|  |  |
| --- | --- |
|  | **CS 462: Cloud Computing**  **Lab Exercise 5 (Friday, 11th October 2024)** |

**Instructions**

You are to work individually on this lab exercise. You can discuss aspects of this course work in a team of three members, but remember that each member should be fully engaged, so that you can do the work and submit individually. Ashesi Honour Code applies!

**Submission**: A submission slot will be provided slot on CANVAS.

Complete this project and submit two deliverables:

1. website link, and
2. a short recording explaining how you completed this project.

**Deadline is by 11:55 pm today Friday, 11th October 2024**.

**Introduction to Docker Compose [35 MARKS]**

Docker Compose is a tool that simplifies defining and running multi-container Docker applications. It allows you to manage multiple services using a single configuration file (docker-compose.yml).

Docker focuses on running individual containers, while Docker Compose orchestrates and manages multiple containers as services, simplifying the deployment of complex applications.

For this lab you will use Docker Compose to link two services: a **Flask** application provided and **Redis**, a caching service.

**Part 1: Setting Up the Environment**

1. Create a VM on GCP/Azure/AWS
   * + Log in to your GCP, Azure or AWS account.
     + Create a new VM instance or select the one used in the previous lab if still available
2. Access the VM
   * + SSH into your VM instance from the GCP/Azure/AWS console.
3. Update System Packages
   * + Update the system's packages using:

***sudo apt update***

***sudo apt upgrade***

**Part 2: Installing Docker Compose**

1. Install Docker Compose on your VM using:

***sudo apt install docker-compose***

NB: If this is not the VM from the previous lab, you need to install docker as well.

1. Verify that Docker Compose is installed by checking the version using:

***docker-compose --version***

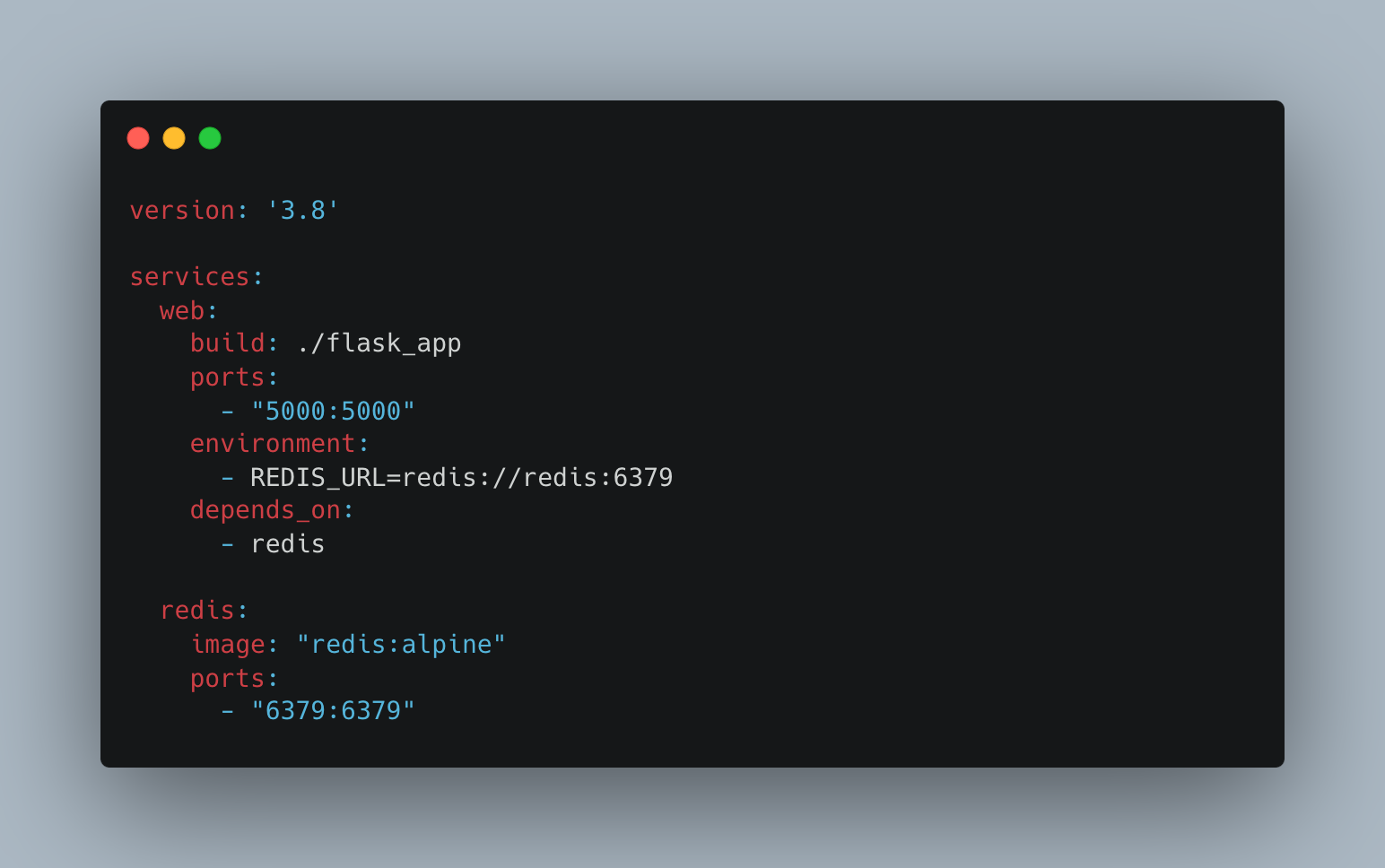
**Part 3: Building the Flask Application**

