Universidad de las Américas Puebla
Cloud Computing and Big Data
LIS4102-1

Hands-On:

Services as a unit of construction

Author:
Fernando Nieto Morales (ID: 163930)

Professor:
Genoveva Vargas Solar

Introduction

With the development of Cloud Computing, different services through Internet has been deployed to let an effective connection between different applications and databases to work. The XaaS models (*X* as a Service) refers to the delivery of any as a service through the network. The most common XaaS are:

- SaaS (Software as a Service): Software distribution model where a third-party provider hosts any application and allows the customers to use them.
- PaaS (Platform as a Service): Model where a third-party provider hosts application development platforms and tools on its own infrastructure and the customers to use them.
- IaaS (Infrastructure as a Service): Model where a third-party provider hosts servers, storage and resources and allows the customers to use them. (Bigelow, 2017)

To understand how to use the IaaS ans Paas, the main objective of this Hands-On is to deploy a Python web app to App Service on Linux, Azure's highly scalable, self-patching web hosting service (PaaS service). This will be hosted on Google Colab (IaaS) and will require the functional architecture of Figure 1 to work.

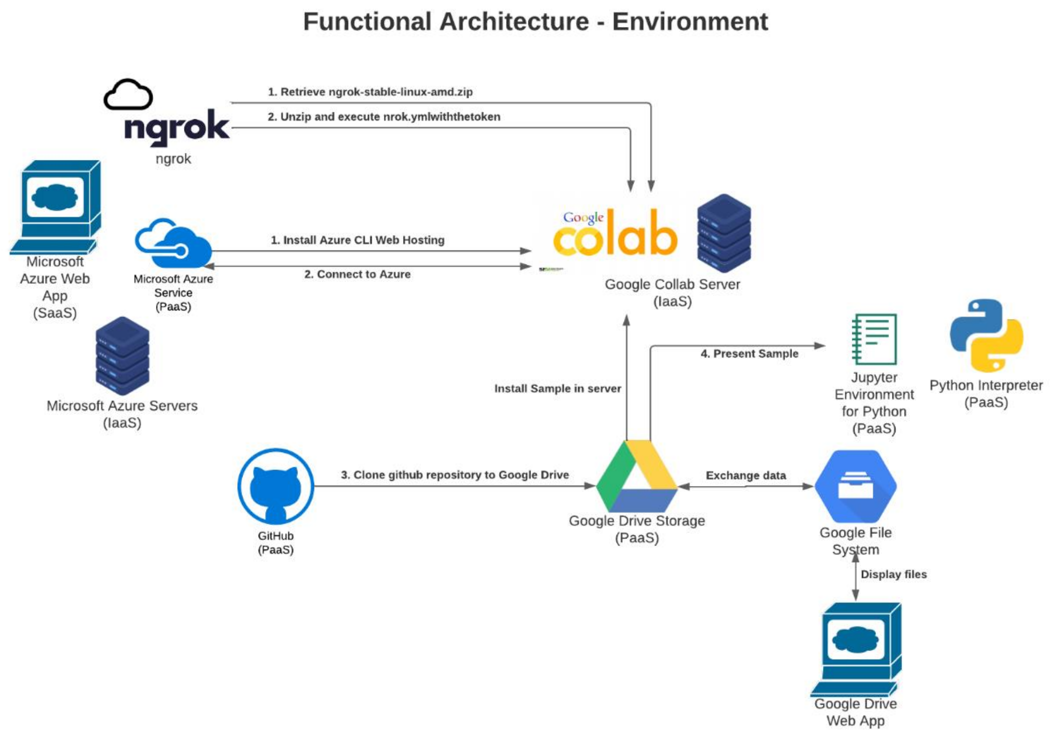

Functional Architecture - Environment
ngrok
1. Retrieve ngrok-stable-linux-amd.zip
2. Unzip and execute nrok.ymlwiththetoken
Microsoft Azure Web App (SaaS)
Microsoft Azure Service (PaaS)
1. Install Azure CLI Web Hosting
2. Connect to Azure
Google colab
Google Collab Server (IaaS)
Microsoft Azure Servers (IaaS)
Install Sample in server
4. Present Sample
Jupyter Environment for Python (PaaS)
Python Interpreter (PaaS)
3. Clone github repository to Google Drive
GitHub (PaaS)
Google Drive Storage (PaaS)
Exchange data
Google File System
Display files
Google Drive Web App

Figure 1: Functional architecture of Hands-On 1.

