Polytechnique Montreal
LOG8415: Lab 2
Deploying Webapps on Containers on AWS

Abstract

In this lab assignment, you will get hands-on experience running containers on AWS in order to serve web applications. In the first part of the assignment, you will have install docker. Next, you will deploy our custom containers on your instances in order to serve your web application. Your web application will be a simple flask app that runs inference on a ML model. You are asked to report your results and analysis by producing a report using LATEX format.

Objectives

The overall goals of this lab assignment are:

- Get hands-on experience running containers on AWS.
- Understand containerization technologies such as Docker.
- Run inference on ML models hosted on the cloud.
- Use the codebase you've developed for your 1st assignment to setup your instances.

Instructions on setting up Docker

You will find detailed instructions about how to install and run Docker on an Ubuntu instance in here: Install Docker Engine on Ubuntu [1]. You should use the "Install using the Apt repository" section. Then, build your own docker container using docker compose by following the instructions on here: "Docker Compose overview" [2]. Follow the instructions in the "Try Docker Compose" section in order to setup a single flask application in a container on your instance.

Experiments with Orchestration

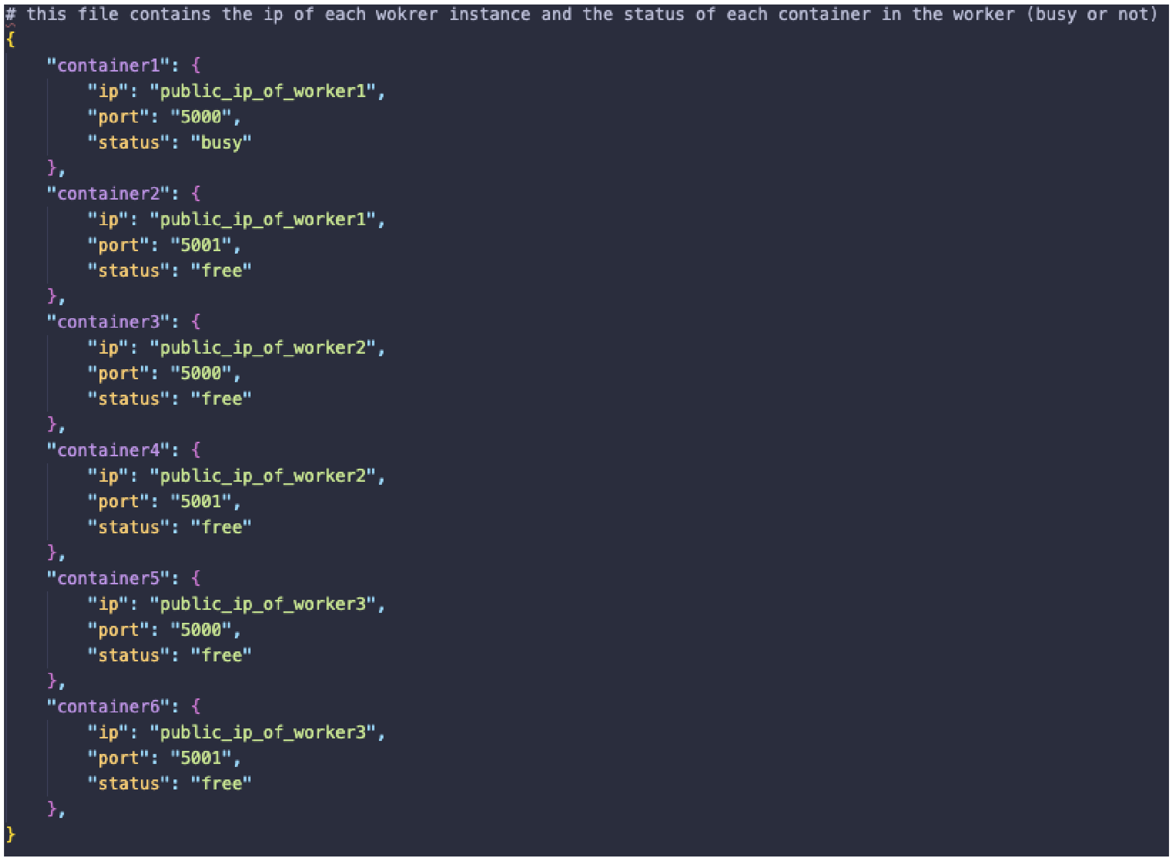
As you are going to be using machine learning models inside your instances, you need to make sure to orchestrate which machine should respond to each request. A request should be sent to a machine that is not already processing a previous request. For this purpose, you will need to implement a simple cluster manager instance (orchestrator). The orchestrator is responsible for:

- Receiving the requests from users.
- Keeping track of which container is available for processing requests.
- Returning the results to the users.

Receiving requests from users

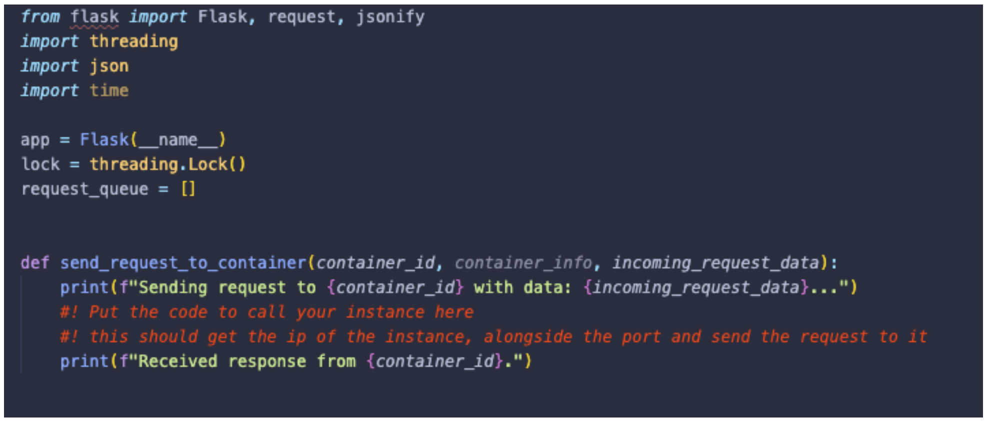
This is handled the same way as your previous assignment. You will run a simple Flask application on the orchestrator which accepts requests on port 80. The orchestrator will manage 4 worker instances. Each worker instance will be running two similar containers. The orchestrator will be monitoring which container in each instance is processing a request and which is free by using an internal status file. When a new request is received, the orchestrator will check the status file and if a container is free, will forward the request to the mentioned container. If no containers are free for processing an incoming request, the orchestrator will store the request in a queue and will forward it to a container as soon as one is free.

Below is an example of a JSON file that is used to keep track of the status of workers:

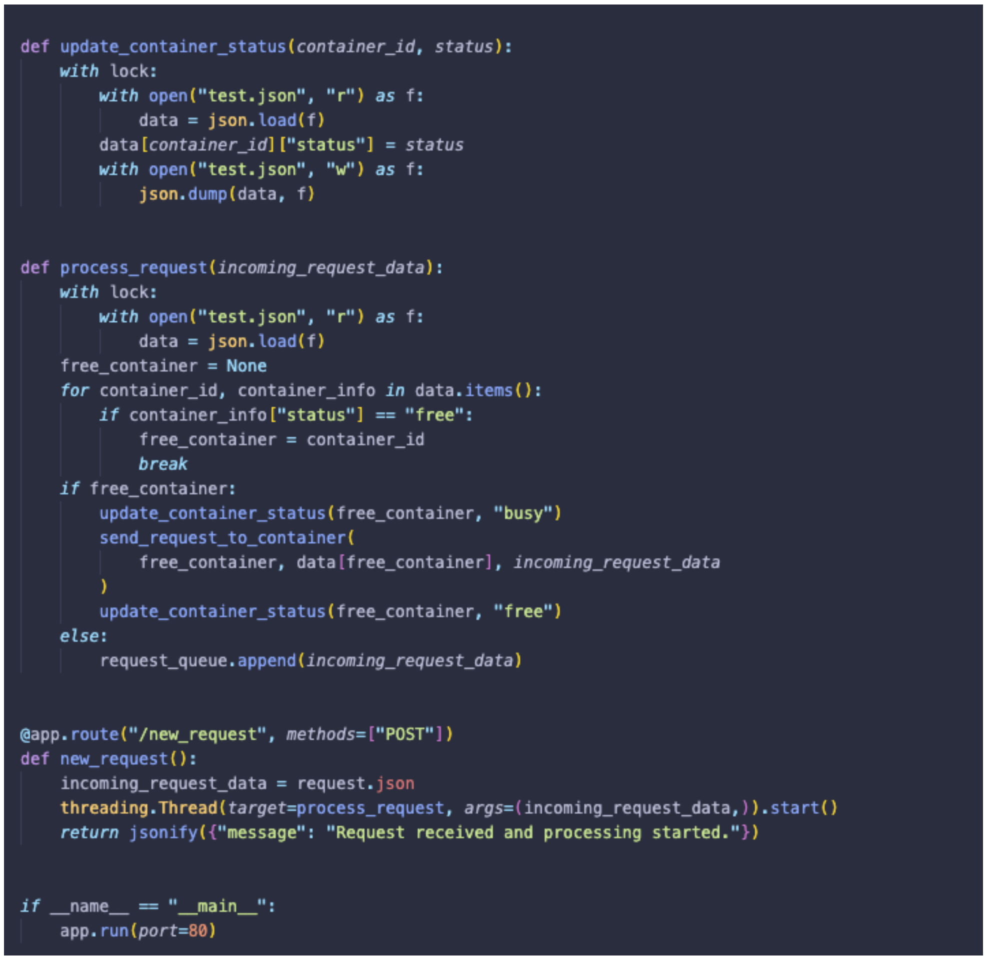


```
# this file contains the ip of each wokrer instance and the status of each container in the worker (busy or not)
{
    "container1": {
        "ip": "public_ip_of_worker1",
        "port": "5000",
        "status": "busy"
    },
    "container2": {
        "ip": "public_ip_of_worker1",
        "port": "5001",
        "status": "free"
    },
    "container3": {
        "ip": "public_ip_of_worker2",
        "port": "5000",
        "status": "free"
    },
    "container4": {
        "ip": "public_ip_of_worker2",
        "port": "5001",
        "status": "free"
    },
    "container5": {
        "ip": "public_ip_of_worker3",
        "port": "5000",
        "status": "free"
    },
    "container6": {
        "ip": "public_ip_of_worker3",
        "port": "5001",
        "status": "free"
    },
}
```

Below is an example of the code you need to run on your orchestrator. Note that you need to write the code for calling your instance in the part that is written in red. Feel free to change or modify this example as you wish.



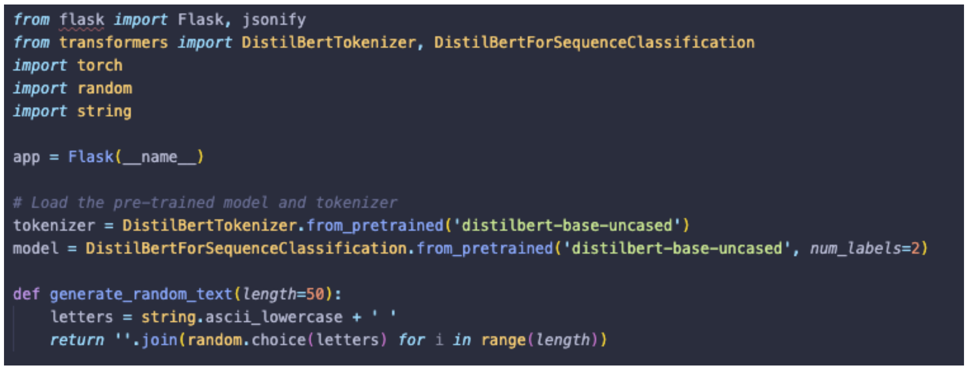
```python
from flask import Flask, request, jsonify
import threading
import json
import time

app = Flask(__name__)
lock = threading.Lock()
request_queue = []


def send_request_to_container(container_id, container_info, incoming_request_data):
    print(f"Sending request to {container_id} with data: {incoming_request_data}...")
    #! Put the code to call your instance here
    #! this should get the ip of the instance, alongside the port and send the request to it
    print(f"Received response from {container_id}.")
```



