Tasks:

1. **Create a Kubernetes cluster on GCP If possible share a script / code which can be used to create the cluster.**

➔ I created using Google Kubernetes Engine through google cloud UI.
Rest Call to use to create it using script

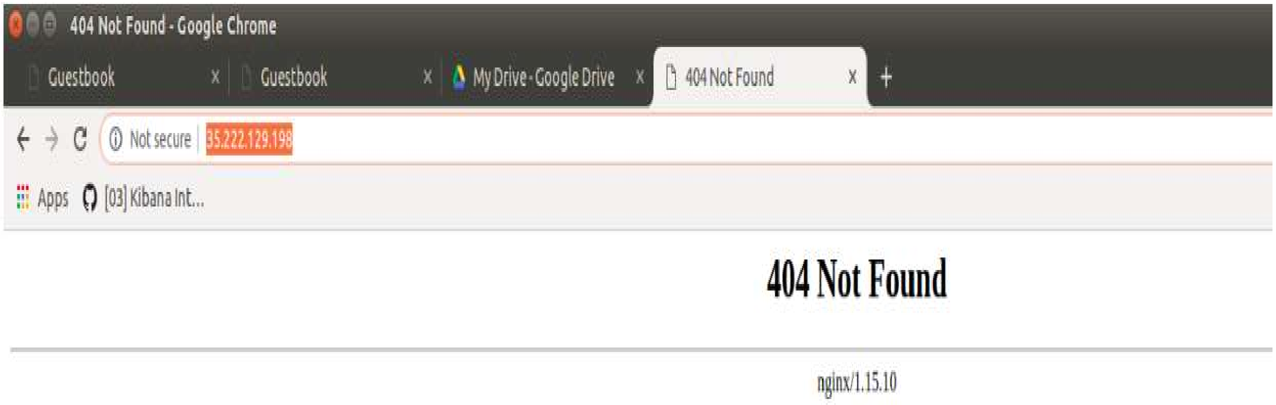
```
POST https://container.googleapis.com/v1beta1/projects/pelagic-bison-243518/zones/us-east1-c/clusters {
"cluster": { "name": "test", "masterAuth": { "clientCertificateConfig": {} }, "loggingService": "logging.googleapis.com",
"monitoringService": "monitoring.googleapis.com", "network": "projects/pelagic-bison-
243518/global/networks/default", "addonsConfig": { "httpLoadBalancing": {}, "horizontalPodAutoscaling": {},
"kubernetesDashboard": { "disabled": true }, "istioConfig": { "disabled": true } }, "subnetwork": "projects/pelagic-
bison-243518/regions/us-east1/subnetworks/default", "nodePools": [ { "name": "default-pool", "config": {
"machineType": "n1-standard-1", "diskSizeGb": 100, "oauthScopes": [
"https://www.googleapis.com/auth/devstorage.read_only", "https://www.googleapis.com/auth/logging.write",
"https://www.googleapis.com/auth/monitoring", "https://www.googleapis.com/auth/servicecontrol",
"https://www.googleapis.com/auth/service.management.readonly",
"https://www.googleapis.com/auth/trace.append" ], "imageType": "COS", "diskType": "pd-standard" },
"initialNodeCount": 3, "autoscaling": {}, "management": { "autoUpgrade": true, "autoRepair": true }, "version":
"1.12.8-gke.6" } ], "networkPolicy": {}, "ipAllocationPolicy": {}, "masterAuthorizedNetworksConfig": {},
"privateClusterConfig": {}, "databaseEncryption": { "state": "DECRYPTED" }, "initialClusterVersion": "1.12.8-gke.6",
"location": "us-east1-c" } }
```

2. **Install nginx ingress controller on the cluster. For now, we consider that the user will add public IP of ingress LoadBalancer to their /etc/hosts file for all hostnames to be used. So do not worry about DNS resolution.**

➔ Ingress controller needs a specific namespace, service account, cluster role bindings, configmaps etc. One can create all the kubernetes objects mentioned using the yaml file from official ingress repo.

