Building an Event Listener SPI (Plugin) for KeyCloak

Adwait Thattey · Follow
7 min read · Dec 22, 2019

In this blog, I will talk about how to build an event listener plugin (called an SPI) for KeyCloak

So, what is **Keycloak**?

Keycloak is an Open Source Identity and Access Management Framework built by **RedHat**. It provides a lot of advanced features like SSO, Social Auth, support for multiple auth protocols, etc. Read more here: https://www.keycloak.org/

But one of the most important features is the ability to extend any functionality of KeyCloak by simply building a plugin.

During my internship this summer, we needed to log all the events of users (and admins) happening within KeyCloak and send them to external systems for analysis. This is needed in many situations. *(one example is if you are using an external SIEM to log and analyze incidents)*.

By default, KeyCloak logs don't contain user/admin events. And even if we enable that, it would be difficult to build an external system which monitors and parses the logs to extract required events. **Instead, we can build a plugin for KeyCloak to hook into the system and do *"something"* whenever an event occurs *(In our case, fire external API calls)***

So, let's build one :)

> A quick note: The entire SPI (plugin) is available in this repository:

adwait-thattey/keycloak-event-listener-spi

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or...

github.com

I would be using Maven here for managing dependencies and building project.

So let's get the pom.xml sorted out first.

*(If you are not familiar with Maven, we use a **pom.xml** file in Maven to list all the project details including all the dependencies)*

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4
    <parent>
        <groupId>org.keycloak</groupId>
        <artifactId>keycloak-parent</artifactId>
        <version>6.0.1</version>
    </parent>

    <name>Sample Event Listener</name>
    <description/>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>sample_event_listener</artifactId>
    <packaging>jar</packaging>

    <dependencies>
        <dependency>
            <groupId>org.keycloak</groupId>
            <artifactId>keycloak-core</artifactId>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.keycloak</groupId>
            <artifactId>keycloak-server-spi</artifactId>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.keycloak</groupId>
            <artifactId>keycloak-server-spi-private</artifactId>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.keycloak</groupId>
            <artifactId>keycloak-services</artifactId>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.keycloak</groupId>
            <artifactId>keycloak-saml-core-public</artifactId>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.jboss.logging</groupId>
            <artifactId>jboss-logging</artifactId>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.jboss.spec.javax.ws.rs</groupId>
```

```
            <artifactId>jboss-jaxrs-api_2.1_spec</artifactId>
        </dependency>
    </dependencies>

    <build>
        <finalName>sample-event-listener</finalName>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.wildfly.plugins</groupId>
                <artifactId>wildfly-maven-plugin</artifactId>
                <configuration>
                    <skip>false</skip>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

(if the above gist is not visible, you can find the file here)

In `pom.xml`, we define the parent details, project name `Sample Event Listener`), version, artifact-id (here `sample_event_listener`), dependencies and build configuration.

. . .

The next step is to implement the SPI. For this, we need to implement 2 classes. `Provider` and `ProviderFactory`

so let's create our package in `src/main/java`.
Here the package name is `com.coderdude.sampleeventlistenerprovider.provider`

`coderdude` : because my dev alias is coderdude :D

`sampleeventlistenerprovider` : Could be shorter but let's leave it at that

`provider` : The last provider is there because there can potentially be other modules that you use in your provider.

Now this package is going to contain the 2 above discussed classes.
The `Provider` class contains the actual logic of the plugin. The `ProviderFactory` is a wrapper that initializes the provider. **The difference is important.**

- The `Factory` is initialized only when KeyCloak is started. A new instance of `Provider` is created by `Factory` every time required. *(In our case every time an event occurs)*
- Only 1 instance of `Factory` will exist. Multiple providers can exist at the same time (say 2 events occur at the same time).
- Providers are destroyed as soon as they complete their tasks. The Factory exists as long as KeyCloak is running.
- Any error in Factory will crash KeyCloak. An error in Provider will simply go to the logs and rest of Keycloak will function normally

So let's start by creating a Provider.
The name of the class will be `SampleEventListenerProvider` which implements the `EventListenerProvider` interface *(This interface is provided by KeyCloak)*

```
package com.coderdude.sampleeventlistenerprovider.provider;

import org.keycloak.events.Event;
import org.keycloak.events.EventListenerProvider;
import org.keycloak.events.admin.AdminEvent;

import java.util.Map;


public class SampleEventListenerProvider implements
EventListenerProvider {

    public SampleEventListenerProvider() {
    }


}
```

Keep these imports for now. We will need them.

So here, we are just going to print all the events to the console. All events are provided by 2 classes: `org.keycloak.events.Event` and `org.keycloak.events.admin.AdminEvent`

The normal events occur whenever a normal user does something. Admin events occur when administrators do something.

We need to write appropriate methods to convert these class objects to readable strings.

