**OAuth 2.0 OVERVIEW**

welcome to our oath to introduction

let's go over the basics of OAuth 2

including the roles involved how it

works and what benefits it can provide

let's consider the case of John who was

a developer writing a personal finance

application my bucks he's been told by

his boss that he needs to use OAuth 2 to

authorize users of his application by

using a bank's external authorization

server John's application will delegate

the responsibilities of user

authorization to some other service

rather than managing them on its own

first let's take a look at the

high-level roles that exist within an

OAuth 2 framework for simplicity we will

use three roles to start we have the

user the application in the API within

the API there is an authentication

server and a resource server but we will

cover the difference between those two

later with the growing popularity of

OAuth 2 you've almost certainly seen

these roles at play before in your

everyday internet use for instance when

you launch Spotify to listen to your

music you may have chosen the option to

log in with Facebook and Spotify

requests access to your basic

information and profile picture Spotify

is the application in this example and

Facebook is the API now for John's case

it is his my bucks application instead

of Spotify and instead of Facebook he

intends to use the API of various banks

so John is thinking ah I get it now when

I log in to Spotify with my Facebook

account

Spotify grabs my username and password

from Facebook so easy wrong this is one

of the biggest misconceptions about

OAuth 2 frameworks passwords are never

passed around during this process so how

does it actually look let's use John's

personal finance application my bucks

as an example Sarah is a user who wants

to manage her finances

using the dashboard views that Maya

bucks provides so she opens up my box

and wants to connect her Memorial Bank

checking account in order to view her

balances and transaction history in Maya

Bucks dashboards when Sarah clicks to

connect her memorial bank account my box

will make a request to the memorial bag

authorization server which will display

to Sarah a Memorial Bank authorization

screen asking her to authorize my bucks

to access her bank account

you may recognize these types of screens

from other use cases where a prompt will

tell you what permissions an application

is requesting in our Spotify example

earlier Facebook's authorization screen

asked you to authorize Spotify to access

your Facebook account information and

profile picture sarah grants permission

to my bucks to access her account

balances and transaction history via the

Memorial Bank authorization screen that

she was shown that authorization is sent

back to my Bucks by Memorial Bank along

with an authorization code that my Bucks

will use when requesting an access token

to see Sarah's account now is where

technically we come to the distinction

in the API or Memorial Bank between the

authorization server and the resource

server my box is going to take sarah's

authorization grant including the

authorization code that I mentioned and

use it to request an access token from

the authorization server at Memorial

Bank note that none of Sarah's protected

resources her account balances or

information are located on the

authorization server this server is only

responsible for authorizing Sarah as a

user and providing the proper access

token that will eventually allow my

Bucks to retrieve her protected info

from the resource server after accepting

the authorization grant and code that my

Bucks provided on behalf of Sarah

memorial banks authorization server

provides my Bucks with an access token

specifically for Sarah this token will

be included in a request from my Bucks

to the resource server and it provides

my Bucks with access to only the two

things that Sarah granted it permission

to see her account balances in her

transaction history my buck sends the

request to the Memorial Bank resource

server with this access token included

and Memorial Bank identifies that this

token is valid to access those

particular pieces of Sarah's account the

resource server sends the protected

resources that were requested back to my

Bucks and the application now has

Sarah's balances and transaction history

to display in dashboards and metrics for

Sarah note that throughout this workflow

oo-oo 2 serves as the authorization

framework the actual authentication of

Sarah as a user occurs with Open ID

Connect through the use of ID tokens

that are passed along with the

tokens shown here in order for all of

this to work though John needed to

register his my bucks application with

the Memorial Bank API service

independent of any user involvement John

needs to provide the API with my Bucks

name website and the URL to which the

Memorial Bank authorization screen will

redirect the user after they have

authorized access to their account once

my Bucks provides that necessary

information to the Memorial Bank API the

API will send back a set of credentials

to my Bucks these credentials include a

client ID which is a public and unique

identifier that will be used to identify

my bucks as an application and a client

secret which is a private identifier

kept secret between the app and the API

that is used to authenticate my Bucks

when it makes a request for an access

token let's revisit the actual workflow

for a moment remember that the first few

steps in the process are essentially

getting sarah's permission to access the

