ΠΛΗ-518 Services in Cloud and Fog Computing

Winter Semester 2020-2021

1st PROGRAMMING EXERCISE:

EXTENSION OF WEB APPLICATION WITH THE USE OF MICROSERVICES IN DOCKER ENVIRONMENT AND FAMILIARIZATION WITH THE GCP ENVIRONMENT

Assistants: Tsakos Konstantinos (email: ktsakos@isc.tuc.gr), Tzavaras Emilios (email: atzavaras@isc.tuc.gr)

Professor: Petrakis Evripidis

EXERCISE DESCRIPTION

● The purpose of the exercise is to get familiarized with a real cloud computing environment (Google Cloud Platform) and to develop an application using boxes (containers).

● Additional purpose of the exercise is to get familiarized with the asynchronous way of data exchange and the dynamic updating of the content of the pages using the AJAX technology.

● At the same time, an application data storage service will be developed which will accept REST requests and convert them to the corresponding operations on a MongoDB database.

● Also, the aim of the exercise is to become familiarized with the communication of services through REST, while extending the functionality of the application with ready-made services related to security issues of the application.

● Finally, you will need to familiarize yourself with the use of a publish-subscribe mechanism in order to create subscriptions and alerts for users of the platform.

More specifically the requirements of this exercise are the following:

1) DEPLOYMENT OF THE APPLICATION IN DOCKER ENVIRONMENT

Install the necessary services in the form of containers. For example, you will need MySQL (for application and Keyrock data), Keyrock IDM, Orion Context Broker, Pep Proxy Wilma, MongoDB (required for Orion CB), and Apache Server for the logic of the application (see image below). Make sure that the services are properly connected through the network, expose only the doors they need and keep their data even if a container falls. All containers of the application should be described in a docker-compose.yml file.

2) DYNAMIC REFRESHMENT OF PAGE CONTENT THROUGH ASYNCHRONOUS COMMUNICATION (AJAX)

For each creation, edit, deletion in the database, the updated content should appear dynamically on the corresponding page after each action. That is, the page does not need to be updated to see a user change. For example, the admin deletes the user at the touch of a button and the table of users he sees is dynamically updated without refreshing this page. Use AJAX for asynchronous communication with the application server to achieve all of the above.

3) STORE THE DATA OF THE APPLICATION THROUGH DATA STORAGE SERVICE

Application data should now - instead of being stored in a MySQL database - be stored in a MongoDB using the Data Storage Service. Properly design the REST API of the service which should convert REST requests into suitable queries to a NoSQL MongoDB database. Protect this application as well by using Wilma PEP Proxy.

4) EXTEND THE APPLICATION WITH THE USE OF PUB/SUB SERVICE (ORION CB) FOR THE INFORMATION OF THE USERS ABOUT NEW MOVIES THAT ARE INTERESTED OR THAT STOP PLAYING

When creating a Cinema, make sure that the corresponding entity is registered in the Orion Context Broker (FIWARE). Each "Cinema" entity in Orion will represent a cinema and keep track of the different movies being played in that cinema (and the movies in turn may have their own information). Alternatively, entities can be created in Orion for each movie (Movie type). Ordinary users will be able to subscribe to movies and receive notifications in the following cases: a) when a movie is released within a few days of setting (eg the movie "The Lord of the Rings" is now available in the Cinema Attikon) and b) when a movie is no longer being played. This information should be automatically sent to both the database to be displayed to the user when re-entering the application and to the graphical user interface for real-time notification when the movie details are updated by the owners and they should be notified of those in their feed. In other words, the user's feed should display information to change the status of a movie and keep it for a certain period of time. Use for all of the above the pub / sub mechanism provided by the Orion CB via the REST interface. Orion requires a MongoDB database to work in which it stores all its data. But for the functions of Orion, all you have to do is work with its REST API.

5) USE THE SERVICE KEYROCK IDM (FIWARE) FOR THE AUTHDICATION OF THE USERS

Change the current logic of the application for user authentication and use the REST interface of Keyrock for their authentication (user authentication) through the OAUth2 protocol. You will also need to register your application through the Keyrock graphical interface and encode the client_id and client_secret provided by Keyrock. This encoding should be done on base64 (client_id: client_secret). The result of the encoding will be the Authorization Basic header required in Keyrock's REST call for authentication and the provision of a temporary access token to the application. Essentially we remove the admin page of the application and all management is done from the Keyrock dashboard. To access the application, users and movie owners must have signed up for Keyrock and have been confirmed by an administrator: a) both to access Keyrock and b) to have access to the application.