Here is the method to build string for an `Event`

We are capturing all the parameters, errors and details. (hence the map, because the details is an array)

```
private String toString(Event event) {
StringBuilder sb = new StringBuilder();
sb.append("type=");
sb.append(event.getType());
sb.append(", realmId=");
sb.append(event.getRealmId());
sb.append(", clientId=");
sb.append(event.getClientId());
sb.append(", userId=");
sb.append(event.getUserId());
sb.append(", ipAddress=");
sb.append(event.getIpAddress());
if (event.getError() != null) {
sb.append(", error=");
sb.append(event.getError());
}
if (event.getDetails() != null) {
for (Map.Entry<String, String> e : event.getDetails().entrySet()) {
sb.append(", ");
sb.append(e.getKey());
```

```
if (e.getValue() == null || e.getValue().indexOf(' ') == -1) {
sb.append("=");
sb.append(e.getValue());
} else {
sb.append("='");
sb.append(e.getValue());
sb.append("'");
}
}
}
return sb.toString();
}
```

Of course, this is a very naive implementation. What we actually did was define methods to wrap these events in other objects and make API calls to external systems. But this will work for now.
We can build a similar method for `AdminEvent`. You will find it in the main full code.

Once this is done, we need to override 2 methods provided by the `EventListenerProvider` interface. These are `onEvent` and `close`.

Here it is

The `onEvent` is the actual method called whenever an event occurs. We need to overload onEvent twice to capture both `Event` and `AdminEvent`.
Finally, the `close` method is called just before the class is destroyed. Sort of like a destructor. We need to override it even if we don't need to use it.

You can find the full class code (along with string implementation for AdminEvent) here

. . .

Next step is to implement the `ProviderFactory`
The name of the class is `SampleEventListenerProviderFactory` which implements

`EventListenerProviderFactory`

Here is the code:

```java
package com.coderdude.sampleeventlistenerprovider.provider;

import org.keycloak.Config;
import org.keycloak.events.EventListenerProvider;
import org.keycloak.events.EventListenerProviderFactory;
import org.keycloak.models.KeycloakSession;
import org.keycloak.models.KeycloakSessionFactory;



public class SampleEventListenerProviderFactory implements EventListenerProviderFactory {


    @Override
    public EventListenerProvider create(KeycloakSession keycloakSession) {

        return new SampleEventListenerProvider();
    }

    @Override
    public void init(Config.Scope scope) {

    }

    @Override
    public void postInit(KeycloakSessionFactory keycloakSessionFactory) {

    }

    @Override
    public void close() {

    }

    @Override
    public String getId() {
        return "sample_event_listener";
    }
}

```

SampleEventListenerProviderFactory.java hosted with ❤ by GitHub view raw

(if the above gist is not visible, you can find the file here)

We override multiple methods here. The main ones are the `create` and `getId`. The create method should initialize and return an instance of provider (in our case `SampleEventListenerProvider`). The `getId` should return a string with the name of the plugin

. . .

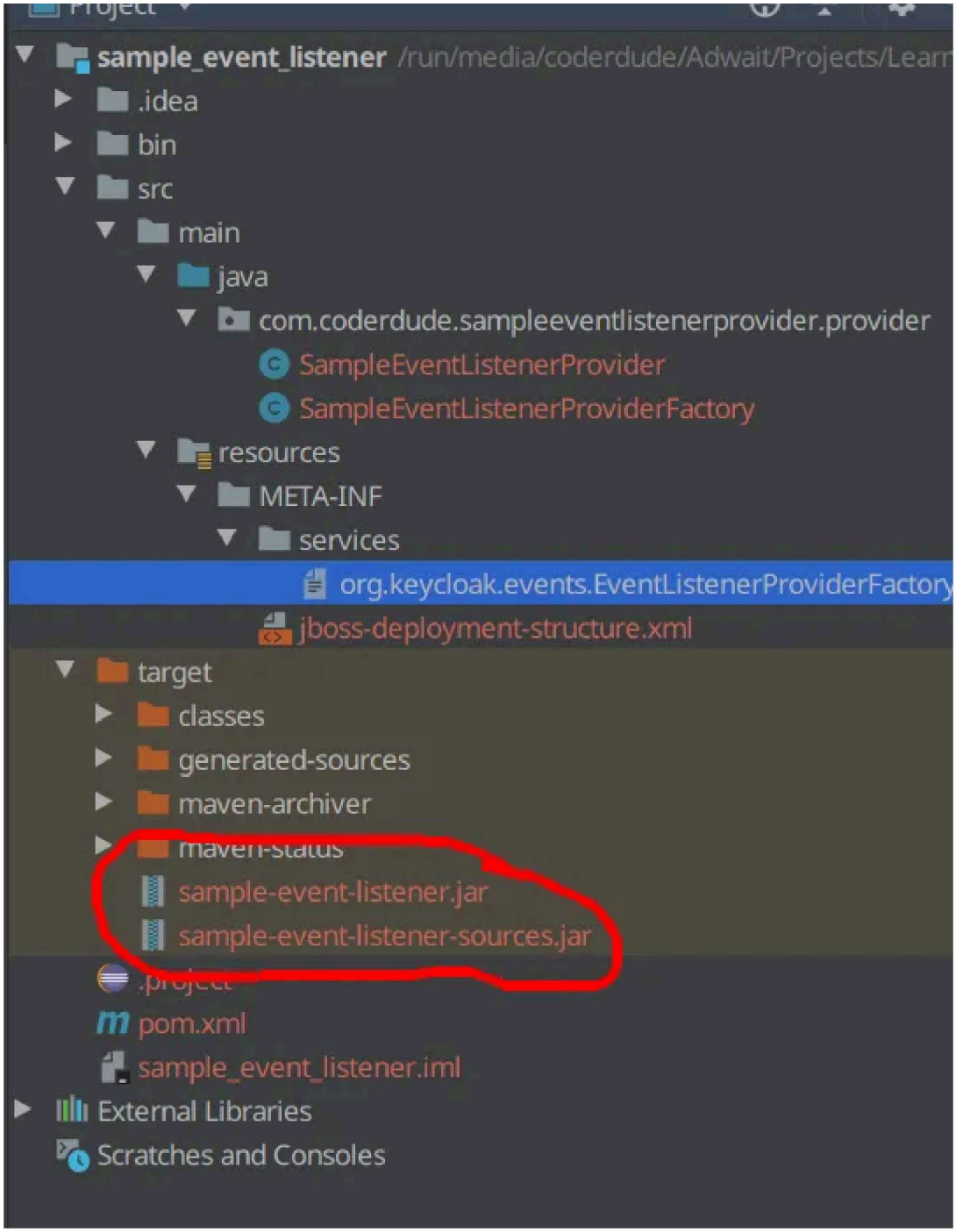
The next and the final task is to provide a link to our class. For this we need to create resources.
create a folder named `resources` in `src/main` (alongside `java` folder)
Now create the following file in `resources/META-INF/services/` named `org.keycloak.events.EventListenerProviderFactory` . Note that full path to location of file is `src/main/resources/META-INF/services/org.keycloak.events.EventListenerProviderFactory`

