

Assignment

1. Context

In the last years, the number of Internet users has seen an unprecedented growth. Whether these users are human beings or machines (IoT devices, bots, other services, mobile clients etc.) they place a great burden on the systems they are requesting services from. As a consequence, the administrators of those systems had to adapt and come up with solutions for efficiently handling the increasing traffic. In this assignment, we will focus on one of them and that is **load balancing**.

A **load balancer** is a proxy meant to distribute traffic across a number of servers, usually called **services**. By adding a load balancer in a distributed system, the **capacity** and **reliability** of those services greatly increase. That is why in almost every modern **cloud architecture** there is at least one layer of load balancing.

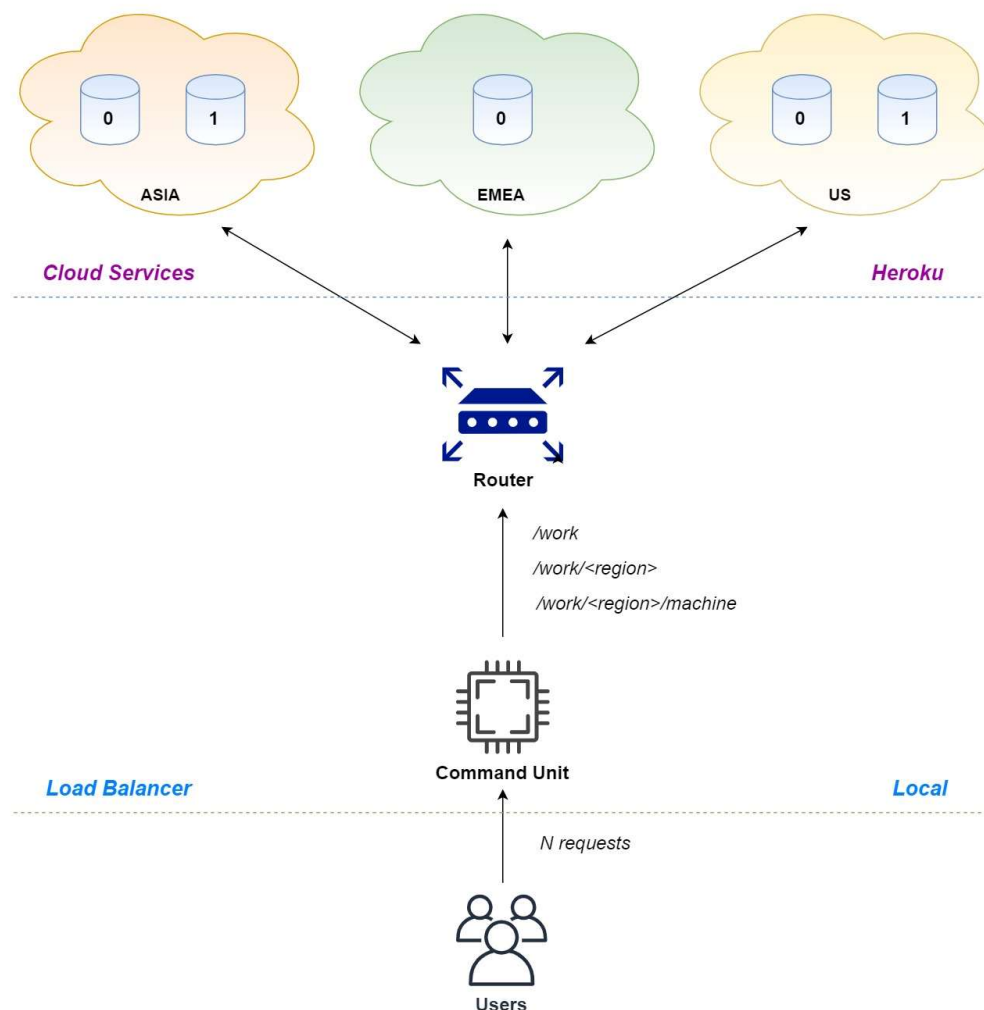
2. Architecture

We propose a **cloud system** that acts as a global service, replicated across multiple **machines/workers** and **regions**, which must serve the clients as efficiently as possible. Supposing that all the requests are coming from users found in the same country, the **latencies** expected from the cloud regions differ. Moreover, the **number of machines** available on each region vary and all these aspects can have an impact on the overall performance of the system.