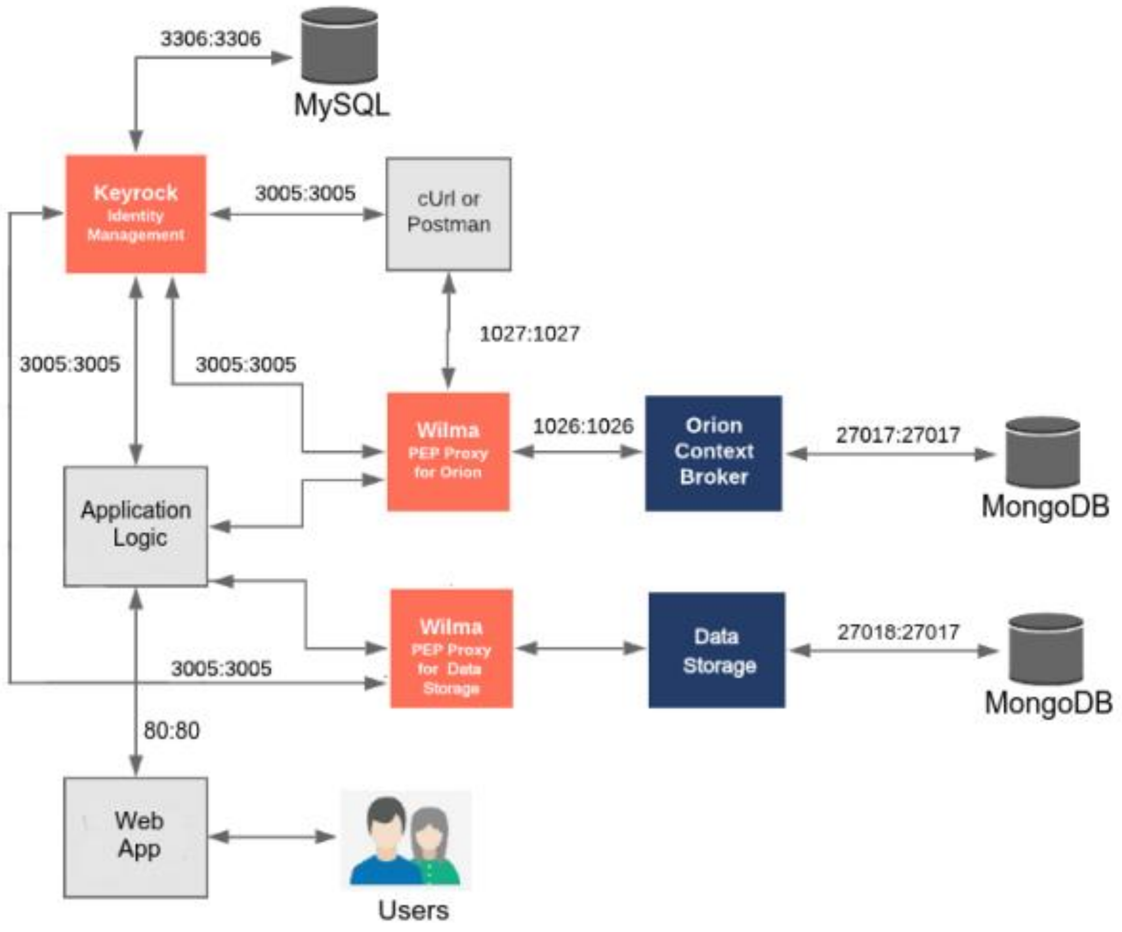
6) USE THE SERVICE PEP PROXY - WILMA (FIWARE) FOR THE PROTECTION OF THE BACKEND APPLICATIONS FROM NOT AUTHORIZED USERS

With the Pep Proxy service, every request to the Orion Context Broker service is checked for whether or not it will be forwarded to it. Thus, requests to the Orion CB are made through the Pep Proxy, which confirms the correctness of the masterkey given in the request header and, only if this is correct, forwards the request to the Orion. This protects the service from malicious users and applications.

7) MIGRATION OF THE APPLICATION AT THE INSTANCE OF THE GOOGLE CLOUD PLATFORM

Create an instance with Ubuntu operating system, install Docker on it and launch the application there. You will need to transfer your password files and the corresponding Dockerfiles, docker-compose, etc. to the virtual machine. You will also need to "open" outside the VM (expose) the doors for the application server and Keyrock for access. in their Web UIs.

The following is an auxiliary / indicative diagram of the architecture that will be composed by the above services.



Ενδεικτικό διάγραμμα αρχιτεκτονικής υπηρεσιών

<u>USEFULL LINKS:</u>

https://fiware-orion.readthedocs.io/en/master/

https://fiware-pep-proxy.readthedocs.io/en/latest/

https://fiware-idm.readthedocs.io/en/latest/index.html

https://documenter.getpostman.com/view/513743/RWMLLRuihttps://documenter.getpostman.com/view/513743/RWaHxUgP

https://documenter.getpostman.com/view/513743/fiware-getting-started/RVu5kp1c

https://www.tutorialspoint.com/mongodb/mongodb_php.htm https://www.php.net/manual/en/mongodb.tutorial.library.php

NOTATION:

● Make sure the application is enjoyable and easy to use (images, easy-to-use menus, fonts, colors, form data entry validation, etc.).

● You are free to use any Web Framework that serves both the creation of the graphical environment (frontend) and the implementation of the application logic (backend).

●Make sure that the database data is stored during the migration of the application to another host machine.

DELIVERED:

● Code scripts with explanatory comments.

● Half page report with remarks about your work: how you did it migration to GCP, what networking exists between the services and how they collaborating, technologies and frameworks used and possible shortcomings of the application that were not implemented or partially implemented, the way data representation in Orion, what you consider to be an entity and what attributes has, the rest API reference of Data Storage which you designed and implemented.

● Sql file (backup file of mysql database)