# Conteúdo

# Pre requisite

**Billing:** If you have a new account created on google cloud, ensure that you activated your billing account details or your free testing period on google.
**APIs:** Enable the Cloud Build API, Google Kubernetes Engine APIs e Cloud SQL Admin API. Cloud Deployment Manager V2 API,
**SDK:** if you do not have the SDK already installed on your desktop, follow the procedure for Installing/Configuring SDK as detailed on this document Apendice A.
**Python framework**: install Python and the necessary modules as detailed on Appendix B.:

# SDK command line to create Hello World

Download the project from git:

Change to the project folder downloaded from git

Execute the Google Cloud SDK Shell as administrator

```
gcloud config configurations create deciocfg
gcloud auth login
```

Efetue authenticação na sua conta google cloud

```
gcloud projects create prjdbjsolucx --name prjdbjsolucx
gcloud config set project prjdbjsolucx
```

Access you google console account and access the newly created project and link it with your billing account. Enable the necessary APIs.

```
gcloud services enable cloudbuild.googleapis.com
gcloud services enable container.googleapis.com
gcloud services enable sqladmin.googleapis.com
gcloud services enable iam.googleapis.com
gcloud services enable iamcredentials.googleapis.com
gcloud services enable stackdriver.googleapis.com
gcloud services enable compute.googleapis.com
gcloud services enable deploymentmanager.googleapis.com
gcloud services enable cloud.googleapis.com
gcloud services enable sts.googleapis.com
gcloud services enable securetoken.googleapis.com
gcloud services enable dns.googleapis.com
gcloud services enable binaryauthorization.googleapis.com
gcloud services enable meshca.googleapis.com
gcloud services enable privateca.googleapis.com
gcloud services enable publicca.googleapis.com
gcloud services enable prod-tt-sasportal.googleapis.com
gcloud services enable pubsub.googleapis.com
```

Create the SQL instance on google.

gcloud sql instances create **dbjmysql** --database-version=MYSQL_5_7 --cpu=2 --memory=8GB --zone=us-central1-a --root-password=password123

Connect to the mysql instance and creates: database, table and one record

gcloud sql connect dbjmysql --user=root

Connecting to database with SQL user [root].Enter password:**password123**

```
mysql> create database exemplo\g
mysql> use exemplo\g
mysql>    create table tabela(
          linha INT NOT NULL AUTO_INCREMENT,
          mensagem VARCHAR(100) NOT NULL,
          PRIMARY KEY ( linha )
          )\g
mysql> insert into tabela (mensagem) VALUES ("Hello World!")\g
mysql> exit
```

Enable High availability

gcloud sql instances patch dbjmysql --availability-type REGIONAL --enable-bin-log --backup-start-time=04:00

Install kubectl if not installed (needs administrator privilege).

gcloud components install kubectl

## Containerizing an app with Cloud Build

**Dockerfile:** To containerize an app, create a new file named Dockerfile in the same directory as the source files, and copy the following content:

```
# Use the official lightweight Python image.
# https://hub.docker.com/_/python
FROM python:3.7-slim

# Copy local code to the container image.
ENV APP_HOME /app
WORKDIR $APP_HOME
COPY . ./

# Install production dependencies.
RUN pip install Flask gunicorn

# Run the web service on container startup. Here we use the gunicorn
# webserver, with one worker process and 8 threads.
# For environments with multiple CPU cores, increase the number of workers
# to be equal to the cores available.
CMD exec gunicorn --bind :$PORT --workers 1 --threads 8 app:app
```

**.dockerignore:** file to ensure that local files don't affect the container build process

```
Dockerfile
README.md
*.pyc
*.pyo
*.pyd
__pycache__
```

After the successful creation of the above files (Dockerfile  e .dockerignore) on the same sub-dir of the hello world application, execute the following command on the same sub-dir:

```
gcloud builds submit --tag gcr.io/prjdbjsolucx/helloworld-gke .
```

## Creating a GKE cluster

```
gcloud config set project prjdbjsolucx
gcloud config set compute/region us-central1
gcloud config set compute/zone  us-central1-a

gcloud container clusters create helloworld-gke
```

(*) The command above will create a cluster with 3 instances on the us-central1-a.