That is why, a dedicated proxy that decides where to route those requests coming from the clients had to be added into the system. Obviously, this proxy is our **load balancer** and in this particular scenario it is divided into **2 main components**:

1. A **router**, meant to **forward the requests** coming from the clients to an actual machine available on a certain region
2. A **command unit** that is supposed to **identify the number of requests** that are about to hit our system and decide, based on this number, how to **efficiently utilize the routing unit**

You can have an overview on the proposed architecture by looking at the diagram below:



In this assignment, you will be focusing on the **Command Unit**, the other components of the system being already implemented by us and deployed in **Docker images**.

3. REST API

In simple terms, a **RESTful API** is a programming interface that uses **HTTP methods** to interact with data or **resources**. At the moment, it is the most popular **client-server** interaction and is largely used in various cloud systems. Usually, those resources are served to the client in fixed formats (e.g. **HTML**, **XML**, **JSON**), depending on the context. For this assignment, the responses coming from the workers will be represented in the **JSON** [https://www.w3schools.com/js/js_json_syntax.asp] format.

For simplicity, the **routing unit** is also exposing a **REST API** through which the **command unit** can choose what machine and region the request should be forwarded to. The available **endpoints** are:

- **/work** - forwards the request to a random machine from a random region
- **/work/<string:region>** - forwards the request to a random machine from the specified region
- **/work/<string:region>/<int:machine>** - forwards the request to a specific machine from a specific region

You can quickly go through the **main principles** of a **REST API** by visiting this link [<https://searcharchitecture.techtarget.com/definition/RESTful-API>]. Besides, you might find this example [<https://restfulapi.net/rest-api-design-tutorial-with-example/>] useful.

The region names and machine numbers are the ones represented in the diagram above. Please note that the region names are expected to be **lowercase**.

4. Docker

Docker is one of the most popular **container** solutions available right now. It helps you isolate a service in a provisioned environment and deploy that entire environment fast using **images**. In our system, each worker is placed in a separate Docker image, which can be easily used as-is and deployed locally or on a cloud service. The same happens with the **forwarding unit**, also called **master**.

To understand more about Docker containers and how they work, please briefly go through those links:

- What is Docker? [<https://opensource.com/resources/what-docker>]
- What are Docker containers? [<https://www.docker.com/resources/what-container>]
- Brief Docker Tutorial [<https://github.com/docker/labs/tree/master/beginner/>]

All the necessary images for this assignment can be found in this Docker Hub repository [<https://hub.docker.com/u/vstefanescu96>].

5. Heroku

In order to simulate a **real-world environment** in which the requested services cannot be found locally, we chose to host the **workers** on a cloud **platform as a service** (PaaS) provider. Due to its attractive pricing (it's free 😊), we chose **Heroku**.

You can find more about it here [<https://www.heroku.com/what>].

6. Implementation

You have to implement the **command unit** component of the **load balancer** in **Python 3** and deploy it **locally**. Having a number of requests **N** as input, try various strategies of calling the 3 endpoints available on the **forwarding unit** so that your clients experiment response times as low as possible. There are **no constraints** applied to how you read the number of requests, what Python library you use to call the forwarding unit or how you plot the results.

This assignment **must** be developed in **Python 3**!

However, we strongly suggest you to work in a **virtual environment** where you install all your **pip dependencies**. By doing so, you will keep your global workspace free of useless packages and you can easily specify just those packages required to run your code:

```
pip freeze > requirements.txt
```

Please note that we will definitely apply penalties if the **requirements.txt** file contains packages that are not used. Always imagine that you are in a real production environment 😊. You can find out more about **virtual environments** here [<https://docs.python.org/3/tutorial/venv.html>].

