



### Agent Customer Market Analyzer (ACMA)

**Name:** Agent Customer Market Analyzer

**Domain:** P&C (Property and Casualty) Insurance

**Module:** Super Admin

#### Introducing Keycloak

Keycloak is an open source Identity and Access Management tool with a focus on modern applications such as single-page applications(SPA's), mobile applications, and REST APIs.

Keycloak is a lightweight and easy-to-install solution. It is highly scalable and provides high availability through clustering capabilities and highly customizable, extendable as when needed.

Keycloak has a large number of extension points where we can implement and deploy custom code to Keycloak to modify existing behavior or add completely new capabilities.

Examples of extensions that can be written to Keycloak include custom authentication mechanisms, integrations with custom user stores, and the custom manipulation of tokens. we can even implement our own custom login protocols.

The project was started in 2014 with a strong focus on making it easier for developers to secure their applications. It has since grown into a well-established open source project with a strong community and user base.

It is used in production for scenarios ranging from small webapplications with only a handful of users up to large enterprises with millions of users.

Keycloak provides fully customizable login pages, including support for strong authentication, and built-in capabilities such as the recovery of passwords, requiring users to regularly update their passwords, accepting terms and conditions, and a lot more.

All of this without any need to add anything to our applications, or any coding at all. All pages visible to our users support custom themes, making it very easy to modify the look and feel of the pages to integrate with our corporate branding and existing applications.

By delegating authentication to Keycloak, our applications do not need to worry about different authentication mechanisms, or how to safely store passwords. This approach also provides a higher level of security as applications do not have direct access to user credentials; they are instead provided with security tokens that give them only access to what they need.

Keycloak supports a wide range of authentication factors, allowing us to easily enable Multi-Factor Authentication (MFA) and Strong Authentication (SA) for our applications.

Authenticating Users, with only a few steps, we are able to choose from authenticating our users using OTPs, security devices and WebAuthn, passwords, or any combination of these.

## JAVA Real time Project Using Microservices Architecture and React JS

Keycloak provides single sign-on as well as session management capabilities, allowing users to access multiple applications, while only having to authenticate once. Both users themselves and administrators have full visibility in to where users are authenticated, and can **remotely terminate sessions** when required.

Keycloak builds on industry-standard protocols supporting **OAuth 2.0, OpenID Connect, and SAML 2.0**. Using industry-standard protocols is important from both a security perspective and in terms of making it easier to integrate with existing and new applications.

Keycloak comes with its own user database, which makes it very easy to get started. We can also easily integrate with existing identity infrastructure. Through its **identity brokering capabilities**, we can plug in existing user bases from social networks, or other enterprise identity providers. This is called as JIT Provisioning. It can also integrate with existing user directories, such as Active Directory and LDAP servers.

### **Installing and running Keycloak**

Keycloak provides a few options on how it can be installed, including the following:

- Running as a container on Docker
- Installing and running Keycloak locally (which will require a Java virtual machine, such as OpenJDK)
- Running Keycloak on Kubernetes
- Using the Keycloak Kubernetes Operator

### **Installing and Running Keycloak as docker Container**

If we already have Docker installed on our workstation, this is the recommended approach as it is simpler to get up and running as a container.

If we don't have Docker installed, it is easier to get started by installing and running it locally. The only dependency required is a Java virtual machine.

Keycloak can also be easily deployed to Kubernetes, where we have the option of using the Keycloak Kubernetes Operator, which makes installation, configuration, and management even simpler.

### **Running Keycloak on Docker**

With Docker, it is very easy to run Keycloak as we don't need to install a Java virtual machine. To run Keycloak on Docker, simply execute the following command:

```
docker run -e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=admin -p 8080:8080
quay.io/keycloak/keycloak start-dev
```

