DS Servers**. To improve performance, ensure that you have properly-sized directory servers for your external CTS stores. In addition, you can enable token compression as discussed in "*Managing CTS Tokens*". When enabled, token compression reduces load requirements on the network connection between token stores in exchange for processing time-compressing tokens.
- **Consider Dedicated Replication Servers**. Once configured, the DS server replicates CTS data transmitted from AM servers to connected DS servers. The amount of replication traffic can be

significant, especially if replication proceeds over a WAN. You can limit this replication traffic by separating DS instances into directory and replication servers as seen in "Core Token Service For Global Session Failover". For more information on how this is done with DS, see the DS documentation on *Standalone Servers*.

Chapter 2

Deployment Architectures

You can deploy CTS token stores in a number of deployment architectures depending on your system requirements:

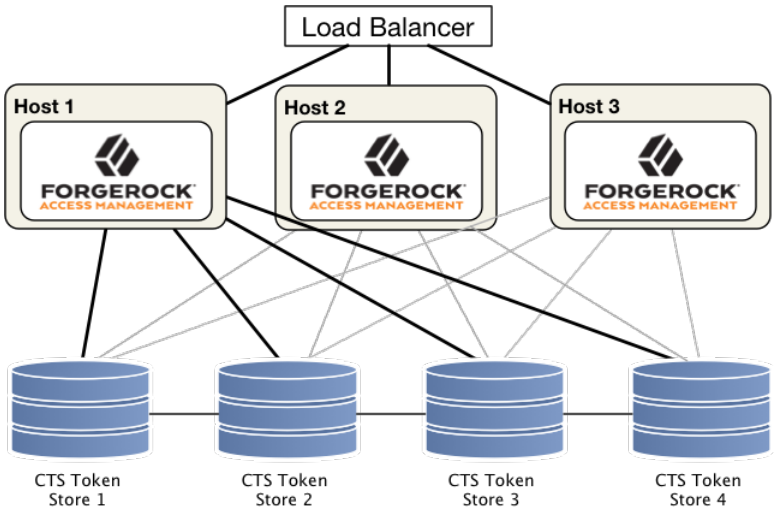
 Affinity CTS Token Store	 Site CTS Token Store
Use a DS affinity deployment for the CTS token store for high availability in the same datacenter.	Create site deployments in different geographic locations for high availability across datacenters.

CTS Affinity Deployment

In an *affinity* deployment, AM balances LDAP requests across one or more directory servers. AM always routes LDAP requests for a specific CTS token to the same directory server. This prevents attempts to read a token that has been written to another directory server, but not replicated yet. Affinity deployments are well suited for deployments with many AM servers.

Use AM's Connection String(s) property on the AM console to configure server affinity without a load balancer. For more information on the Connection String(s) property, see "External Store Configuration" in the *Reference*.

A CTS Affinity Deployment



For more information on CTS affinity deployments, see Best practice for using Core Token Service (CTS) Affinity based load balancing in AM (All versions) and OpenAM 13.5.1 in the *ForgeRock Knowledge Base*.

Note

The connection strings to the data or identity stores are static and not hot-swappable. This means that, if you expand or contract your DS affinity deployment, AM will not detect the change.

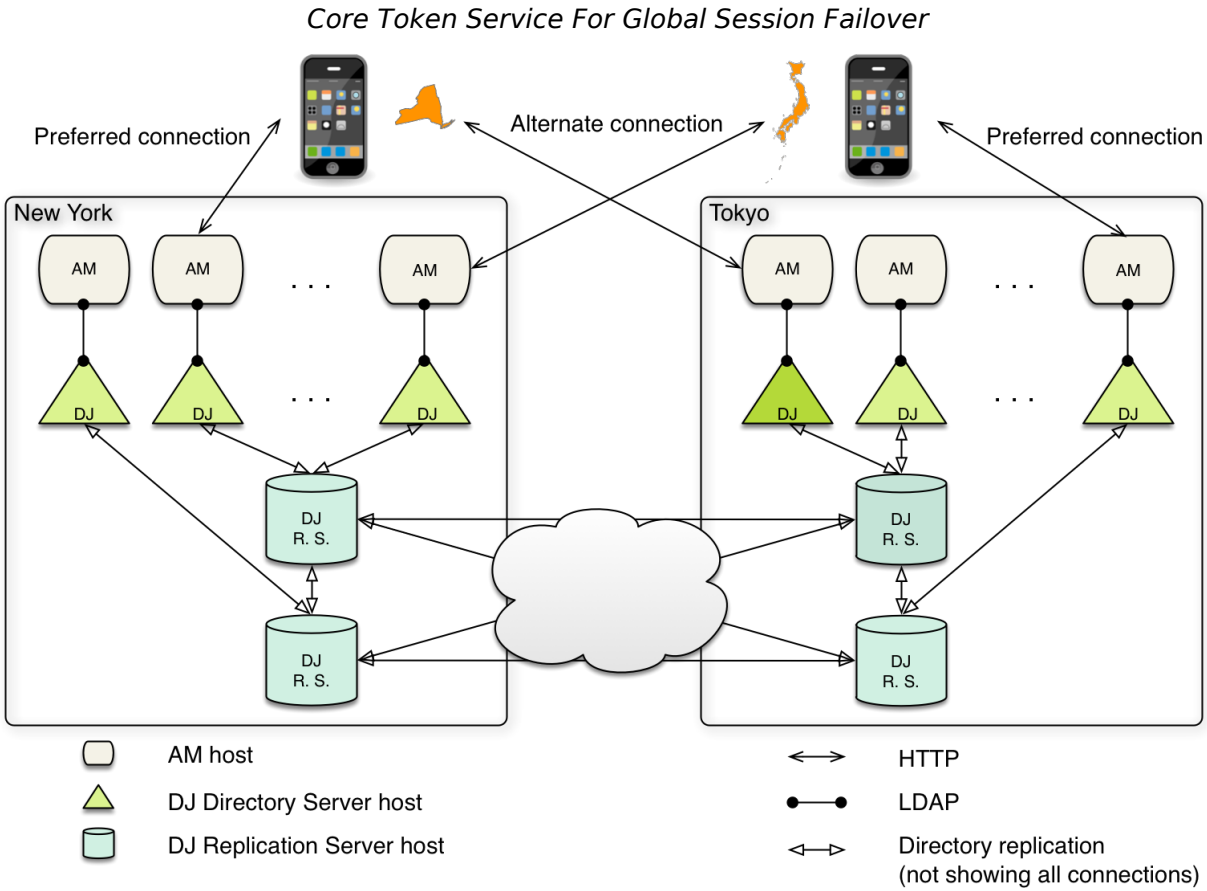
To work around this, either:

- Manually add or remove the instances from the connection string and restart AM or the container where it runs.
- Configure a DS proxy in front of the DS instances to distribute data across multiple DS *shards*, and configure the proxy's URL in the connection string.

CTS Site Deployment

CTS supports uninterrupted session availability in deployments with multiple sites if all sites use the same global underlying CTS store replicated across all sites. If an entire site fails or becomes unavailable, AM servers in another site can detect the failure of the site's load balancer and attempt to use sessions from the global Core Token Service.

In the event of a failure, client applications can connect to an AM server in an active data center as shown in "Core Token Service For Global Session Failover":



For more information on CTS for global session high availability with DS server, see the DS documentation on *Replication*.

Chapter 3
Configuring CTS Token Stores

The following table summarizes the high-level tasks you need to perform to configure a new instance of the CTS token store:

Task	Resources
Prepare a DS Server Prepare the DS schema for CTS data.	"To Install and Configure Directory Services for CTS Data"
Configure AM to Use the New CTS Token Store Configuring a new CTS data store *does not* migrate existing data from the old store to the new one. In most cases, this will mean that users will need to log in again so that AM stores their sessions and tokens in the new token store.	"To Configure CTS"
Test Session High-Availability For this test, you need to have a site with more than one instance. The idea is to log in a user into the first instance, shut it down, and check that the session is still available to the second instance.	"To Test Session High Availability"

To Install and Configure Directory Services for CTS Data

Directory Services 6.5 added support for *setup profiles* to greatly simplify initial configuration.

Using a setup profile will create the backend, schema, bind user, and indexes required for use with CTS data.

1. To install DS using a setup profile, follow the steps in "*DS for AM CTS*" in the *Directory Services 7.1 Installation Guide*.
2. Share the CTS store certificate with the AM container to prepare for TLS/LDAPS. The CTS store should communicate over secure connections for security reasons.

   DS 7 or later is configured to require secure connections by default; therefore, share its certificate with the AM container before continuing.

   + *Sharing the DS Certificate with AM*

     1. Export the DS server certificate:

```
$ /path/to/opendj/bin/dskeymgr export-ca-cert \
--deploymentKey $DEPLOYMENT_KEY \
--deploymentKeyPassword password \
--alias ds-ca-cert \
--outputFile ds-ca-cert.pem
```

Note that `$DEPLOYMENT_KEY` is a Unix variable that contains the DS deployment key, so that it is not logged in the user's command history.

The default DS server certificate only has the hostname you supplied at setup time, and `localhost`, as the value of the `SubjectAlternativeName` attribute; however, certificate hostname validation is strict.

Ensure that the certificate matches the hostname (or the FQDN) of the DS server before continuing.

2. Import the DS certificate into the AM truststore:

```
$ keytool \
-importcert \
-alias ds-ca-cert \
-file ds-ca-cert.pem \
-keystore /path/to/openam/security/keystores/truststore
```

For more information on configuring AM's truststore, see "Preparing a Truststore" in the *Installation Guide*.

3. Proceed to configuring the CTS store in AM. See "To Configure CTS".

   The bind DN of the service account to use when configuring the CTS store in AM is `uid=openam_cts,ou=admins,ou=famrecords,ou=openam-session,ou=tokens`.

To Configure CTS

This procedure assumes that the AM instances communicate with a single CTS instance (or load balancer), which has an FQDN of `cts.example.com` and is running on port `1636`.

Important

If AM cannot access the CTS token store, you will be unable to log in to the AM console.

Back up your deployment before making any changes to your CTS token store configuration.

Perform the following steps to configure an external CTS token store:

1. In the AM console, go to Configure > Server Defaults, and then click CTS.
2. On the CTS Token Store tab, configure the parameters as follows:

CTS Token Store Parameters

Parameter	Value	Notes
Store Mode	`External Token Store`	
Root Suffix	`ou=famrecords,ou=openam-session, ou=tokens`	
Max Connections	`17`	For production, this value needs to be tuned. Consider 2^n+1, where n=4, 5, 6, and so on. For example, try setting this to 17, 33, 65, and test performance under load.

3. On the External Store Configuration tab, configure the parameters as follows:

External Store Configuration Parameters

Parameter	Value	Notes
SSL/TLS Enabled	True	When connecting to DS 7, or a DS server in production mode, enable secure connections. When you enable SSL/TLS, make sure the AM server can trust the DS server certificate and that the certificate matches the CTS store FQDN.
Connection String(s)	`cts.example.com:1636`	Use the LDAPS port. Alternatively, use the LDAP port with StartTLS, as in `cts.example.com:1389`.
Login ID	`uid=openam_cts,ou=admins, ou=famrecords,ou=openam-session, ou=tokens`	The bind DN of the service account AM uses to connect to the directory service.
Password	*5up35tr0ng*	Use a strong bind password for production systems.
Heartbeat	`10`	For production, this value needs to be tuned.