The previous architecture (Figure 1) is similar to the one provided by the Azure tutorial. The main difference between these is that the hands-on is completely externalized from the local machine. Instead of using a local machine application (VSCode & AzureTools) as the Azure tutorial mentions, this will be done through the Google Colab infrastructure service. Furthermore, to know the main services provided in the functional architecture, the next Figures represents the implementations done through the used services:

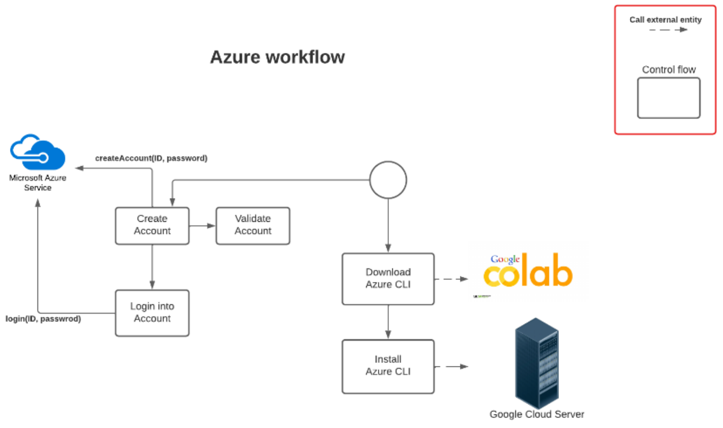


Figure 2: Azure workflow. Create account; Validate account; Login; Download & Install Azure CLI into the server

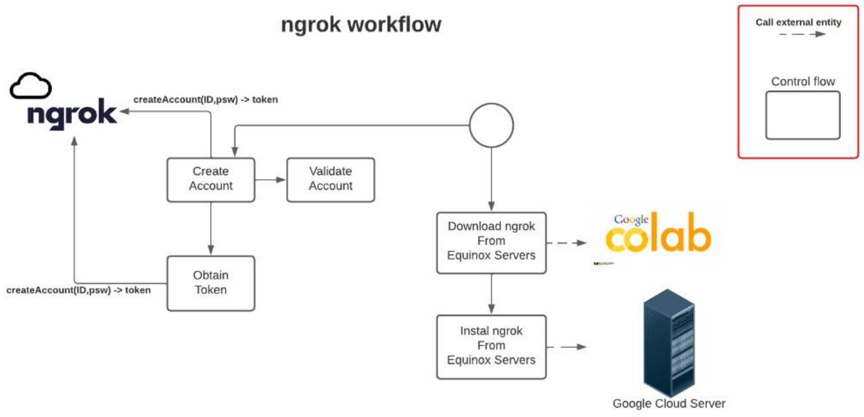

ngrok workflow
Call external entity
Control flow
ngrok
createAccount(ID,psw) -> token
Create Account
Validate Account
Obtain Token
createAccount(ID,psw) -> token
Download ngrok From Equinox Servers
Instal ngrok From Equinox Servers
colab
Google Cloud Server

Figure 3: ngrok workflow: Create an account; Validate Account; Obtain token; Download & Install ngrok into the server

Methodology

The complete process can be divided into the next steps:

1. Create a Google Colab account and retrieve the data located in the next repository: https://gist.github.com/javieraespinosa/8d9e93cff61c675d74bd862975a84d5e
2. Install Azure CLI into Google Collab and check that it was correctly installed by checking the version.
3. Login into the Microsoft Azure account.
4. Install ngrok into Google Colab and enter the authtoken.

5. Clone the code sample given by the tutorial.
6. Install the code requirements.
7. Run the code sample and launch it through a web application. Get the URL of the application and check that is working correctly.
8. Check the log tail to see the recent requests.
9. Delete the resources from the application.

As a general visualization of the workflow, we have the next Figure:

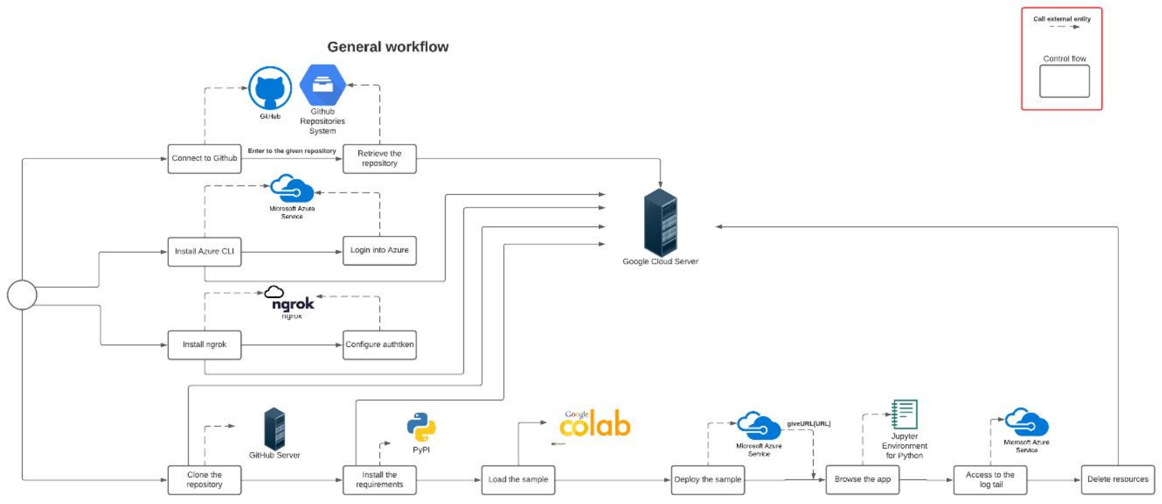


Figure 4.1: Process diagram of the Hands.-on.