7. Objectives and Evaluation

All the necessary files required for the prerequisites can be found here.

I. (20p) Prerequisites

A. (3p) Heroku Account

Create a Heroku account using your student email. You can register here [<https://signup.heroku.com/>]. While you register, you can use this information:

- **Company:** UPB
- **Role:** student
- **Primary Development Language:** Python

Make sure you **confirm** your **account registration** and setup a password!

B. (3p) Docker Hub Account

Create a Docker Hub account using your student email. You can register here [<https://hub.docker.com/signup>].

Make sure you **confirm** your **account registration**!

C. (4p) Docker Installation

In order to use Docker on your **local machine**, you must first install the **Docker Engine**. Please follow one of these guides:

- Windows Hosts [<https://docs.docker.com/v17.09/docker-for-windows/install/>]
- Mac Hosts [<https://docs.docker.com/v17.09/docker-for-mac/install/>].
- Ubuntu Hosts [<https://docs.docker.com/engine/install/ubuntu/>]

If you are using **Windows** as host, **Hyper-V** must be enabled. This feature should be enabled automatically by the installer, but if it isn't, please check out this article [<https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/quick-start/enable-hyper-v?redirectedfrom=MSDN>].

D. (4p) Heroku CLI Installation

In order to interact with Heroku through the command line, you must first install **Heroku CLI** by following this guide [<https://devcenter.heroku.com/articles/heroku-cli>].

E. (4p) Heroku Deployment

Before moving forward, make sure that the **Docker Engine is started** and you are **logged in** with **Heroku**, **Heroku Registry** (the official Heroku repository for Docker images) and **Docker Hub**. You can do that by running these commands:

```
heroku login
heroku container:login
docker login
```

Run the **cloud deployment script** with your **cs.curs username as argument** .

Please note that the script removes all the existing Docker images on your system before pulling the assignment images and after your tagged images are pushed to Heroku. If you want to bypass these steps, comment out the following command:

```
docker rmi -f $(docker images -aq) 2> /dev/null
```

Linux/MacOS:

```
./deploy_dockers.sh vlad.stefanescu
```






Windows:

```
deploy_dockers.bat vlad.stefanescu
```

Make sure you don't misspell your username. 😊

The script will:

1. Create 5 applications in your Heroku account [<https://dashboard.heroku.com/apps>]. In the end, your dashboard should look similar to this one:

Personal ▾		New ▾
<input type="text" value="Filter apps and pipelines"/>		
	vladstefanescu-worker-asia-0	container · Europe ★
	vladstefanescu-worker-asia-1	container · Europe ★
	vladstefanescu-worker-emea-0	container · Europe ★
	vladstefanescu-worker-us-0	container · Europe ★
	vladstefanescu-worker-us-1	container · Europe ★

2. Remove any existing Docker image
3. Pull the Docker images containing the workers from Docker Hub [<https://hub.docker.com/u/vstefanescu96>]
4. Tag (rename) the images according to the Heroku requirements
5. Push the tagged images in the Heroku Registry
6. Run the workers from the pushed images
7. Remove the pulled images

If any of the above steps fails, you can comment out parts of the script after you figured out what the problem was. You don't have to rerun the entire script. It is there just to help you automate the deployment process.

F. (2p) Local Deployment

Run the **local deployment script** with your **cs.curs username as argument**.

Linux/MacOS:

```
./run.sh vlad.stefanescu
```

Windows:

```
run.bat vlad.stefanescu
```

Make sure you don't misspell your username. 😊

The script will:

1. Pull the Docker image containing the **forwarding unit** if it's run for the first time
2. Start the **forwarding unit** on <http://localhost:5000> [<http://localhost:5000>]

Make sure the **port 5000** is **available**!

