Demos

# Swagger Demo

Open the swagger editor

Look at petstore

Let’s create a simple API (GET + POST) from scratch (I need a sample).

/home/mjrw/Documents/BBC-Microservices/Source/swagger/SimpleDemo

Start with step1.yaml

Gradually convert from echo to member services

1) change meta data

2) basepath → member

3) What does it return?

**Create a definition:**

definitions:

member:

type: object

properties:

id:

type: string

firstName:

type: string

age:

type: integer

format: int32

**Modify the get definition to:**

/:

get:

responses:

'200':

description: List of members

schema:

title: members

type: array

items:

$ref: '#/definitions/member'

**Add Produces and Consumes**

consumes:

- application/json

- text/xml

produces:

- application/json

- text/html

**Now, add a new get method taking an argument**

/{memberId}:

get:

description: Gets details of a member for the specfied ID

parameters:

- name: memberId

in: path

type: string

required: true

responses:

'200':

description: returns the member detail

schema:

$ref: '#/definitions/member'lly either lined with concrete or the coffi

Now, add a POST – unknown ID:

$ref: '#/definitions/member'

post:

parameters:

- name: member

in: body

description: The member JSON you want to post

schema:

$ref: '#/definitions/member'

required: true

responses:

200:

description: Make a new member

Show Mocking

1) Add some example data to the definitions

Go to Swagger Hub

2) Hit + | Create new API

3) Paste new YAML over it

4) Save

5) … Publish

6) Use the Try it out buttons in the UI

(CURL command shows real UI).

## Proxy DEMO

Based on the YAML file from /home/mjrw/Documents/BBC-Microservices/Source/swagger/SimpleDemo/echo.yaml

Create a NodeJS server

Unzip the server into a folder

Edit EchoServer/controllers/DefaultService.js

Change return from rootPOST to

res.end("Name:" + args.name.value + " Year:" + args.year.value);

npm start

\*\*\*\* Edit the yaml file to specify the correct port number!!!!

Use the docs URL http://localhost:8080/docs

Test the post method: should see our values echoed

Explain the generated code produces the proxy

Generate Client

Optional: to create a container

Docker file:

FROM node:6-onbuild

EXPOSE 8092

docker build -t mjrw/echo-server .

Docker run … -p 8092:8092 mjrw/echo-server

# AWS DEMO

Use my account.

Walk thru the whole process of key creation, instance selection, starting an instance and connecting to it.

# DOCKER DEMO

//Start a container with MYSQL

docker run --name mysql-container -e MYSQL\_ROOT\_PASSWORD=MyPassword -d mysql:8.0.1

docker ps // Show it is running

docker inspect mysql-container

Get the host IP – it’s near the end of the output

// Launch another container point out that is is linked – allows ports to be shared across instances

// This container is runing the mysql client but could be running any client application

// -h is the ip address to connect to (from inspect)

docker run -it --rm mysql sh -c 'exec mysql -h 172.17.0.5 -P 3306 -uroot -p"MyPassword"'

exit

In the instance

show databases;

use mysql;

show tables;

Exit the container and run:

docker run -it --rm mysql bash

Show there is a real file system

# Docker Registry

1) Add a new database to the mysql image

docker run --name mysql-container-demo -v /home/mjrw/Documents/BBC-Microservices/Source/docker/MySQLDemo:/sqldata -e MYSQL\_ROOT\_PASSWORD=MyPassword -d mysql:8.0.1

Note that the key thing here is the mount of our local dir into /docker-entrypoint-initdb.d

(refer to the library/mysql docs). The minimal.sql file is automatically read and creates a new db called servicedb.

Use docker exec -it mysql-container-demo bash to connect to the image

Copy the minimal.sql file to the /docker-entrypoint-initdb.d dir

cp sqldata/\* docker-entrypoint-initdb.d

Exit bash

Stop the container

docker stop mysql-container-demo

2) Save the image with:

docker commit mysql-container-demo bbcdemo/mysqldemo

// Not the tag starts with the account name

docker run --name mysql-container-from-image -d bbcdemo/mysqldemo

// Critical there is no password!