1. $ kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/master/deploy/static/mandatory.yaml
2. Deploying nginx-ingress as LoadBalancer
   $ kubectl apply –f ingress-nginx_service.yaml

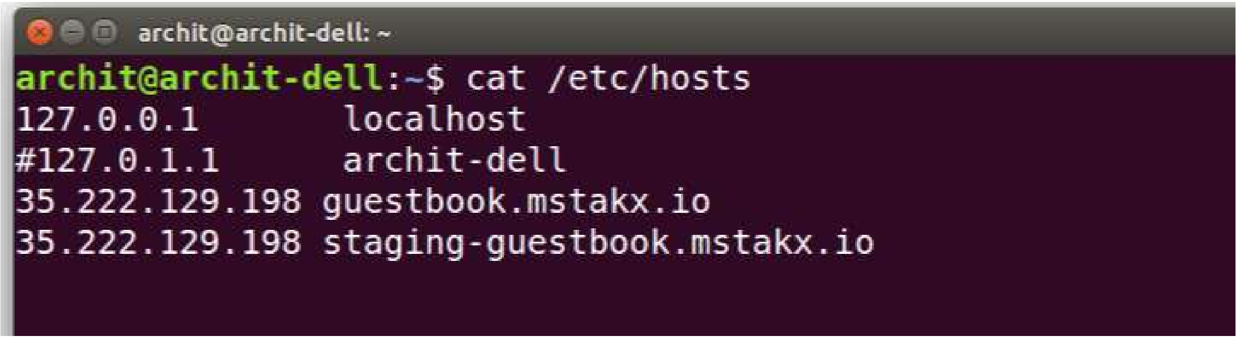
Default response of nginx after installation, listening on public IP address: 35.222.129.198



404 Not Found

nginx/1.15.10

Also updated /etc/hosts file on local machine



```
archit@archit-dell:~$ cat /etc/hosts
127.0.0.1       localhost
#127.0.1.1      archit-dell
35.222.129.198 guestbook.mstakx.io
35.222.129.198 staging-guestbook.mstakx.io
```

3. **On this cluster, create namespaces called staging and production.**
   1. $ kubectl create namespace staging
      namespe/staging created
   2. $ kubectl create namespace production
      namespace/production created

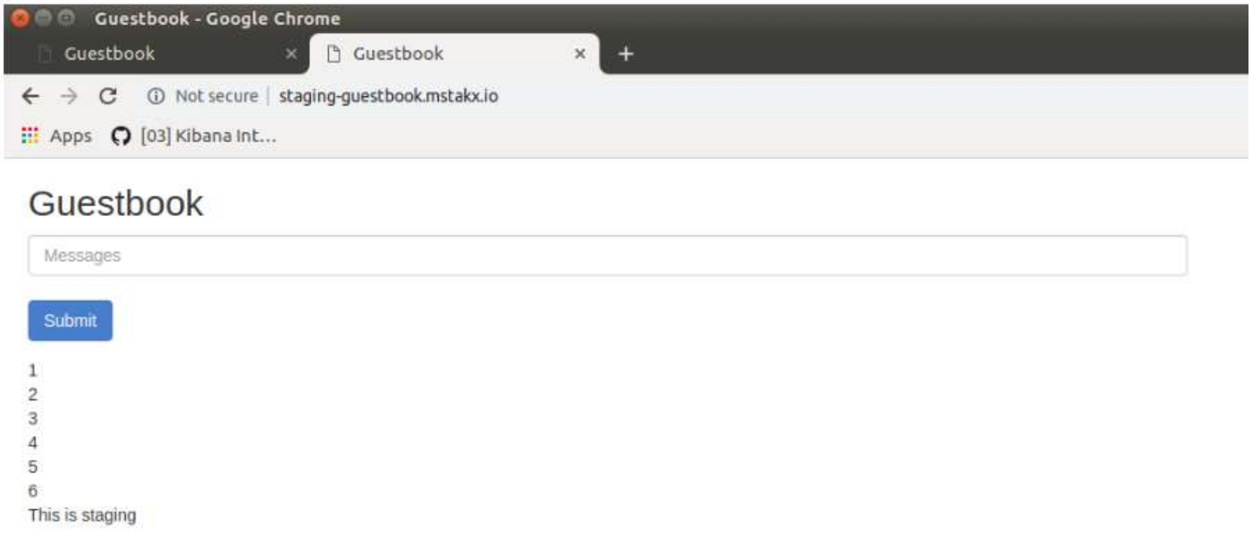
4. **Install guest-book application on both namespaces.**
   1. Installing guest-book application in staging namespace
      $ kubectl create -f redis-master-deployment.yaml -n staging
      $ kubectl create -f redis-master-service.yaml   -n staging
      $ kubectl create -f redis-slave-deployment.yaml -n staging
      $ kubectl create -f redis-slave-service.yaml   -n staging
      $ kubectl create -f frontend-deployment.yaml   -n staging
      $ kubectl create -f frontend-service_nginx.yaml -n staging

