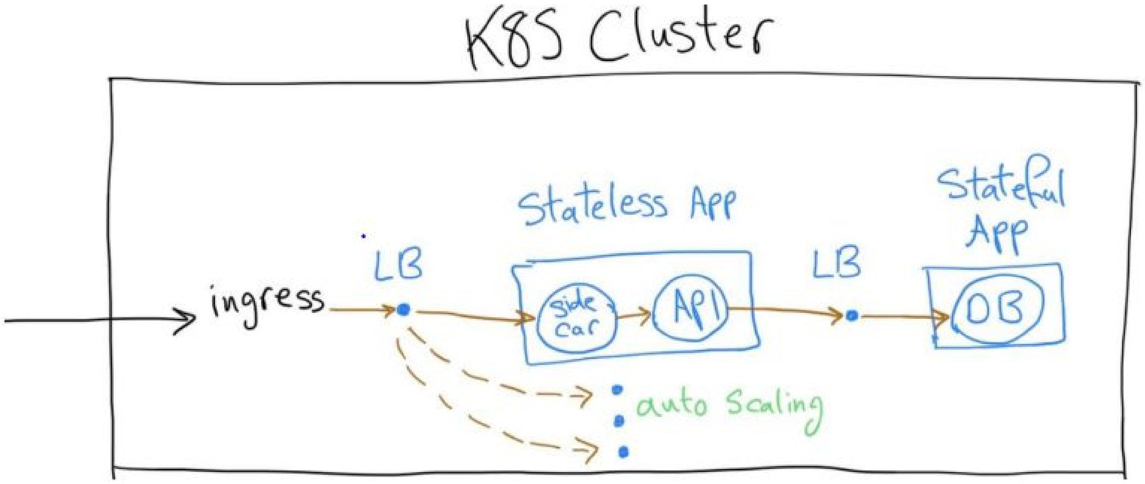
Travix Technical Assignment

Introduction

This document describes the detailed steps followed in travix technical assignment to deploy an application in kubernetes cluster(GKE).

Application architecture to be deployed is shown in the below image(From Original Question):



Step 1: Setup Cluster in GKE

We need to create an account in GKE to setup the cluster. I have created a free account for a year and registered my user with my email address.

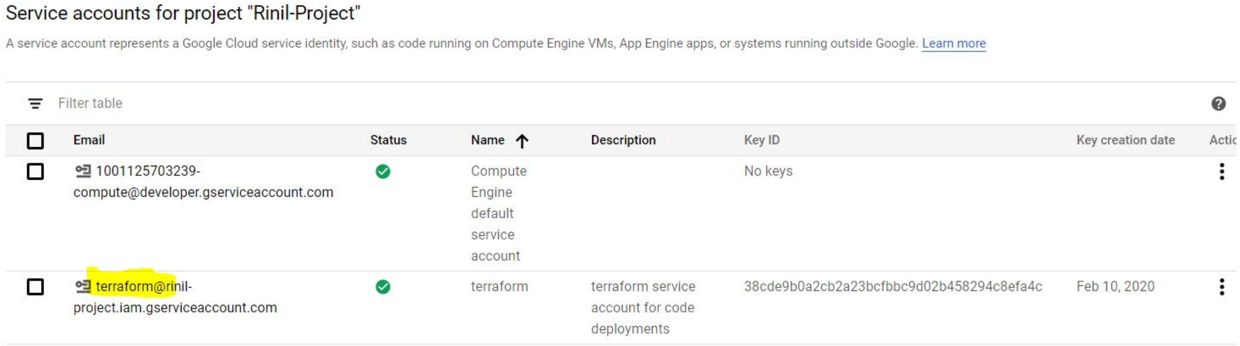
Build Cluster Using Terraform:

Create Service Account:

Now we need to build a kubernetes cluster from the GCP console using terraform. Before we can run the code, we need to have a service account to be used to authenticate and connect to GCP API.

In order to create service account, navigate to:
IAM & Admin > Service Accounts, and click Create Service Account.

I have created “terraform” account as below:

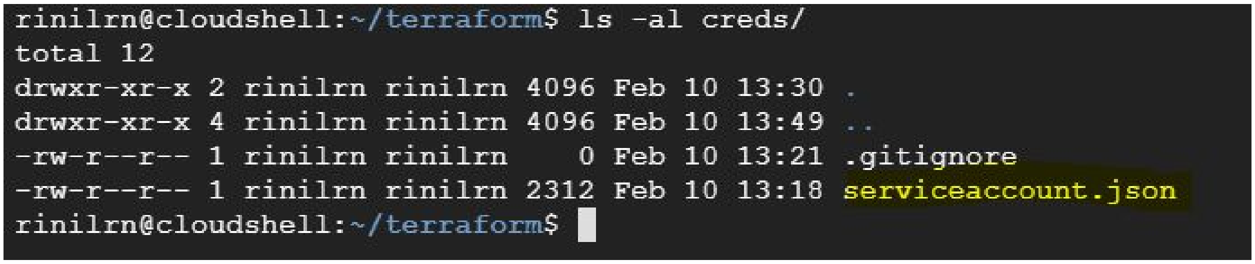


Service accounts for project "Rinil-Project"

A service account represents a Google Cloud service identity, such as code running on Compute Engine VMs, App Engine apps, or systems running outside Google. Learn more

□	Email	Status	Name ↑	Description	Key ID	Key creation date	Actic
□	1001125703239-compute@developer.gserviceaccount.com	✓	Compute Engine default service account		No keys		⋮
□	terraform@rinil-project.iam.gserviceaccount.com	✓	terraform	terraform service account for code deployments	38cde9b0a2cb2a23bcfbbc9d02b458294c8efa4c	Feb 10, 2020	⋮

Create key for this service account and download in json format. Save this key in folder “creds”. We will use this key for authentication while running terraform code.

