**Spring Boot + OAuth 2 Client Credentials Grant - Hello World Example**

OAuth (Open Authorization) is a simple way to publish and interact with protected data.  
It is an open standard for token-based authentication and authorization on the Internet. It allows an end user's account information to be used by third-party services, such as Facebook, without exposing the user's password.  
When using OAuth2, grant type is the way an application gets the access token. Following are the grant types according to OAuth2 specification-

* Authorization code grant
* Implicit grant
* Resource owner credentials grant
* Client credentials grant
* Refresh token grant

Spring Boot Security - Implementing OAuth2

[Spring Boot Security - Introduction to OAuth](https://www.javainuse.com/spring/spring-boot-oauth-introduction)

[Spring Boot OAuth2 Part 1 - Getting The Authorization Code](https://www.javainuse.com/spring/spring-boot-oauth-authorization-code)

[Spring Boot OAuth2 Part 2 - Getting The Access Token And Using it to fetch data.](https://www.javainuse.com/spring/spring-boot-oauth-access-token)

[**Spring Boot + OAuth 2 Client Credentials Grant - Hello World Example.**](https://www.javainuse.com/spring/springboot-oauth2-client-grant)

[Spring Boot + OAuth 2 Password Grant - Hello World Example.](https://www.javainuse.com/spring/springboot-oauth2-password-grant)

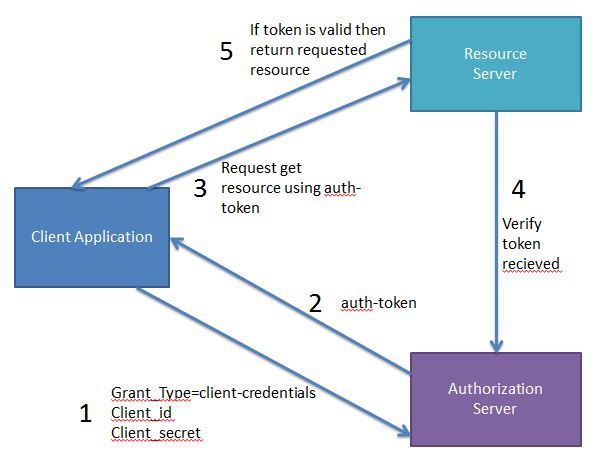
[Facebook Authentication Using Spring Boot + Spring Social Simple Example.](https://www.javainuse.com/spring/spring-social)

Video

This tutorial is explained in the below Youtube Video.

In a previous series we had seen the [Authorization Code Grant](https://www.javainuse.com/spring/spring-boot-oauth-introduction) in detail.  
The Client Credentials Grant involves machine to machine authentication. Oauth usually consists of following actors -

* **Resource Owner(User) -** An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user.
* **Client Application -** The machine that needs to be authenticated.
* **Authorization Server -** The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization
* **Resource Server -** The resource server is the OAuth 2.0 term for your API server. The resource server handles authenticated requests after the application has obtained an access token.

In case of Client credentials grant type the user has no role to play. As previously stated it is machine to machine communication. This is typically used by clients to access resources about themselves rather than to access a user's resources.  
  
This type of Authentication does not involve any end-user. Unlike Authorization Grant where the end user had to authenticate himself using Authorization Server like Gmail, here the machine it self authenticates itself to access a protected resource.

For example consider Trivago, a hotel aggregator portal which will be our client application.  
  
Trivago server will be accessing several third party APIs to show search results. Machine to machine authentication will be done by the Trivago server to access the third party API's to get the hotel data. Suppose it wants search data from makemytrip.com, so Trivago Server will authenticate itself by calling makemytrip's authorization server to get access token and then using this token access the makemytrip resource server to get the search result. So here-

* Client Application(Trivago Server) - Trivago Server which will need to get some reources from MakeMyTrip.com.
* Authorization Server(MakeMyTrip Authorization Server)- MakeMyTrip Authorization Server. Here Trivago should have already registered itself to the MakeMyTrip Authorization Server so that it can be authenticated and issued token.
* Resource Server(MakeMyTrip Resource Server) - MakeMyTrip application will then use the token it recieved from the Authorization Server to get resource from the MakeMyTrip Resource Server. MakeMyTrip ResourceServer will verify if the token recieved is valid by calling the Authorization server which issued it. If its valif it will return the requested resource

So 2 calls are required to be made by the client application to get the resource-

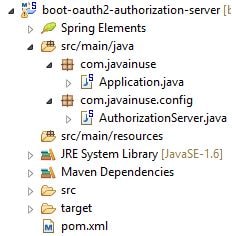
* Call to the Authorization Server to get the token.

|  |  |
| --- | --- |
| **Parameter** | **Value** |
| grant\_type (required) | client\_credentials |
| client\_id(required) | The client id |
| client\_secret(required) | The client secret key |

* After getting the token from the authorization server, the client application then needs to use this for getting resource from the resource server.

Lets Begin?