4. Click Save Changes.

5. Restart AM or the web container where it runs for the changes to take effect.

To Test Session High Availability

To test session high availability, use two browsers: Chrome and Firefox. You can use any two browser types, or run the browsers in incognito mode. You can also view tokens using an LDAP browser.

1. In Chrome, log in to the second AM instance with the `amAdmin` user, select the realm, and then click on `sessions`.
2. In Firefox, log in to the first AM instance with a test user.
3. In Chrome, verify that the test user exists in the first AM instance's session list and not in the second instance.
4. Shut down the first AM instance.
5. In Firefox, rewrite the URL to point to the second AM instance. If successful, the browser should not prompt for login.
6. Confirm the session is still available. In Chrome, list the sessions on the second instance, the test user's session should be present.
7. Restart the first AM instance to complete the testing.

Chapter 4
Managing CTS Tokens

You can configure AM to encrypt or compress CTS tokens as they are stored in the token store. The following properties, disabled by default, are associated with token encryption and compression:

`com.sun.identity.session.repository.enableEncryption`

Supports encryption of CTS tokens. Default: `false`.

`com.sun.identity.session.repository.enableCompression`

Enables GZip-based compression of CTS tokens. Default: `false`.

`com.sun.identity.session.repository.enableAttributeCompression`

Supports compression over and above the GZip-based compression of CTS tokens. Default: `false`.

Important

Compression can undermine the security of encryption. You should evaluate this threat depending on your use cases before enabling compression and encryption together.

To Configure AM to Encrypt and / or Compress CTS Tokens for Storage

Warning

When encryption or compression properties are changed, all previous tokens in the LDAP store will be unreadable; thus, invalidating any user's sessions. As a result, the user will be required to log in again.

1. Navigate to Configure > Server Defaults > Advanced.
2. Find the property you want to enable in the Property Name column.
3. Replace the `false` value with `true` in the Property Value column.
4. Save your changes.
5. Enable the same property on every AM instance within the site. Failure to do so may cause unexpected issues storing and reading tokens across the environment.
6. Restart the AM servers for the changes to take effect.

Tip

Configuring the CTS to encrypt and store tokens incurs a performance penalty for AM. If you need to encrypt the stored tokens in your environment, consider configuring the CTS token store DS instance to encrypt the data instead. For more information about encrypting a DS instance, see the *ForgeRock Directory Services Security Guide*.

Chapter 5

CTS Backups and DS Replication Purge Delay

Replication is the process of copying updates between directory servers to help all servers converge to identical copies of directory, token, session, SAML v2.0, and OAuth 2.0 data. DS uses advanced data replication methods to ensure that directory services remain available in the event of a server crash or network interruption.

The historical information needed to resolve the latest changes is periodically purged to prevent it from becoming an unmanageable size. The age at which the information is purged is known as the `replication-purge-delay`.

With CTS, the default `replication-purge-delay` for DS is 3 days. Unless you have configured a separate DS server for CTS data, you may have to balance the needs for backups, the requirements for replication, disk space, and different useful lifetimes for CTS tokens and other DS data. Adjustments may be required. For example, to set the `replication-purge-delay` period to four hours, use the following command:

```
$ /path/to/opendj/bin/dsconfig set-synchronization-provider-prop \
--provider-name "Multimaster Synchronization" \
--set replication-purge-delay:4h \
--hostname 'cts.example.com' \
--port 4444 \
--usePkcs12TrustStore /path/to/opendj/config/keystore \
--trustStorePasswordFile /path/to/opendj/config/keystore.pin \
--bindDN uid=admin \
--bindPassword str0ngAdm1nPa55word \
--no-prompt
```

At this point, you need to understand whether CTS data backups are important in your deployment. Session, SAML v2.0, and OAuth 2.0 token data is often short-lived. In some deployments, the worst-case scenario is that users have to log in again.

If CTS data backups are important in your deployment, note that DS backups that are older than the `replication-purge-delay` are useless and must be discarded. You can use the DS **dsbackup** command to schedule backups. For more information, see *Backup and Restore* in the *ForgeRock Directory Services Maintenance Guide*.

If you adjust the time periods associated with `replication-purge-delay` and backups, you need to backup more frequently so that the change log records required to restore date are not lost.

Chapter 6
Configuring the CTS Reaper

Tokens in the CTS store have a time to live after which they must be pruned. AM can manage expired tokens by either using its own reaper (the default) or by delegating the task to DS. Both AM and DS are equally adept at managing expired tokens. However, choosing DS to manage token expiration in the place of AM's reaper frees resources in the AM servers than can then be used for policy or authorization requests.

Review the following list for more information about the different configurations:

AM CTS Reaper (Default)

When an AM server modifies a token in the CTS token store, it also takes responsibility to manage it when it expires. To determine which tokens have expired, each AM server maintains a local cache of which tokens to delete and when. This reduces the number of relatively slow queries to the CTS store.

Use of the local reaper cache means fewer searches to the CTS store to determine expired tokens to delete, which improves overall cluster performance. A search of the CTS store for expired tokens is still performed as a fail safe, to ensure expired tokens are not missed when a server in the cluster goes down.

The AM CTS reaper is enabled by default. However, if you have configured the DS expiration and deletion feature, and you need to enable the AM CTS reaper again, see "To Enable the AM Reaper".

DS Entry Expiration and Deletion Feature

When the AM CTS reaper is disabled, AM relies on DS capabilities to delete expired tokens.

By default, DS does not reap its contents, so you need to configure the entry expiration and deletion feature manually. This feature uses an ordering index to find those tokens that have reached their time to live and expires (deletes) them.