   2. Installing guest-book application in staging namespace
      $ kubectl create -f redis-master-deployment.yaml -n production
      $ kubectl create -f redis-master-service.yaml   -n production
      $ kubectl create -f redis-slave-deployment.yaml -n production
      $ kubectl create -f redis-slave-service.yaml   -n production
      $ kubectl create -f frontend-deployment.yaml   -n production
      $ kubectl create -f frontend-service_nginx.yaml -n production

- **Note**: frontend-deployment.yaml has little change then the one mentioned on github, added auto scale part and reduced CPU requirement, so that can show auto scaling of pod with lower traffic.

5. Expose staging application on hostname staging-guestbook.mstakx.io

➔ $ kubectl create –f ingress_staging_guestbook_routing.yaml –n staging



Guestbook

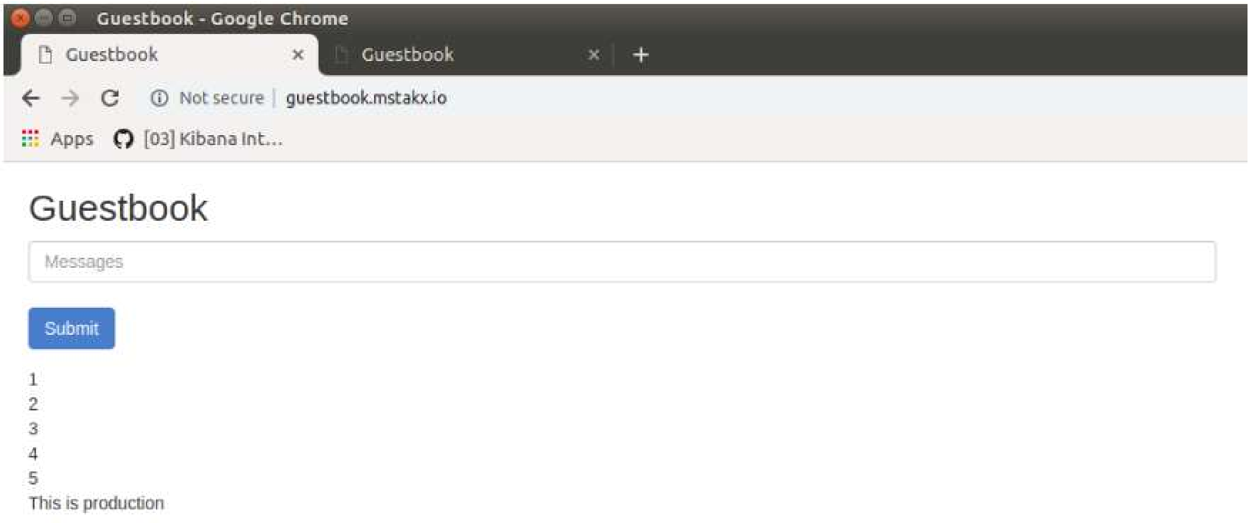
Messages

Submit

1
2
3
4
5
6
This is staging

6. Expose production application on hostname guestbook.mstakx.io

➔ $ kubectl create –f ingress_production_guestbook_routing.yaml –n production



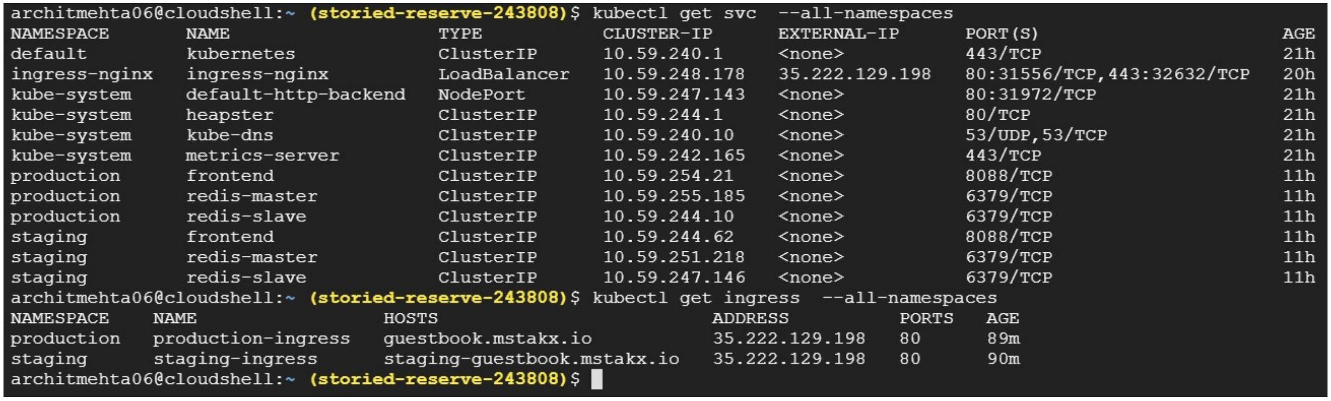
Guestbook

Messages

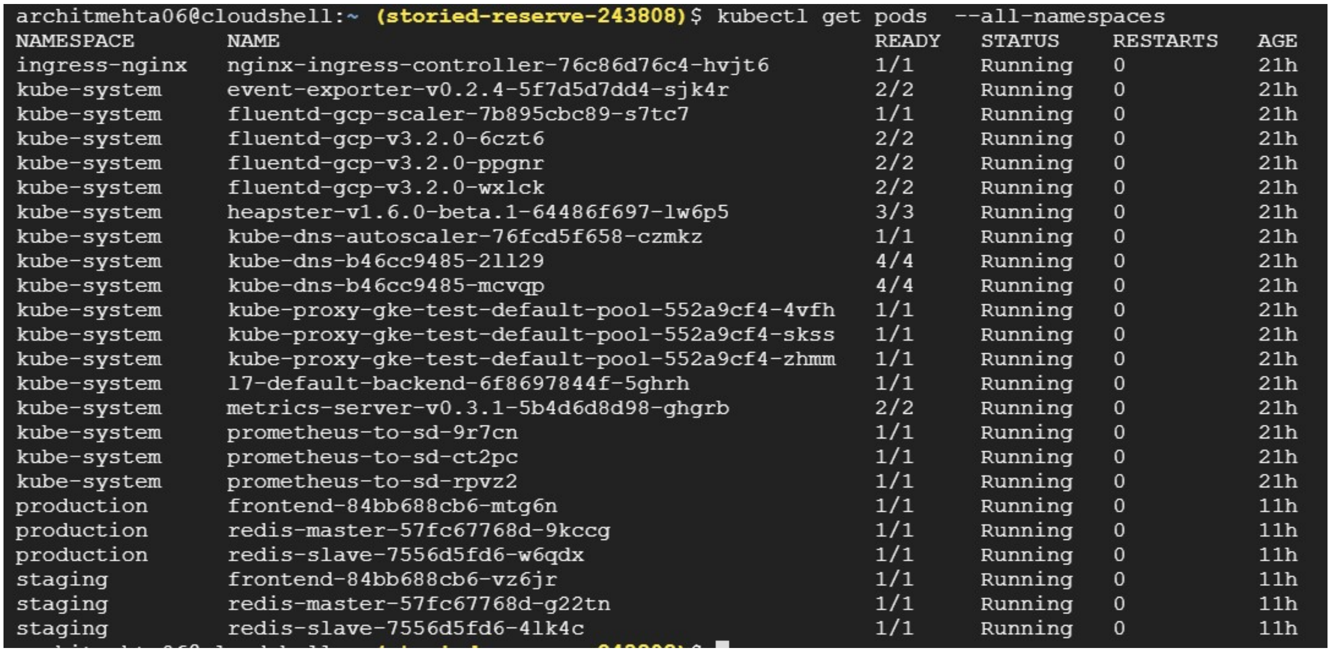
Submit

1
2
3
4
5
This is production

State of Kubernetes' pod, service and namespace after executing all above steps



```
architmehta06@cloudshell:~ (storied-reserve-243808)$ kubectl get svc  --all-namespaces
NAMESPACE      NAME                     TYPE          CLUSTER-IP      EXTERNAL-IP      PORT(S)                       AGE
default        kubernetes               ClusterIP     10.59.240.1     <none>           443/TCP                       21h
ingress-nginx  ingress-nginx            LoadBalancer  10.59.248.178   35.222.129.198   80:31556/TCP,443:32632/TCP    20h
kube-system    default-http-backend     NodePort      10.59.247.143   <none>           80:31972/TCP                  21h
kube-system    heapster                 ClusterIP     10.59.244.1     <none>           80/TCP                        21h