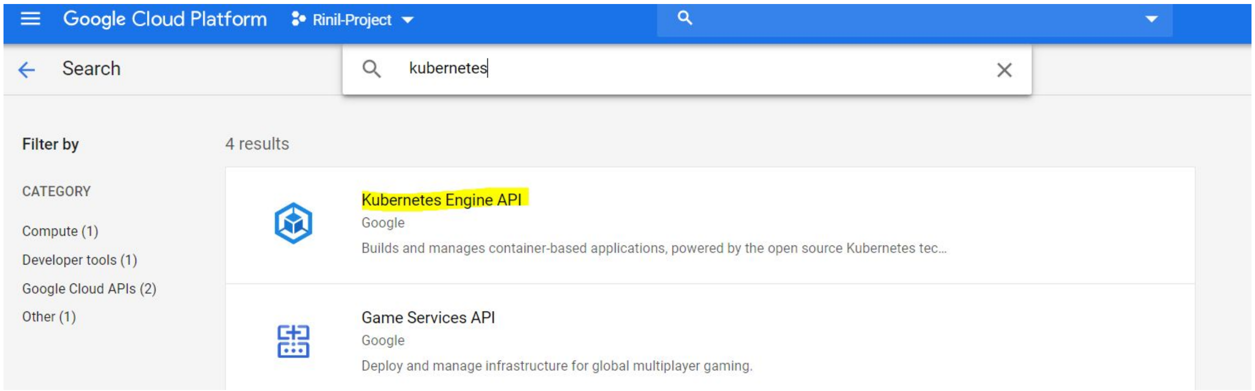


```
rinilrn@cloudshell:~/terraform$ ls -al creds/
total 12
drwxr-xr-x 2 rinilrn rinilrn 4096 Feb 10 13:30 .
drwxr-xr-x 4 rinilrn rinilrn 4096 Feb 10 13:49 ..
-rw-r--r-- 1 rinilrn rinilrn    0 Feb 10 13:21 .gitignore
-rw-r--r-- 1 rinilrn rinilrn 2312 Feb 10 13:18 serviceaccount.json
rinilrn@cloudshell:~/terraform$
```

Created a .gitignore in this folder to prevent this file from being exposed by git.

Enable Kubernetes Engine API:

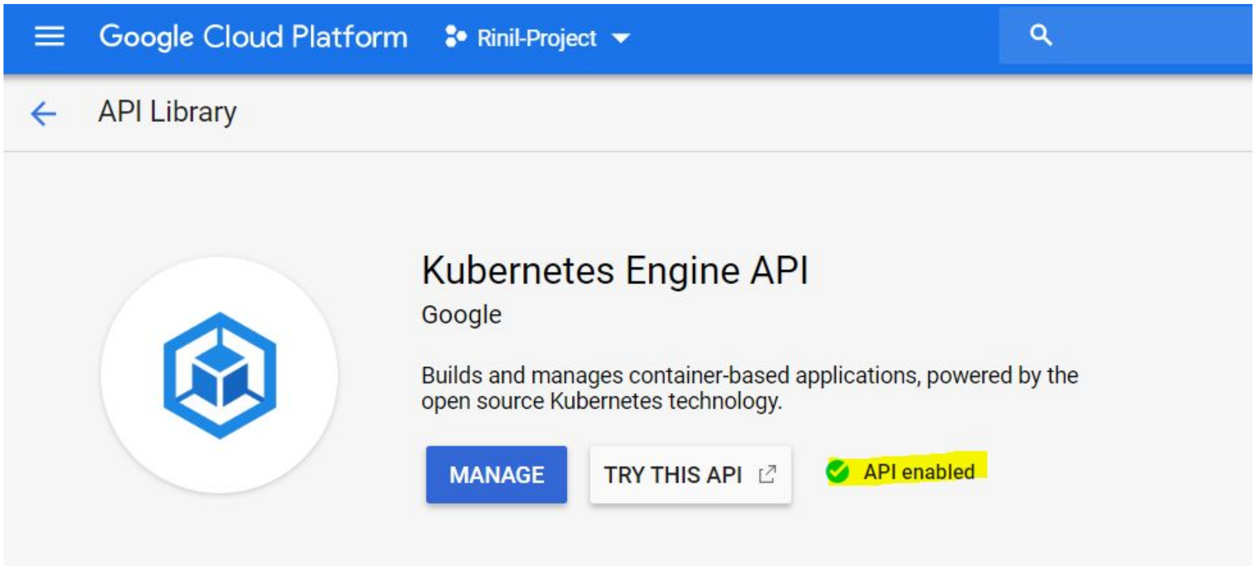
Navigate to APIs & Services > Dashboard, then click Enable APIs and Services. Search for “ Kubernetes Engine API " and enable it.



4 results

Kubernetes Engine API
Google
Builds and manages container-based applications, powered by the open source Kubernetes tec...

Game Services API
Google
Deploy and manage infrastructure for global multiplayer gaming.



Kubernetes Engine API

Google

Builds and manages container-based applications, powered by the open source Kubernetes technology.

Build cluster from code:

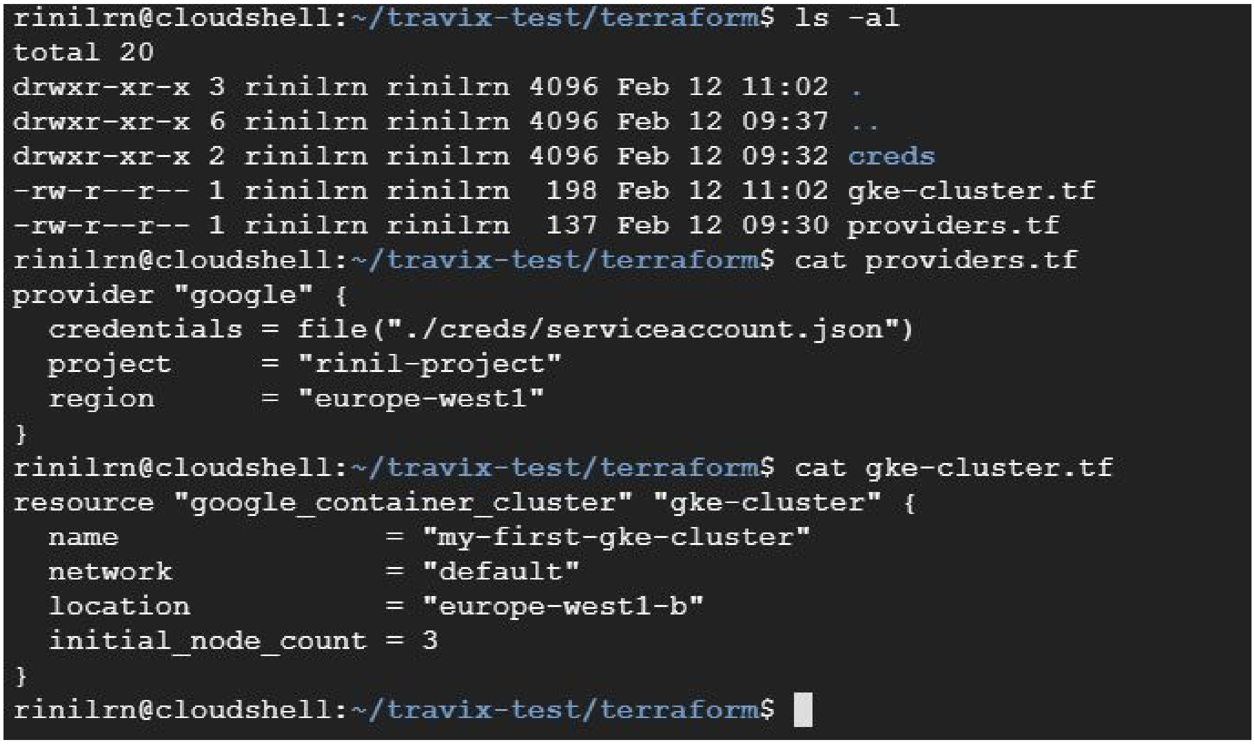
Access cloud shell and save the provider definition(provider.tf) and cluster code(gke-cluster.tf) in terraform directory as below:

Provider.tf: Defines the provider as google, credential path for service account, project name and region in it.

```
===========================
provider "google" {
  credentials = file("./creds/serviceaccount.json")
  project     = "rinil-project"
  region      = "europe-west1"
}
===========================
```

Gke-cluster.tf: Cluster definition which includes Name, Region, initial node count etc.

```
===========================
resource "google_container_cluster" "gke-cluster" {
  name               = "my-first-gke-cluster"
  network            = "default"
  location           = "europe-west1-b"
  initial_node_count = 3
}
===========================
```

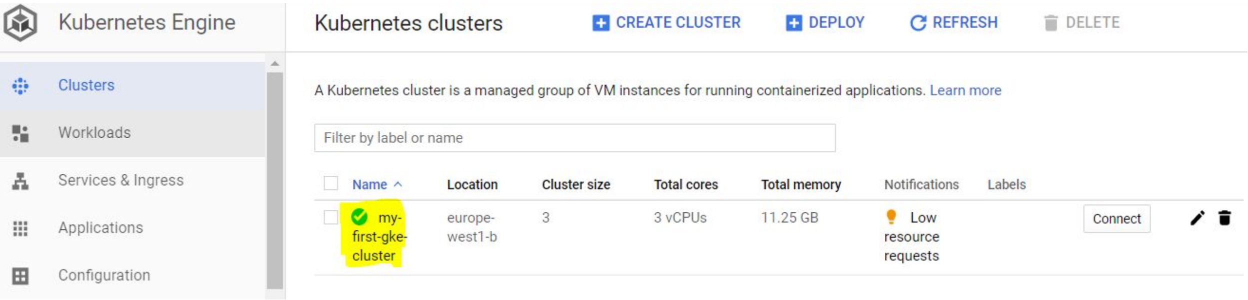


```
rinilrn@cloudshell:~/travix-test/terraform$ ls -al
total 20
drwxr-xr-x 3 rinilrn rinilrn 4096 Feb 12 11:02 .
drwxr-xr-x 6 rinilrn rinilrn 4096 Feb 12 09:37 ..
drwxr-xr-x 2 rinilrn rinilrn 4096 Feb 12 09:32 creds
-rw-r--r-- 1 rinilrn rinilrn  198 Feb 12 11:02 gke-cluster.tf
-rw-r--r-- 1 rinilrn rinilrn  137 Feb 12 09:30 providers.tf
rinilrn@cloudshell:~/travix-test/terraform$ cat providers.tf
provider "google" {
  credentials = file("./creds/serviceaccount.json")
  project     = "rinil-project"
  region      = "europe-west1"
}
rinilrn@cloudshell:~/travix-test/terraform$ cat gke-cluster.tf
resource "google_container_cluster" "gke-cluster" {
  name               = "my-first-gke-cluster"
  network            = "default"
  location           = "europe-west1-b"
  initial_node_count = 3
}
rinilrn@cloudshell:~/travix-test/terraform$
```

Initialize terraform and execute terraform code to build cluster:

```
#terraform init
#terraform plan -out myplan
#terraform apply "myplan"
```

On successful completion, you will be able to see the cluster under the respective project:



Kubernetes clusters

A Kubernetes cluster is a managed group of VM instances for running containerized applications. Learn more

□	Name	Location	Cluster size	Total cores	Total memory	Notifications	Labels
□	✓ my-first-gke-cluster	europe-west1-b	3	3 vCPUs	11.25 GB	Low resource requests	

Step 2: Implementation of test Architecture:

Cluster is now ready and we are proceeding with the implementation of test architecture in question.

Create Stateful App(DB):

We need to create a mysql DB service with persistent volume to meet the requirement here.