## JAVA Real time Project Using Microservices Architecture and React JS

```
C:\Users\narsi>docker run -e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=admin -p 8080:8080 quay.io/keycloak/keycloak start-dev
Unable to find image 'quay.io/keycloak/keycloak:latest' locally
latest: Pulling from keycloak/keycloak
5f328c14e09d: Pull complete
fb2206f30d41: Pull complete
2fc1c8dd8426: Pull complete
09489a99764b: Pull complete
Digest: sha256:bf788a3b7fd737143f98d4cb514cb9599c896acee01a26b2117a10bd99e23e11
Status: Downloaded newer image for quay.io/keycloak/keycloak:latest
Updating the configuration and installing your custom providers, if any. Please wait.
2024-09-03 00:58:52,234 INFO [io.qua.dep.QuarkusAugmentor] (main) Quarkus augmentation completed in 8946ms
2024-09-03 00:58:57,708 INFO [org.infinispan.CONTAINER] (ForkJoinPool.commonPool-worker-1) ISPN000556: Starting user marshaller 'org
.infinispan.jboss.marshalling.core.JBossUserMarshaller'
2024-09-03 00:58:58,531 INFO [org.keycloak.broker.provider.AbstractIdentityProviderMapper] (main) Registering class org.keycloak.bro
ker.provider.mappersync.ConfigSyncEventListener
2024-09-03 00:58:58,564 INFO [org.keycloak.connections.infinispan.DefaultInfinispanConnectionProviderFactory] (main) Node name: node
_521552, Site name: null
2024-09-03 00:58:59,676 INFO [org.keycloak.quarkus.runtime.storage.legacy.liquibase.QuarkusJpaUpdaterProvider] (main) Initializing d
atabase schema. Using changelog META-INF/jpa-changelog-master.xml

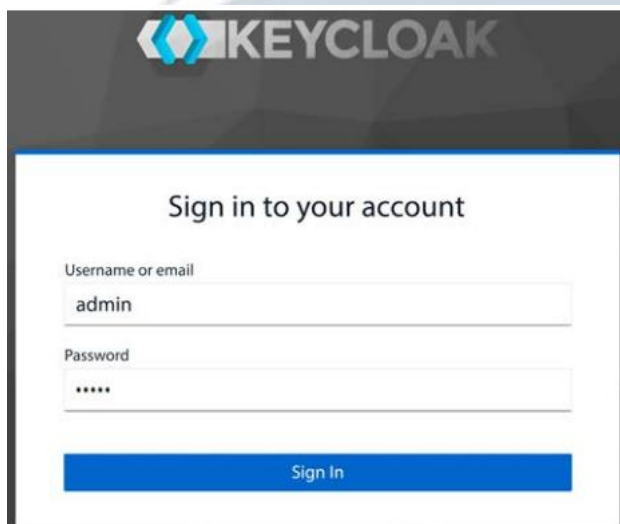
UPDATE SUMMARY
Run: 134
Previously run: 0
Filtered out: 0
-----
Total change sets: 134

2024-09-03 00:59:03,086 INFO [org.keycloak.services] (main) KC-SERVICES0050: Initializing master realm
2024-09-03 00:59:05,470 INFO [org.keycloak.services] (main) KC-SERVICES0009: Added user 'admin' to realm 'master'
2024-09-03 00:59:05,621 INFO [io.quarkus] (main) Keycloak 25.0.4 on JVM (powered by Quarkus 3.8.5) started in 13.242s. Listening on:
http://0.0.0.0:8080. Management interface listening on http://0.0.0.0:9000.
2024-09-03 00:59:05,622 INFO [io.quarkus] (main) Profile dev activated.
2024-09-03 00:59:05,622 INFO [io.quarkus] (main) Installed features: [agroal, cdi, hibernate-orm, jdbc-h2, keycloak, logging-gelf, n
```

As Keycloak does not ship with a default admin account, passing the environment variables, KEYCLOAK\_ADMIN and KEYCLOAK\_ADMIN\_PASSWORD, makes it easy to create an initial admin account.

We are also using -p 8080:8080 to publish the port used by Keycloak to the host, so as to make it easy to access Keycloak.

After a few seconds, we can verify that Keycloak is running by opening <http://localhost:8080/admin> and log in with the username admin and password admin.

The image shows the Keycloak login page. At the top, there is a Keycloak logo. Below it, the text "Sign in to your account" is displayed. There are two input fields: "Username or email" with the value "admin" and "Password" with masked characters "\*\*\*\*\*". A blue "Sign In" button is located at the bottom of the form.

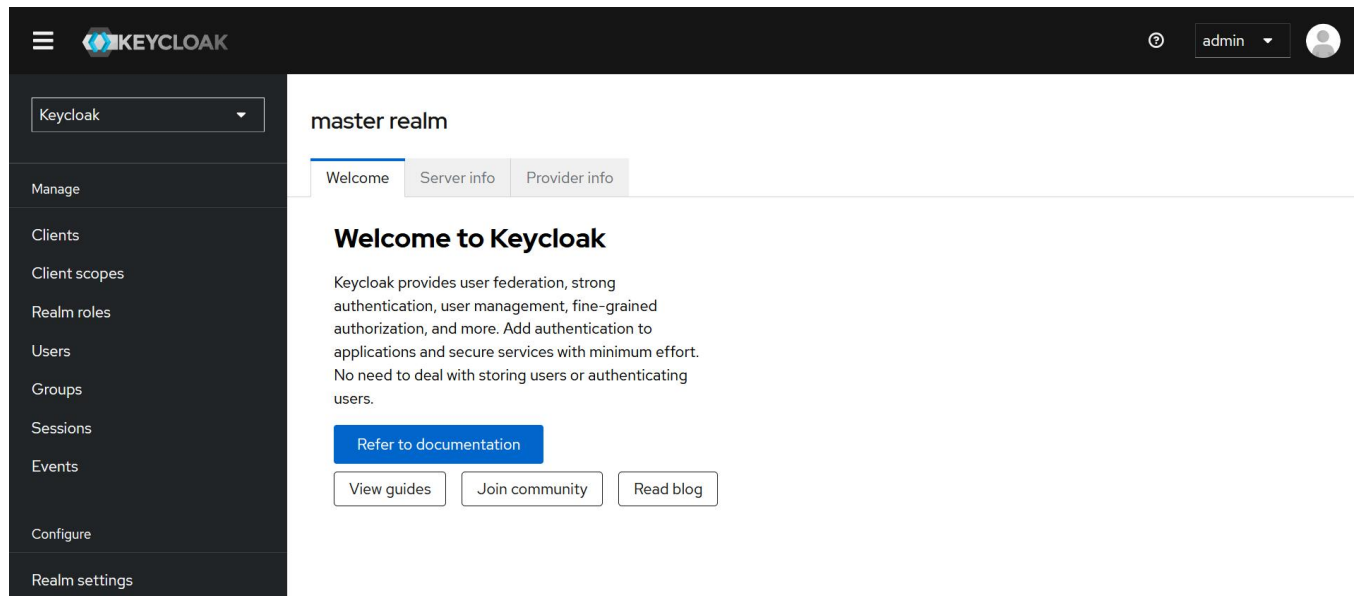
## JAVA Real time Project Using Microservices Architecture and React JS