kube-system    kube-dns                 ClusterIP     10.59.240.10    <none>           53/UDP,53/TCP                 21h
kube-system    metrics-server           ClusterIP     10.59.242.165   <none>           443/TCP                       21h
production     frontend                 ClusterIP     10.59.254.21    <none>           8088/TCP                      11h
production     redis-master             ClusterIP     10.59.255.185   <none>           6379/TCP                      11h
production     redis-slave              ClusterIP     10.59.244.10    <none>           6379/TCP                      11h
staging        frontend                 ClusterIP     10.59.244.62    <none>           8088/TCP                      11h
staging        redis-master             ClusterIP     10.59.251.218   <none>           6379/TCP                      11h
staging        redis-slave              ClusterIP     10.59.247.146   <none>           6379/TCP                      11h
architmehta06@cloudshell:~ (storied-reserve-243808)$ kubectl get ingress  --all-namespaces
NAMESPACE    NAME                 HOSTS                         ADDRESS          PORTS   AGE
production   production-ingress   guestbook.mstakx.io           35.222.129.198   80      89m
staging      staging-ingress      staging-guestbook.mstakx.io   35.222.129.198   80      90m
architmehta06@cloudshell:~ (storied-reserve-243808)$
```



```
architmehta06@cloudshell:~ (storied-reserve-243808)$ kubectl get pods  --all-namespaces
NAMESPACE       NAME                                          READY   STATUS    RESTARTS   AGE
ingress-nginx   nginx-ingress-controller-76c86d76c4-hvjt6     1/1     Running   0          21h
kube-system     event-exporter-v0.2.4-5f7d5d7dd4-sjk4r        2/2     Running   0          21h
kube-system     fluentd-gcp-scaler-7b895cbc89-s7tc7           1/1     Running   0          21h
kube-system     fluentd-gcp-v3.2.0-6czt6                      2/2     Running   0          21h
