Hands On 1:
Service as unit of construction

María Fernanda Flores Luna
ID. 162064

Cloud Computing and Big Data
LIS 4102-1
20th february 2022

Spring 2022

Index

Introduction

The objective of the laboratory of the course is to play with different cloud solutions and understand its purposes and motivations. The first hands-on presented the opportunity to interact with two different services on the cloud: Google Collab and Microsoft Azure.

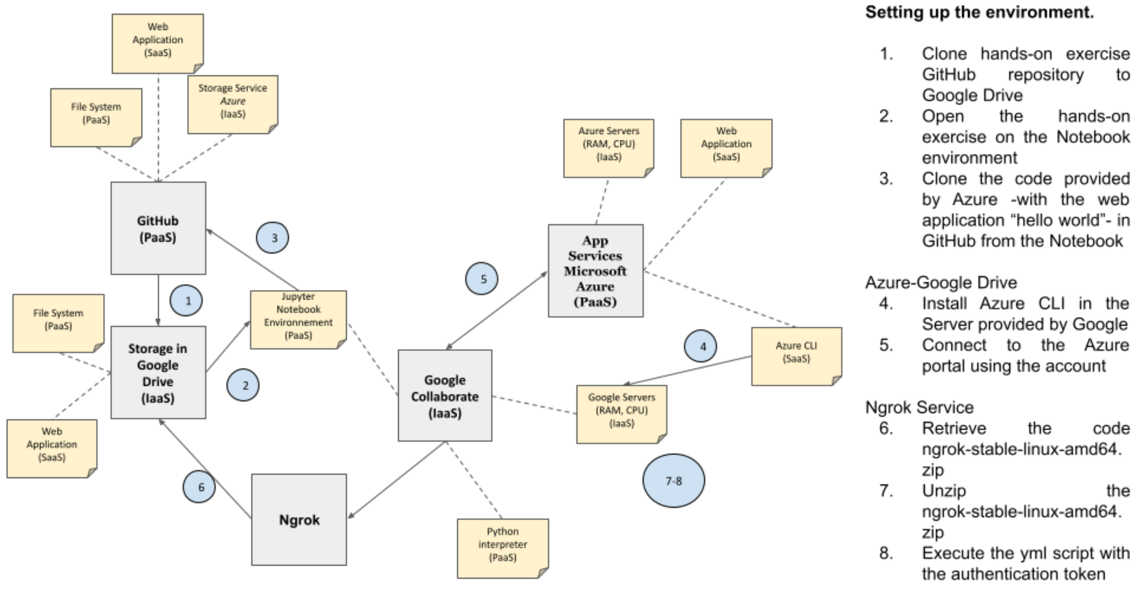
- ★ Google Collab was used as a virtual machine to avoid the installation and the limitations of our hardware.
- ★ Microsoft Azure was used as a platform to launch our web application.

In the following report we are making an attempt to explain the interactions that happened in the experiment between the services and how they made it possible.

Functional Architecture

In this section, we are going to show how the different cloud services interact for building the application using a functional architecture schema. Creating a functional architecture schema means: to draw boxes and arrows that represent the interaction between the elements during the execution of the application. For the schemas, the experiment was segmented into 4 parts: setting up the environment, running the web application on Google Collab, the deployment of the application on Microsoft Azure and running the web application on Microsoft Azure.

The first stage is setting up the environment. This stage includes everything to prepare the services before running the application. The functional architecture can be seen in figure 1. The diagram was designed using the instructions given in class.


Web Application (SaaS)
File System (PaaS)
Storage Service Azure (IaaS)
GitHub (PaaS)
3
1
File System (PaaS)
Jupyter Notebook Environnement (PaaS)
Storage in Google Drive (IaaS)
2
Web Application (SaaS)
6
Ngrok
Azure Servers (RAM, CPU) (IaaS)
Web Application (SaaS)
App Services Microsoft Azure (PaaS)
5
Google Collaborate (IaaS)
4
Azure CLI (SaaS)
Google Servers (RAM, CPU) (IaaS)
7-8
Python interpreter (PaaS)

Setting up the environment.

