

National Software Development Framework

Anar Framework Identity and Access Management Documentation

Big Data Directorate Team

Introduction	3
Single Sign On and Single Sign Out (SSO)	3
Kerberos bridge	3
Identity Brokering and Social Login	3
User Federation	4
Client Adapters	4
Gatekeeper	4
Admin Console	4
Account Management Console	4
Standard Protocols	5
Authorization Services	5
Ubiquitous Language	5
Resource Server	5
Resource	6
Scope	6
Permission	6
Policy	7
Policy Provider	7
Permission Ticket	7
Authorization Services	8
Architecture	10
Policy Administration Point (PAP)	10
Policy Decision Point (PDP)	10
Policy Enforcement Point (PEP)	11
Policy Information Point (PIP)	11
Installation	11
System Requirements	11
Installing Distribution Files	12
Distribution Directory Structure	12
bin/	12
domain/	12
modules/	13
providers/	13
standalone/	13
themes/	13

Choosing an Operating Mode	13
Standalone Mode	13
Domain Clustered Mode	14
Cross-Datacenter Replication Mode	15
Database	15
Infinispan caches	15
Authentication sessions	15
Action tokens	16
Caching and invalidation of persistent data	16
User sessions	16
Brute force protection	16
Relational Database Setup	17
RDBMS Setup Checklist	17
Server Cache Configuration	17
Conclusion	18
Reference	18



Introduction

Single Sign On (SSO), Single Sign Out, Identity Brokering, User Federation, Client Adapters to work with multiple programming languages, Multi-tenancy, Identity and Account Management, Flexible Authorization services, implementation of standard security protocols, and administration console are the vital element for Anar Framework's identity and access management service. To attain the required functionalities, we have relied on an open source solution named Keycloak. Keycloak is an open source Identity and Access Management solution aimed at modern applications and services. It makes it easy to secure applications and services with little to no code.

Single Sign On and Single Sign Out (SSO)

Users authenticate and authorize with Identity and Access Management (IAM) service rather than individual applications. This means that other applications don't have to deal with login forms, authenticating users, and storing users. Once logged-in to IAM service, users don't have to login again to access a different application.

This also applied to logout. IAM service provides single-sign out, which means users only have to logout once to be logged-out of all applications that use IAM service.

Kerberos bridge

If your users authenticate to workstations with Kerberos (LDAP or active directory) they can also be automatically authenticated to IAM service without having to provide their username and password again after they log on to the workstation.

Identity Brokering and Social Login

Enabling login with social networks is easy to add through the admin console. It's just a matter of selecting the social network you want to add. No code or changes to your application is required.

IAM service can also authenticate users with existing OpenID Connect or SAML 2.0 Identity Providers. Again, this is just a matter of configuring the Identity Provider through the admin console.



User Federation

IAM service has built-in support to connect to existing LDAP or Active Directory servers. You can also implement your own provider if you have users in other stores, such as a relational database.

Client Adapters

IAM service Client Adapters makes it really easy to secure applications and services. We have adapters available for a number of platforms and programming languages, but if there's not one available for your chosen platform don't worry. IAM service is built on standard protocols so you can use any OpenID Connect Resource Library or SAML 2.0 Service Provider library out there.

Gatekeeper

You can also opt to use a proxy to secure your applications which removes the need to modify your application at all.

Admin Console

Through the admin console administrators can centrally manage all aspects of the IAM server. They can enable and disable various features. They can configure identity brokering and user federation. They can create and manage applications and services, and define fine-grained authorization policies. They can also manage users, including permissions and sessions.

Account Management Console

Through the account management console users can manage their own accounts. They can update the profile, change passwords, and setup two-factor authentication. Users can also manage sessions as well as view history for the account. If you've enabled social login or identity brokering users can also link their accounts with additional providers to allow them to authenticate to the same account with different identity providers.



Standard Protocols

IAM server is based on standard protocols and provides support for OpenID Connect, OAuth 2.0, and SAML.

Authorization Services

If role based authorization doesn't cover your needs, IAM server provides fine-grained authorization services as well. This allows you to manage permissions for all your services from the admin console and gives you the power to define exactly the policies you need.

Ubiquitous Language

Before going further, it is important to understand these terms and concepts introduced by IAM Authorization Services.

Resource Server

Per OAuth2 terminology, a resource server is the server hosting the protected resources and capable of accepting and responding to protected resource requests.

Resource servers usually rely on some kind of information to decide whether access to a protected resource should be granted. For RESTful-based resource servers, that information is usually carried in a security token, typically sent as a bearer token along with every request to the server. Web applications that rely on a session to authenticate users usually store that information in the user's session and retrieve it from there for each request.