Create a new GSA (Google Service Account) and grants roles/cloudsql.client to it

```
gcloud iam service-accounts create helloworld-gsa --display-name="Helloworld GSA for SQL proxy"

gcloud projects add-iam-policy-binding prjdbjsolucx --member="serviceAccount:helloworld-gsa@prjdbjsolucx.iam.gserviceaccount.com" --role=”roles/cloudsql.client”
```

Create a credential file for your service account key

```
gcloud iam service-accounts keys create key.json --iam-account helloworld-gsa@prjdbjsolucx.iam.gserviceaccount.com
```

Turn your service account key into a Secret

```
kubectl create secret generic deciomysasecret --from-file=credentials=key.json
```

Create a secret to securely connect to the database.

```
kubectl create secret generic dbjmysqlsecret --from-literal=username=root --from-literal=password=password123 --from-literal=database=exemplo
```

## Deploying to the Google Cloud Engine

1. Create the **deployment.yaml** file in the same directory as your other files and copy the following content, replacing *$GCLOUD_PROJECT* with your Google Cloud project ID:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: helloworld-gke
spec:
  selector:
    matchLabels:
      app: hello
  template:
    metadata:
      labels:
        app: hello
    spec:
      serviceAccountName: helloworld-gke-ksa
      containers:
      - name: hello-app
        # Replace $GCLOUD_PROJECT with your project ID
        image: gcr.io/prjdbjsolucx/helloworld-gke:latest
        # This app listens on port 8080 for web traffic by default.
        ports:
        - containerPort: 8080
        env:
          - name: PORT
            value: "8080"
        - name: DB_USER
          valueFrom:
            secretKeyRef:
              name: dbjmysqlsecret
              key: username
        - name: DB_PASS
          valueFrom:
            secretKeyRef:
              name: dbjmysqlsecret
              key: password
        - name: DB_NAME
          valueFrom:
            secretKeyRef:
              name: dbjmysqlsecret
              key: database
      - name: cloud-sql-proxy
        image: gcr.io/cloudsql-docker/gce-proxy:1.17
        command:
          - "/cloud_sql_proxy"
          - "-instances= prjdbjsolucx:us-central1:dbjmysql=tcp:3306"
        securityContext:
          runAsNonRoot: true
        resources:
          requests:
            memory: "2Gi"
            cpu:    "1"
```

Note: the line "name: helloworld-gke" specifies the cluster name created on the step before.

After the **deployment.yaml** file creation, execute the following command on the same sub-dir to deploy the resources to the cluster:

```
kubectl apply -f deployment.yaml
```

Wait until the deployment is ready to continue with this procedure. You may check the deployment status with the following command:

```
kubectl get deployments
```

2. Deploy a service: create the **service.yaml** file to deploy the service in the same directory as your other files and copy the following content:

```
# The hello service provides a load-balancing proxy over the hello-app
# pods. By specifying the type as a 'LoadBalancer', Kubernetes Engine will
# create an external HTTP load balancer.
apiVersion: v1
kind: Service
metadata:
  name: hello
spec:
  type: LoadBalancer
  selector:
    app: hello
  ports:
  - port: 80
    targetPort: 8080
```

Create the Hello World Service with the following command:

```
kubectl apply -f service.yaml
```

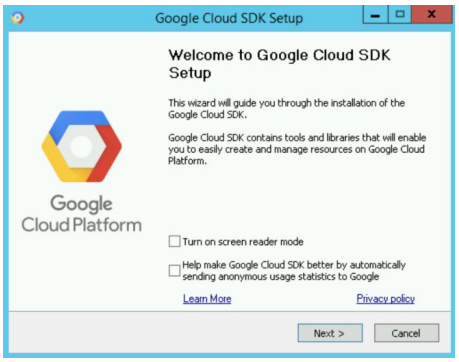
Get the Service's external IP address:

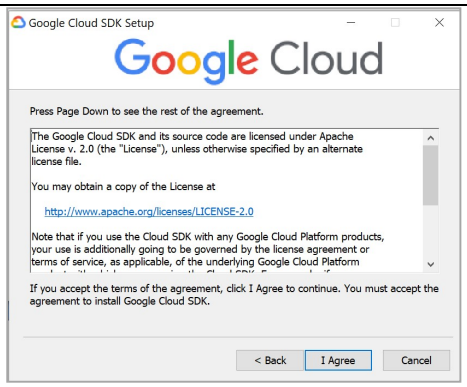
```
kubectl get services
```

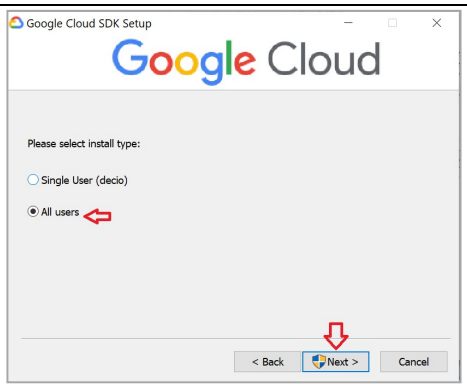
Wait until the EXTERNAL_IP address is acquired.

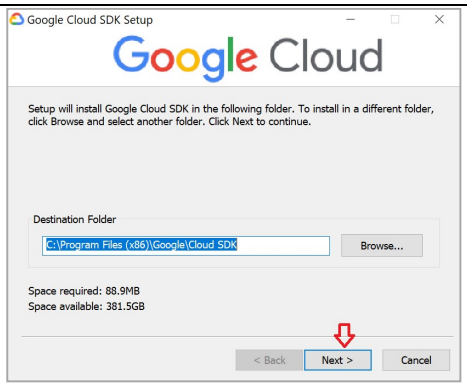
# APENDICE A – Installing/Configuring Cloud SDK

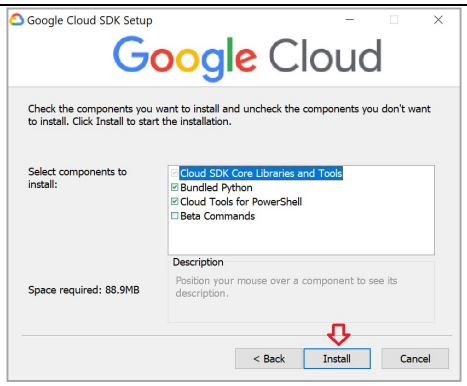
Download the SDk application from: https://cloud.google.com/sdk/docs/install

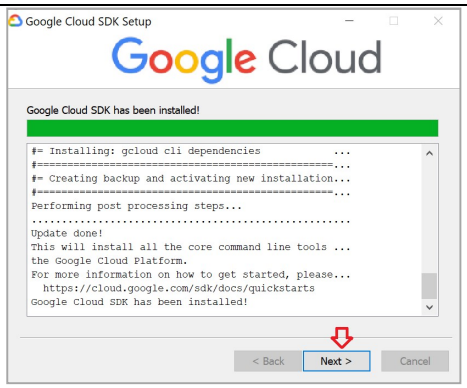


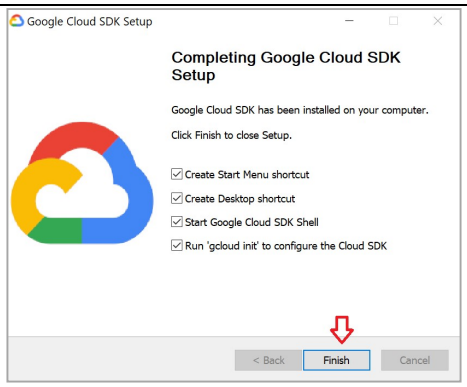



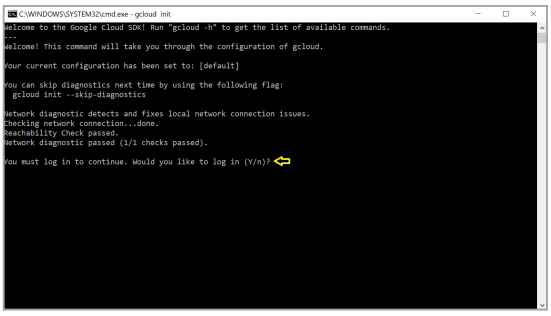

Cloud SDK requires Python; supported versions are **Python 3** (preferred, 3.5 to 3.8) and Python 2 (2.7.9 or higher).

The installer will install all necessary dependencies, including the needed Python version. While Cloud SDK currently uses Python 3 by default, you can use an existing Python installation if necessary by unchecking the option to Install Bundled Python.

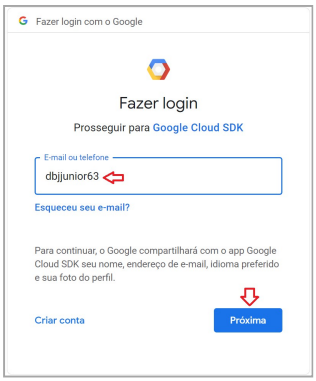
## Configuring SDK

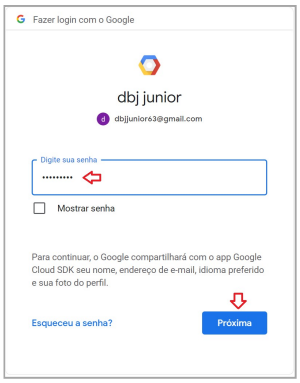