This file just contains one line with the package and name of our factory class

```
com.coderdude.sampleeventlistenerprovider.provider.SampleEventListen
erProviderFactory
```

That's it. We have written the plugin. Now let's build and package it.
I have used maven to build and package

Once packaging is complete, you should see the `jar` and `sources` in the target directory
Here is my final directory structure



Directory Structure

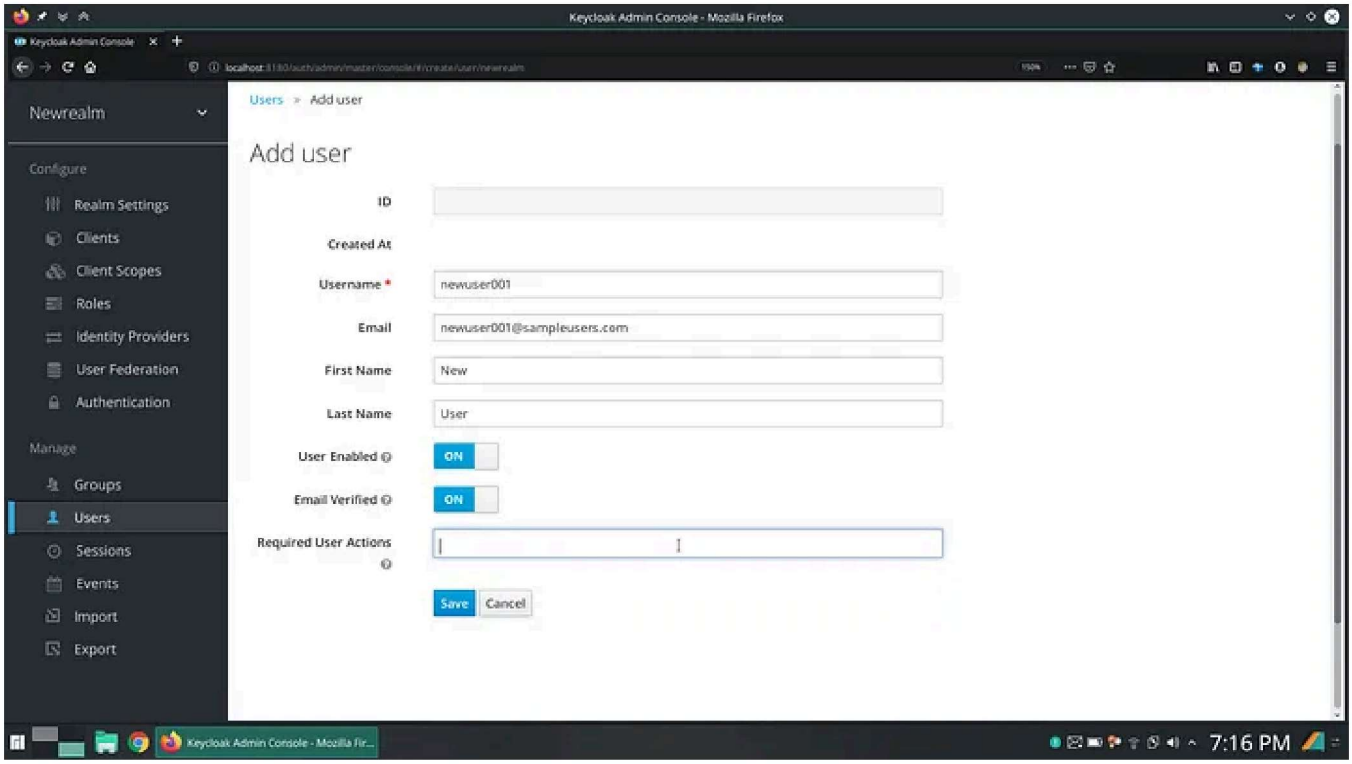
> *We will only need the* `sample-event-listener.jar`

. . .