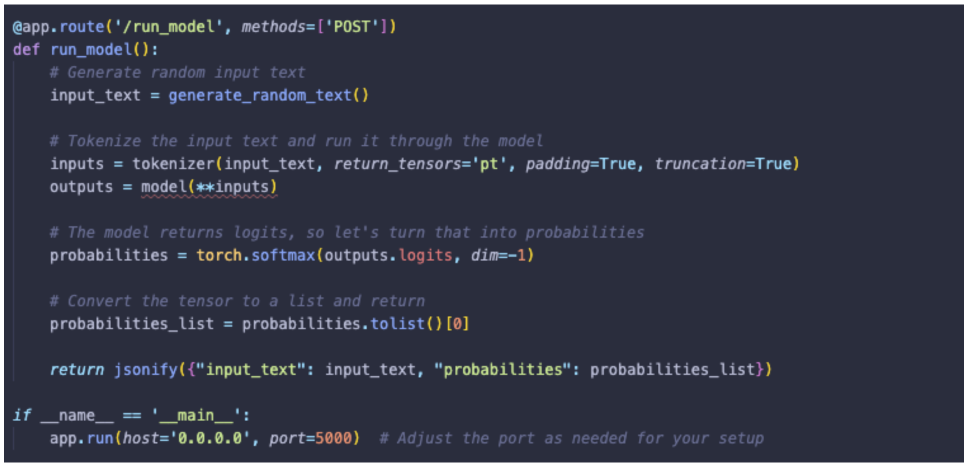
```python

def update_container_status(container_id, status):
    with lock:
        with open("test.json", "r") as f:
            data = json.load(f)
        data[container_id]["status"] = status
        with open("test.json", "w") as f:
            json.dump(data, f)


def process_request(incoming_request_data):
    with lock:
        with open("test.json", "r") as f:
            data = json.load(f)
    free_container = None
    for container_id, container_info in data.items():
        if container_info["status"] == "free":
            free_container = container_id
            break
    if free_container:
        update_container_status(free_container, "busy")
        send_request_to_container(
            free_container, data[free_container], incoming_request_data
        )
        update_container_status(free_container, "free")
    else:
        request_queue.append(incoming_request_data)


@app.route("/new_request", methods=["POST"])
def new_request():
    incoming_request_data = request.json
    threading.Thread(target=process_request, args=(incoming_request_data,)).start()
    return jsonify({"message": "Request received and processing started."})


if __name__ == "__main__":
    app.run(port=80)
```

Deploying an ML application on your workers using flask

This part is similar to how deployed your flask apps in your first assignment. The difference is that now instead of returning a simple string, your flask app calls a function which is responsible for running inference on an ML model and returns a JSON response. Below, is an example of the code you can use on your worker instances:



```python
from flask import Flask, jsonify
from transformers import DistilBertTokenizer, DistilBertForSequenceClassification
import torch
import random
import string

app = Flask(__name__)

# Load the pre-trained model and tokenizer
tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')
model = DistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased', num_labels=2)

def generate_random_text(length=50):
    letters = string.ascii_lowercase + ' '
    return ''.join(random.choice(letters) for i in range(length))
```