Create Persistent volume:

‘Persistent volume claim’ and ‘storage class’ definition for MySQL data volume are given below:

mysql-pv.yml

```
==========================
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: mysql-pv-claim
spec:
 storageClassName: faster
 accessModes:
   - ReadWriteOnce
 resources:
   requests:
     storage: 10Gi
==========================
```

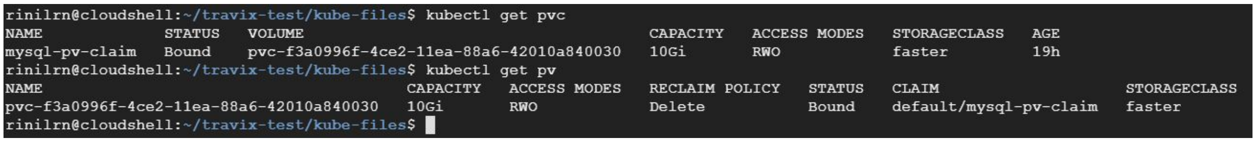
Storageclass.yml

```
==========================
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: faster
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-ssd
==========================
```

Deploy the code to create persistent volume for mysql:

```
# kubectl apply -f mysql-pv.yml
```

Now you will be able to see the persistent volume claim and volume created:



```
rinilrn@cloudshell:~/travix-test/kube-files$ kubectl get pvc
NAME             STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
mysql-pv-claim   Bound    pvc-f3a0996f-4ce2-11ea-88a6-42010a840030   10Gi       RWO            faster         19h
rinilrn@cloudshell:~/travix-test/kube-files$ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                    STORAGECLASS
pvc-f3a0996f-4ce2-11ea-88a6-42010a840030   10Gi       RWO            Delete           Bound    default/mysql-pv-claim   faster
rinilrn@cloudshell:~/travix-test/kube-files$
```

Create docker image for MySQL:

Docker file for this image is saved as Dockerfile in github

```
==================================
FROM mysql:5.6
ENV MYSQL_DATABASE company
==================================
```

This will also create a database named “company”.

Build and push docker image to dockerhub:

Run the commands from dockerfile location.

```
#docker build -t mysql-test .
#docker tag <image ID> username/mysql-test:latest
#docker push username/mysql-test:latest
```

Deploy MySQL Service:

MySQL can now be deployed with the below yml file which will use the persistent volume and mysql-test docker image created in the above section.

mysql-deployment.yml

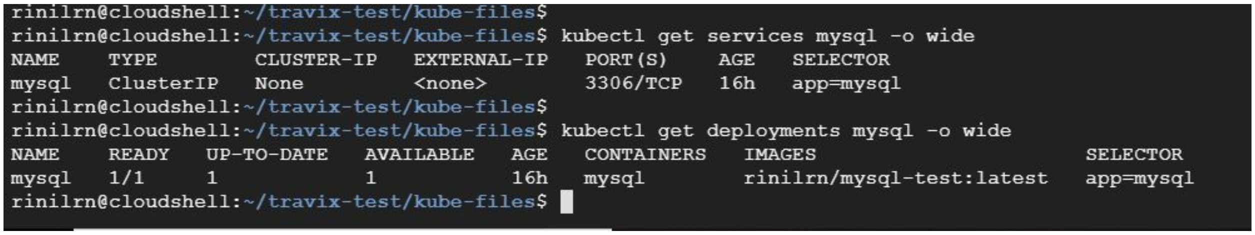
```
====================================
apiVersion: v1
kind: Service
metadata:
  name: mysql
spec:
  ports:
  - port: 3306
  selector:
    app: mysql
  clusterIP: None
```

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql
spec:
  selector:
    matchLabels:
      app: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
      - image: rinilrn/mysql-test:latest
        name: mysql
        env:
          # Use secret in real usage
        - name: MYSQL_ROOT_PASSWORD
          value: password
        - name: MYSQL_DATABASE
          value: company
        - name: MYSQL_USER
          value: root
        - name: MYSQL_PASSWORD
          value: password
        args: ["--default-authentication-plugin=mysql_native_password"]
        ports:
        - containerPort: 3306
          name: mysql
        volumeMounts:
        - name: mysql-persistent-storage
          mountPath: /var/lib/mysql
      volumes:
      - name: mysql-persistent-storage
        persistentVolumeClaim:
          claimName: mysql-pv-claim
=========================================
```

Deploy MySQL from console:

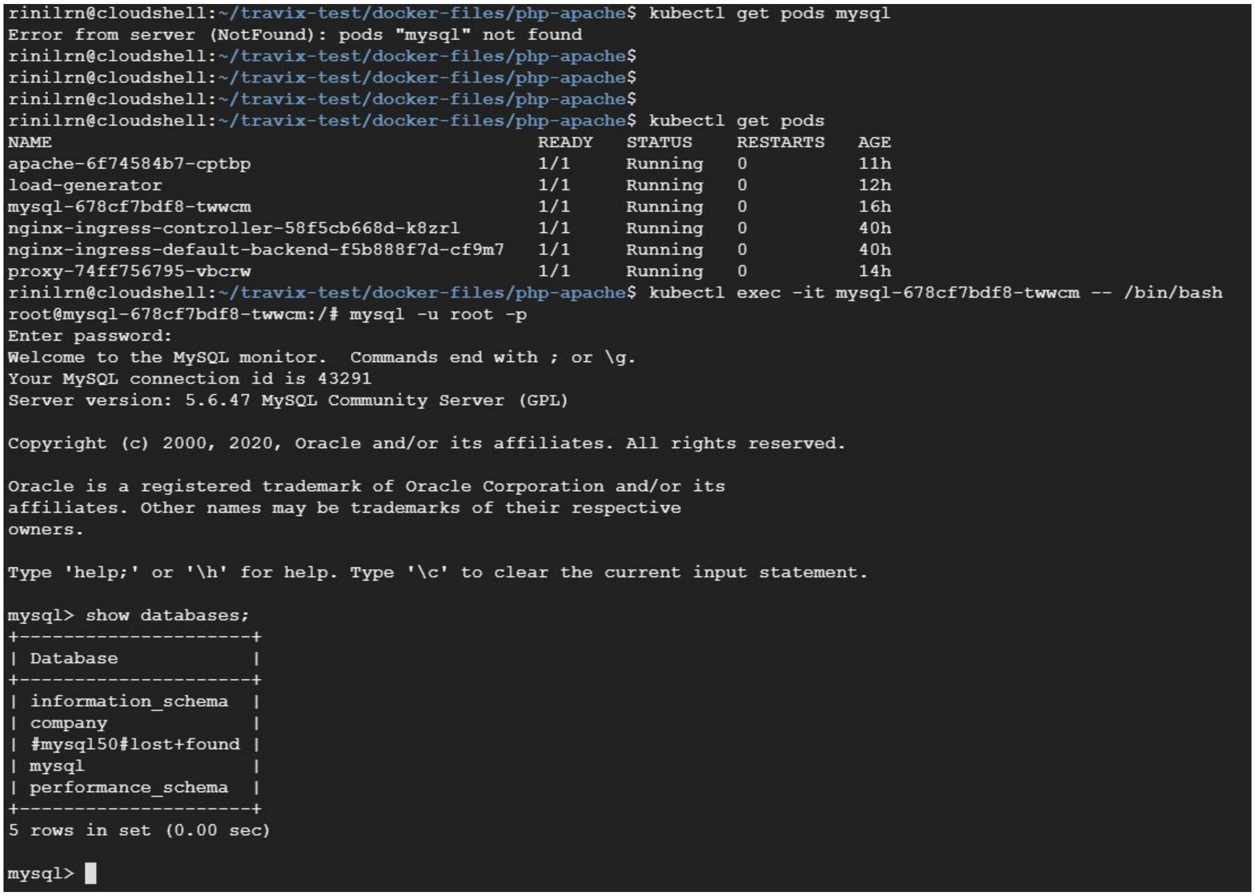
```
#kubectl apply -f mysql-deployment.yml
```

Now you can see the deployment and service for mysql with a persistent data volume path.