kube-system     fluentd-gcp-v3.2.0-ppgnr                      2/2     Running   0          21h
kube-system     fluentd-gcp-v3.2.0-wxlck                      2/2     Running   0          21h
kube-system     heapster-v1.6.0-beta.1-64486f697-lw6p5        3/3     Running   0          21h
kube-system     kube-dns-autoscaler-76fcd5f658-czmkz          1/1     Running   0          21h
kube-system     kube-dns-b46cc9485-21129                      4/4     Running   0          21h
kube-system     kube-dns-b46cc9485-mcvqp                      4/4     Running   0          21h
kube-system     kube-proxy-gke-test-default-pool-552a9cf4-4vfh   1/1     Running   0          21h
kube-system     kube-proxy-gke-test-default-pool-552a9cf4-skss   1/1     Running   0          21h
kube-system     kube-proxy-gke-test-default-pool-552a9cf4-zhmm   1/1     Running   0          21h
kube-system     l7-default-backend-6f8697844f-5ghrh           1/1     Running   0          21h
kube-system     metrics-server-v0.3.1-5b4d6d8d98-ghgrb        2/2     Running   0          21h
kube-system     prometheus-to-sd-9r7cn                        1/1     Running   0          21h
kube-system     prometheus-to-sd-ct2pc                        1/1     Running   0          21h
kube-system     prometheus-to-sd-rpvz2                        1/1     Running   0          21h
production      frontend-84bb688cb6-mtg6n                     1/1     Running   0          11h
production      redis-master-57fc67768d-9kccg                 1/1     Running   0          11h
production      redis-slave-7556d5fd6-w6qdx                   1/1     Running   0          11h
staging         frontend-84bb688cb6-vz6jr                     1/1     Running   0          11h
staging         redis-master-57fc67768d-g22tn                 1/1     Running   0          11h
staging         redis-slave-7556d5fd6-41k4c                   1/1     Running   0          11h
```