Now it's time to deploy the plugin to KeyCloak.

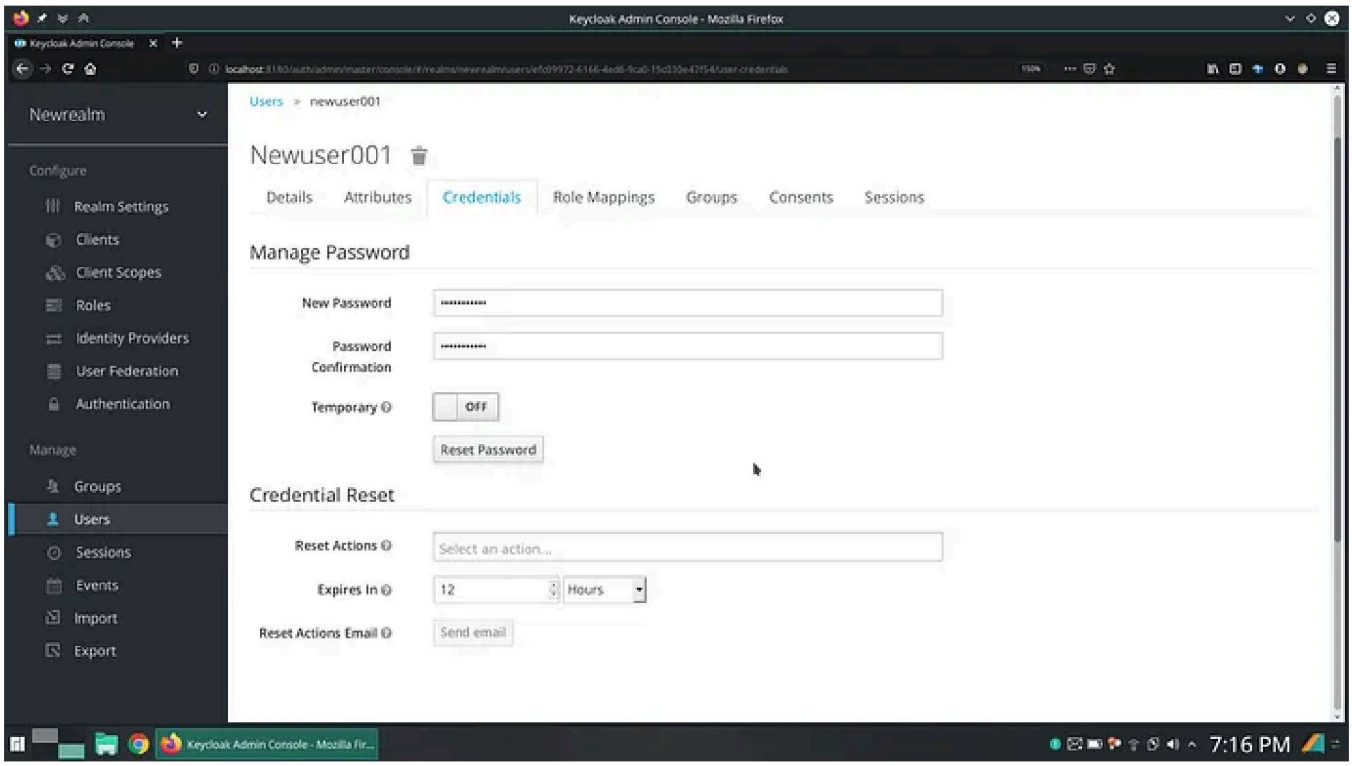
Let's get setup with a KeyCloak first. You will find the getting started guide here https://www.keycloak.org/docs/latest/getting_started/index.html.

Quickly download and create an admin user and login to KeyCloak.
Now let's create a new realm named `newrealm` and add a user named `newuser001` in the new realm.



Ad user in newly created realm in keycloak

Let's also create a password for this new user



Set password for user

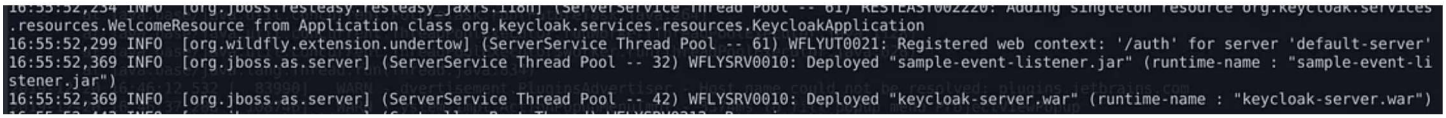
It's time to deploy our awesome plugin

The deployment process is pretty straightforward. We need to copy the `sample-event-listener.jar` to `$KEYCLOAK_DIR/standalone/deployments/` where $KEYCLOAK_DIR is the main KeyCloak directory (after unzipping)

KeyCloak supports hot-reloading. So as soon we copy the jar file, keycloak should reload and deploy the plugin. But just to be sure, let's restart the Keycloak server.