1. Clone hands-on exercise GitHub repository to Google Drive
2. Open the hands-on exercise on the Notebook environment
3. Clone the code provided by Azure -with the web application "hello world"- in GitHub from the Notebook

Azure-Google Drive

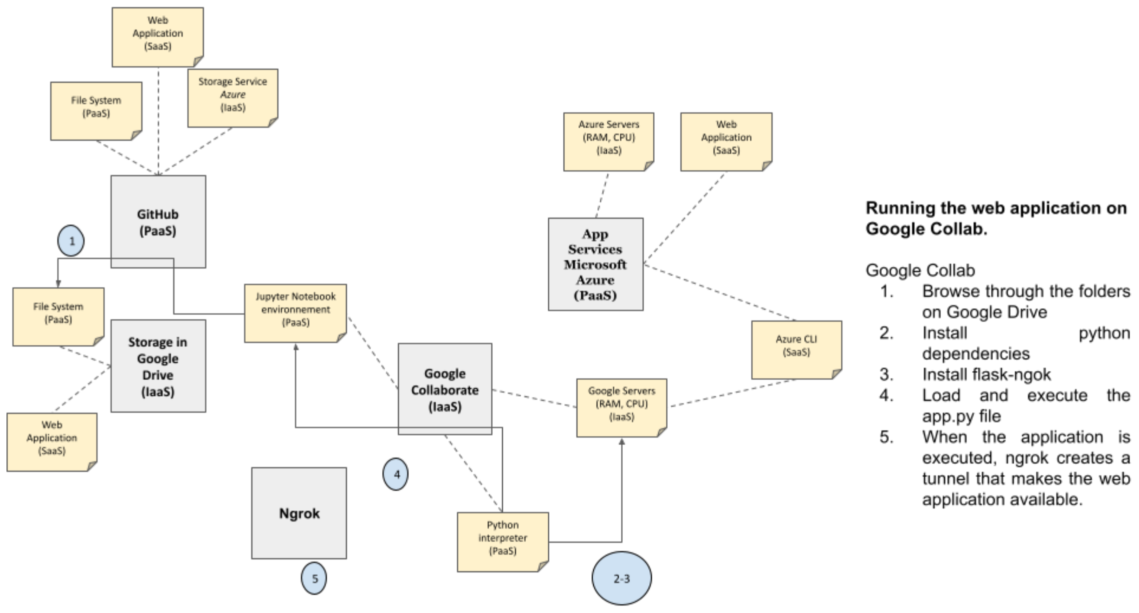
4. Install Azure CLI in the Server provided by Google
5. Connect to the Azure portal using the account

Ngrok Service

6. Retrieve the code ngrok-stable-linux-amd64.zip
7. Unzip the ngrok-stable-linux-amd64.zip
8. Execute the yml script with the authentication token

Figure 1. Functional architecture of setting up the environment.

The second stage is running everything on Collab. This stage includes from the installation of python libraries to the application running. The functional architecture can be seen in figure 2. It is important to highlight that the arrows that flow from the python interpreter mean that the command is written in Python and executed later.



Web Application (SaaS)
File System (PaaS)
Storage Service Azure (IaaS)
GitHub (PaaS)
1
File System (PaaS)
Storage in Google Drive (IaaS)
Web Application (SaaS)
Jupyter Notebook environnement (PaaS)
Azure Servers (RAM, CPU) (IaaS)
Web Application (SaaS)
App Services Microsoft Azure (PaaS)
Azure CLI (SaaS)
Google Collaborate (IaaS)
Google Servers (RAM, CPU) (IaaS)
4
Ngrok
5
Python interpreter (PaaS)
2-3