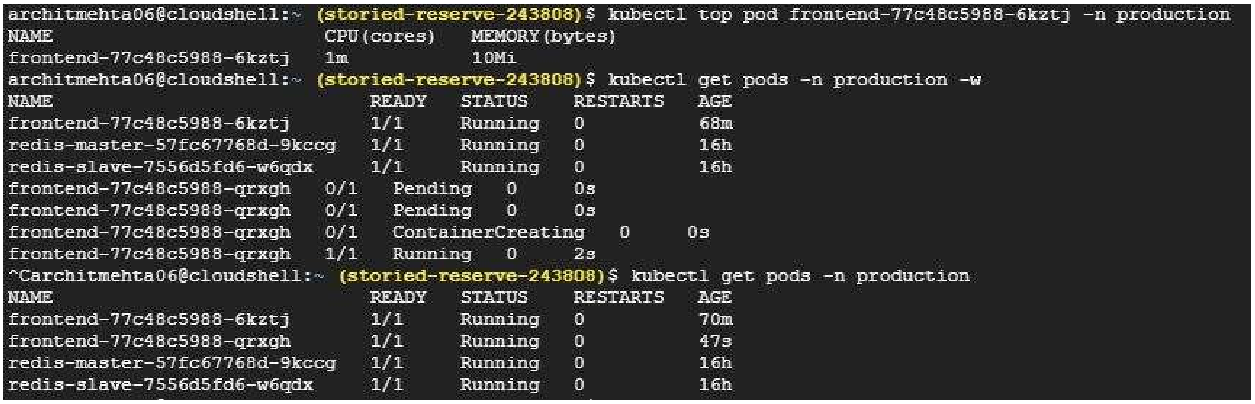
7. **Implement a pod autoscaler on both namespaces which will scale frontend pod replicas up and down based on CPU utilization of pods.**

➔ Added “HorizontalPodAutoscaler” kind with minReplicas: 1 and maxReplicas: 3 with targetCPUUtilizationPercentage: 3 , this component added to “frontend-deployment.yaml”

8. **Write a script which will demonstrate how the pods are scaling up and down by increasing/decreasing load on existing pods.**

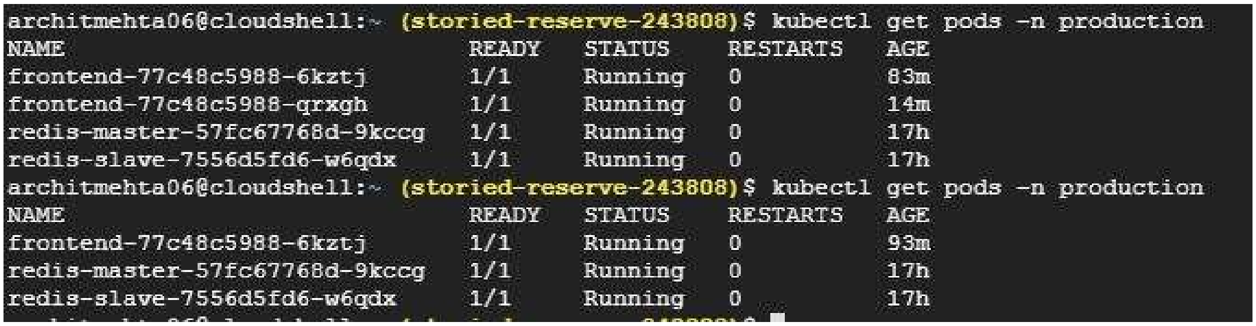
➔ Wrote shell script to send http request to front end in infinite loop, and started watch on “kubectl get pod” command in other terimnal.
Script: load_generator.sh (added to github)