You should see a line like this

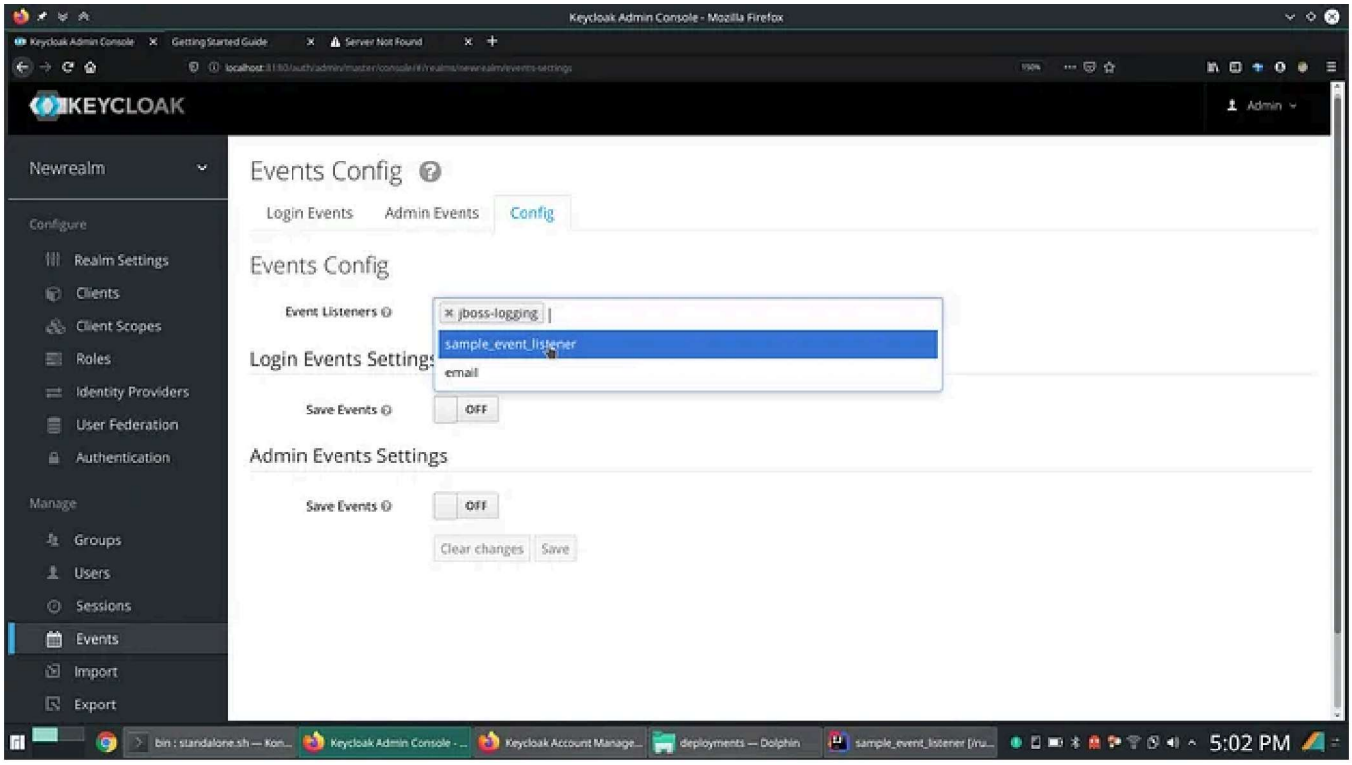
```
Deployed "sample-event-listener.jar" (runtime-name : "sample-event-
listener.jar")
```



Looks like this on my system

Now we need to allow this plugin to listen to events.
Go to `newrealm->manage->events->config` or this url `/auth/admin/master/console/#/realms/newrealm/events-settings` **Make sure to replace newrealm with the name of the realm you created**

In the config, event-listeners, add `sample_event_listener` to the list and hit save.



Adding our plugin to event listeners

Now our plugin should be able to capture all events.

. . .

Lets test this

Login to the newrealm using the user that was created above.

You should see an event occuring in the console

```
17:03:01,797 INFO  [stdout] (default task-5) Event
Occurred:type=LOGIN, realmId=newrealm, clientId=account,
userId=efc09972-6166-4ed6-9ca0-15c030e47f54, ipAddress=127.0.0.1,
auth_method=openid-connect, auth_type=code,
redirect_uri=http://localhost:8180/auth/realms/newrealm/account/logi
n-redirect, consent=no_consent_required, code_id=78db58ed-3c99-4d42-
aced-b69873c59f12, username=newuser001
```

