**CLOUD COMPUTING**

**EXERCISE ONE**

**Step 1**) The first thing after setting up folder structure and downloading the file is to create a docker file for running User Service in port 5002.

**Step 2**) We need to create docker compose file as well for the User Service and postgres db so that we can test few api’s for user Service. We did a setup for network and volumes for efficient communication and persistent storage.

**Step 3**) After the setup we can run the docker compose build command to test if everything is working fine.

A screen shot of a computer

Description automatically generated

**Step 4)** Let’s add the user’s using the post api

A screenshot of a computer

Description automatically generated

**Step 5)** Let’s get the details of all the users we added using the get api

A screenshot of a computer

Description automatically generated

**Step 6)** Let’s get details of single user using the get request using params

A screenshot of a computer

Description automatically generated

**Step 7)** Let’s update the data of a particular user using the put request

A screenshot of a computer

Description automatically generated

**Step 8**) Let’s see if the data is updated in the database using the get request

A screenshot of a computer

Description automatically generated

**Step 9**) Let’s delete one record for the db, using the userid.

A screenshot of a computer

Description automatically generated

**Step 10**) Let’s check if the record is being deleted or not

A screenshot of a computer

Description automatically generated

Now lets replicate the same thing for book Service

**Step 1**) The first thing after setting up folder structure and downloading the file is to create a docker file for running Book Service in port 5006.

**Step 2**) We need to create docker compose file as well for the Book Service and postgres db so that we can test few api’s for Book Service. We did a setup for network and volumes for efficient communication and persistent storage.

**Step 3**) After the setup we can run the docker compose build command to test if everything is working fine.

A screen shot of a computer

Description automatically generated

A computer screen with blue text

Description automatically generated

**Step 4**) Let’s add the book using the post api

A screenshot of a computer

Description automatically generated

**Step 5**) Let’s get the details of all the books we added using the get api

A screenshot of a computer

Description automatically generated

**Step 6**) Let’s get details of single book using the get request using params

A screenshot of a computer

Description automatically generated

If no book with a particular id is found it return the below message

A screenshot of a computer

Description automatically generated

**Step 7**) Let’s update the data of a particular user using the put request

A screenshot of a computer

Description automatically generated

**Step 8**) Let’s see if the data is updated in the database using the get request

A screenshot of a computer

Description automatically generated

**Step 9**) Let’s delete one record for the db

A screenshot of a computer

Description automatically generated

**Step 10**) Let’s check if the record is being deleted or not

A screenshot of a computer

Description automatically generated

So, this means if the bookid is not found, then the book is deleted successfully.

**Exercise Two**

**Step 1**) We need to duplicate exercise one to exercise two

**Step 2**)We need to add rabbitmq to the Docker compose.For rabbitmq we define two ports in docker-compose file.One is default port(5672) and one is default management

**Step 3**) We need to modify the .env file for rabbitmq credentials

**Step 4**) We need to modify the user Service for it to connect with rabbitmq.We need to update main.py code for this.

**Step 5**) We need to create new Service i.e borrow Service that will listen to messages from user service and process them asynchronously

**Step 6**)We need to update docker-compose.yml to include BorrowService

**Step 7**)Build and Run the Services:

Now lets test the new endpoint which we have added in borrow Service which is /users/borrow/request

A screenshot of a computer

Description automatically generated

We did hit this api multiple times to test some conditions in future.

**Step 8**) We have created a api for list of books for a particular student id  
if we hit that api, we can get list of books for a particular student id that logic is wrriten in the borrow service and its an get api with params

A screenshot of a computer

Description automatically generated

So here we get list of books borrowed by a user.So as we can see we get this list in the response.

**Step 9**) Let’s check the database, and to see how many records.

A screenshot of a computer screen

Description automatically generated

**Step 10**) As we know that a particular Student cannot borrow more than 5 books at a time,so we need fetch list of books borrowed by a given user and then add condition that if he has borrowed 5 books, it would throw an error when he is borrowing the 6th book.

A screenshot of a computer

Description automatically generated

As we saw in the early api response, this student has borrowed 5 books and when he borrows the 6th book it gives and error like this.

**Exercise three**

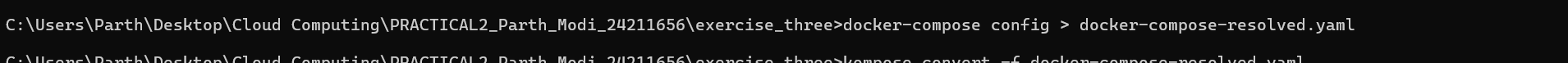
**Step 1**) Copy the exercise two and make it exercise three

**Step 2**) Start minikube cluster

A black screen with white text

Description automatically generated

**Step 3**) Then we run the following command so that it validates and merges the configuration in a docker-compose.yml file, resolving variables and defaults, and then outputs the resulting configuration into a new file named docker-compose-resolved.yaml.



**Step 4**) The following command converts the Docker Compose configuration specified in docker-compose-resolved.yaml into Kubernetes resource files, enabling deployment to a Kubernetes cluster.

A black screen with white text

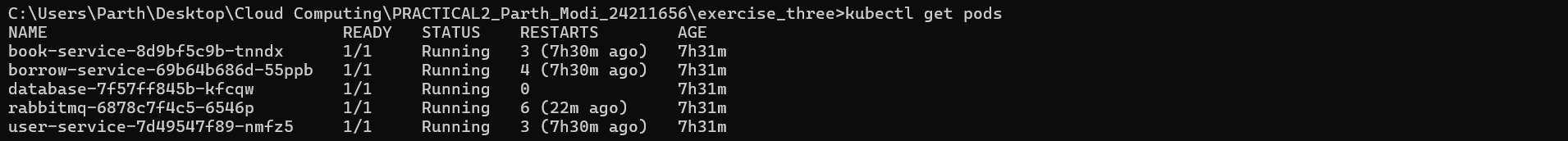
Description automatically generated

**Step 5**) The following command applies all Kubernetes configuration files in the current directory, creating or updating the resources (like pods, services, deployments) defined in them in the Kubernetes cluster.

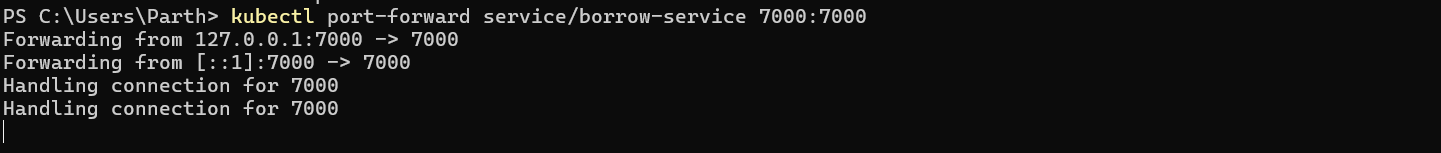
A black screen with white text

Description automatically generated

**Step 6**) Just check the status of the pods which you have created



Everything is running fine  
  
**Step 7**) After that, we just need to do the port forwarding, for user, borrow and book service.



A black screen with white text

Description automatically generated

A black screen with white text

Description automatically generated

**Conclusion**: We have successfully completed the deployment of whole services on docker-compose and Kubernetes.