This process was created from an Azure tutorial that works in a local machine instead of an externalized manner. This has a workflow that requires the usage of less services but cannot provide a deployment out of the local machine. The tutorial works in the next process:

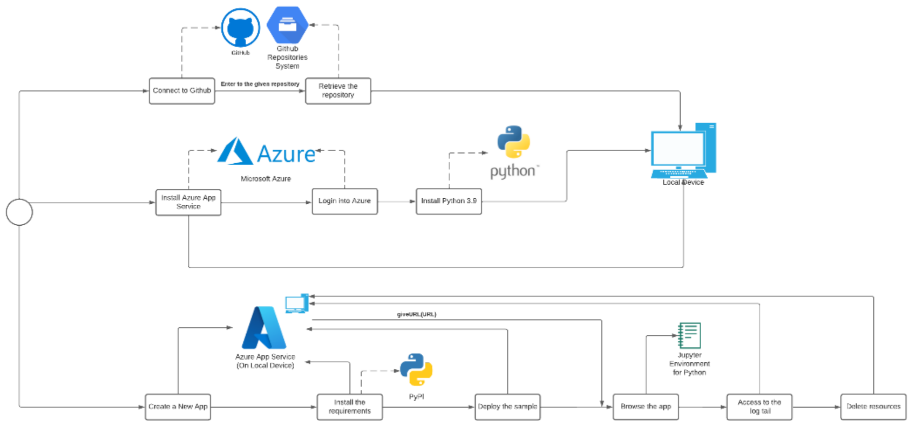


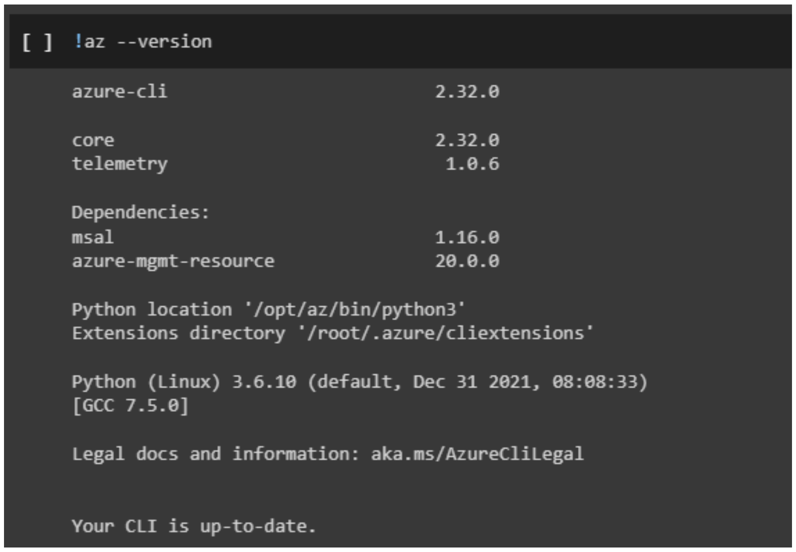
Figure 4.2: Process diagram of the Azure Tutorial.

As a comparison, the tutorial merely works with installed applications in a local machine, while the externalized version requires the implementation of different services to provide the deployment. The main difference resides in the architecture required: instead of using Ngrok and

Google Colab, it works in a local machine; The Microsoft Azure Service is replaced by its local version, the Azure App Service.

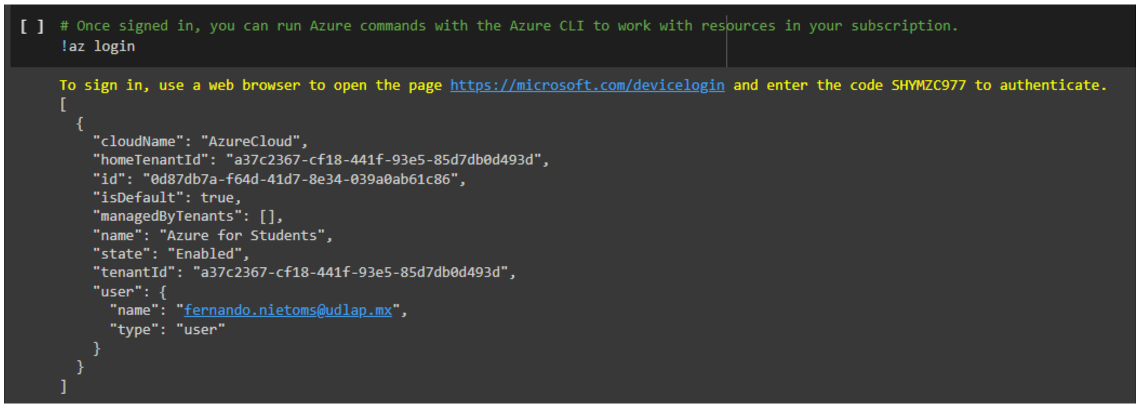
Results

After entering to Google Colab, I installed Azure CLI and checked the installed version (See Figure 5), logged into my account (See Figure 6) and installed ngrok (See Figure 7). Furthermore, it is important to enter the authentication token provided by ngrok (See Figure 8) and clone the sample with its given requirements (See Figure 9 & 10). After loading the sample into the host service (See Figure 11), it is possible to deploy the code as a web application (See Figure 12 &13). We can check that it was correctly deployed by entering the given URL (See Figure 14) and check the log tail to see any request done to the web application. As example of this, we can see on Figure 15 a request done. Finally we can delete the resources of the web application and check that it was correctly erased (See Figure 16 as example after deleting the web application).



