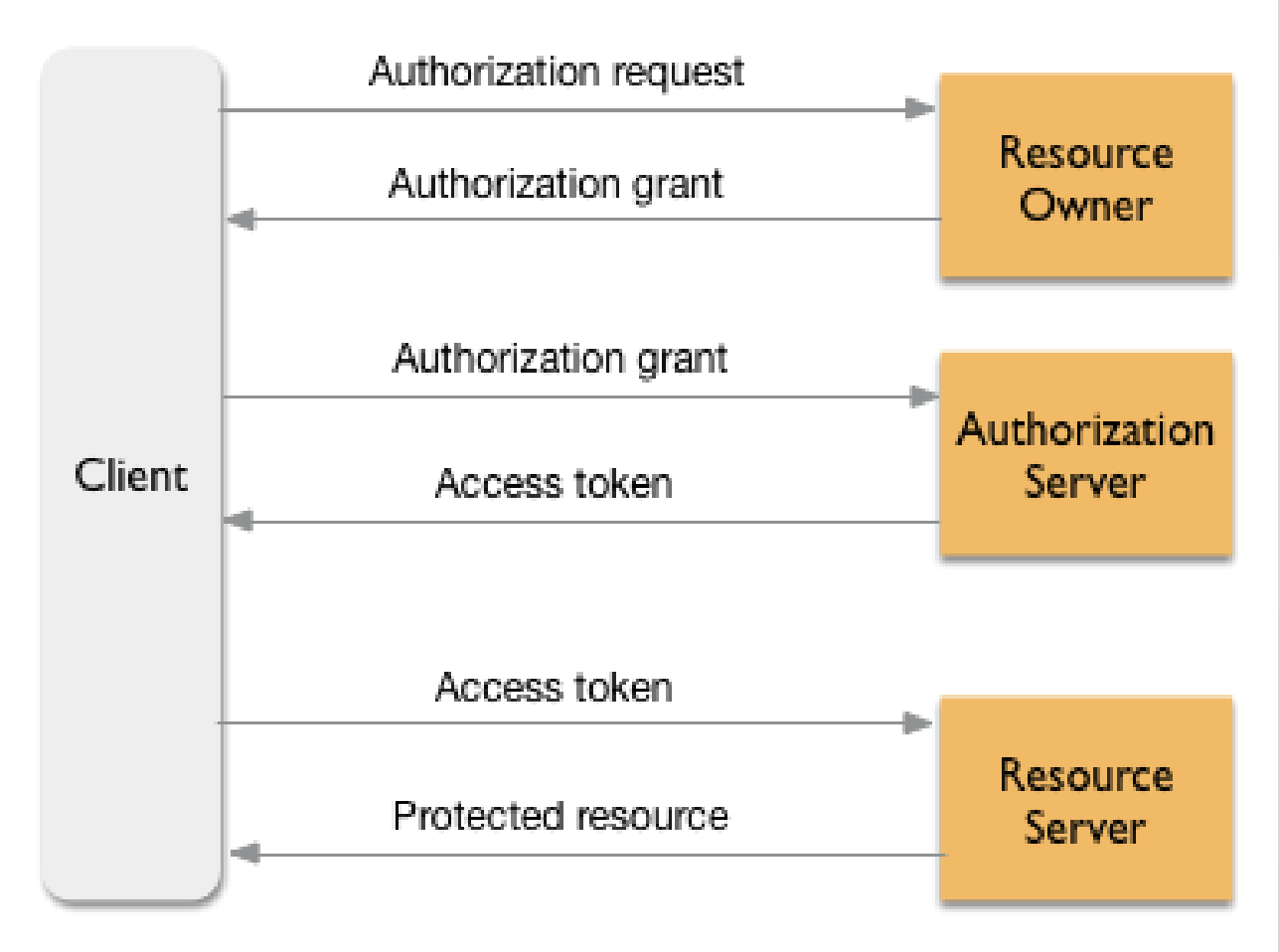
OAUTH 2.0

INTRODUCTION TO OAUTH 2.0

What is OAuth 2.0

"The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf."