```
rinilrn@cloudshell:~/travix-test/kube-files$
rinilrn@cloudshell:~/travix-test/kube-files$ kubectl get services mysql -o wide
NAME    TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE   SELECTOR
mysql   ClusterIP   None         <none>        3306/TCP   16h   app=mysql
rinilrn@cloudshell:~/travix-test/kube-files$
rinilrn@cloudshell:~/travix-test/kube-files$ kubectl get deployments mysql -o wide
NAME    READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS   IMAGES                         SELECTOR
mysql   1/1     1            1           16h   mysql        rinilrn/mysql-test:latest      app=mysql
rinilrn@cloudshell:~/travix-test/kube-files$ 
```

Connect and test MySQL service in any pod:



```
rinilrn@cloudshell:~/travix-test/docker-files/php-apache$ kubectl get pods mysql
Error from server (NotFound): pods "mysql" not found
rinilrn@cloudshell:~/travix-test/docker-files/php-apache$
rinilrn@cloudshell:~/travix-test/docker-files/php-apache$
rinilrn@cloudshell:~/travix-test/docker-files/php-apache$
rinilrn@cloudshell:~/travix-test/docker-files/php-apache$ kubectl get pods
NAME                                            READY   STATUS    RESTARTS   AGE
apache-6f74584b7-cptbp                          1/1     Running   0          11h
load-generator                                  1/1     Running   0          12h
mysql-678cf7bdf8-twwcm                          1/1     Running   0          16h
nginx-ingress-controller-58f5cb668d-k8zrl       1/1     Running   0          40h
nginx-ingress-default-backend-f5b888f7d-cf9m7   1/1     Running   0          40h
proxy-74ff756795-vbcrw                          1/1     Running   0          14h
rinilrn@cloudshell:~/travix-test/docker-files/php-apache$ kubectl exec -it mysql-678cf7bdf8-twwcm -- /bin/bash
root@mysql-678cf7bdf8-twwcm:/# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 43291
Server version: 5.6.47 MySQL Community Server (GPL)

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+---------------------+
| Database            |
+---------------------+
| information_schema  |
| company             |
| #mysql50#lost+found |
| mysql               |
| performance_schema  |
+---------------------+
5 rows in set (0.00 sec)

mysql> 
```

Deploy Application Server:

Create Docker Image for php-apache:

We need to deploy an application server with apache, which will initiate a data request to MySQL server from php code.

I have uploaded the necessary docker files for php-apache image with index page(index.php) and dataloader(dataloader.php) to github under docker-files/php-apache/ .

dataloader.php >> a php file included with the image to load sample data to MySQL database.
Index.php >> default index page with an sql query to display MySQL table data.

Build a docker image(php-apache) with these files and use it for application server deployment.

Run from docker file location:

```
#docker build -t php-apache .
#docker tag <image ID> username/php-apache:latest
#docker push username/php-apache:latest
```

Once the image is created, we are ready for apache deployment:

PHP-Apache Deployment:

Deploy application server with the below yml.

app-api-apache.yml

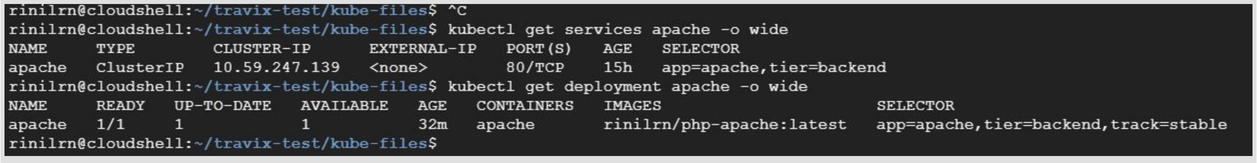
```
================================
apiVersion: v1
kind: Service
metadata:
  name: apache
spec:
  selector:
    app: apache
    tier: backend
  ports:
  - protocol: TCP
    port: 80
    targetPort: http
---
apiVersion: apps/v1
```