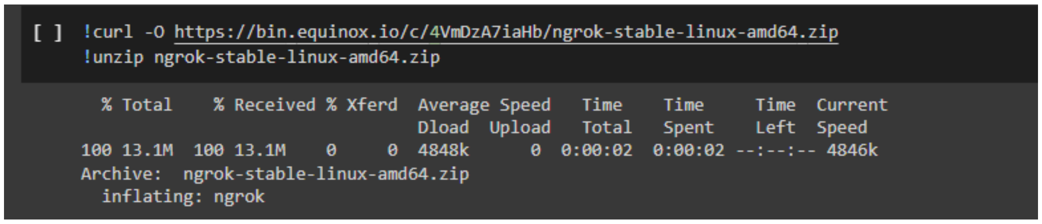
```
[ ]  !az --version

     azure-cli                         2.32.0

     core                              2.32.0
     telemetry                          1.0.6

     Dependencies:
     msal                              1.16.0
     azure-mgmt-resource               20.0.0

     Python location '/opt/az/bin/python3'
     Extensions directory '/root/.azure/cliextensions'

     Python (Linux) 3.6.10 (default, Dec 31 2021, 08:08:33)
     [GCC 7.5.0]

     Legal docs and information: aka.ms/AzureCliLegal


     Your CLI is up-to-date.
```

Figure 5: Check the Azure CLI version.



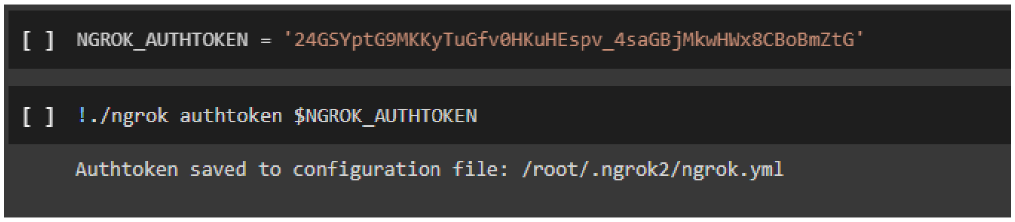
```
[ ]  # Once signed in, you can run Azure commands with the Azure CLI to work with resources in your subscription.
     !az login

     To sign in, use a web browser to open the page https://microsoft.com/devicelogin and enter the code SHYMZC977 to authenticate.
     [
       {
         "cloudName": "AzureCloud",
         "homeTenantId": "a37c2367-cf18-441f-93e5-85d7db0d493d",
         "id": "0d87db7a-f64d-41d7-8e34-039a0ab61c86",
         "isDefault": true,
         "managedByTenants": [],
         "name": "Azure for Students",
         "state": "Enabled",
         "tenantId": "a37c2367-cf18-441f-93e5-85d7db0d493d",
         "user": {
           "name": "fernando.nietoms@udlap.mx",
           "type": "user"
         }
       }
     ]
```

Figure 6: Login into my Microsoft Azure account.



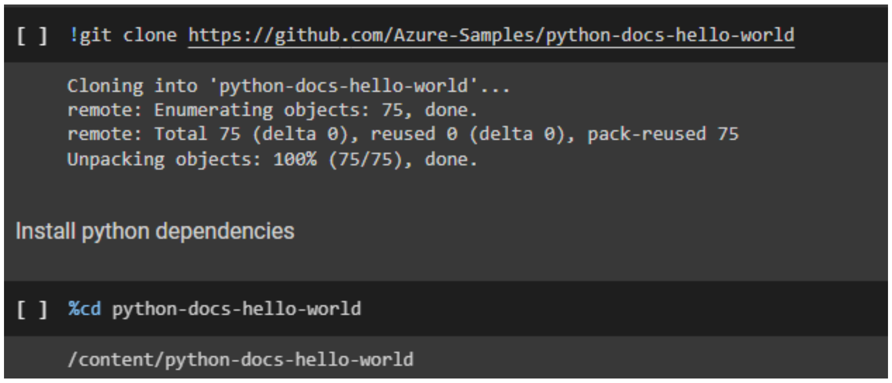
```
[ ] !curl -O https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-amd64.zip
    !unzip ngrok-stable-linux-amd64.zip

      % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                     Dload  Upload   Total   Spent    Left  Speed
    100 13.1M  100 13.1M    0     0  4848k      0  0:00:02  0:00:02 --:--:-- 4846k
    Archive:  ngrok-stable-linux-amd64.zip
      inflating: ngrok
```

Figure 7: Install ngrok.



```
[ ] NGROK_AUTHTOKEN = '24GSYptG9MKKyTuGfv0HKuHEspv_4saGBjMkwHWx8CBoBmZtG'

[ ] !./ngrok authtoken $NGROK_AUTHTOKEN

    Authtoken saved to configuration file: /root/.ngrok2/ngrok.yml
```

Figure 8: ngrok authtoken.