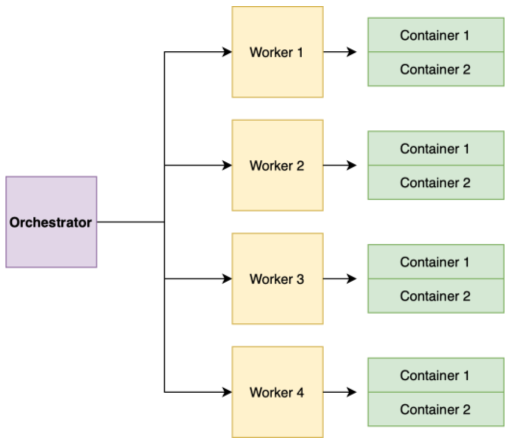


```python
@app.route('/run_model', methods=['POST'])
def run_model():
    # Generate random input text
    input_text = generate_random_text()

    # Tokenize the input text and run it through the model
    inputs = tokenizer(input_text, return_tensors='pt', padding=True, truncation=True)
    outputs = model(**inputs)

    # The model returns logits, so let's turn that into probabilities
    probabilities = torch.softmax(outputs.logits, dim=-1)

    # Convert the tensor to a list and return
    probabilities_list = probabilities.tolist()[0]

    return jsonify({"input_text": input_text, "probabilities": probabilities_list})

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)  # Adjust the port as needed for your setup
```

DistilBert is a small model and should run without any problems on an M4.large instance. However, you are allowed to use even smaller models as the purpose of this assignment is getting familiar with ML model deployment. You can find more information about this model and how to use the HhuggingFace library in here [3], [4].

Instructions

You will need to setup five instances in total. One M4.large instance to act as the orchestrator, and 4 other M4.large instances to act as the workers. Each worker must run 2 containers that listen on different ports for incoming requests. For this assignment, there is no need to setup a loadbalanacer and target groups. The figure below displays the architecture of your cluster:



After setting up your cluster, similar to your first assignment, you will send multiple requests to your cluster at the same time. Use the multiprocessing library in Python or any other library depending on the programming language that you are using to send requests simultaneously. You will need to send at least 5 requests at the same time. Note that in the examples, the models work on randomly generated text that is generated as soon as the model is called. Therefore, you don't need to send any strings to your workers. A request to just invoke the processing is sufficient.
Note that the ip addresses in the provided snippets are examples and you need to change them depending on how you deploy your cluster.

How does this relate to your 1st assignment

In your first assignment, you practiced setting up your clusters and installing applications on them using code and running commands on your machines remotely. Here, the underlying principles are the same:

- Setting up the machines using code is the same.
- You have already used docker for your 1st assignment. Here, you need to customize your docker files to build the containers that are required for this assignment.
- You don't need to setup a loadbalancer.
- You will install the Flask, PyTorch, and HuggingFace libraries the same way that you installed them for your first assignment.
- You will send the requests to the orchestrator the same way that you did for your first assignment. Here, instead of a loadbalancer, the orchestrator will decide which instance and which container inside an instance should respond to the incoming request.

Working with Groups

You should work in groups and submit only one report along with all necessary code.

Report

One submission per group is required for this assignment. To present your findings and answers questions, you have to write a lab report on your results using LATEX template. In your report should write:

- Experiments with your docker container.
- How did you compose your docker containers.
- How the orchestrator manages the queue when all instances are busy.
- How you deployed your flask application on the orchestrator.
- How did you deploy your flask apps on your workers.
- A sample of what your cluster returns for an incoming request.

- Summary of results and instructions to run your code.

Evaluation

A single final submission for this assignment is due on the date specified in Moodle for each group. You need to submit one PDF file per group. All necessary codes, scripts and programs must be sufficiently commented and attached to your submission. A demo will be organized after the submission for your work. All team members should be participating in the demo.

Please submit your PDF report and push your code to a GitHub repository.

Acknowledgement

We would like to thank Amazon Web Services for supporting us through their and AWS Educate grants.

References

[1] https://docs.docker.com/engine/install/ubuntu/

[2] https://docs.docker.com/compose/

[3] https://huggingface.co/docs/transformers/model_doc/distilbert

[4] https://huggingface.co/docs/transformers/index