For this lab, you have been provided with a very simple Flask application that incorporates Redis for caching. This application uses a counter to tell how many times a web page has been visited.

Using either FileZilla or Git, transfer the files provided onto your VM.

**Part 4: The Docker Compose File**

As part of the files that you have been provided is a docker-compose.yml file. This file defines both the Flask and Redis services that would be used by the application.



The *version: '3.8'* at the top specifies which version of the Docker Compose file format is being used. The rest of the file is where you define your services.

**Services Section:**

Docker Compose allows you to define multiple services under the services key. Each service represents a different container that will be managed by Docker Compose.

The first service is **web**, which is typically used for the main application server.

* **build: ./flask\_app**: This tells Docker Compose to build the image for this service from the contents of the flask\_app directory. Inside this directory, there should be a Dockerfile that will define how the Flask application should be set up (Python version, dependencies, and app code).
* **ports:** This is a port mapping configuration. The format is "HOST\_PORT:CONTAINER\_PORT". In this case:
  + "5000:5000" means that Docker will expose the container's internal port 5000 (where Flask is running) to port 5000 on the host machine (in your case, the VM).
  + This allows external access to the Flask application through port 5000 on the VM's IP address.
* **environment:** Here, you're setting an environment variable for the Flask application:
  + REDIS\_URL=redis://redis:6379 tells the Flask app how to connect to the Redis service.
  + redis://redis:6379 means that the Flask app can reach the Redis service at hostname redis on port 6379. This works because Docker Compose automatically creates a network where services can communicate with each other by their service names (in this case, redis).
* **depends\_on:** This tells Docker Compose that the web service depends on the redis service. This ensures that the Redis container is started before the Flask container is started. Note that it doesn’t wait for Redis to be “ready,” just for the container to be started.

The second service is **redis**, which represents the Redis service.

* **image:** "redis:alpine": This tells Docker Compose to use the official Redis image from Docker Hub, specifically the lightweight Alpine version, which is much smaller and more efficient.
* **ports:** This exposes Redis's default port, 6379, to the host.

"6379:6379" means that port 6379 inside the container is accessible on port 6379 on the VM.

**Part 5: Running the Application**

To run the application, you need to start Docker Compose first. In the directory where the docker-compose.yml file is located, run the following command:

***sudo docker-compose up --build***

Since the Flask application is going to use port 5000 (from the Docker Compose file), you will need to ensure that the port (5000) is open in your VM’s firewall settings. You may need to add a firewall rule to allow traffic on port 5000.

Once this is complete, enter the following URL in your web browser:

http://<YOUR\_EXTERNAL\_IP\_ADDRESS>:5000

**Setting Up an Additional Service [5 MARKS]**

Docker Compose can handle multiple services. Modify the application and the docker-compose file to include and use an additional service of your choice (e.g. PostgreSQL) and demonstrate how this new service works in conjunction with the existing application. Alternatively, you could create a new application that uses at least three services, all managed by Docker Compose.