The OAuth 2.0 flow



Terms you should know

- Client
- Resource owner
- Resource server
- Authorization server
- Authorization grant
- Access token
- Protected resource

Client

Also called "the app". It can be an app running on a mobile device or a traditional web app. The app makes requests to the resource server for protected assets on behalf of the resource owner. The resource owner must give the app permission to access the protected resources.

Resource owner

Also called an "end user". This is generally the person (or other entity) who is capable of granting access to a protected resource.

For example, if an app needs to use data from one of your social media sites, then you are the resource owner -- the only person who can grant the app access to your data.

Resource server

Think of the resource server as a service like Facebook, Google, or Twitter; or an HR service on your intranet; or a partner service on your B2B extranet. The resource server needs some kind of authorization before it will serve up protected resources to the app.

Authorization server

The authorization server is implemented in compliance with the OAuth 2.0 specification, and it is responsible for validating **authorization grants** and issuing the **access tokens** that give the app access to the user's data on the resource server.

Authorization grant

Gives the app permission to retrieve an access token on behalf of the end user. OAuth 2.0 defines four specific "grant types"

Access token

A long string of characters that serves as a credential used to access protected resources.

Protected resource

Data owned by the resource owner. For example, the user's contact list, account information, or other sensitive data.

Authorization Grant Types

authorization code

- Considered the most secure grant type. Before the authorization server issues an access token, the app must first receive an authorization code from the resource server.
- Typically, this grant type is used when the app resides on a server rather than on the client.
- This grant type is considered highly secure because the client app never handles or sees the user's username or password for the resource server (that is, for example, the app never sees or handles your Twitter credentials). This grant type flow is also called "three-legged" OAuth.

Authorization Grant Types

implicit

- Considered a simplified version of authorization code.
- Typically this grant type is used when the app resides on the client.
- In this grant type flow, the authorization server returns an access token directly when the user is authenticated, rather than issuing an authorization code first.
- Implicit grants can improve app responsiveness in some cases, but this advantage needs to be weighed against possible security implications as described in the IETF specification.

Authorization Grant Types

resource owner password credentials

- In this flow, the client is issued an access token when the user's username/password are validated by the authorization server.
- This flow is recommended for highly trusted applications.
- An advantage of this flow over, say, basic authentication, is that the user only presents their username/password once. From then on, the access token is used.

Authorization Grant Types

client credentials

- Consider using for situations where the client app is acting on its own behalf. That is, the client is also the resource owner.
- This grant type is typically used when the app needs to access a backend data storage service, for example.
- The app needs to use the service to do its work, and the service is otherwise opaque to the end user.
- With this grant type, an app can receive an access token by presenting it's client ID and client secret keys to the authorization server.

Access Token

An access token is a long string of characters that serves as a credential used to access protected resources. Resources tokens (also called bearer tokens) are passed in Authorization headers, like this:

```
$ curl -H "Authorization: Bearer UAj2yiGAcMZGxfN2DhcUbl9v8WsR" \
  http://myorg-test.demogateway.com/v0/weather/forecastrss?w=12797282
```

Access Token

- An access token is a string representing an authorization issued to the client. The string is usually opaque to the client. Tokens represent specific scopes and durations of access, granted by the resource owner, and enforced by the resource server and authorization server.

- The token may denote an identifier used to retrieve the authorization information or may self-contain the authorization information in a verifiable manner (i.e., a token string consisting of some data and a signature).

- Additional authentication credentials, which are beyond the scope of this specification, may be required in order for the client to use a token.

Refresh Token

- Refresh tokens are credentials used to obtain access tokens.
- Refresh tokens are issued to the client by the authorization server and are used to obtain a new access token when the current access token becomes invalid or expires, or to obtain additional access tokens with identical or narrower scope (access tokens may have a shorter lifetime and fewer permissions than authorized by the resource owner).
- Issuing a refresh token is optional at the discretion of the authorization server. If the authorization server issues a refresh token, it is included when issuing an access token

DEMO AND HANDS ON OAUTH 2.0

Lets apply Our Knowledge of OAuth 2.0 in spring boot and spring cloud