Now, the **forwarding unit** is ready to receive **GET** requests in the following endpoints:

- **/work**
- **/work/<string:region>**
- **/work/<string:region>/<int:machine>**

(40p) II. Implementation

(30p) A. Solution

Write the **command unit** component of the **load balancer**. Your solution should try various ways of calling the exposed endpoints of the **forwarding unit** depending on the number of requests your system must serve. For instance, if you only have 10 requests, you might get away by just calling a certain endpoint, but if this number increases, then you might want to try something more complex.

The number of requests your system should serve is not imposed, but you should definitely try a sufficiently large range of request batches in order to properly evaluate your policies. Choosing a relevant number is part of the task. 😊

You should have **at least 5** load balancing policies!

Response Object

As stated above, the **JSON** format was chosen for the response of a request. Before writing a single line of code, you should check the content of these responses to know what information you have at hand.

Heroku Logs

You might find relevant information about the response times of the workers by looking at the logs. You can use it to establish a load balancing policy or evaluate your solution.

```
heroku logs -a <application_name> [--tail]
```

Heroku Free Tier

Because we are using the **free tier** of Heroku, the workers will become **idle** if no request is received throughout a period of **30 minutes**. That's great because you don't want to waste your free hours, but please be aware that when a worker is woken up it would take an abnormal amount of time to respond to the first request. You should handle this situation in your implementation.

Moreover, the free tier allows you to keep the workers up and running for **550 hours in a month, excluding the idle time**. This limit is considered across all running services, so because 5 workers are deployed, then you actually have **110 hours** available for the workers in a single month. Implementing and testing your solution within this limitation is part of the assignment. 😊

You can check your remaining hours here [<https://dashboard.heroku.com/account/billing>].

(10p) B. Bonus

Deploy your solution in a **Docker image** and make sure it can be run with a **runtime argument** representing the number of requests your system should serve. The container you created should be able to communicate with the **Forwarding Unit**.

(40p) III. Evaluation

(15p) A. System Limits Analysis

Before implementing your own load balancing policies, you should first analyse the **limits of the system**:

- How many requests can be handled by a single machine?
- What is the latency of each region?
- What is the computation time for a work request?
- What is the response time of a worker when there is no load?
- What is the latency introduced by the **forwarding unit**?
- How many requests must be given in order for the **forwarding unit** to become the bottleneck of the system? How would you solve this issue?
- What is your estimation regarding the latency introduced by Heroku?
- What downsides do you see in the current architecture design?

Your observations should be written in the **Performance Evaluation Report** together with some **relevant charts** (if applicable).

(25p) B. Load Balancing Policies Comparison

Compare your **load balancing policies** for a relevant range of request batch sizes and write your observations in the **Performance Evaluation Report** file together with some **relevant charts**

(10p) IV. Documentation

You should write a high quality **Performance Evaluation Report** document which:

- should explain your **implementation** and **evaluation** strategies
- present the **results**
- can have a **maximum of 3 pages**
- should be readable, easy to understand and aesthetic
- on the **first page** it should contain the following:
 - your name
 - your group number
 - which parts of the assignment were completed
 - what grade do you consider that your assignment should receive

8. Assignment Upload

The **solution archive** (.zip) should only contain:

- the **Python modules** used in the implementation

- a **requirements.txt** file to easily install all the necessary **pip dependencies**
- a **Dockerfile** used for easily deploying the command unit (**bonus**)
- a **Performance Evaluation Report** in the form of a **PDF** file

The assignment has to be uploaded **here** [<https://curs.upb.ro/2021/mod/assign/view.php?id=85256>] by **23:55 on December 6th 2021**. This is a **HARD deadline**.

Questions regarding the assignment should be addressed **here** [<https://curs.upb.ro/2021/mod/forum/view.php?id=85261>].

If the submission does not include the Report / a Readme file, the assignment will be graded with ZERO!

To emphasise this, we are writing it again in bold:

If the submission does not include the Report / a Readme file, the assignment will be graded with ZERO!

ep/teme/01.txt • Last modified: 2021/11/24 11:53 by cezar.craciunoiu