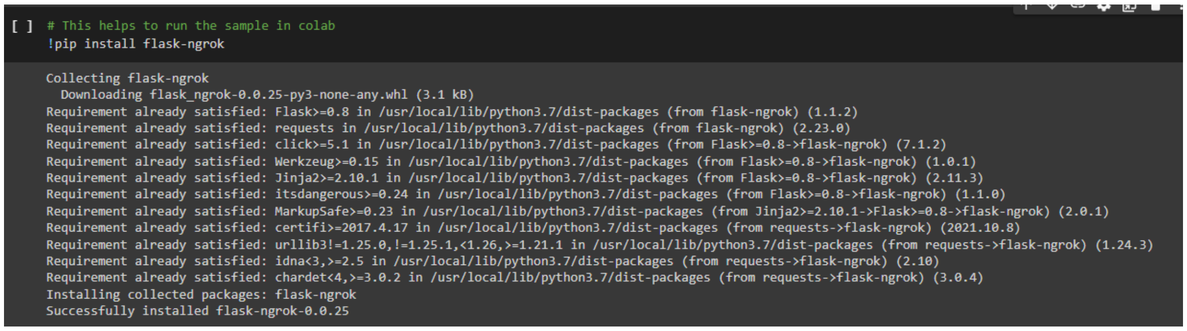


```
[ ] !git clone https://github.com/Azure-Samples/python-docs-hello-world

    Cloning into 'python-docs-hello-world'...
    remote: Enumerating objects: 75, done.
    remote: Total 75 (delta 0), reused 0 (delta 0), pack-reused 75
    Unpacking objects: 100% (75/75), done.
```

Install python dependencies

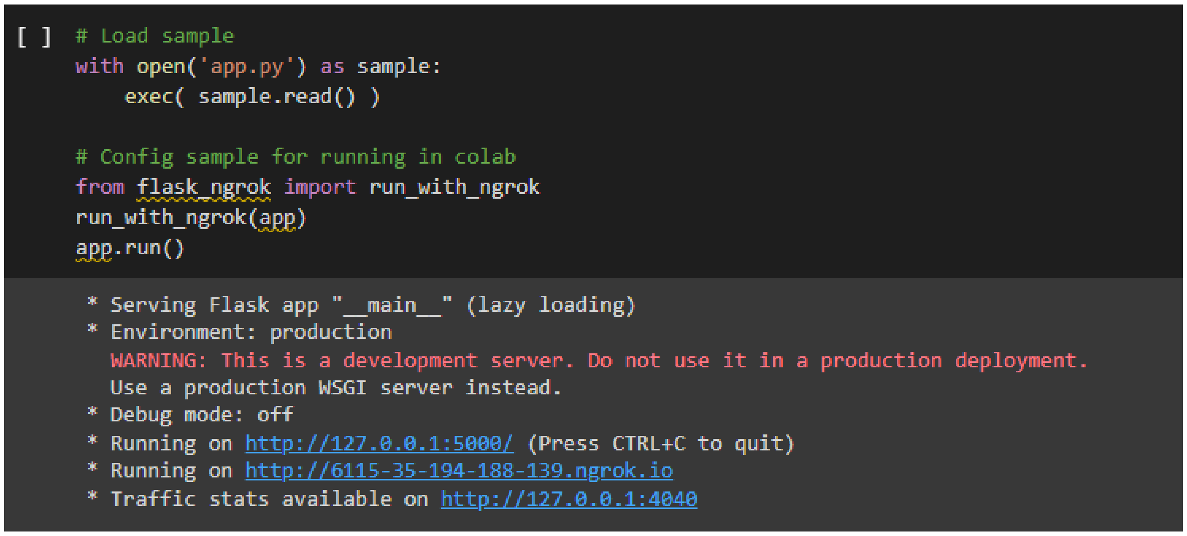
```
[ ] %cd python-docs-hello-world

    /content/python-docs-hello-world
```

Figure 9: Clone the python code sample.



```
[ ] # This helps to run the sample in colab
    !pip install flask-ngrok

    Collecting flask-ngrok
      Downloading flask_ngrok-0.0.25-py3-none-any.whl (3.1 kB)
    Requirement already satisfied: Flask>=0.8 in /usr/local/lib/python3.7/dist-packages (from flask-ngrok) (1.1.2)
    Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from flask-ngrok) (2.23.0)
    Requirement already satisfied: click>=5.1 in /usr/local/lib/python3.7/dist-packages (from Flask>=0.8->flask-ngrok) (7.1.2)
    Requirement already satisfied: Werkzeug>=0.15 in /usr/local/lib/python3.7/dist-packages (from Flask>=0.8->flask-ngrok) (1.0.1)
    Requirement already satisfied: Jinja2>=2.10.1 in /usr/local/lib/python3.7/dist-packages (from Flask>=0.8->flask-ngrok) (2.11.3)
    Requirement already satisfied: itsdangerous>=0.24 in /usr/local/lib/python3.7/dist-packages (from Flask>=0.8->flask-ngrok) (1.1.0)
    Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-packages (from Jinja2>=2.10.1->Flask>=0.8->flask-ngrok) (2.0.1)
    Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests->flask-ngrok) (2021.10.8)
    Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests->flask-ngrok) (1.24.3)
    Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->flask-ngrok) (2.10)
    Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->flask-ngrok) (3.0.4)
    Installing collected packages: flask-ngrok
    Successfully installed flask-ngrok-0.0.25
```