In IAM, any confidential client application can act as a resource server. This client's resources and their respective scopes are protected and governed by a set of authorization policies.



Resource

A resource is part of the assets of an application and the organization. It can be a set of one or more endpoints, a classic web resource such as an HTML page, and so on. In authorization policy terminology, a resource is the object being protected.

Every resource has a unique identifier that can represent a single resource or a set of resources. For instance, you can manage a Banking Account Resource that represents and defines a set of authorization policies for all banking accounts. But you can also have a different resource named Alice's Banking Account, which represents a single resource owned by a single customer, which can have its own set of authorization policies.

Scope

A resource's scope is a bounded extent of access that is possible to perform on a resource. In authorization policy terminology, a scope is one of the potentially many verbs that can logically apply to a resource.

It usually indicates what can be done with a given resource. Example of scopes are view, edit, delete, and so on. However, scope can also be related to specific information provided by a resource. In this case, you can have a project resource and a cost scope, where the cost scope is used to define specific policies and permissions for users to access a project's cost.

Permission

Consider this simple and very common permission:

A permission associates the object being protected with the policies that must be evaluated to determine whether access is granted.

- X CAN DO Y ON RESOURCE Z
 - X represents one or more users, roles, or groups, or a combination of them. You can also use claims and context here.
 - Y represents an action to be performed, for example, write, view, and so on.
 - Z represents a protected resource, for example, "/accounts".

IAM provides a rich platform for building a range of permission strategies ranging from simple to very complex, rule-based dynamic permissions. It provides flexibility and helps to:

- Reduce code refactoring and permission management costs
- Support a more flexible security model, helping you to easily adapt to changes in your security requirements
- Make changes at runtime; applications are only concerned about the resources and scopes being protected and not how they are protected.

Policy

A policy defines the conditions that must be satisfied to grant access to an object. Unlike permissions, you do not specify the object being protected but rather the conditions that must be satisfied for access to a given object (for example, resource, scope, or both). Policies are strongly related to the different access control mechanisms (ACMs) that you can use to protect your resources. With policies, you can implement strategies for attribute-based access control (ABAC), role-based access control (RBAC), context-based access control, or any combination of these.

IAM leverages the concept of policies and how you define them by providing the concept of aggregated policies, where you can build a "policy of policies" and still control the behavior of the evaluation. Instead of writing one large policy with all the conditions that must be satisfied for access to a given resource, the policies implementation in IAM Authorization Services follows the divide-and-conquer technique. That is, you can create individual policies, then reuse them with different permissions and build more complex policies by combining individual policies.


Policy Provider

Policy providers are implementations of specific policy types. IAM provides built-in policies, backed by their corresponding policy providers, and you can create your own policy types to support your specific requirements.

IAM provides a SPI (Service Provider Interface) that you can use to plug in your own policy provider implementations.

Permission Ticket

A permission ticket is a special type of token defined by the User-Managed Access (UMA) specification that provides an opaque structure whose form is determined by the authorization server. This structure represents the resources and/or scopes being requested by a client, the access context, as well as the policies that must be applied to a request for authorization data (requesting party token [RPT]).



In UMA, permission tickets are crucial to support person-to-person sharing and also person-to-organization sharing. Using permission tickets for authorization workflows enables a range of scenarios from simple to complex, where resource owners and resource servers have complete control over their resources based on fine-grained policies that govern the access to these resources.

In the UMA workflow, permission tickets are issued by the authorization server to a resource server, which returns the permission ticket to the client trying to access a protected resource. Once the client receives the ticket, it can make a request for an RPT (a final token holding authorization data) by sending the ticket back to the authorization server.


Authorization Services

IAM supports fine-grained authorization policies and is able to combine different access control mechanisms such as:

- Attribute-based access control (ABAC)
- Role-based access control (RBAC)
- User-based access control (UBAC)
- Context-based access control (CBAC)
- Rule-based access control
- Using JavaScript
- Using JBoss Drools
- Time-based access control
- Support for custom access control mechanisms (ACMs) through a Policy Provider Service Provider Interface (SPI)

IAM is based on a set of administrative UIs and a RESTful API, and provides the necessary means to create permissions for your protected resources and scopes, associate those permissions with authorization policies, and enforce authorization decisions in your applications and services.


Resource servers (applications or services serving protected resources) usually rely on some kind of information to decide if access should be granted to a protected resource. For RESTful-based resource servers, that information is usually obtained from a security token, usually sent as a bearer token on every request to the server. For web applications that rely on a session to authenticate users, that information is usually stored in a user's session and retrieved from there for each request.