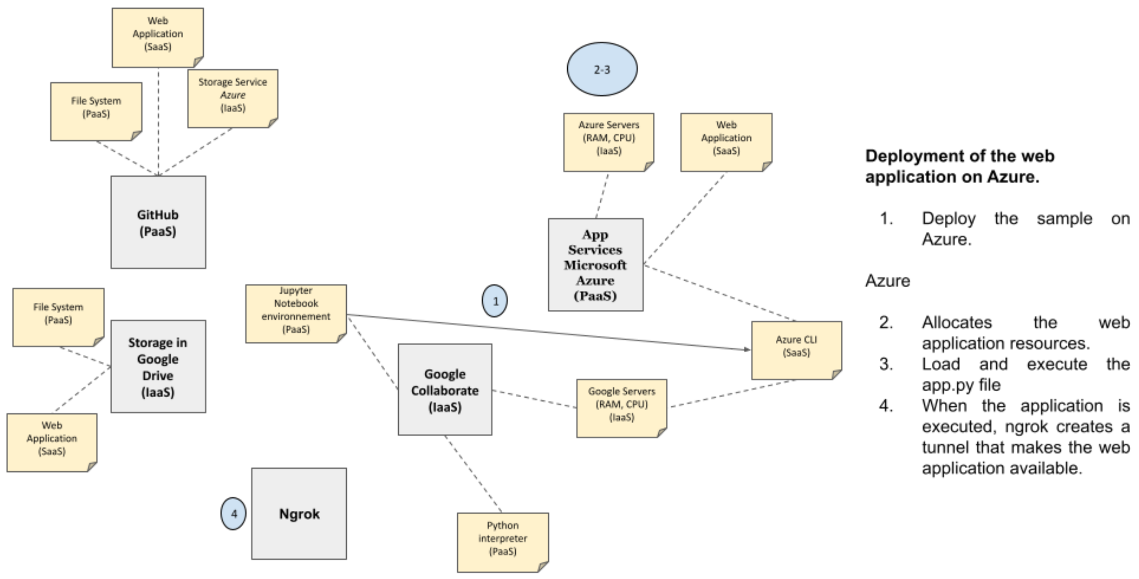
Running the web application on Google Collab.

Google Collab

1. Browse through the folders on Google Drive
2. Install python dependencies
3. Install flask-ngok
4. Load and execute the app.py file
5. When the application is executed, ngrok creates a tunnel that makes the web application available.

Figure 2. Functional architecture of running the web application on Google Collab.

The third stage is deploying the application on Azure. This stage includes the operations executed on Collab and on Azure. The functional architecture can be seen in figure 3.



Web Application (SaaS)
File System (PaaS)
Storage Service Azure (IaaS)
GitHub (PaaS)
File System (PaaS)
Storage in Google Drive (IaaS)
Web Application (SaaS)
Jupyter Notebook environnement (PaaS)
2-3
Azure Servers (RAM, CPU) (IaaS)
Web Application (SaaS)
App Services Microsoft Azure (PaaS)
1
Azure CLI (SaaS)
Google Collaborate (IaaS)
Google Servers (RAM, CPU) (IaaS)
4
Ngrok
Python interpreter (PaaS)

Deployment of the web application on Azure.

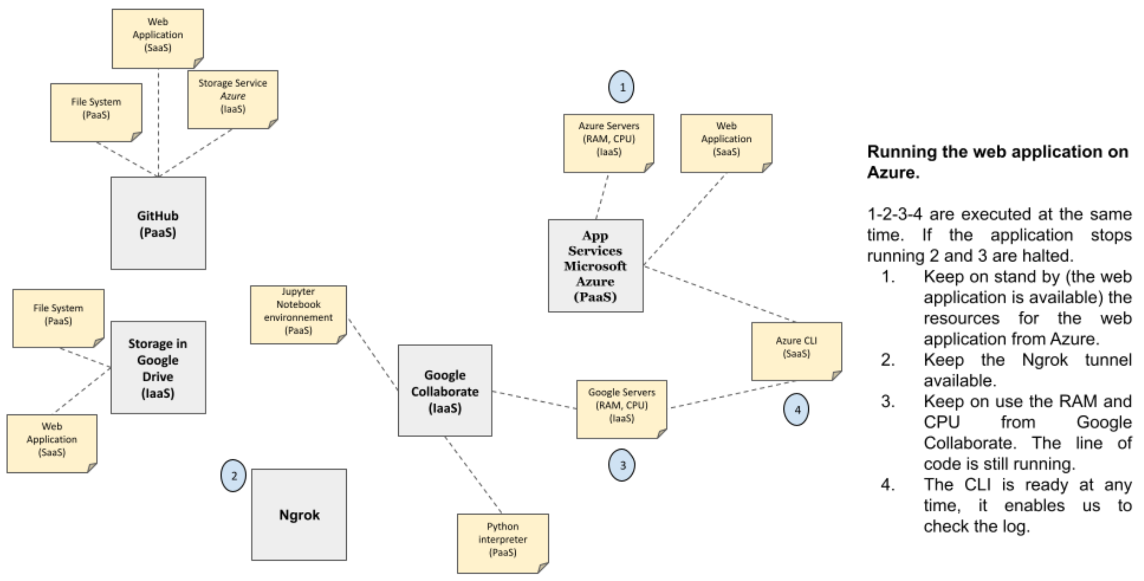
1. Deploy the sample on Azure.