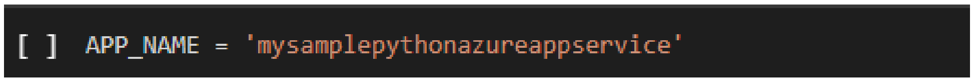
Figure 10: Install the code requirements from Figure 2.



```
[ ] # Load sample
    with open('app.py') as sample:
        exec( sample.read() )

    # Config sample for running in colab
    from flask_ngrok import run_with_ngrok
    run_with_ngrok(app)
    app.run()
```

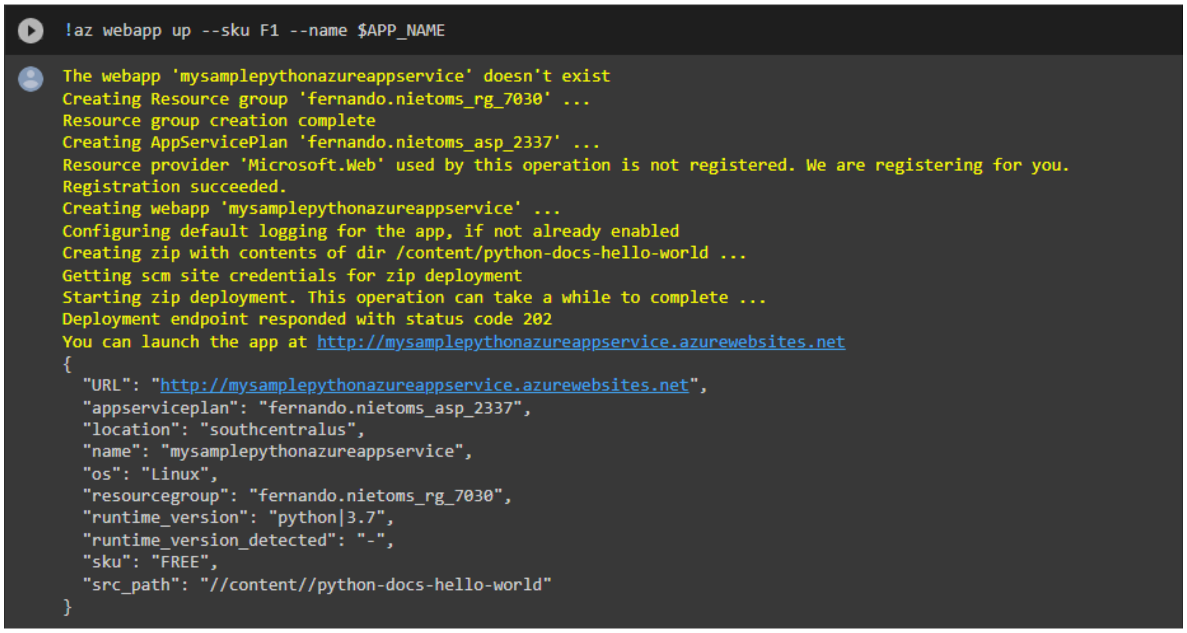
```
 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Running on http://6115-35-194-188-139.ngrok.io
 * Traffic stats available on http://127.0.0.1:4040
```

Figure 11: Load the code sample.



```
[ ] APP_NAME = 'mysamplepythonazureappservice'
```

