# Membership Application

Application manages member details and the products to which they subscribe

Provides the following User Interfaces (UIs)

1) Self-registration / detail editing for members

2) End user facility to add / delete subscriptions

3) Admin to allow the management of subscription products

4) Payment UI

Provides API’s

1) Query if a particular member has a subscription

2) List all members holding a subscription

3) List all members

4) List details for a member

Has a time or event driven component which notifies users if subscriptions are due to expire

# Identifying Microservices

Using the application description \*\*\* as a starting point, define an initial (candidate) set of services for the application.

1. Define a domain – we’ve done that for you with the description.
2. Examine the description and play “spot the noun” this should yield a list of all possible services
3. Refine the list. You should consider:
   1. Does each service map onto a real world concept?
   2. Is it likely that creating a service will provide a benefit to the system (in terms of flexibility or scalability)?
4. For each of the user / system interactions outlined in the description, consider if the services provide the capabilities needed.
   1. Add any extra services identified to our system

# Identifying Service Interfaces

In our version of the solution to the previous exercise, we have identified a member service. Examining the service, we have identified these use-case titles for the service

Add new member

Update member details

List members

Get details of member

Archive member

1. Examine the list of use-cases and decide
   1. What operations the service should support
   2. What the request and response messages for those services should look like

## Bonus

Other parts of the system need to know that member details have been modified. Consider the operations you would need to support some form of publish / subscribe interface to support this notification.

# Docker Ex 1

Start a Docker machine from an image

Connect to it.

Start stop / explore

Then use Portainer:dmjrw

sudo docker run -d -p 9000:9000

-v /var/run/docker.sock:/var/run/docker.sock portainer/portainer

# Docker Ex 2

Create an empty directory

Create a dockerfile

FROM httpd:2.4

COPY ./public-html/ /usr/local/apache2/htdocs/

Add a /public-html dir to the image with an index.html file

Run the image – map port to 80:80

docker exec -it sad\_franklin /bin/bash

Explore

# Swagger

Create a service interface

swagger.io

Select tools / editor – online editor

Create a swagger specification for a Member service

Use the heoroku pet store as a template

Methods

get given and id returns an object representing a member

name

address

email

post adds a new member

put updates a member

delete deletes a member

# Mongo DB

docker run --name some-more-mongo -v /home/mjrw/Documents/BBC-Microservices/Resources:/mydata -d mongo

docker exec -it some-more-mongo bash

use testdb

foo = { name :"foo", location: { x: 3, y: 5 } }

bar = { name : "bar", location: { x: 1, y: 2, z: 4 } }

zep = { name : "zep", location: { x: -1, z: 4 } }

db.testdata.insert( foo )

db.testdata.insert( bar )ul if eate

db.testdata.insert( zep )

db.testdata.find()

mongoimport --db dbsubs --collection subs --type csv --headerline --file /mydata/subscriptions.csv

db.subs.find({"Member\_Name" : /Michael.\*/});

Do some sample queries (see 4500)

Give them a query to figure out

\*\*\* Integrate with Node?

# CASSANDRA

docker run --name some-cassandra -v /home/mjrw/Documents/BBC-Microservices/Resources:/mydata -d cassandra

docker exec -it some-cassandra bash

cqlsh

(May have to wait for this to work and then retry)

CREATE KEYSPACE subscriptions WITH REPLICATION = { 'class' :

'NetworkTopologyStrategy', 'datacenter1' : 2 };

(Network replicated)

use subscriptions;

CREATE TABLE subscriptions ( id varchar, name varchar, member\_id varchar, type varchar, renew boolean, startdate timestamp, enddate timestamp, PRIMARY KEY(id));

COPY crimedata (id , name , member\_id , type , renew , startdate , enddate )

FROM '/mydata/subscriptions.csv;

Neo4J

docker run \

--publish=7474:7474 --publish=7687:7687 \

--volume=$HOME/neo4j/data:/data \

neo4j

Connect to 7474:7474

Run the code sample

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?

db.testdata.find()