Before you decide to configure DS to manage CTS token expiration, consider the following points:

- The CTS token store must be in DS version 6.0 or later.
- Deletion of entries is not replicated across DS servers. Therefore, you must ensure all the CTS store replicas are configured in the same way. For more information, see *Automating Entry Expiration and Deletion* in the *ForgeRock Directory Services Maintenance Guide*.
- Completely disabling the AM reaper impacts session-related functionality, such as sending notifications about session expiration or timeouts to agents.

Tip

A common implementation to manage CTS tokens is to configure AM to reap session tokens only, while using DS capabilities to manage the non-session tokens. This configuration ensures your environment still can make use of all session functionality, while benefiting from DS's capabilities as well.

You can configure the AM reaper to manage different subsets of tokens other than session tokens, too, if your environment requires it.

The following is a list of session functionality that is not available when the AM reaper is completely disabled:

- Implementations of `org.forgerock.openam.cts.reaper.TokenDeletionStrategy`, which are responsible for deleting expiring tokens. Any custom logic for specific token types that is not related to simply deleting tokens no longer work.
- Implementations of `org.forgerock.openam.cts.continuous.watching.ContinuousListener` and `org.forgerock.openam.cts.continuous.watching.ContinuousWatcher` no longer receive deletion notifications in case of session timeout (active logout is not impacted).
- Code built on top of `com.iplanet.dpro.session.service.SessionEventListener` no longer receives `IDLE_TIMEOUT` or `MAX_TIMEOUT` events. This affects:
  - Session timeout monitoring
  - Session timeout auditing
  - Session timeout logging
  - Session timeout notifications for agents (PLL, WebSockets)
  - Timeout handlers configured via `openam-session-timeout-handler-list`
  - Session web hooks registered during authentication
- Code built on top of `com.iplanet.dpro.session.watchers.listeners.SessionDeletionListener` no longer receives notifications when the session idle or maximum lifetime is exceeded (common usage includes caches).
- Listeners built on top of `com.sun.identity.plugin.session.SessionListener` no longer receive notifications on active logout, nor when the maximum session time is reached (idle timeout is ignored).

If your environment does not require the session capabilities listed above, see "To Enable DS to Manage the Tokens in the CTS Store (Impacted Session Capabilities Not Required) ".

If your environment requires the session capabilities above, see "To Enable DS to Manage the Tokens in the CTS Store (Impacted Session Capabilities Required)".

To Enable DS to Manage the Tokens in the CTS Store (Impacted Session Capabilities Not Required)

This procedure assumes you have ensured that your environment does not require the session capabilities impacted by stopping the AM CTS reaper. Perform the steps in this procedure to configure DS to manage expiration and deletion of all CTS tokens in your environment:

1. Perform one of the following options depending on whether you are installing a new Directory Services instance for CTS, or are modifying an existing instance:

   - If you are creating a *new instance* of Directory Services for use with CTS, perform the following step:

     - Use a setup profile, and set the `tokenExpirationPolicy` property to `ds`.

       For more information, see *DS for AM CTS* in the *Directory Services Installation Guide*.

   - If you are modifying an *existing instance* of Directory Services for use with CTS, perform the following step:

     - Configure the DS entry expiration and deletion feature for the `coreTokenExpirationDate` attribute. Note that the attribute is already indexed, but requires the TTL-related properties to be enabled. For example:

       ```
       $ /path/to/opendj/bin/dsconfig set-backend-index-prop \
       --hostname 'cts.example.com' \
       --port 4444 \
       --usePkcs12TrustStore /path/to/opendj/config/keystore \
       --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
       --bindDN uid=admin \
       --bindPassword str0ngAdm1nPa55word \
       --backend-name ctsStore \
       --index-name coreTokenExpirationDate \
       --set ttl-enabled:true \
       --set ttl-age:10s \
       --no-prompt
       ```

       Deletion of entries is not replicated across DS servers. Therefore, you must ensure all the CTS store replicas are configured in the same way. For more information, see *Entry Expiration* in the *ForgeRock Directory Services Maintenance Guide*.

2. Disable the AM CTS reaper by navigating to Configure > Server Defaults > Advanced and setting the `org.forgerock.services.cts.store.reaper.enabled` property to `false` on all the AM servers on your deployment.

   The changes are effective immediately.

   For more information about these advanced server properties, see "Advanced Properties" in the *Reference*.

To Enable DS to Manage the Tokens in the CTS Store (Impacted Session Capabilities Required)

This procedure assumes that your environment requires the session capabilities that would be impacted by stopping the AM CTS reaper. Perform the steps in this procedure to configure AM to reap `SESSION` tokens only, while using DS capabilities to manage the rest:

1. Perform one of the following options depending on whether you are installing a new Directory Services instance for CTS, or are modifying an existing instance:

   - If you are creating a *new instance* of Directory Services for use with CTS, perform the following step:

     - Use a setup profile, and set the `tokenExpirationPolicy` property to `am-sessions-only`.

       For more information, see *DS for AM CTS* in the *Directory Services Installation Guide*

   - If you are modifying an *existing instance* of Directory Services for use with CTS, perform the following steps:

     a. (Optional) If you have existing tokens in the CTS store, you must copy the value of the `coreTokenExpirationDate` attribute to the `coreTokenTtlDate` for all the existing tokens **except** for `SESSION` tokens.

        AM will manage the `SESSION` tokens separately.