Authorization Server

Lets first create the Authorization Server which will generate a token for client. Maven Project will be as follows-  
  


In the Maven we need the Spring oauth dependency. Maven will be as follows-

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.javainuse</groupId>

<artifactId>boot-oauth2-authorization-server</artifactId>

<version>0.0.1.SNAPSHOT</version>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.3.0.RELEASE</version>

</parent>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

**<dependency>**

**<groupId>org.springframework.security.oauth</groupId>**

**<artifactId>spring-security-oauth2</artifactId>**

**</dependency>**

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

Define the Spring Boot Main Application.

package com.javainuse;

import java.security.Principal;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.security.oauth2.config.annotation.web.configuration.EnableResourceServer;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

@RestController

@EnableResourceServer

@SpringBootApplication

public class Application {

public static void main(String[] args) {

SpringApplication.run(Application.class, args);

}

//This method will be used to check if the user has a valid token to access the resource

@RequestMapping("/validateUser")

public Principal user(Principal user) {

return user;

}

}

Configure the Authorization Server. The @EnableAuthorizationServer annotation is used to configure the OAuth 2.0 Authorization Server mechanism and defines the behaviour of various endpoints when interacting with the authorization server.

package com.javainuse.config;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.oauth2.config.annotation.configurers.ClientDetailsServiceConfigurer;

import org.springframework.security.oauth2.config.annotation.web.configuration.AuthorizationServerConfigurerAdapter;

import org.springframework.security.oauth2.config.annotation.web.configuration.EnableAuthorizationServer;

import org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationServerEndpointsConfigurer;

@Configuration

@EnableAuthorizationServer

public class AuthorizationServer extends AuthorizationServerConfigurerAdapter {

@Autowired

private AuthenticationManager authenticationManager;

@Override

public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {

endpoints.authenticationManager(authenticationManager);

}

@Override

public void configure(ClientDetailsServiceConfigurer clients) throws Exception {

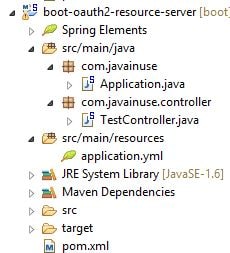
clients.inMemory().withClient("javainuse-client").secret("javainuse-secret")

**.authorizedGrantTypes("client\_credentials")**.scopes("resource-server-read", "resource-server-write");

}

}

Resource Server

Next create the Resource Server which has the reource to be accessed by the client. Maven Project will be as follows-  
  


In the Maven we need the Spring oauth dependency. Maven will be as follows-

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.javainuse</groupId>

<artifactId>boot-oauth2-resource-server</artifactId>

<version>0.0.1.SNAPSHOT</version>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.3.0.RELEASE</version>

</parent>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

**<dependency>**

**<groupId>org.springframework.security.oauth</groupId>**

**<artifactId>spring-security-oauth2</artifactId>**

**</dependency>**

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

Define the Spring Boot Main Application. Configure the Resource Serverusing @EnableResourceServer annotation. It means the service expects an access token in order to process the request. Access token should be obtained from Authorization Server by OAuth 2.0 Client before calling the Resource Server.

package com.javainuse;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.security.oauth2.config.annotation.web.configuration.EnableResourceServer;

import org.springframework.web.bind.annotation.RestController;

@EnableResourceServer

@SpringBootApplication

public class Application {

public static void main(String[] args) {

SpringApplication.run(Application.class, args);

}

}

Define the Controller, to expose API which can be accessed only using valid token.

package com.javainuse.controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

@RestController

public class TestController {

@RequestMapping("/test")

public String test() {

return "Hello World";

}

}

Next define the url of the authorization server to be called by the resource server for verifying the token in the application.yml as follows.

security:

oauth2:

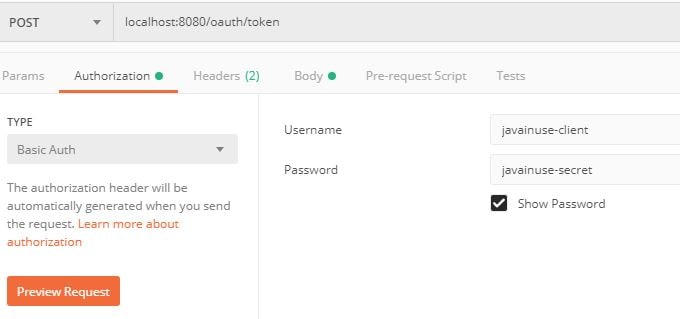
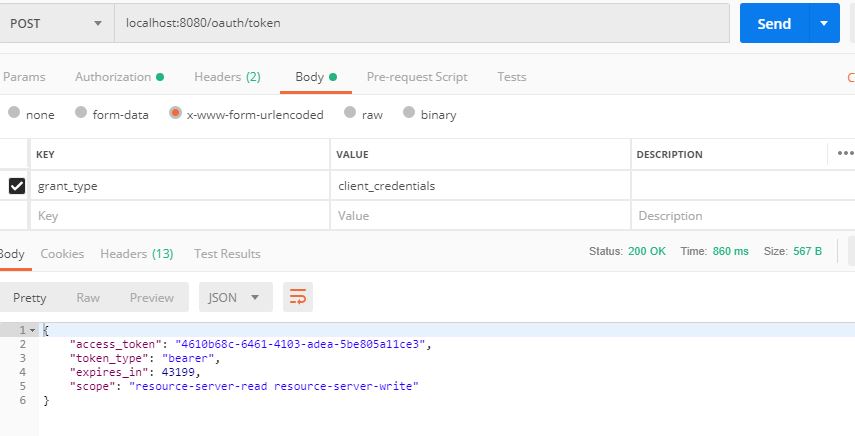
resource:

userInfoUri: http://localhost:8080/validateUser

server:

port: 9090

Finally start the Authorization Server and the Resource Server.

* First get the Access Token by making a POST request to localhost:8080/oauth/token
  + Specify the client\_id and client\_secret in the header using base64 encoding.  
    
  + Next specify the grant type as Client Credentials in body and send the request.  
      
    We get the token as response
* Get the Resource using the access token received above and making a GET call to localhost:9090/test.  
  The token is specified as Authorization Bearer.  
  