protected resources in her account in

OAuth 2 there are four different grant

types for different use cases the grant

type that we use in our example is the

authorization code grant type which is

how Sarah granted access for my bucks to

see her Memorial bank account

information this grant type is used for

applications running on web servers

there are three additional grant types

that we won't detail in this video

implicit grants password grants and

client credentials grants once

everything is implemented we have happy

and informed parties all around Sarah

gets to log in once to access all of her

account balances across different bank

accounts and John's boss is thrilled

because my buck supports easy and simple

integration with various banks through

OAuth 2 and it simplifies the customer

experience for Sarah all of the OAuth 2

capabilities that we've described at a

high level in this video are available

with inter systems cache a which can

serve in any or all of the roles shown

in this overview visit learning enter

systems comm to learn more about inter

systems products and technologies

**DOCKER**

docker is the leading software container

platform on the market today and inner

system's iris allows you to use docker

containers as a vehicle for application

integration and deployment so what are

docker containers containers packaged

applications in the platform independent

fully portable runtime solutions that

satisfy and isolate all dependencies

starting from an executable container

image a container is a runtime instance

of that image think of the image as a

software package and the container

itself as the installed instance of that

software these containers can be used in

both public and private cloud

deployments as well as on physical

machines they've become so widely used

that all major public cloud providers

such as Google Amazon and Microsoft

Azure support container services so what

benefits can docker containers bring to

your inner systems iris implementation

containers allow you to cleanly move an

application between different software

environments and platforms for example

from a development system to a test

system to a production system this ease

of movement allows developers to focus

on writing their code and operations

engineers to focus on the overall

solution infrastructure without

individual build discrepancies in

overhead containers help you to support

your application provisioning and

deployment by separating two important

and distinct application phases the

build phase and the run phase

containers also help you to keep a clear

separation of concerns between code and

data providing full separation of

concerns like these enables you to

easily deploy and upgrade your

applications faster than ever before

containers are extremely efficient a

container utilizes no more memory on the

host machine than any other executable

and is completely isolated from its host

environment an application within a

container only receives the elements

needed to run the containerized software

software such as inner systems iris

so how do containers actually work a

docker image is defined by a docker file

a text file containing all the

information docker needs to build the

container it includes specifications for

a base runtime environment and any

actions that need to be taken during the

building of the container such as

copying or downloading files setting

environment variables or installing

applications

ultimately the image is created using a

docker build command that specifies the

location of this docker file a docker

run command will create and start a new

container from that image once the image

has been built and put into a registry

if you are experienced with inner

system's iris running on Linux it will

not matter what type of physical virtual

or cloud systems and operating systems

your containers are running on you will

interact with them all in the same way

just as you would with traditional inner

systems iris instances running on Linux

systems one important distinction

however is the storage of instance

specific data when running an instance

of inner systems iris in a container

since a containerized application is

isolated from its host environment it

does not write persistent data whatever

it writes inside the container will be

lost when that container is removed and

replaced by a new one thus an important

part of containerized application

deployment is arranging a solution for

data to be stored outside of the

container this is achieved using the

durable cyst feature in inner systems

iris durable cysts enables persistent

storage of instance specific data for

example to upgrade an inner systems iris

instance running in a container you can

run a new container from an upgraded

image because durable cysts stores

instant specific data such as user

definitions audit records log files and

journal files on an external file system

that can be mounted as a volume within

the container the upgraded instance can

be started and run using this data

becoming an upgraded version of the

previous instance note that the approach

used by durable cysts should also be

used for your own data separating

concerns like these will enable you to

be more efficient creating new versions

of your applications in containers if

you want to move your applications to a

cloud native solution docker containers

allow you to embrace a devops process

easily enabling simple fast and

repeatable deployment of your inner

systems iris applications