Azure

2. Allocates the web application resources.
3. Load and execute the app.py file
4. When the application is executed, ngrok creates a tunnel that makes the web application available.

Figure 3. Functional architecture of deployment of the web application on Azure.

The final stage is running the application on Azure. This stage is the usage of resources from every active service. The functional architecture can be seen in figure 4.


Web Application (SaaS)
File System (PaaS)
Storage Service Azure (IaaS)
GitHub (PaaS)
File System (PaaS)
Storage in Google Drive (IaaS)
Web Application (SaaS)
Jupyter Notebook environnement (PaaS)
2
Ngrok
Google Collaborate (IaaS)
Python interpreter (PaaS)
1
Azure Servers (RAM, CPU) (IaaS)
Web Application (SaaS)
App Services Microsoft Azure (PaaS)
Azure CLI (SaaS)
Google Servers (RAM, CPU) (IaaS)
3
4

Running the web application on Azure.

1-2-3-4 are executed at the same time. If the application stops running 2 and 3 are halted.

1. Keep on stand by (the web application is available) the resources for the web application from Azure.
2. Keep the Ngrok tunnel available.
3. Keep on use the RAM and CPU from Google Collaborate. The line of code is still running.
4. The CLI is ready at any time, it enables us to check the log.

Figure 4. Functional architecture of running the web application on Azure.

Once the diagrams have been presented, let us talk about the type of services they represent. The first step to do that is analyzing the types of services, table 1 presents a list of common services and their type.

Service	Type
Compilers	**Platform as a Service (PaaS).**
DBMS	
Operating System	
Web application	**Software as a Service (SaaS).**
BlackBoard	
App that gives access to the backend of a system	
Storage services	**Infrastructure as a Service (IaaS)**

Table 1. Common services and their type.

After analyzing table 1 it is possible to categorize the services used during the experiment. Table 2 presents the service and their type, for better visualization the color purple represents Google Clouds services and color blue Microsoft Azure services.

	Service	Type
Google Cloud	Google Drive	Platform as a Service (PaaS).
	Jupyter Notebook	
	Python Interpreter	
Microsoft Azure	File System	
	CLI	
	Web App	
Google Cloud	Storage	Infrastructure as a Service (IaaS).
	CPU+RAM	
Microsoft Azure	Storage	
	CPU+RAM	

Table 2. Identifying the experiment components and their type of service.

Process diagrams

In this section, we are going to show the processes that were executed during the experiment using process diagrams. The process diagrams that are going to be presented next do not follow a specific regulation, they are created using a basic nomenclature to explain what was done.

There are 5 process diagrams, each one represents a step of the experiment. Since there is no general diagram to explain how each diagram is related to the others, the processes are going to be presented in a sequential manner.

Step 1. Ngrok authentication. This process is presented on figure 5, it includes the actions executed by the systems of ngrok and the ones taken manually by the user and on Collab.