        If you do not perform this step, neither the AM reaper nor the DS expiration feature will delete these tokens and they will remain untouched in the CTS store until manually removed.

     b. Create an ordering index for the `coreTokenTtlDate` attribute. For example:

        ```
        $ /path/to/opendj/bin/dsconfig create-backend-index \
        --hostname 'cts.example.com' \
        --port 4444 \
        --usePkcs12TrustStore /path/to/opendj/config/keystore \
        --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
        --bindDN uid=admin \
        --bindPassword str0ngAdm1nPa55word \
        --backend-name ctsStore \
        --index-name coreTokenTtlDate \
        --set index-type:ordering \
        --no-prompt
        ```

        Deletion of entries is not replicated across DS servers. Therefore, you must ensure all the CTS store replicas are configured in the same way.

     c. Rebuild the new index using the **rebuild-index** command. For example:

```
$ /path/to/opendj/bin/rebuild-index \
--hostname 'cts.example.com' \
--port 4444 \
--useSsl \
--usePkcs12TrustStore /path/to/opendj/config/keystore \
--trustStorePasswordFile /path/to/opendj/config/keystore.pin \
--bindDN uid=admin \
--bindPassword str0ngAdm1nPa55word \
--baseDN "dc=cts,dc=example,dc=com" \
--index coreTokenTtlDate
```

Deletion of entries is not replicated across DS servers. Therefore, you must ensure all the CTS store replicas are configured in the same way.

d. Configure DS's entry expiration and deletion feature in the CTS store for the `coreTokenTtlDate` index created in the previous steps. For example:

```
$ /path/to/opendj/bin/dsconfig set-backend-index-prop \
--hostname 'cts.example.com' \
--port 4444 \
--usePkcs12TrustStore /path/to/opendj/config/keystore \
--trustStorePasswordFile /path/to/opendj/config/keystore.pin \
--bindDN uid=admin \
--bindPassword str0ngAdm1nPa55word \
--index-name coreTokenTtlDate \
--backend-name ctsStore \
--set ttl-enabled:true \
--set ttl-age:10s \
--no-prompt
```

Deletion of entries is not replicated across DS servers. Therefore, you must ensure all the CTS store replicas are configured in the same way.

For more information, see *Entry Expiration* in the *ForgeRock Directory Services Maintenance Guide*.

2. In the AM console, go to Configure > Server Defaults > Advanced and configure the following advanced server properties in all of the AM servers sharing the CTS store cluster:

   - Set `org.forgerock.services.cts.store.ttlsupport.enabled` to `true`.
   - Set `org.forgerock.services.cts.store.ttlsupport.exclusionlist` to `SESSION`.
   - Set `org.forgerock.services.cts.store.reaper.enabled` to `true`.

   The changes are effective immediately.

   For more information about these advanced server properties, see "Advanced Properties" in the *Reference*.

To Enable the AM Reaper

This procedure assumes you enabled the DS expiration and deletion feature and you want to disable it and enable the AM reaper again:

1. (Optional) If you enabled the DS expiration and deletion feature, following the steps in "To Enable DS to Manage the Tokens in the CTS Store (Impacted Session Capabilities Not Required) ", perform the following steps to enable the AM CTS reaper for all tokens:

   a. Disable the DS entry expiration and deletion feature for the `coreTokenExpirationDate` index. Note that you should not delete the index. For example:

   ```
   $ /path/to/opendj/bin/dsconfig set-backend-index-prop \
   --hostname 'cts.example.com' \
   --port 4444 \
   --usePkcs12TrustStore /path/to/opendj/config/keystore \
   --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
   --bindDN uid=admin \
   --bindPassword str0ngAdm1nPa55word \
   --backend-name ctsStore \
   --index-name coreTokenExpirationDate \
   --set ttl-enabled:true \
   --set ttl-age:10s \
   --no-prompt
   ```

   Ensure that all the CTS store replicas are configured in the same way.

   b. Enable the AM CTS reaper by by navigating to Configure > Server Defaults > Advanced and setting the `org.forgerock.services.cts.store.reaper.enabled` property to `true` on all the AM servers on your deployment.

2. (Optional) If you enabled the DS expiration and deletion feature following the steps in "To Enable DS to Manage the Tokens in the CTS Store (Impacted Session Capabilities Required)", perform the following steps to enable the AM CTS reaper for all tokens:

   a. Disable the DS entry expiration and deletion feature for the `coreTokenTtlDate` index. For example:

   ```
   $ /path/to/opendj/bin/dsconfig set-backend-index-prop \
   --hostname 'cts.example.com' \
   --port 4444 \
   --usePkcs12TrustStore /path/to/opendj/config/keystore \
   --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
   --bindDN uid=admin \
   --bindPassword str0ngAdm1nPa55word \
   --index-name coreTokenTtlDate \
   --backend-name ctsStore \
   --set ttl-enabled:false
   ```

   Ensure that all the CTS store replicas are configured in the same way.

   b. Delete the index created for the `coreTokenTtlDate` attribute. For example:

```
$ /path/to/opendj/bin/dsconfig delete-backend-index \
--hostname 'cts.example.com' \
--port 4444 \
--usePkcs12TrustStore /path/to/opendj/config/keystore \
--trustStorePasswordFile /path/to/opendj/config/keystore.pin \
--bindDN uid=admin \
--bindPassword str0ngAdm1nPa55word \
--backend-name ctsStore \
--index-name coreTokenTtlDate \
--no-prompt
```

Ensure that all the CTS store replicas are configured in the same way.

c. Navigate to Configure > Server Defaults > Advanced and configure the following advanced server properties in all of the AM servers sharing the CTS store cluster:

- Set `org.forgerock.services.cts.store.ttlsupport.enabled` to `false`.
- Remove the `org.forgerock.services.cts.store.ttlsupport.exclusionlist` property from the configuration.
- Set `org.forgerock.services.cts.store.reaper.enabled` to `true`.

The changes are effective immediately.

For more information about these advanced server properties, see "Advanced Properties" in the *Reference*.