<u>Scale Up</u>: Below snapshot shows that when script was running, after some time it created another pod



```
architmehta06@cloudshell:~ (storied-reserve-243808)$ kubectl top pod frontend-77c48c5988-6kztj -n production
NAME                        CPU(cores)   MEMORY(bytes)
frontend-77c48c5988-6kztj   1m           10Mi
architmehta06@cloudshell:~ (storied-reserve-243808)$ kubectl get pods -n production -w
NAME                            READY   STATUS    RESTARTS   AGE
frontend-77c48c5988-6kztj       1/1     Running   0          68m
redis-master-57fc67768d-9kccg   1/1     Running   0          16h
redis-slave-7556d5fd6-w6qdx     1/1     Running   0          16h
frontend-77c48c5988-qrxgh   0/1   Pending   0     0s
frontend-77c48c5988-qrxgh   0/1   Pending   0     0s
frontend-77c48c5988-qrxgh   0/1   ContainerCreating   0     0s
frontend-77c48c5988-qrxgh   1/1   Running   0     2s
^Carchitmehta06@cloudshell:~ (storied-reserve-243808)$ kubectl get pods -n production
NAME                            READY   STATUS    RESTARTS   AGE
frontend-77c48c5988-6kztj       1/1     Running   0          70m
frontend-77c48c5988-qrxgh       1/1     Running   0          47s
redis-master-57fc67768d-9kccg   1/1     Running   0          16h
redis-slave-7556d5fd6-w6qdx     1/1     Running   0          16h
```

Scale Down: After that, I stopped script and it automatically deleted the 2nd pod after sometime



```
architmehta06@cloudshell:~ (storied-reserve-243808)$ kubectl get pods -n production
NAME                            READY   STATUS    RESTARTS   AGE
frontend-77c48c5988-6kztj       1/1     Running   0          83m
frontend-77c48c5988-qrxgh       1/1     Running   0          14m
redis-master-57fc67768d-9kccg   1/1     Running   0          17h
redis-slave-7556d5fd6-w6qdx     1/1     Running   0          17h
architmehta06@cloudshell:~ (storied-reserve-243808)$ kubectl get pods -n production
NAME                            READY   STATUS    RESTARTS   AGE
frontend-77c48c5988-6kztj       1/1     Running   0          93m
redis-master-57fc67768d-9kccg   1/1     Running   0          17h
redis-slave-7556d5fd6-w6qdx     1/1     Running   0          17h
```

9. Write a wrapper script which does all the steps above. Mention any pre-requisites in the README.md at the root of your repo.
   The evaluator will proceed by going over the steps mentioned in the README. So try to make this as automated as possible.

➔ automation.py
   1. It installs nginx and guestbook application for list of namespaces
   2. Configures hostnames for both namespaces in nginx controller

In the context of above test, please explain the following:

- What was the node size chosen for the Kubernetes nodes? And why?

➔ Number of nodes chosen: 3 (1 Master 2 Worker)
Just went with default 3 node cluster. That is enough for the demo, for production we need to have backup of master node (works as HA if master goes down) and worker node depends on the compute and application requirement.
Machine Type: n1-standard-1 (1 vCPU, 3.75 GB memory) which makes total 3 vCPU and memory 11.25 GB
While choosing, goal was just to implement this particular assignment. I looked at basic guestbook application for which "cpu: 50m" and "memory: 100Mi" with replicacount: 1 and not being cpu or memory intensive this quota is sufficient.

- What method was chosen to install the demo application and ingress controller on the cluster, justify the method used

➔ Got all the yaml files for guestbook Application, nginxcontroller and adding ingress object, created separate yaml files for deployment and service for better understanding. Used "kubectl create" to apply all yaml files.
This way one can clearly understand the steps and process about how complete application is getting set up and working. Alternative way is to prepare helm chart and with single command we can deploy whole set up.

- What would be your chosen solution to monitor the application on the cluster and why?

➔ ELK/EFK: Application is used to solve business problem and the logging events are also generated according to business logic which are highly dimensional. EFK uses indexing mechanism to store everything and provides option to search. Having to search/monitor on n number of key this works efficiently. It is also industry proven and heavily used with large scale.

➔ For system related metric can be pegged and stored in prometheus

- What additional components/plugins would you install on the cluster to manage it better?

➔ prometheus: To measure cluster health and other metrics (ex: container_cpu_usage_total) and for other system metrics
Grafana: for viewing matrics
Jenkis: for CI/CD