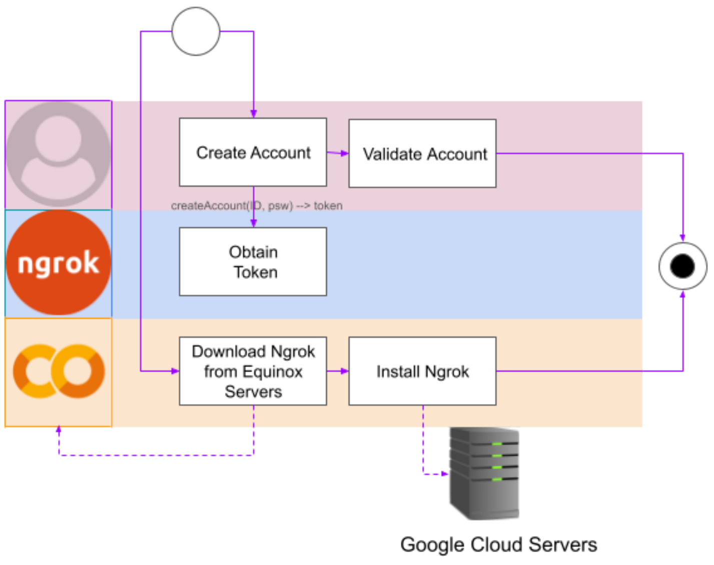


Figure 5. Process diagram of Ngrok authentication.

Step 2. Installing the Azure CLI and configuring it. This process is presented on figure 6, it includes the actions executed by the systems of Microsoft Azure and the ones taken manually by the user and on Collab.

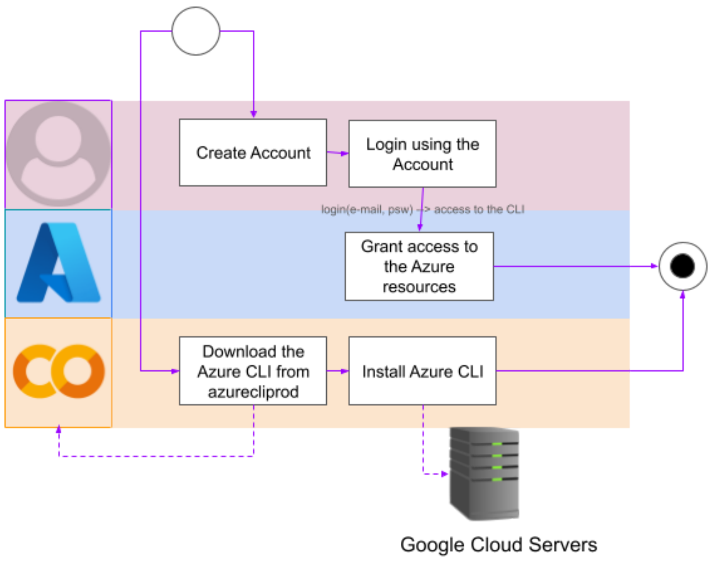

Create Account
Login using the Account
login(e-mail, psw) --> access to the CLI
Grant access to the Azure resources
Download the Azure CLI from azurecliprod
Install Azure CLI
Google Cloud Servers

Figure 6. Process diagram of Installing Azure CLI and its configuration.

Step 3. Cloning from GitHub and configuring Python. This process is presented on figure 7, it includes the actions executed by the systems of GitHub and the ones taken by the user on Collab.