1. Get the IP address – using docker inspect [container\_id]

docker run -it --rm mysql sh -c 'exec mysql -h 172.17.0.3 -P 3306 -uroot -p"MyPassword"'

show databases;

servicedb should be there

Push the image to the registry

docker push bbcdemo/mysqldemo

Prove it works!

Now, on the remote machine:

docker run --name mysql-container-from-image -d bbcdemo/mysqldemo

docker run -it --rm mysql sh -c 'exec mysql -h 172.17.0.2 -P 3306 -uroot -p"MyPassword"'

# Circuit Breaker Demo

$COURSE\_HOME/Source/nodejs-services/CircuitBreaker

Open circuit-breaker.js

Explain:

circuitBreaker is a nodeJS implementation of the Circuit Breaker pattern

<https://www.npmjs.com/package/circuitbreaker>

We have a dummy function loadDataFromRemoteServer which we are wrapping in the circuit breaker

Run with node circuit-breaker.js

Demo order:

Run with calls passing 23 and 32 – works OK

Add two calls with -1

Still works

Add third call with -1

Fails

And keeps failing

Add code called after timeout

Works now :-)

# Zookeeper Demo

~/Documents/BBC-Microservices/Source/docker/Zookeeper-Web$

Docker-compose up

Starts a Web UI and a Zookeeper server

<http://localhost:8000/>

Connection string zookeeper:2181 no credentials

Explain the run the client code

Go back to ZK and show the major key with two sub-keys each holding service access data.

# Docker Swarm

Connect Swarm to AWS

(Use pre-created AWS role).

Create Swarm – show resources being created

Connect to the swarm

Deploy the Composition onto the Swarm

\*\* Connect to a node and see what’s started?

Logs on manager: /var/lib/docker/swarm

User for ssh is docker@

Cloud.docker.io

Create Swarm / AWS

Takes a while to start :: explore the resources being started:

Dynamo, instances, load-balancers, auto-scaling etc

In cloud.docker screen, click on the name of the Swarm – gives a local docker command to run

Run it.

It provides an environment variable DOCKER\_HOST

In ~/Documents/BBC-Microservices/Exercises/Solutions/Composition

docker stack deploy -c docker-compose-swarm.yaml BBC-Demo

(BBC-Demo is the name of the stack)

docker info

docker service ls

Connect to the ELB address (available via edit endpoint in cloud.docker.io).

ELD-HOST:8090/docs (or 8091,8092).

# SQS Client

Qname:

"<https://sqs.us-east-1.amazonaws.com/959486755281/BBC-Demo>";

Producer /home/mjrw/Documents/BBC-Microservices/Source/node-clients/SQSProducer

Consumer /home/mjrw/Documents/BBC-Microservices/Source/node-clients/SQSClient

Show the code for the producer (node index.js)

Run it: puts 50 messages in the Q

Show the consumer

Run (node index.js)

Pulls messages from the Q

Note the duplicates – if there are any!

Run docker -it bbcdemo/sqs-client a couple of times

Run the producer

**API Gateway / Lambda**

Create the Lambda function first

Create Function

Use microservice-http-endpoint as the template (filter with “gateway”)

Specify a name , Create new role from templates and call the role memberRole. Leave the role template as Simple Microservice permission

On completion: click on the gateway name which displays the gateway details

Switch back to the Lambda

**Configuration** tab allows editing of the script

Comment out the switch block

Replace with

console.log(event.httpMethod);

let res = {res : 'OK'};

callback(null, { statusCode: '200', headers: {'Content-Type': 'application/json'}, body: JSON.stringify(res)});

Remove the comment around //console.log('Received event:', JSON.stringify(event, null, 2));

Test the gateway function: use a post with some simple object data {“name”:”value”}

Show the response body

Show the request parameters in the trace (under Endpoint request)

Then add a new resource / method (from the Actions button)

Show that there is an import from Swagger option

Create API – MemberAPI (Select New API)

From Actions button: Create resource (path) (maybe another to illustrate the point)

Create a method