Chapter 7

Tuning the CTS

There are several tuning considerations that you can make for the efficient processing of your CTS token store:

Task	Resources
Configure the Reaper Cache Size The reaper process prunes tokens in the store that have reached their maximum time-to-live.	"Reaper Cache Size"
Manage the Reaper Search Size In addition to the cache, the reaper also uses a search to find expired tokens. You can tune the maximum amount of tokens that the reaper can find and delete in a single invocation.	"Reaper Search Size"
Manage the CTS Queue Size and Timeout Every create, update, and delete operation sent to the CTS token store is placed in an asynchronous queue before being handled. When the queue is full, new operations are blocked until they get into the queue or time out.	"Queue Size and Timeout"
Disable Virtual Attributes To improve CTS throughput, disable unnecessary virtual DS attributes.	"Virtual Attributes"
Change the OAuth 2.0 Storage Scheme If the CTS token store was created on a version of AM earlier than 6.5, configure the `grant-set` scheme to improve the performance of OAuth 2.0-related operations.	"OAuth 2.0 CTS Storage Scheme"

Reaper Cache Size

The size of the AM reaper cache is controlled by the `org.forgerock.services.cts.reaper.cache.size` advanced property. The default size is `500000` tokens.

If an AM server is under sustained heavy load, the reaper cache may reach capacity, causing degraded performance due to the additional slower searches of the CTS store. If the reaper cache is full, messages are logged in the `Session` debug log, such as the following:

```
The CTS token reaper cache is full. This will result in degraded performance.
You should increase the cache size by setting the advanced server property
'org.forgerock.services.cts.reaper.cache.size' to a number higher than 500000.
```

If this debug message appears frequently in the debug logs, increase the value of the `org.forgerock.services.cts.reaper.cache.size` property. To alter the value, in the AM console, go to Configure > Server Defaults > Advanced, and add the property and increased value to the list.

Increasing the size of the reaper cache causes higher memory usage on the AM server. If a cache of the default size of 500000 entries is nearly full, the server memory used could be up to approximately 100 megabytes.

Note

Tune the AM reaper cache size only if the `org.forgerock.services.cts.store.reaper.enabled` advanced server property is set to `true`.

Reaper Search Size

The reaper uses its cache to ensure that expired tokens are removed from the CTS token store, but it also has a built-in mechanism to search for expired tokens that may have not been purged. This could happen, for example, due to an AM instance crash.

When the reaper searches for expired tokens, DS returns a page of records that then the reaper will delete.

To configure the maximum amount of records that DS will return, use the `org.forgerock.services.cts.reaper.search.tokenLimit` advanced server property The default is `5000`.

To configure how often the search runs, and the grace period of the tokens that will be reaped, configure the following properties:

- `org.forgerock.services.cts.reaper.search.pollFrequencyMilliseconds`. Specifies how often to perform a search for expired tokens. The default is `5000` milliseconds.

  The value of this setting should never be higher than a few seconds, since the goal is to launch the search fair often so that the expired tokens do not pile up.

- `org.forgerock.services.cts.reaper.search.gracePeriodMilliseconds`. Specifies the grace period used when searching for expired tokens. Any tokens that expired more than the specified duration ago are returned. The default is `300000` milliseconds.

  The grace period should be larger than the value controlled by the `org.forgerock.services.cts.reaper.cache.pollFrequencyMilliseconds` advanced property. This allows an AM instance sufficient time to delete the token using its cache, rather than search.

  Deleting from the cache is preferred as it avoids expensive searches against the CTS store.

To tune the reaper searches, run load tests in your environment with the default reaper parameters, and then watch if the expired tokens build up. If there is none, then the reaper is sufficiently tuned. If you see token build-up, tune the reaper cache first, since it is the less expensive of the two reaping mechanisms.

It is fine if there is a small amount of expired tokens left on each iteration of the reaper search as long as tokens are cleared up during lower activity periods.

Keeping a large amount of expired tokens in the CTS store will result in increased disk space usage, and in AM not sending expiration notifications in a timely fashion.

Setting the token limit too high could negatively affect the performance of the DS instance. It is better to launch more runs of the reaper search with smaller amounts of records, than fewer runs with larger amounts of records.

Queue Size and Timeout

One tuning consideration is to manage the CTS queue size and timeout for efficient throughput. AM makes CTS requests from the following components:

- AM CTS-based sessions and authentication sessions
- OAuth 2.0
- UMA
- Session blacklist (if enabled for client-based sessions)
- Authentication session whitelist (if enabled for client-based sessions)
- SAML v2.0 (if enabled for Security Token Service token validation and cancellation)
- OAuth 2.0 CTS-based tokens and client-based token blacklist
- Push notification during authentication
- Cluster-wide notification

Every create, update, and delete requests to CTS are placed into an asynchronous buffer before being handled by an asynchronous processor. This ensures that callers performing write operations can continue without waiting for CTS to complete processing.

Once the queue is full, all new operations are "blocked" before being placed in the queue. Once the queue is frees up, the caller can continue as normal.

CTS is designed to automatically throttle throughput when the buffer fills up with requests. Therefore, if you require a balance between performance versus system memory, AM provides two properties that can be used to tune CTS, queue size and queue timeout.

`org.forgerock.services.cts.async.queue.size`

Default size: 5000. Determines the amount of request operations that can be buffered before the queue size becomes full, after which the caller will be required to wait for the buffered requests to complete processing. All CRUDQ operations are converted to tasks, which are placed in the queue, ensuring that operations happen in the correct sequence.

`org.forgerock.services.cts.async.queue.timeout`

Default timeout is 120 seconds. Determines the length of time a caller will wait when the buffer is full. If the timeout expires, the caller receives an error. The timeout property is used in any system configuration where the LDAP server throughput is considerably slower than the AM server, which can result in blocked requests as the backlog increases.