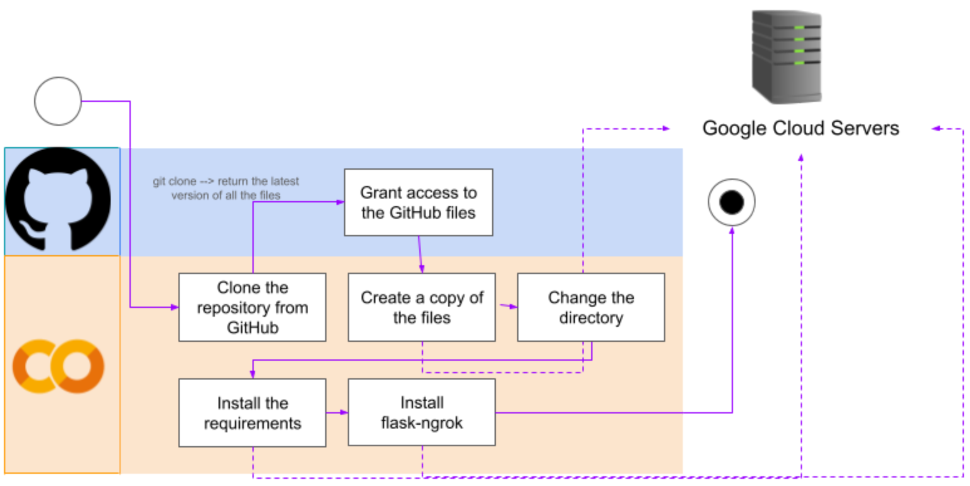

Google Cloud Servers
git clone --> return the latest version of all the files
Grant access to the GitHub files
Clone the repository from GitHub
Create a copy of the files
Change the directory
Install the requirements
Install flask-ngrok

Figure 7. Process diagram of Cloning from GitHub and configuring Python.

Step 4. Running the sample on Collab. This process is presented on figure 8, it includes the actions executed by the systems of ngork, flask and the ones taken manually by the user and on Collab.
Note: there are 2 ways for the process to finish, either the user chooses to stop the execution or leave the program running.

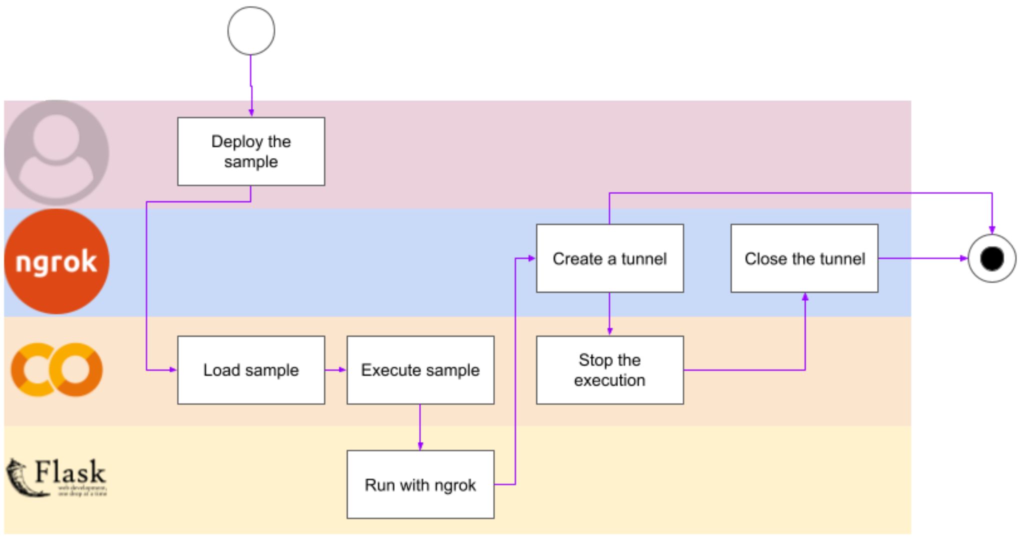


Figure 8. Process diagram of Running the sample on Collab.

Step 5. Running the sample on Azure. This process is presented on figure 9, it includes the actions executed by the systems of ngork, flask, Microsoft Azure and the ones taken manually by the user and on Collab.