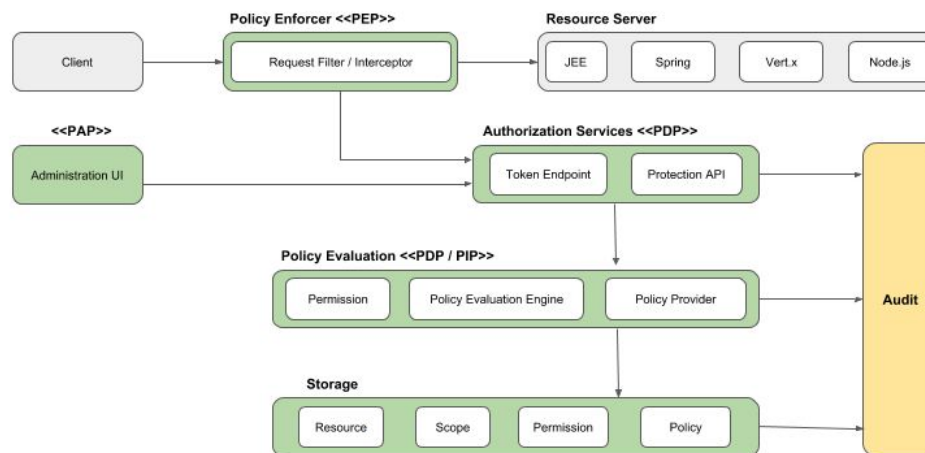
Frequently, resource servers only perform authorization decisions based on role-based access control (RBAC), where the roles granted to the user trying to access protected resources are checked against the roles mapped to these same resources. While roles are very useful and used by applications, they also have a few limitations:

- Resources and roles are tightly coupled and changes to roles (such as adding, removing, or changing an access context) can impact multiple resources
- Changes to your security requirements can imply deep changes to application code to reflect these changes
- Depending on your application size, role management might become difficult and error-prone
- It is not the most flexible access control mechanism. Roles do not represent who you are and lack contextual information. If you have been granted a role, you have at least some access.

Considering that today we need to consider heterogeneous environments where users are distributed across different regions, with different local policies, using different devices, and with a high demand for information sharing, Keycloak Authorization Services can help you improve the authorization capabilities of your applications and services by providing:

- Resource protection using fine-grained authorization policies and different access control mechanisms
 - Centralized Resource, Permission, and Policy Management
 - Centralized Policy Decision Point
 - REST security based on a set of REST-based authorization services
 - Authorization workflows and User-Managed Access
 - The infrastructure to help avoid code replication across projects (and redeploys) and quickly adapt to changes in your security requirements.
- 

Architecture



From a design perspective, Authorization Services is based on a well-defined set of authorization patterns providing these capabilities:

Policy Administration Point (PAP)

Provides a set of UIs based on the Administration Console to manage resource servers, resources, scopes, permissions, and policies. Part of this is also accomplished remotely through the use of the Protection API.

Policy Decision Point (PDP)

Provides a distributable policy decision point to where authorization requests are sent and policies are evaluated accordingly with the permissions being requested. For more information, see Obtaining Permissions.



Policy Enforcement Point (PEP)

Provides implementations for different environments to actually enforce authorization decisions at the resource server side. IAM provides some built-in Policy Enforcers.

Policy Information Point (PIP)

Being based on Authentication Server, you can obtain attributes from identities and runtime environment during the evaluation of authorization policies.

Installation

Installing Keycloak is as simple as downloading it and unzipping it. This section reviews system requirements as well as the directory structure of the distribution.

System Requirements

These are the requirements to run the Keycloak authentication server:

- Can run on any operating system that runs Java
- Java 8 JDK
- zip or gzip and tar
- At least 512M of RAM
- At least 1G of disk space
- A shared external database like PostgreSQL, MySQL, Oracle, etc. it requires an external shared database if you want to run in a cluster. Please see the database configuration section of this guide for more information.
- Network multicast support on your machine if you want to run in a cluster. it can be clustered without multicast, but this requires a bunch of configuration changes. Please see the clustering section of this guide for more information.
- On Linux, it is recommended to use `/dev/urandom` as a source of random data to prevent hanging due to lack of available entropy, unless `/dev/random` usage is mandated by your security policy. To achieve that on Oracle JDK 8 and OpenJDK 8, set the `java.security.egd` system property on startup to `file:/dev/urandom`.