1. Checking the option “**Run gcloud init**”, as illustrated above, will automatically call the gcloud initial configuration tools.

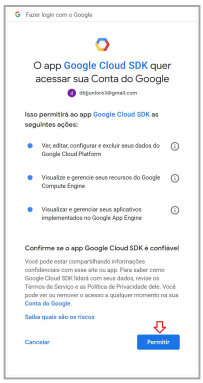
   The configuration process you as your to connect to your google account. Type **y** to connect.



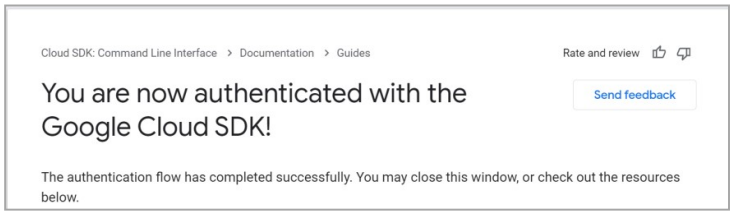

2. The configuration process will activate the web browser to authenticate on your google account.



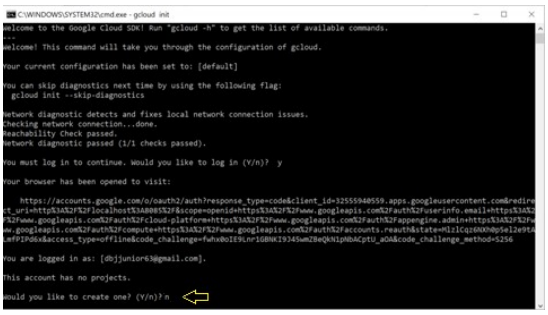




After the successful authentication, the following message will be shown on your browser:



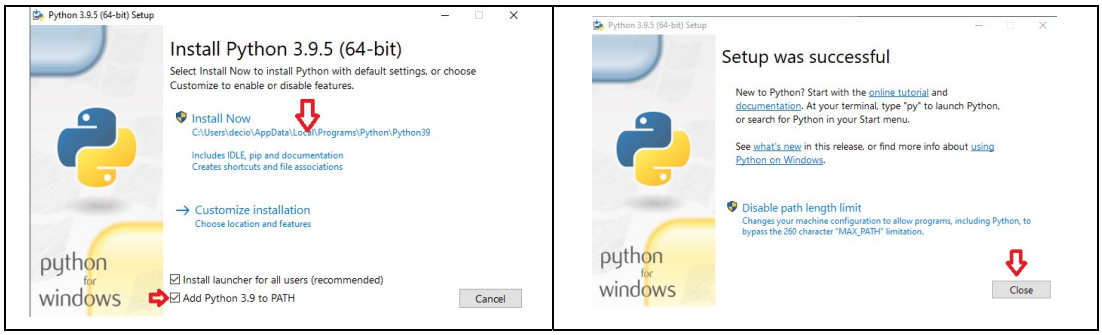

3. The configuration process will identify if you already have projects created on your google account. Answer with “n”. We will create the project latter.



# APPENDIX B – Installing Python

If you don´t have the Python installed, download it from: https://www.python.org/downloads/ and install version Python 3.7.10.

Download and install Python

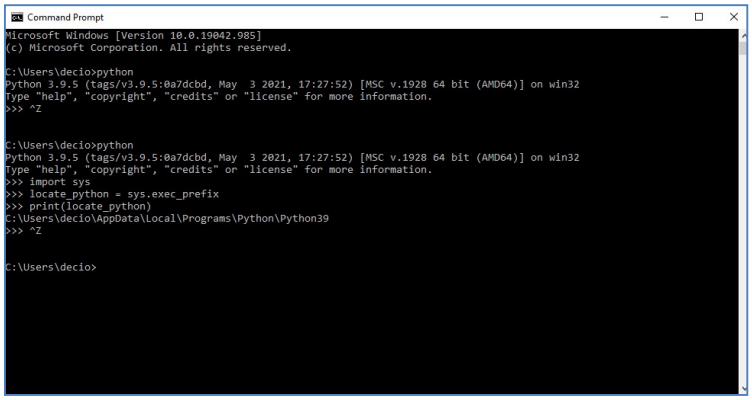



## Instalando o PIP

Executar o seguinte procedimento para instalação do PIP:

1. Verifique onde o Python está instalado conforme ilustrado a seguir:

```
$ python
>>> import sys
>>> locate_python = sys.exec_prefix
>>> print(locate_python)
>>>^Z
```




2. Execute os seguintes comandos para instalação do PIP no sub-dir do Python:

   Execute o CMD com privilegio de administração

   Mude para o diretório de instalação do Python que foi obtido anteriormente

   Exemplo: cd C:\Users\decio\AppData\Local\Programs\Python\Python39

   Baixe o PIP conforme o comando abaixo:

   $ curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py

   Execute o seguinte comando para instalar o PIP:

   $ Python get-pip.py

## Creating a Python environment and installing modules

Execute the following command to create a new Python environment (myenv) and install the desired modules on it :

```
cd \
mkdir Environments
Cd Environments
python -m venv myenv
myenv\Scripts\activate.bat

pip install Flask
pip install sqlalchemy
```

# APPENDIX C – Testing the application locally

## Creating the Environment variables

In this step we will create the necessary env variables to use the SQL proxy.

Create the following variables:

```
DB_HOST: 127.0.0.1:3306
DB_NAME:  exemplo
DB_USER:  root
DB_PASS: password123
```

**Note:** that port number specified on the DB_HOST variable is the default por t number where MySQL will be listening for connections on the production environment. We should use the same port number on the execution of the SQL Auth locally.

## Download the SQL AUTH

Download and move the SQL AUTH to the desired sub-dir. The sub-dir location has no influence on the running application.

Change to the sub-dir where the file cloud_sql_proxy_x64.exe is installed.

Execute the following command on dedicated CMD windows:

```
cloud_sql_proxy_x64.exe -instances=prjdbjsolucx:us-central1:dbjmysql=tcp:127.0.0.1:3306
```

## Testing the application

Change to the sub-dir where the source codes are installed

Execute the Python command

```
py main.py
```

On the web browser type the following address:

```
http://127.0.0.1:8080/
```

**Note:** the address here is the address that was specified on the following line of code inside the application: app.run(host="127.0.0.1", port=8080, debug=True)

# APPENDIX D – alternative method to link the billing account

This can be mode programmatically but I decide not to use because we need to install google beta and the Readme script would need to be manually changed any way.

gcloud beta billing accounts list

gcloud alpha billing accounts projects link prjdbjsolucx (--account-id=ACCOUNT_ID | --billing-account=ACCOUNT_ID) [GCLOUD_WIDE_FLAG …]