Institut Polytechnique de Paris

M2 Parallel and distributed systems

Logo

Description automatically generated

Cloud Infrastructures CS5004

**Scaling Horizontally a Web Service**

Authors:  
Jana Toljaga  
Timothée Zerbib  
Aleksandar Maksimovic  
Dimitrije Panic

Content

[1 Introduction 3](#_Toc124080679)

[2 The Pdfmagic Service 3](#_Toc124080680)

[2.1 *Overview* 3](#_Toc124080681)

[2.2 *A Container To Hold The Magic* 3](#_Toc124080682)

[3 Using a Container Registry 4](#_Toc124080683)

[4 Deploying The Service 5](#_Toc124080684)

[5 Evaluating The Pdf Magic Service 5](#_Toc124080685)

[5.1 *A synthetic workload* 5](#_Toc124080686)

[5.2 *Multiple Clients* 6](#_Toc124080687)

[6 Scaling Out The Service 6](#_Toc124080688)

# Introduction

During this lab session we firstly containerized the Pdfmagic service which is used for converting images to pdf files. Afterwards, we deployed the container in Kubernetes. Then we evaluate the performance of the Pdf Magic Service with different loads – a single client with multiple requests, multiple clients with multiple requests. After understanding the initial service load, we scaled the service to try to improve its responsiveness.

# The Pdfmagic Service

### *Overview*

In this section we took a closer look at the implementation of the Pdfmagic service. It’s important to note here that the service gives a unique and random identifier to each request (uuid). Also, it depends on two external libraries – delegator and webpy. We need to take this into account when writing the dockerfile.

*[Q]* Quickly browse through the source code of the Pdfmagic service. Where is located the pdf document on the server before being sent back to the client?

The pdf document will be located in **/tmp** folder.  
The document will be named **<uuid>.jpg.**

### *A Container To Hold The Magic*

In this section we wrote the dockerfile and ran it locally.

The ***Dockerfile*** we used is shown below.

Text

Description automatically generated

We used python:2 as the base image. Because the container is an isolated unit of computation, we had to add commands to install the external libraries the Pdfmagic service depends upon. Enabling the access to our container from the outside we exposed its port (8080). At the end of the file we wrote the command that will be executed in the container which runs the pdfmagic.py python script.

Then, we can create the Docker image from the dockerfile adding the tag **pdfimage**.



Finally, we can start the container, making a bridge between host and containers ports 8080.

A picture containing shape

Description automatically generated

[Q] Run the container locally and access it to create a pdf document from a jpg image. Convert the logo located here. Where is the pdf document on the server?

Once the service is used from a container, the pdf document will be in the **/tmp** folder of the container.

# Using a Container Registry

Now that we have created the container image for the Pdfmagic service we are going to publish it on the DockerHub. We published as jacikot/pdfmagic which can be seen on the picture below. Our image is now publicly available.

Graphical user interface, application, Teams

Description automatically generated

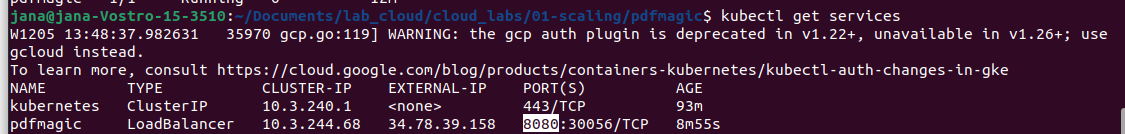
# Deploying The Service

Next, we want to deploy our container in Kubernetes. This is important as we can use Kubernetes to scale our containerized application.

Graphical user interface

Description automatically generated with medium confidenceFirstly, we need to create a pod for our container, as pod is the smallest deployable unit of computation in Kubernetes. We used the image we created in the last section. For this we used the pdfmagic.yml given below.

Now, we have to expose our pod to create a service so our pod can be accessed from the outside. We have to run the command kubectl get services to know the external ip address of the service. The out we get is given in the image below.



# Evaluating The Pdf Magic Service

In this section we evaluate the performance of the previously deployed service. Firstly, for one client, and afterwards for ten clients using client.sh and parallel.sh scripts respectively.

### *A synthetic workload*

Chart

Description automatically generatedWith one client and 100 request we get that the average response time is: **1.22699s**. While the time distribution is given on the plot below.

The average service response time is very high, which is probably because of the internet connection and time needed to convert a .jpg to .pdf. However, for almost all the requests the time response time is similar so there is no degree of degradation in the responsiveness. However, only the first couple of requests took longer which is probably because of the slow start of the containers.

### *Multiple Clients*

With 10 clients each sending 100 requests in parallel we start to notice worse performances. The parallel script ran for approximately 46min. The result is given as a plot showcasing the minimum, maximum and average latency time of each request.

Chart

Description automatically generated

We conclude that our service is incapable of handling multiple clients accessing the service at the same time.

# Scaling Out The Service

As we have previously seen our system needs to be upgraded in order to try to solve the issue of multiple clients accessing it at the same time. We try to accomplish this by increasing the number of pods – horizontally scaling the system. For this we use the deployment.yml given at the next page. We have increased the number of pods running in parallel (from 1 to 3) by adding replicas. Also, we use the same image as before.

Graphical user interface

Description automatically generated with medium confidence

After creating the deployment, exposing it and finally accessing it with 10 clients each sending 100 requests we get the result given below.

Chart, histogram

Description automatically generated

As multiple clients can see their request served at the same time, the average and minimum latency significantly decreased. However the maximum latency is still high as the number of request highly outnumbers the number of available pods. To conclude, horizontal scaling did not satisfy the expectations. This is expected as 3 pods running in parallel is not enough for 10 clients.

***[Q]*** *Analyze the performance of a single instance of Pdfmagic. Do you think that the idea of scaling-out the service was the right one? What is the underlying problem with the implementation of Pdfmagic?*

No. Scaling-out the service was not the right decision. This is because the Pdfmagic can only process one request at a time. Meaning it receives a request, and then converts it. During this time it is blocked.