Installing Distribution Files

The Keycloak Server has three downloadable distributions:

- 'keycloak-8.0.1.[ziptar.gz]'
- 'keycloak-overlay-8.0.1.[ziptar.gz]'
- 'keycloak-demo-8.0.1.[ziptar.gz]'

The '**keycloak-8.0.1.[ziptar.gz]**' file is the server only distribution. It contains nothing other than the scripts and binaries to run the Keycloak Server. To unpack this file just run your operating system's unzip or gunzip and tar utilities.

The '**keycloak-overlay-8.0.1.[ziptar.gz]**' file is a WildFly add-on that allows you to install Keycloak Server on top of an existing WildFly distribution. We do not support users that want to run their applications and Keycloak on the same server instance. To install the Keycloak Service Pack, just unzip it in the root directory of your WildFly distribution, open the bin directory in a shell and run `./jboss-cli.[sh|bat] --file=keycloak-install.cli`.

The '**keycloak-demo-8.0.1.[ziptar.gz]**' contains the server binaries, all documentation and all examples. It is preconfigured with both the OIDC and SAML client application adapters and can deploy any of the distribution examples out of the box with no configuration. This distribution is only recommended for those that want to test drive Keycloak. We do not support users that run the demo distribution in production.

To unpack these files run the unzip or gunzip and tar utilities.

Distribution Directory Structure

This section walks you through the directory structure of the server distribution.

bin/

This contains various scripts to either boot the server or perform some other management action on the server.

domain/

This contains configuration files and working directory when running IAM in domain mode.



modules/

These are all the Java libraries used by the server.

providers/

If you are writing extensions to keycloak, you can put your extensions here. See the Server Developer Guide for more information on this.

standalone/

This contains configuration files and working directory when running Keycloak in standalone mode.

themes/

This directory contains all the html, style sheets, JavaScript files, and images used to display any UI screen displayed by the server. Here you can modify an existing theme or create your own. See the Server Developer Guide for more information on this.

Choosing an Operating Mode

Before deploying Keycloak in a production environment you need to decide which type of operating mode you are going to use. Will you run Keycloak within a cluster? Do you want a centralized way to manage your server configurations? Your choice of operating mode effects how you configure databases, configure caching and even how you boot the server.

Standalone Mode

Standalone operating mode is only useful when you want to run one, and only one Keycloak server instance. It is not usable for clustered deployments and all caches are non-distributed and local-only. It is not recommended that you use standalone mode in production as you will have a single point of failure. If your standalone mode server goes down, users will not be able to log in. This mode is really only useful to test drive and play with the features of Keycloak



Domain Clustered Mode

Domain mode is a way to centrally manage and publish the configuration for your servers. Running a cluster in standard mode can quickly become aggravating as the cluster grows in size. Every time you need to make a configuration change, you have to perform it on each node in the cluster. Domain mode solves this problem by providing a central place to store and publish configuration. It can be quite complex to set up, but it is worth it in the end. This capability is built into the WildFly Application Server which IAM derives from.

Here are some of the basic concepts of running in domain mode.

domain controller

The domain controller is a process that is responsible for storing, managing, and publishing the general configuration for each node in the cluster. This process is the central point from which nodes in a cluster obtain their configuration.

host controller

The host controller is responsible for managing server instances on a specific machine. You configure it to run one or more server instances. The domain controller can also interact with the host controllers on each machine to manage the cluster. To reduce the number of running processes, a domain controller also acts as a host controller on the machine it runs on.


domain profile

A domain profile is a named set of configuration that can be used by a server to boot from. A domain controller can define multiple domain profiles that are consumed by different servers.

server group

A server group is a collection of servers. They are managed and configured as one. You can assign a domain profile to a server group and every service in that group will use that domain profile as their configuration.

In domain mode, a domain controller is started on a master node. The configuration for the cluster resides in the domain controller. Next a host controller is started on each machine in the cluster. Each host controller deployment configuration specifies how many Keycloak server instances will be started on that machine. When the host controller boots up, it starts as many



Keycloak server instances as it was configured to do. These server instances pull their configuration from the domain controller.

Cross-Datacenter Replication Mode

Cross-Datacenter Replication mode is for when you want to run Keycloak in a cluster across multiple data centers, most typically using data center sites that are in different geographic regions. When using this mode, each data center will have its own cluster of Keycloak servers.

Database

IAM uses a relational database management system (RDBMS) to persist some metadata about realms, clients, users, and so on.

Details of DB setup are out-of-scope for Keycloak, however many RDBMS vendors like MariaDB and Oracle offer replicated databases and synchronous replication. We test Keycloak with these vendors:

- Oracle Database 12c Release 1 (12.1) RAC
- Galera 3.12 cluster for MariaDB server version 10.1.19-MariaDB

Infinispan caches

This section begins with a high level description of the Infinispan caches. More details of the cache setup follow.