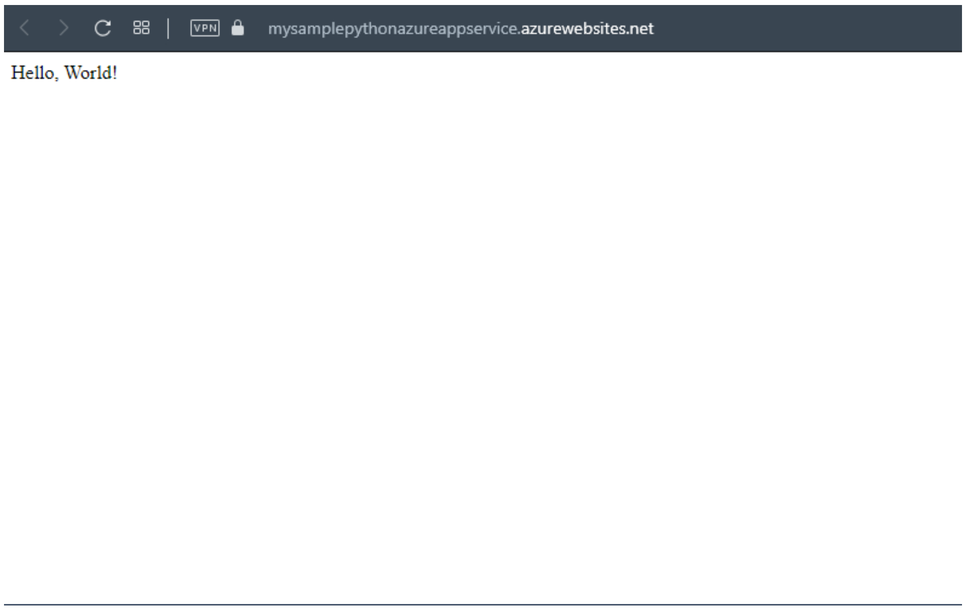
Figure 12: Application name for URL.



```
!az webapp up --sku F1 --name $APP_NAME
```

```
The webapp 'mysamplepythonazureappservice' doesn't exist
Creating Resource group 'fernando.nietoms_rg_7030' ...
Resource group creation complete
Creating AppServicePlan 'fernando.nietoms_asp_2337' ...
Resource provider 'Microsoft.Web' used by this operation is not registered. We are registering for you.
Registration succeeded.
Creating webapp 'mysamplepythonazureappservice' ...
Configuring default logging for the app, if not already enabled
Creating zip with contents of dir /content/python-docs-hello-world ...
Getting scm site credentials for zip deployment
Starting zip deployment. This operation can take a while to complete ...
Deployment endpoint responded with status code 202
You can launch the app at http://mysamplepythonazureappservice.azurewebsites.net
{
  "URL": "http://mysamplepythonazureappservice.azurewebsites.net",
  "appserviceplan": "fernando.nietoms_asp_2337",
  "location": "southcentralus",
  "name": "mysamplepythonazureappservice",
  "os": "Linux",
  "resourcegroup": "fernando.nietoms_rg_7030",
  "runtime_version": "python|3.7",
  "runtime_version_detected": "-",
  "sku": "FREE",
  "src_path": "//content//python-docs-hello-world"
}
```

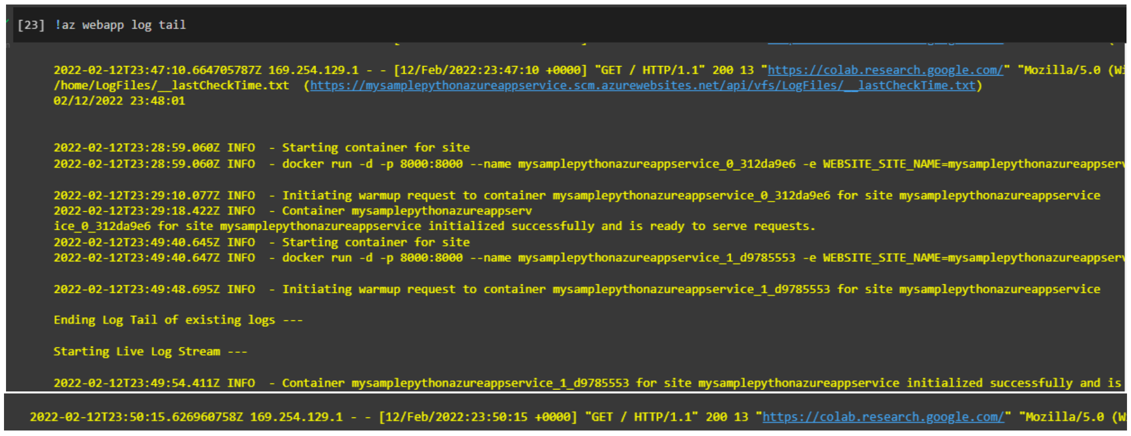
Figure 13: Deploy the code sample.



```
mysamplepythonazureappservice.azurewebsites.net
```

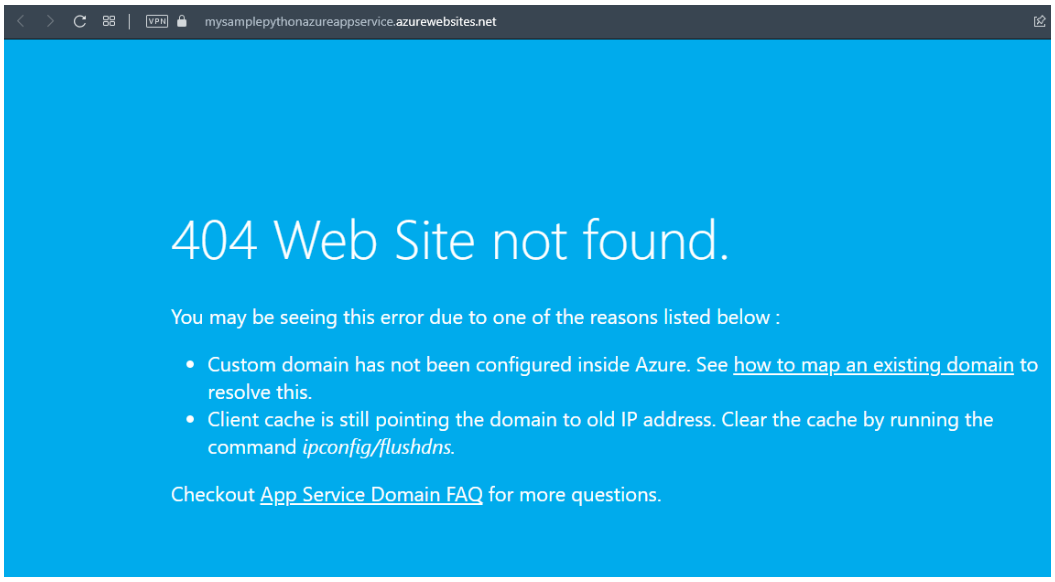
Hello, World!