Logout should also be captured

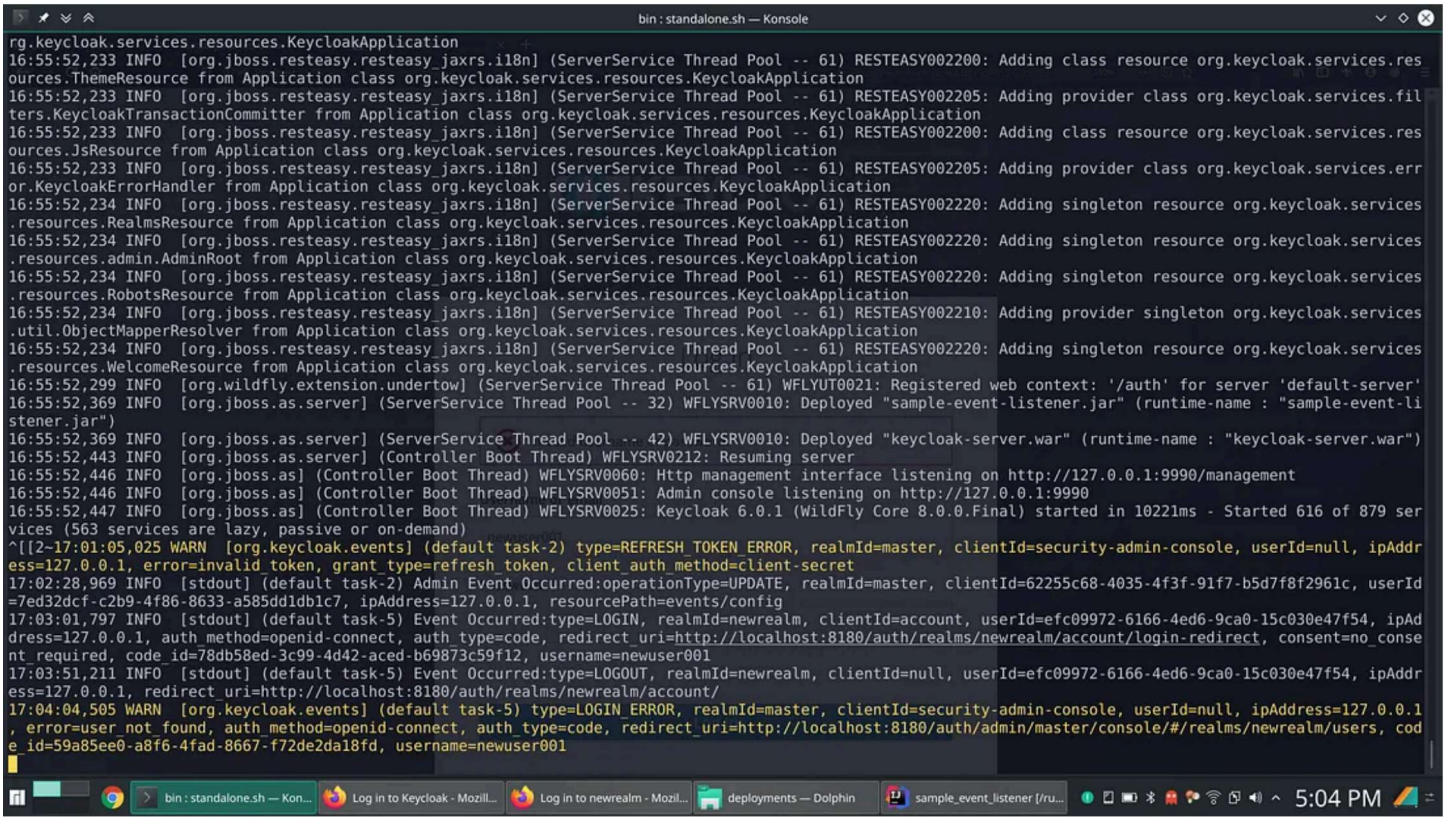
```
17:03:51,211 INFO  [stdout] (default task-5) Event
Occurred:type=LOGOUT, realmId=newrealm, clientId=null,
```

```
userId=efc09972-6166-4ed6-9ca0-15c030e47f54, ipAddress=127.0.0.1,
redirect_uri=http://localhost:8180/auth/realms/newrealm/account/
```

Trying to login with incorrect password is also captured (because we were also capturing errors)

```
17:04:04,505 WARN  [org.keycloak.events] (default task-5)
type=LOGIN_ERROR, realmId=master, clientId=security-admin-console,
userId=null, ipAddress=127.0.0.1, error=user_not_found,
auth_method=openid-connect, auth_type=code,
redirect_uri=http://localhost:8180/auth/admin/master/console/#/realm
s/newrealm/users, code_id=59a85ee0-a8f6-4fad-8667-f72de2da18fd,
username=newuser001
```

Looks like this in my console



Console log of my KeyCloak

Voilà! Our plugin is able to capture events

. . .

Wrapping Up:

Once again, the entire code is available here:

adwait-thattey/keycloak-event-listener-spi

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or...

github.com

This is a very basic example. We can do lots more. There is a lot more useful information that keycloak events provide that can be captured. Like current realm, ip address of the person trying to login, access token IDs if it is an api login, etc.

Bye!