To set the queue size and timeout properties in the AM console, go to Configure > Server Defaults > Advanced, enter the key name and value, and then click Add.

For additional information on tuning CTS, see "Tuning CTS Store LDAP Connections" in the *Maintenance Guide*.

Virtual Attributes

DS supports a number of virtual attributes, which dynamically generate entry values that are not persisted in the token store. By default, the following DS virtual attributes are enabled at install:

- collective-attribute-subentries
- entity-tag
- entry-dn
- entry-uuid
- governing-structure-rule
- has-subordinates
- is-member-of
- num-subordinates
- password-expiration-time
- password-policy-subentry
- structural-object-class
- subschema-subentry

To improve CTS throughput, you can disable your virtual attributes using the **dsconfig** command, *except* for the Entity Tag virtual attribute, which must remain enabled else it will lead to an inoperable server.

To disable a virtual attribute, use the **dsconfig** command on DS to disable, for example, the Collective Attribute Subentries virtual attribute:

```
./dsconfig set-virtual-attribute-prop \
--name=collective-attribute-subentries-virtual-attribute \
--set enabled: false \
--hostname localhost \
--port 4444 \
--bindDN uid=admin \
--bindPassword password \
--no-prompt
```

Important

The DS `entity-tag-virtual-attribute` is required and should never be disabled.

OAuth 2.0 CTS Storage Scheme

AM 6.5 introduced a new scheme for storing OAuth 2.0 tokens in the CTS store, called the *grant-set* scheme.

The `grant-set` scheme groups multiple authorizations for a given OAuth 2.0 client and resource owner pair and stores them in a single CTS `OAUTH2_GRANT_SET` entry. This implementation reduces the size and quantity of entries stored, as well as the number of calls required to perform OAuth 2.0 operations.

The `one-to-one` scheme stores the state of multiple authorizations for a given OAuth 2.0 client and resource owner pair across multiple entries, and is more resource intensive. You should upgrade to the `grant-set` scheme once all the servers on your environment have been upgraded to AM 6.5 or later.

The `grant-set` scheme is backwards-compatible with existing entries stored in the CTS store. Therefore, any access or refresh token issued before configuring the `grant-set` scheme is still valid. Existing tokens will be retained in their original form until the refresh token expires or it is actively revoked.

Users will not notice any change in the tokens they receive, and there is no change to the OAuth 2.0 endpoints.

To enable the `grant-set` scheme, navigate to Configure > Global Services > OAuth2 Provider > Global Attributes and set the CTS Storage Scheme drop-down to Grant-Set Storage Scheme. Then, save your changes.

New OAuth 2.0 tokens stored in the CTS after the change will use the new scheme automatically.

Glossary

Access control	Control to grant or to deny access to a resource.
Account lockout	The act of making an account temporarily or permanently inactive after successive authentication failures.
Actions	Defined as part of policies, these verbs indicate what authorized identities can do to resources.
Advice	In the context of a policy decision denying access, a hint to the policy enforcement point about remedial action to take that could result in a decision allowing access.
Agent administrator	User having privileges only to read and write agent profile configuration information, typically created to delegate agent profile creation to the user installing a web or Java agent.
Agent authenticator	Entity with read-only access to multiple agent profiles defined in the same realm; allows an agent to read web service profiles.
Application	In general terms, a service exposing protected resources. In the context of AM policies, the application is a template that constrains the policies that govern access to protected resources. An application can have zero or more policies.
Application type	Application types act as templates for creating policy applications. Application types define a preset list of actions and functional logic, such as policy lookup and resource comparator logic.

	Application types also define the internal normalization, indexing logic, and comparator logic for applications.
Attribute-based access control (ABAC)	Access control that is based on attributes of a user, such as how old a user is or whether the user is a paying customer.
Authentication	The act of confirming the identity of a principal.
Authentication chaining	A series of authentication modules configured together which a principal must negotiate as configured in order to authenticate successfully.
Authentication level	Positive integer associated with an authentication module, usually used to require success with more stringent authentication measures when requesting resources requiring special protection.
Authentication module	AM authentication unit that handles one way of obtaining and verifying credentials.
Authorization	The act of determining whether to grant or to deny a principal access to a resource.
Authorization Server	In OAuth 2.0, issues access tokens to the client after authenticating a resource owner and confirming that the owner authorizes the client to access the protected resource. AM can play this role in the OAuth 2.0 authorization framework.
Auto-federation	Arrangement to federate a principal's identity automatically based on a common attribute value shared across the principal's profiles at different providers.
Bulk federation	Batch job permanently federating user profiles between a service provider and an identity provider based on a list of matched user identifiers that exist on both providers.
Circle of trust	Group of providers, including at least one identity provider, who have agreed to trust each other to participate in a SAML v2.0 provider federation.
Client	In OAuth 2.0, requests protected web resources on behalf of the resource owner given the owner's authorization. AM can play this role in the OAuth 2.0 authorization framework.
Client-based OAuth 2.0 tokens	After a successful OAuth 2.0 grant flow, AM returns a token to the client. This differs from CTS-based OAuth 2.0 tokens, where AM returns a *reference* to token to the client.
Client-based sessions	AM sessions for which AM returns session state to the client after each request, and require it to be passed in with the subsequent