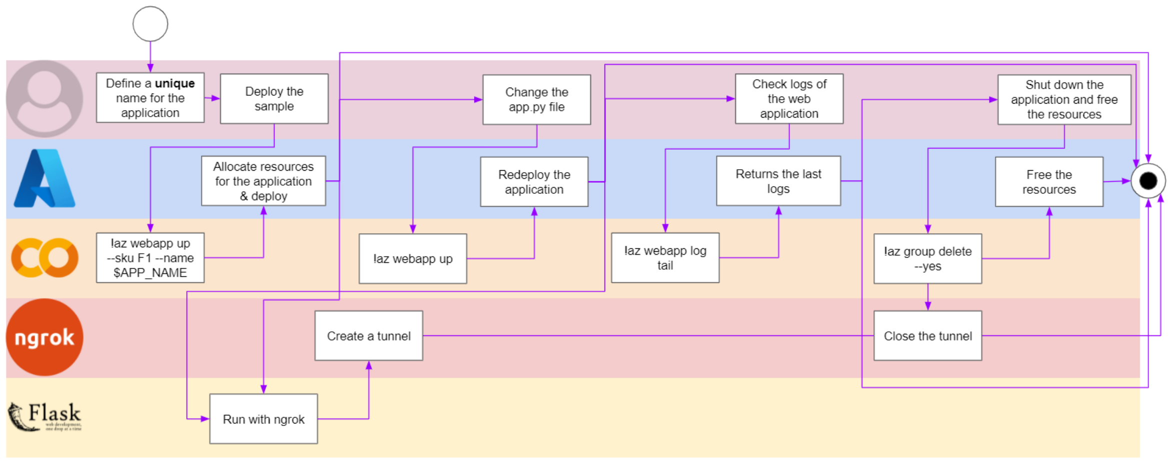

Define a unique name for the application
Deploy the sample
Change the app.py file
Check logs of the web application
Shut down the application and free the resources
Allocate resources for the application & deploy
Redeploy the application
Returns the last logs
Free the resources
!az webapp up --sku F1 --name $APP_NAME
!az webapp up
!az webapp log tail
!az group delete --yes
Create a tunnel
Close the tunnel
Run with ngrok

Figure 9. Process diagram of Running the sample on Azure.

Comparison

In this section, we are going to compare the experiment performed in the laboratory sessions and a tutorial by Azure. For this part we are going to use a comparison table (see table 3).

Lab Tutorial	Azure Tutorial
The web application source code is retrieved from GitHub.	
The requirements installation is performed on the cloud (Google Collab).	The requirements installation is performed locally (VS Code in your Windows machine).
Since we are working in the cloud, there is no need for a virtual environment.	We need a virtual environment, we create one using the following commands: `py -m venv .venv` `.venv\scripts\activate`
To monitor the http traffic we use ngrok. We installed a middleware-like application: flask.ngrok.	There is no tool to monitor http traffic.
Both of the web applications are created using Flask.	
The Azure environment is set up from the CLI. The only thing done using the Azure portal is the identity verification.	The Azure environment can be set up either from the Azure portal, VS Code or the CLI.
The stream logs were accessed briefly from the Azure CLI.	The final part of the tutorial includes an introduction on stream logs.

The clean up of the resources is performed from the Azure CLI.	The clean up of the resources can be done either from the Azure portal, VS Code or the CLI.
The application includes a text message.	The application includes Azure logo, a text message, a text input bar and a button.

Table 3. Comparison table between the performed experiment and the one suggested by Azure.

Conclusions

The free trial or limited accounts are useful for cloud solutions to present their services to possible clients, you can try if it is the best choice for you before starting to pay for the service!

Cloud solutions erase the limitations of regular computers, for example using IaaS you can have a computer with a limited processor and memory running complex applications.

Glossary

Azure: is a cloud oriented service by Microsoft. It was built to let users have a ‘hands-on’ experience with data centers- building, testing, deploying and managing applications and services.

Command Line Interface (CLI): it is an application that allows the users to ask specific tasks from the computer using short commands.

Cloud: is making available computing services -storage, servers, databases, networking, etc- through the Internet.

Flask: is a web framework written in Python, since it does not require any particular tools or extra libraries it is possible to call them microframework. Flask allows the programmer to develop web applications fast and as simple as possible

Magic: is an informal term for abstraction. It mainly refers to the code that handles complex tasks offering a high level of transparency to the user, making it seem as if it were easy.

Ngrok: is an authentication support that allows the application to perform requests on the web.

Server: in client-server architecture a server is an entity - hardware or software - that answers the requests of a client. In other words, a server manages a centralized service.

Notebook: it is an interactive digital environment created to enable the creation, development, documentation and execution of Julia/Python/R programs.