Adwait Thattey,

https://adwait-thattey.github.io/

Java Keycloak Single Sign On Event Listener Keycloak Spi

Follow

Written by Adwait Thattey

5 Followers

Recommended from Medium

Wesley Matlock

Mastering Dependencies: A Guide to Using Swift Package Manager

Introduction

Jun 20 6 1

Bhuvanesh Kamaraj

Implementing MFA using SMS OTP in Keycloak

In this document, we will outline the steps to implement Multi-Factor Authentication (MFA) using SMS One-Time Password (OTP) in Keycloak...

Feb 29 3 3

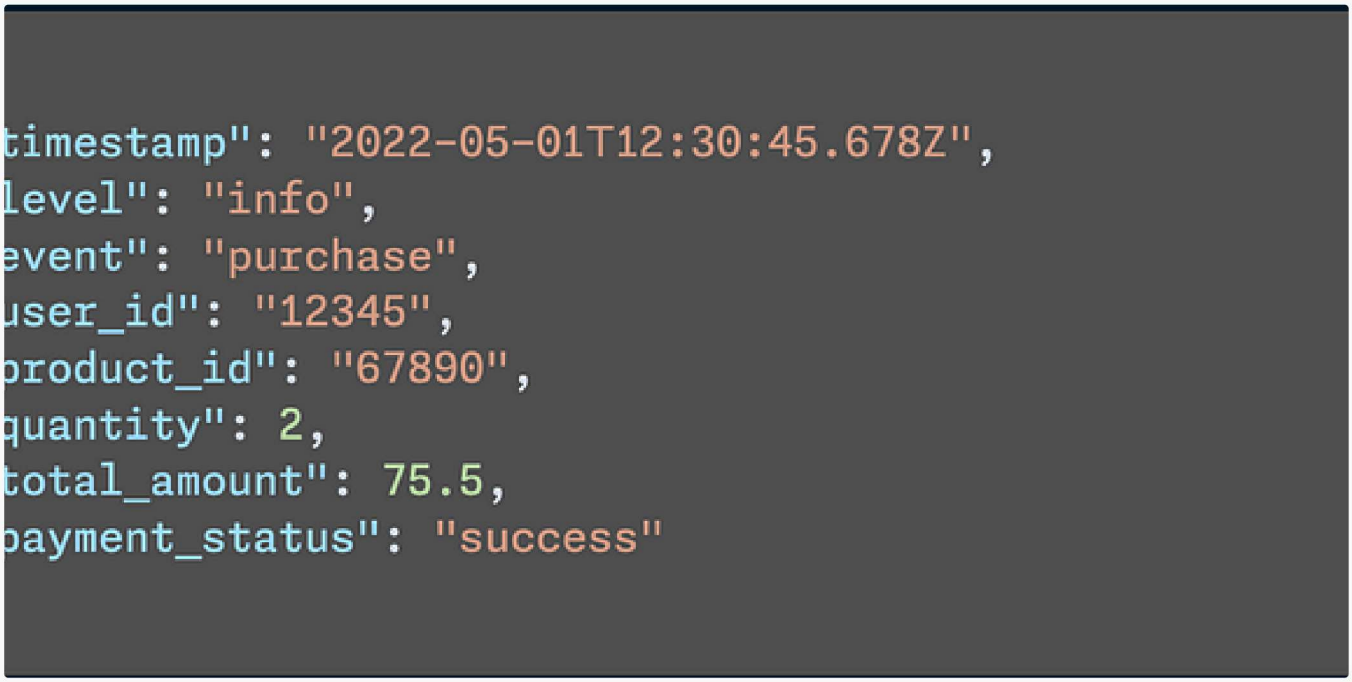
Lists

General Coding Knowledge

20 stories · 1346 saves

data science and AI

40 stories · 196 saves

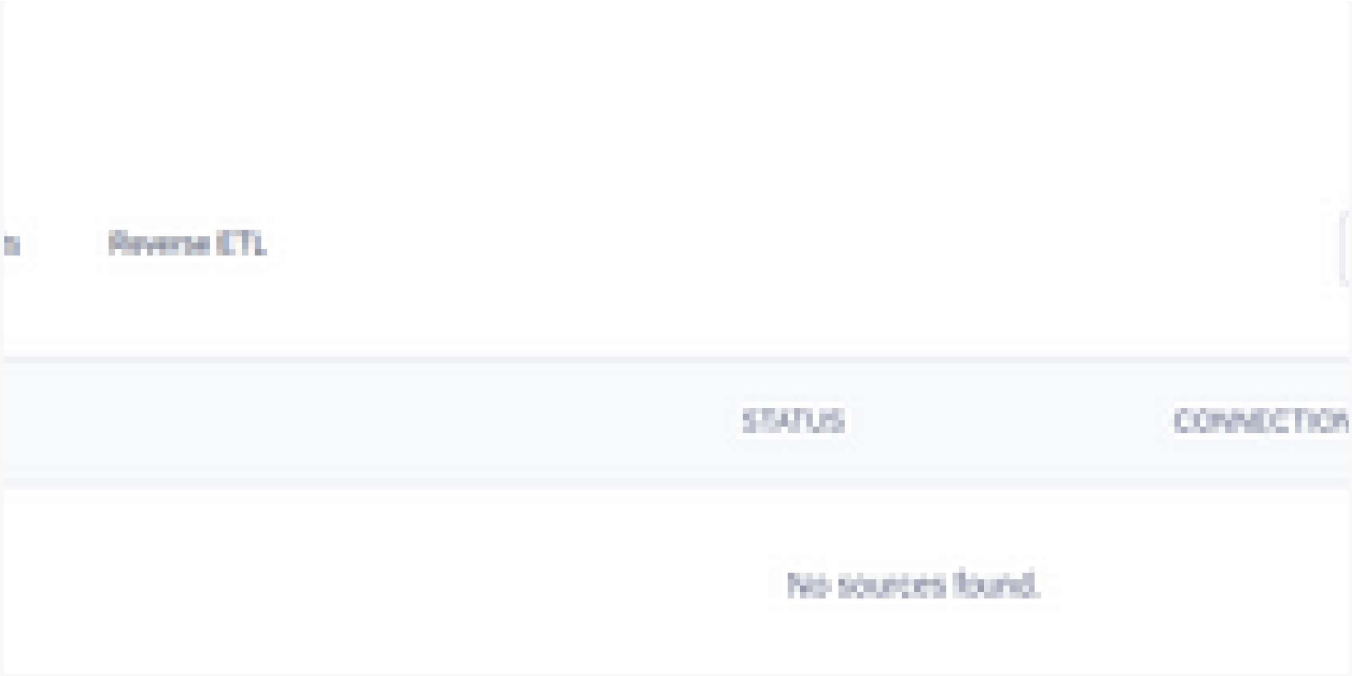