### Getting started with the Keycloak admin console

The Keycloak admin console provides an extensive and friendly interface for administrators and developers to configure and manage Keycloak.

To access the admin console, open <http://localhost:8080/admin> in a browser. We will be redirected to the Keycloak login page, where we can log in with the admin username and password, we created while installing Keycloak.

Once we have logged in, we will see the configuration for the master realm in Keycloak, as shown in the following screenshot:



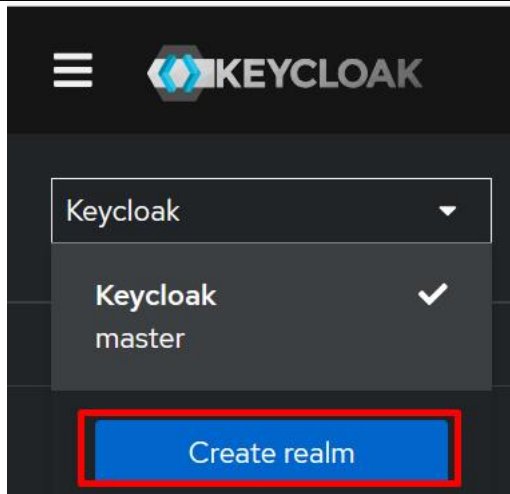
### Creating and configuring a realm

The first thing we want to do is create a realm for our applications and users. Think of a realm as a tenant. A realm is fully isolated from other realms; it has its own configuration, and its own set of applications and users.

This allows a single installation of Keycloak to be used for multiple purposes. For example, we may want to have one realm for internal applications and employees, and another realm for external applications and customers.

To create a new realm, click on the menu icon in the top-left corner (on the left side of the Keycloak logo) to expand the menu. Now, click on the realm selector to see a list of realms, including a button to create a new realm. Click on the Create Realm button:

## JAVA Real time Project Using Microservices Architecture and React JS



On the next page, enter a name for the realm. As the name is used in URLs, the name should ideally not use special characters that need escaping in URLs (such as spaces).

Once created, we can set a human-friendly display name. For example, use **acma** for the name, and acma for the display name.

### Creating a user

Once we have created the realm, let's create the first user in the realm:

1. From the left-hand menu, click on Users, and then click on Create new user button.
2. Enter a memorable username, and also enter a value of your choice for email, first name, and last name.
3. The Email Verified option can be selected by an administrator if they know this is the valid email address for the user.
4. Required User Actions allows an administrator to require a user to perform some initial actions on the next login; for example, to require the user to review their profile, or to verify their email address.
5. Remember to click on Create after you have completed the form:



## JAVA Real time Project Using Microservices Architecture and React JS

Users > Create user

### Create user

Required user actions ⓘ

Username \*

Email

Email verified ⓘ ☐ No

First name

Last name

Groups ⓘ

A user has a few standard built-in attributes, such as First name, but it is also possible to add any custom attributes through the Attributes tab.

Before the user can log in, we have to create an initial temporary password. To do this, click on the Credentials tab. In this tab, click on the Set Password button and follow the instructions to set a new password to the user.

If the Temporary option is enabled, the user will be required to change their password when logging in for the first time.

### Creating a group

Next, let's create a group and add the user we previously created to the group. From the menu on the left-hand side, click on Groups, and then click on the Create group button.

Enter a name for the group, for example, mygroup, and then click on Create. Once we have created the group, we can see the mygroup group in the group list. By clicking on it, you can edit the group settings.

For instance, we can add attributes to the group. A user inherits all the attributes from a group it belongs to.

This can be useful if, for example, we have a group for all employees in an office and want to add the office address to all employees in this group. We can also grant roles to a group, which again are inherited by all members of the group.

To add the user to the group, go back to the Users page and select the user you created previously. Next, click on the Groups tab. In this tab, click Join Group, select the group you created previously, and click on Join to add the user to the group.

Note:

All the Soft Copy of the Project Implementation Guides, Materials and class recordings will be uploaded to TechhubVault app.