Figure 14: Presentation of the launched application.



```
[23] !az webapp log tail

    2022-02-12T23:47:10.664705787Z 169.254.129.1 - - [12/Feb/2022:23:47:10 +0000] "GET / HTTP/1.1" 200 13 "https://colab.research.google.com/" "Mozilla/5.0 (Wi
    /home/LogFiles/__lastCheckTime.txt  (https://mysamplepythonazureappservice.scm.azurewebsites.net/api/vfs/LogFiles/__lastCheckTime.txt)
    02/12/2022 23:48:01


    2022-02-12T23:28:59.060Z INFO  - Starting container for site
    2022-02-12T23:28:59.060Z INFO  - docker run -d -p 8000:8000 --name mysamplepythonazureappservice_0_312da9e6 -e WEBSITE_SITE_NAME=mysamplepythonazureappserv

    2022-02-12T23:29:10.077Z INFO  - Initiating warmup request to container mysamplepythonazureappservice_0_312da9e6 for site mysamplepythonazureappservice
    2022-02-12T23:29:18.422Z INFO  - Container mysamplepythonazureappserv
    ice_0_312da9e6 for site mysamplepythonazureappservice initialized successfully and is ready to serve requests.
    2022-02-12T23:49:40.645Z INFO  - Starting container for site
    2022-02-12T23:49:40.647Z INFO  - docker run -d -p 8000:8000 --name mysamplepythonazureappservice_1_d9785553 -e WEBSITE_SITE_NAME=mysamplepythonazureappserv

    2022-02-12T23:49:48.695Z INFO  - Initiating warmup request to container mysamplepythonazureappservice_1_d9785553 for site mysamplepythonazureappservice

    Ending Log Tail of existing logs ---

    Starting Live Log Stream ---

    2022-02-12T23:49:54.411Z INFO  - Container mysamplepythonazureappservice_1_d9785553 for site mysamplepythonazureappservice initialized successfully and is

  2022-02-12T23:50:15.626960758Z 169.254.129.1 - - [12/Feb/2022:23:50:15 +0000] "GET / HTTP/1.1" 200 13 "https://colab.research.google.com/" "Mozilla/5.0 (W
```

Figure 15: Log tail of the application. The last line represents a request done.



404 Web Site not found.

You may be seeing this error due to one of the reasons listed below :

- Custom domain has not been configured inside Azure. See how to map an existing domain to resolve this.
- Client cache is still pointing the domain to old IP address. Clear the cache by running the command *ipconfig/flushdns*.

Checkout App Service Domain FAQ for more questions.

Figure 16: Web page after deleting the resources.

Conclusion

Understanding the different XaaS models is important into the cloud computing to see how the services provided through the Internet are hosted and launched to any customer in any other place The deployment of this type of techniques, where there is no need of the usage of the customer's machine, lets the organizations and companies to distribute applications that does not need specific or minimum requirements installed on a machine, the only feature that anyone needs is access to Internet.

In conclusion, the services provided through the Internet can have the same efficiency than working similar applications in a local machine. Cloud computing and big data not only develops new database strategies, but also lets new methods of working with different platforms and infrastructures beyond the network.

Glossary

- Azure CLI (Command-line interface) is a cross-platform command-line tool designed to work with the different services that Azure provides. Focused on the automation, CLI was created in order to create and manage Azure resources. It allows to execute any command through a terminal by an interactive command-line prompts, or via scripts.[1]
- Ngrok is a service used to create a tunnel between a running web server and a port on a local machine. It also provides a real-time web UI where you can check any HTTP traffic running over each tunnel. To grant accessibility to a specific user to the features that ngrok provides, it requires an authentication token (authtoken) that is set through the configuration file or by the command *ngrok authtoken <YOUR_AUTHTOKEN>*.[2]

[1] https://docs.microsoft.com/en-us/cli/azure/
[2] https://ngrok.com/docs

- Google Colab (Colaboratory) is a Google Research product that allows people to work in a single Python program and run it in any web browser. It is a hosted notebook service of Jupyter that does not require any configuration to work with and lets free resources.[3]
- Flask is a Python's web microframework used to develop web applications. This WSGI (Web Server Gateway Interface) web app frameworks is based on the Werkzeg WSGI toolkit (implements requests, responses, objects and utility functions to the web frame) and the Jinja2 engine (allows to pass Python variables into HTML templates).[4]
- Jupyter Notebook is a web application used to create computational documents. It supports around 40 programming languages (including Python). These can be shared through Jupyter Notebook Viewer or any repository (Github).[5]
- CURL is an open source software used in command-lines and scripts to transfer data through URLs. Used principally on Internet engine, this lets different applications to send data through different protocols.[6]

[3] https://research.google.com/colaboratory/faq.html
[4] https://pythonbasics.org/what-is-flask-python/
[5] https://jupyter.org
[6] https://curl.se