Authentication sessions

In Keycloak we have the concept of authentication sessions. There is a separate Infinispan cache called `authenticationSessions` used to save data during authentication of particular user. Requests from this cache usually involve only a browser and the Keycloak server, not the application. Here we can rely on sticky sessions and the `authenticationSessions` cache content does not need to be replicated across data centers, even if you are in Active/Active mode.



Action tokens

We also have the concept of action tokens, which are used typically for scenarios when the user needs to confirm an action asynchronously by email. For example, during the forget password flow the actionTokens Infinispan cache is used to track metadata about related action tokens, such as which action token was already used, so it can't be reused second time. This usually needs to be replicated across data centers.

Caching and invalidation of persistent data

IAM uses Infinispan to cache persistent data to avoid many unnecessary requests to the database. Caching improves performance, however it adds an additional challenge. When some IAM server updates any data, all other servers in all data centers need to be aware of it, so they invalidate particular data from their caches. IAM uses local Infinispan caches called realms, users, and authorization to cache persistent data.


We use a separate cache, work, which is replicated across all data centers. The work cache itself does not cache any real data. It is used only for sending invalidation messages between cluster nodes and data centers. In other words, when data is updated, such as the user john, the node sends the invalidation message to all other cluster nodes in the same data center and also to all other data centers. After receiving the invalidation notice, every node then invalidates the appropriate data from their local cache.

User sessions

There are Infinispan caches called sessions, clientSessions, offlineSessions, and offlineClientSessions, all of which usually need to be replicated across data centers. These caches are used to save data about user sessions, which are valid for the length of a user's browser session. The caches must handle the HTTP requests from the end user and from the application. As described above, sticky sessions can not be reliably used in this instance, but we still want to ensure that subsequent HTTP requests can see the latest data. For this reason, the data are usually replicated across data centers.

Brute force protection

Finally the loginFailures cache is used to track data about failed logins, such as how many times the user john entered a bad password. The details are described here. It is up to the admin whether this cache should be replicated across data centers. To have an accurate count of login failures, the replication is needed. On the other hand, not replicating this data can save some



performance. So if performance is more important than accurate counts of login failures, the replication can be avoided.

Relational Database Setup

IAM comes with its own embedded Java-based relational database called H2. This is the default database that IAM will use to persist data and really only exists so that you can run the authentication server out of the box. We highly recommend that you replace it with a more production ready external database. The H2 database is not very viable in high concurrency situations and should not be used in a cluster either.

IAM uses two layered technologies to persist its relational data. The bottom layered technology is JDBC. JDBC is a Java API that is used to connect to a RDBMS. There are different JDBC drivers per database type that are provided by your database vendor.

The top layered technology for persistence is Hibernate JPA. This is a object to relational mapping API that maps Java Objects to relational data. Most deployments of IAM will never have to touch the configuration aspects of Hibernate.


RDBMS Setup Checklist

These are the steps you will need to perform to get an RDBMS configured for IAM.

- Locate and download a JDBC driver for your database
- Package the driver JAR into a module and install this module into the server
- Declare the JDBC driver in the configuration profile of the server
- Modify the datasource configuration to use your database's JDBC driver
- Modify the datasource configuration to define the connection parameters to your database

Server Cache Configuration

IAM has two types of caches. One type of cache sits in front of the database to decrease load on the DB and to decrease overall response times by keeping data in memory. Realm, client, role, and user metadata is kept in this type of cache. This cache is a local cache. Local caches do not use replication even if you are in the cluster with more servers. Instead, they only keep copies



locally and if the entry is updated an invalidation message is sent to the rest of the cluster and the entry is evicted. There is separate replicated cache work, which task is to send the invalidation messages to the whole cluster about what entries should be evicted from local caches. This greatly reduces network traffic, makes things efficient, and avoids transmitting sensitive metadata over the wire.

The second type of cache handles managing user sessions, offline tokens, and keeping track of login failures so that the server can detect password phishing and other attacks. The data held in these caches is temporary, in memory only, but is possibly replicated across the cluster.

Conclusion

As the security, reliability, interoperability, transparency and extensibility are becoming vital for systems, we have relied on an open source identity and access management platform (Keycloak) to provide Single Sign On (SSO), Single Sign Out, Identity Brokering, User Federation, Client Adapters to work with multiple programming languages, Multi-tenancy, Identity and Account Management, Flexible Authorization services, implementation of standard security protocols, and administration console for applications which are built on top of Anar Framework.

Reference

<https://www.keycloak.org/documentation.html>