	request. For browser-based clients, AM sets a cookie in the browser that contains the session information. For browser-based clients, AM sets a cookie in the browser that contains the session state. When the browser transmits the cookie back to AM, AM decodes the session state from the cookie.
Conditions	Defined as part of policies, these determine the circumstances under which which a policy applies. Environmental conditions reflect circumstances like the client IP address, time of day, how the subject authenticated, or the authentication level achieved. Subject conditions reflect characteristics of the subject like whether the subject authenticated, the identity of the subject, or claims in the subject's JWT.
Configuration datastore	LDAP directory service holding AM configuration data.
Cross-domain single sign-on (CDSSO)	AM capability allowing single sign-on across different DNS domains.
CTS-based OAuth 2.0 tokens	After a successful OAuth 2.0 grant flow, AM returns a *reference* to the token to the client, rather than the token itself. This differs from client-based OAuth 2.0 tokens, where AM returns the entire token to the client.
CTS-based sessions	AM sessions that reside in the Core Token Service's token store. CTS-based sessions might also be cached in memory on one or more AM servers. AM tracks these sessions in order to handle events like logout and timeout, to permit session constraints, and to notify applications involved in SSO when a session ends.
Delegation	Granting users administrative privileges with AM.
Entitlement	Decision that defines which resource names can and cannot be accessed for a given identity in the context of a particular application, which actions are allowed and which are denied, and any related advice and attributes.
Extended metadata	Federation configuration information specific to AM.
Extensible Access Control Markup Language (XACML)	Standard, XML-based access control policy language, including a processing model for making authorization decisions based on policies.
Federation	Standardized means for aggregating identities, sharing authentication and authorization data information between trusted providers, and

	allowing principals to access services across different providers without authenticating repeatedly.
Fedlet	Service provider application capable of participating in a circle of trust and allowing federation without installing all of AM on the service provider side; AM lets you create Java Fedlets.
Hot swappable	Refers to configuration properties for which changes can take effect without restarting the container where AM runs.
Identity	Set of data that uniquely describes a person or a thing such as a device or an application.
Identity federation	Linking of a principal's identity across multiple providers.
Identity provider (IDP)	Entity that produces assertions about a principal (such as how and when a principal authenticated, or that the principal's profile has a specified attribute value).
Identity repository	Data store holding user profiles and group information; different identity repositories can be defined for different realms.
Java agent	Java web application installed in a web container that acts as a policy enforcement point, filtering requests to other applications in the container with policies based on application resource URLs.
Metadata	Federation configuration information for a provider.
Policy	Set of rules that define who is granted access to a protected resource when, how, and under what conditions.
Policy agent	Java, web, or custom agent that intercepts requests for resources, directs principals to AM for authentication, and enforces policy decisions from AM.
Policy Administration Point (PAP)	Entity that manages and stores policy definitions.
Policy Decision Point (PDP)	Entity that evaluates access rights and then issues authorization decisions.
Policy Enforcement Point (PEP)	Entity that intercepts a request for a resource and then enforces policy decisions from a PDP.
Policy Information Point (PIP)	Entity that provides extra information, such as user profile attributes that a PDP needs in order to make a decision.
Principal	Represents an entity that has been authenticated (such as a user, a device, or an application), and thus is distinguished from other entities.

	When a Subject successfully authenticates, AM associates the Subject with the Principal.
Privilege	In the context of delegated administration, a set of administrative tasks that can be performed by specified identities in a given realm.
Provider federation	Agreement among providers to participate in a circle of trust.
Realm	AM unit for organizing configuration and identity information. Realms can be used for example when different parts of an organization have different applications and identity stores, and when different organizations use the same AM deployment. Administrators can delegate realm administration. The administrator assigns administrative privileges to users, allowing them to perform administrative tasks within the realm.
Resource	Something a user can access over the network such as a web page. Defined as part of policies, these can include wildcards in order to match multiple actual resources.
Resource owner	In OAuth 2.0, entity who can authorize access to protected web resources, such as an end user.
Resource server	In OAuth 2.0, server hosting protected web resources, capable of handling access tokens to respond to requests for such resources.
Response attributes	Defined as part of policies, these allow AM to return additional information in the form of "attributes" with the response to a policy decision.
Role based access control (RBAC)	Access control that is based on whether a user has been granted a set of permissions (a role).
Security Assertion Markup Language (SAML)	Standard, XML-based language for exchanging authentication and authorization data between identity providers and service providers.
Service provider (SP)	Entity that consumes assertions about a principal (and provides a service that the principal is trying to access).
Authentication Session	The interval while the user or entity is authenticating to AM.
Session	The interval that starts after the user has authenticated and ends when the user logs out, or when their session is terminated. For browser-based clients, AM manages user sessions across one or more applications by setting a session cookie. See also CTS-based sessions and Client-based sessions.

Session high availability	Capability that lets any AM server in a clustered deployment access shared, persistent information about users' sessions from the CTS token store. The user does not need to log in again unless the entire deployment goes down.
Session token	Unique identifier issued by AM after successful authentication. For a CTS-based sessions, the session token is used to track a principal's session.
Single log out (SLO)	Capability allowing a principal to end a session once, thereby ending her session across multiple applications.
Single sign-on (SSO)	Capability allowing a principal to authenticate once and gain access to multiple applications without authenticating again.
Site	Group of AM servers configured the same way, accessed through a load balancer layer. The load balancer handles failover to provide service-level availability. The load balancer can also be used to protect AM services.
Standard metadata	Standard federation configuration information that you can share with other access management software.
Stateless Service	Stateless services do not store any data locally to the service. When the service requires data to perform any action, it requests it from a data store. For example, a stateless authentication service stores session state for logged-in users in a database. This way, any server in the deployment can recover the session from the database and service requests for any user. All AM services are stateless unless otherwise specified. See also Client-based sessions and CTS-based sessions.
Subject	Entity that requests access to a resource When an identity successfully authenticates, AM associates the identity with the Principal that distinguishes it from other identities. An identity can be associated with multiple principals.
Identity store	Data storage service holding principals' profiles; underlying storage can be an LDAP directory service or a custom `IdRepo` implementation.
Web Agent	Native library installed in a web server that acts as a policy enforcement point with policies based on web page URLs.