```
kind: Deployment
metadata:
  name: apache
spec:
  selector:
    matchLabels:
      app: apache
      tier: backend
      track: stable
  replicas: 1
  template:
    metadata:
      labels:
        app: apache
        tier: backend
        track: stable
    spec:
      containers:
        - name: apache
          image: rinilrn/php-apache:latest
          ports:
            - name: http
              containerPort: 80
================================
```

Run deploy command from console:

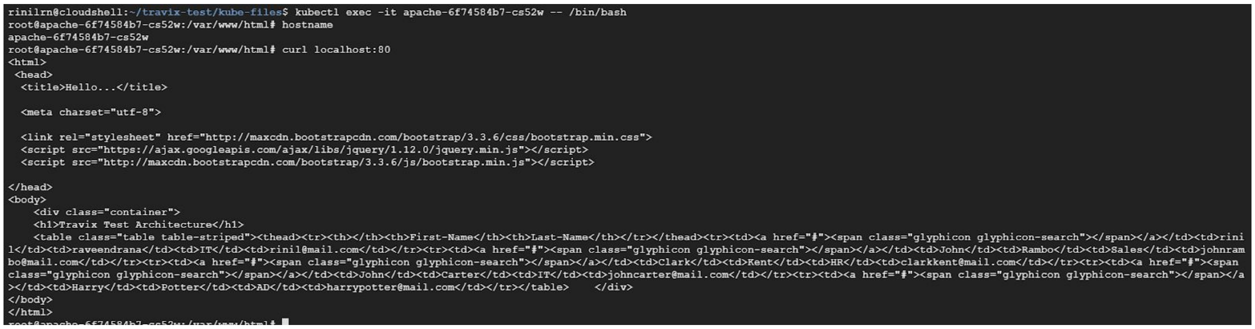
```
#kubectl apply -f app-api-apache.yml
```

This will create a deployment and a service named apache.
Verify service and deployment status with kubectl:



```
rinilrn@cloudshell:~/travix-test/kube-files$ ^C
rinilrn@cloudshell:~/travix-test/kube-files$ kubectl get services apache -o wide
NAME     TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE   SELECTOR
apache   ClusterIP   10.59.247.139   <none>        80/TCP    15h   app=apache,tier=backend
rinilrn@cloudshell:~/travix-test/kube-files$ kubectl get deployment apache -o wide
NAME     READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS   IMAGES                      SELECTOR
apache   1/1     1            1           32m   apache       rinilrn/php-apache:latest   app=apache,tier=backend,track=stable
rinilrn@cloudshell:~/travix-test/kube-files$
```

Also we can verify the connectivity of the app to MySQL from the pod as below:



```
rinilrn@cloudshell:~/travix-test/kube-files$ kubectl exec -it apache-6f74584b7-cs52w -- /bin/bash
root@apache-6f74584b7-cs52w:/var/www/html# hostname
apache-6f74584b7-cs52w
root@apache-6f74584b7-cs52w:/var/www/html# curl localhost:80
<html>
 <head>
  <title>Hello...</title>

  <meta charset="utf-8">

  <link rel="stylesheet" href="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.0/jquery.min.js"></script>
  <script src="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js"></script>

</head>
<body>
    <div class="container">
    <h1>Travix Test Architecture</h1>
    <table class="table table-striped"><thead><tr><th></th><th>First-Name</th><th>Last-Name</th></tr></thead><tr><td><a href="#"><span class="glyphicon glyphicon-search"></span></a></td><td>rini
l</td><td>raveendrana</td><td>IT</td><td>rinil@mail.com</td></tr><tr><td><a href="#"><span class="glyphicon glyphicon-search"></span></a></td><td>John</td><td>Rambo</td><td>Sales</td><td>johnram
bo@mail.com</td></tr><tr><td><a href="#"><span class="glyphicon glyphicon-search"></span></a></td><td>Clark</td><td>Kent</td><td>HR</td><td>clarkkent@mail.com</td></tr><tr><td><a href="#"><span
class="glyphicon glyphicon-search"></span></a></td><td>John</td><td>Carter</td><td>IT</td><td>johncarter@mail.com</td></tr><tr><td><a href="#"><span class="glyphicon glyphicon-search"></span></a
></td><td>Harry</td><td>Potter</td><td>AD</td><td>harrypotter@mail.com</td></tr></table>    </div>
</body>
</html>
root@apache-6f74584b7-cs52w:/var/www/html#
```

This shows that the MySQL table data is fetched from apache successfully.

Autoscaling for Application:

Deploy a HorizontalPodAutoscaler for apache service deployment using below yml.

Application will scale up/down(between 1-10 pods) based on the resource usage:
Thresholds given are:

RAM > 100MB
CPU > 50%

autoscale.yml

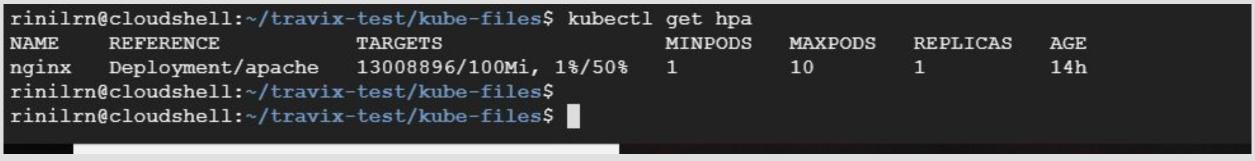
```
=================================
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
  name: nginx
spec:
  maxReplicas: 10
  minReplicas: 1
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: apache
  metrics:
  - type: Resource
    resource:
      name: cpu
      targetAverageUtilization: 50
  - type: Resource
    resource:
      name: memory
```

        targetAverageValue: 100Mi
================================

Run deploy command from console:

```
# kubectl apply -f autoscale.yml
```

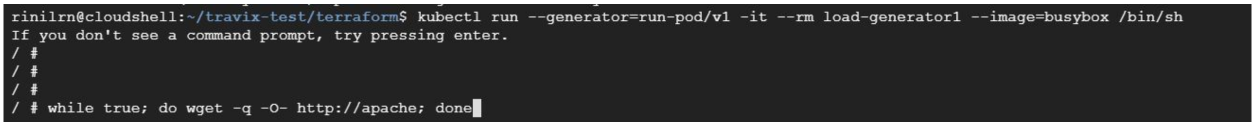
Verify if hpa is properly deployed as below.



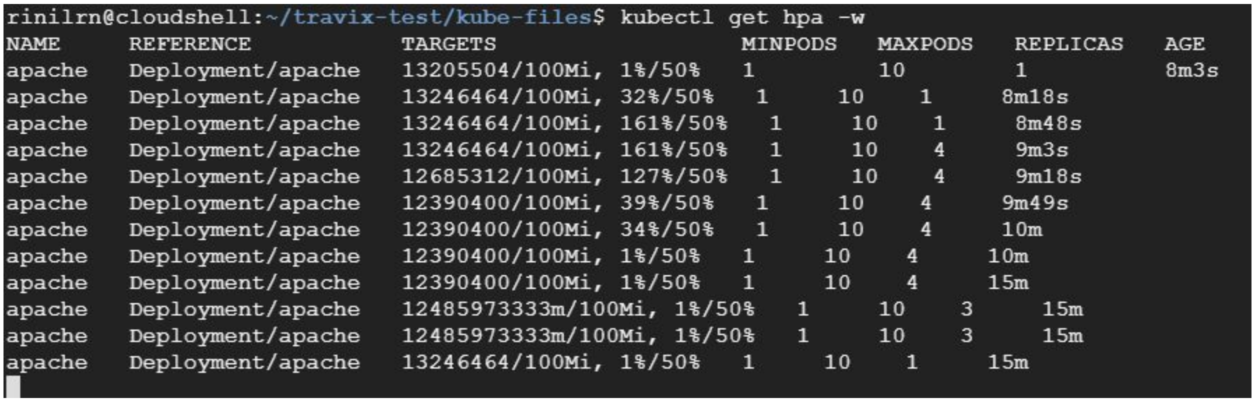
```
rinilrn@cloudshell:~/travix-test/kube-files$ kubectl get hpa
NAME    REFERENCE           TARGETS                  MINPODS   MAXPODS   REPLICAS   AGE
nginx   Deployment/apache   13008896/100Mi, 1%/50%   1         10        1          14h
rinilrn@cloudshell:~/travix-test/kube-files$
rinilrn@cloudshell:~/travix-test/kube-files$ 
```

Also we can monitor its behaviour on high load to make sure that it is working as expected. Pasting some results I have observed in my deployment.

For testing: execute a stress script from a busybox to apache and monitor the hpa status. You can see the number of pods varies with the load.



```
rinilrn@cloudshell:~/travix-test/terraform$ kubectl run --generator=run-pod/v1 -it --rm load-generator1 --image=busybox /bin/sh
If you don't see a command prompt, try pressing enter.
/ #
/ #
/ #
/ # while true; do wget -q -O- http://apache; done
```



```
rinilrn@cloudshell:~/travix-test/kube-files$ kubectl get hpa -w
NAME     REFERENCE           TARGETS                      MINPODS    MAXPODS    REPLICAS    AGE
apache   Deployment/apache   13205504/100Mi, 1%/50%    1          10         1           8m3s
apache   Deployment/apache   13246464/100Mi, 32%/50%    1     10     1     8m18s
apache   Deployment/apache   13246464/100Mi, 161%/50%    1     10     1     8m48s
apache   Deployment/apache   13246464/100Mi, 161%/50%    1     10     4     9m3s
apache   Deployment/apache   12685312/100Mi, 127%/50%    1     10     4     9m18s
apache   Deployment/apache   12390400/100Mi, 39%/50%    1     10     4     9m49s
apache   Deployment/apache   12390400/100Mi, 34%/50%    1     10     4     10m
apache   Deployment/apache   12390400/100Mi, 1%/50%   1     10     4     10m
apache   Deployment/apache   12390400/100Mi, 1%/50%   1     10     4     15m
apache   Deployment/apache   12485973333m/100Mi, 1%/50%   1     10     3     15m
apache   Deployment/apache   12485973333m/100Mi, 1%/50%   1     10     3     15m
apache   Deployment/apache   13246464/100Mi, 1%/50%   1     10     1     15m
```

Nginx sidecar deployment:

Having a sidecar with the application API server was one of the requirements in the question. Here we will use nginx reverse proxy configuration to achieve this.

Create docker image for nginx reverse proxy:

I have uploaded the nginx proxy configuration and dockerfile for nginx image to be used, in the github under “docker-files/nginx-reverse-proxy/”.

Nginx proxy configuration will route the connection to the apache application server. Nginx configuration will be as below:

nginx-proxy.conf

```
================
upstream apache {
    server apache;
}

server {
    listen 80;

    location / {
        proxy_pass http://apache;
    }
}

================
```

Dockerfile

```
================
FROM nginx
RUN rm /etc/nginx/conf.d/*
ADD nginx-proxy.conf /etc/nginx/conf.d/
================
```

Build and push image from docker file location:

```
#docker build -t nginx-rinil .
#docker tag <image ID> username/nginx-rinil:latest
#docker push username/nginx-rinil:latest
```

Deploy nginx reverse proxy with below yml:

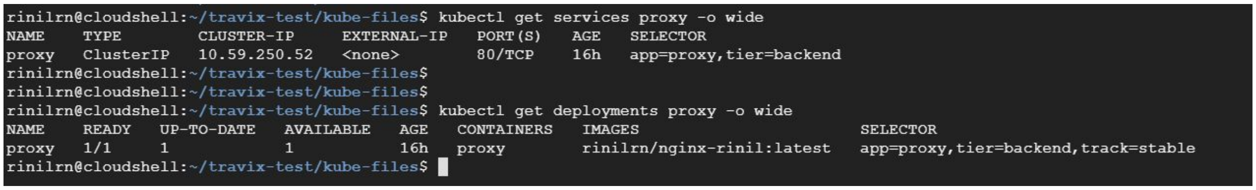
App-sidecar-nginx.yml

```
==============================
apiVersion: v1
kind: Service
metadata:
  name: proxy
spec:
  selector:
    app: proxy
    tier: backend
  ports:
  - protocol: TCP
    port: 80
    targetPort: http
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: proxy
spec:
  selector:
    matchLabels:
      app: proxy
      tier: backend
      track: stable
  replicas: 1
  template:
    metadata:
      labels:
        app: proxy
        tier: backend
        track: stable
    spec:
      containers:
        - name: proxy
          image: rinilrn/nginx-rinil:latest
          ports:
            - name: http
              containerPort: 80
==============================
```

Deploy the code from console:

```
#kubectl apply -f app-sidecar-nginx.yml
```

Also verify the deployment and service:



```
rinilrn@cloudshell:~/travix-test/kube-files$ kubectl get services proxy -o wide
NAME    TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE   SELECTOR
proxy   ClusterIP   10.59.250.52   <none>        80/TCP    16h   app=proxy,tier=backend
rinilrn@cloudshell:~/travix-test/kube-files$
rinilrn@cloudshell:~/travix-test/kube-files$
rinilrn@cloudshell:~/travix-test/kube-files$ kubectl get deployments proxy -o wide
NAME    READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS   IMAGES                       SELECTOR
proxy   1/1     1            1           16h   proxy        rinilrn/nginx-rinil:latest   app=proxy,tier=backend,track=stable
rinilrn@cloudshell:~/travix-test/kube-files$
```

Ingress controller/LB Deployment

Now we have application server, DB server and nginx reverse proxy ready for service. In order to access this application from an external network, we need to deploy an ingress application also. This will give a loadbalancer public ip address to access the application.

I have used a nginx ingress controller installed using helm.

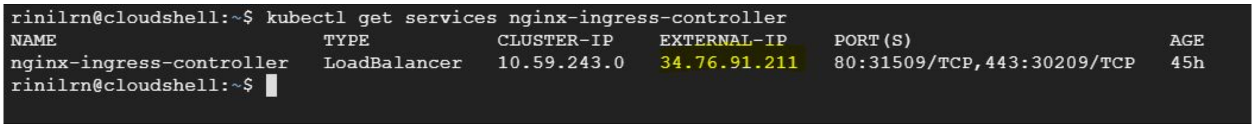
Initialize helm:

```
# kubectl create serviceaccount --namespace kube-system tiller
# kubectl create clusterrolebinding tiller-cluster-rule
--clusterrole=cluster-admin --serviceaccount=kube-system:tiller
# helm init --service-account tiller
```

install nginx ingress controller:

```
helm install --name nginx-ingress stable/nginx-ingress --set
rbac.create=true --set controller.publishService.enabled=true
```

Now you can see the ingress controller service running with an external IP address:



```
rinilrn@cloudshell:~$ kubectl get services nginx-ingress-controller
NAME                       TYPE           CLUSTER-IP    EXTERNAL-IP    PORT(S)                      AGE
nginx-ingress-controller   LoadBalancer   10.59.243.0   34.76.91.211   80:31509/TCP,443:30209/TCP   45h
rinilrn@cloudshell:~$
```

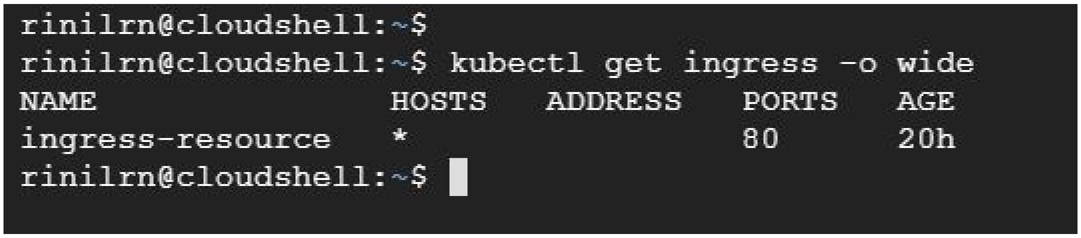
Now we need to route this controller to our application using an ingress resource deployment. Need to deploy ingress resource yml below:

ingress-resource.yaml

```
==================================
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: ingress-resource
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
spec:
  rules:
  - http:
      paths:
      - path: /
        backend:
          serviceName: proxy
          servicePort: 80
==================================
```

Deploy from console and verify the status as below:

```
# kubectl apply -f ingress-resource.yaml
```



```
rinilrn@cloudshell:~$
rinilrn@cloudshell:~$ kubectl get ingress -o wide
NAME                HOSTS   ADDRESS   PORTS   AGE
ingress-resource    *                 80      20h
rinilrn@cloudshell:~$
```

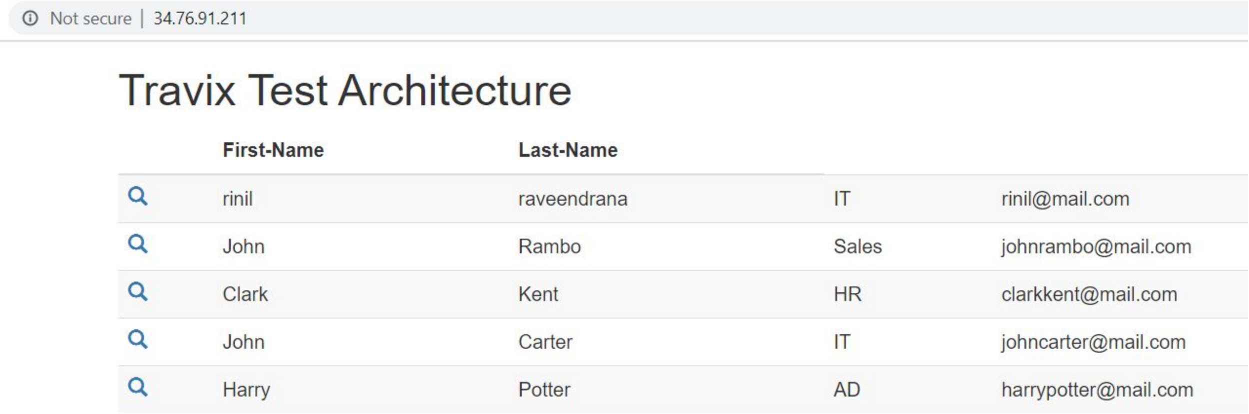
This ingress resource will route connections to ‘proxy’ service which is our nginx reverse proxy for apache application.

Step 3: Access Application from browser:

All components of the architecture are ready and running fine now.

You can load the sample data using the dataloader script “http://34.76.91.211/dataloader.php”

Application should be accessible from the browser with the external ip address of the Loadbalancer “http://34.76.91.211/”.



Travix Test Architecture

	First-Name	Last-Name		
🔍	rinil	raveendrana	IT	rinil@mail.com
🔍	John	Rambo	Sales	johnrambo@mail.com
🔍	Clark	Kent	HR	clarkkent@mail.com
🔍	John	Carter	IT	johncarter@mail.com
🔍	Harry	Potter	AD	harrypotter@mail.com