Nouhoum TRAORE

Structured Logs: Best Practices for Effective Management

The logs play a crucial role in monitoring and debugging applications. They are the go-to resource when an issue arises in production. They...

Jan 4 53

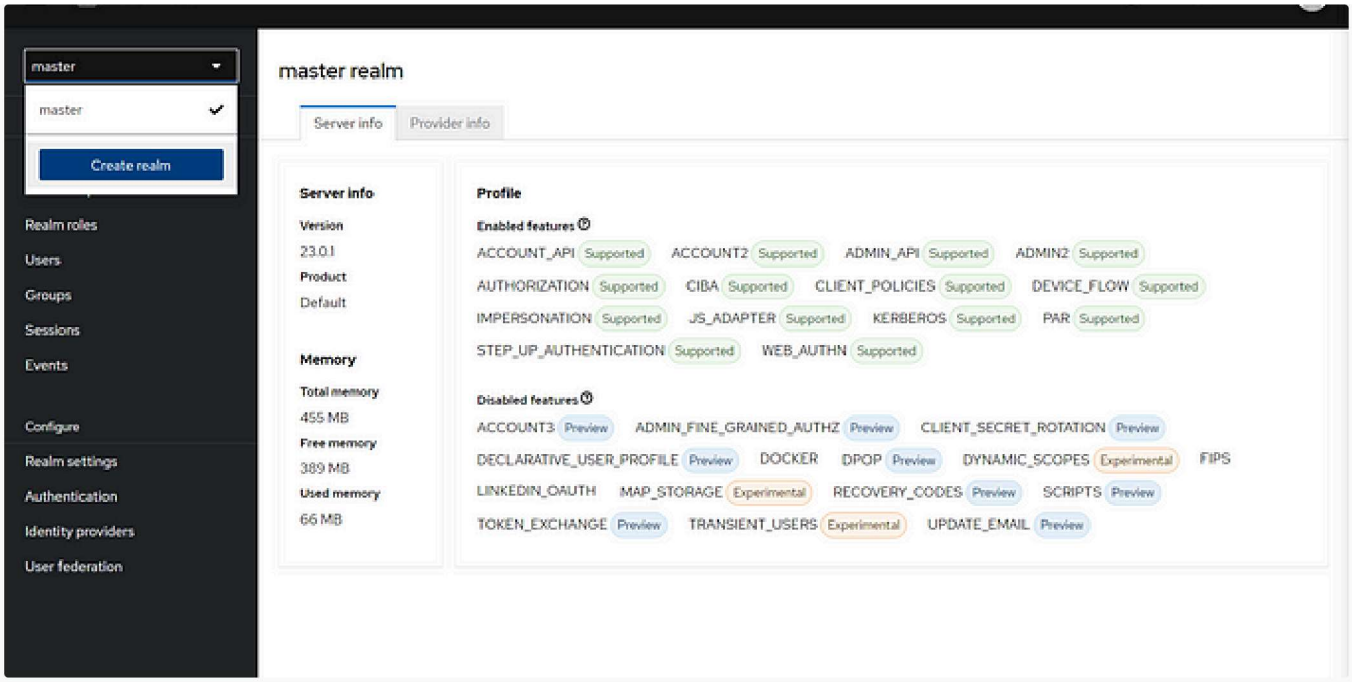


Kawthar Asma Mahboubi

Implementing Keycloak Custom Event Listener for Sending Tracking Events to Segment

This tutorial assumes you already know what keycloak and segment are.

Jan 7 1



Padmakar Kasture

Resource And Scope Based Authorization in Keycloak

Jan 14 5

Vasko Jovanoski

Spring Boot and Testcontainers—A love test story

Hi everyone!

Feb 25 10

See more recommendations