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" } }

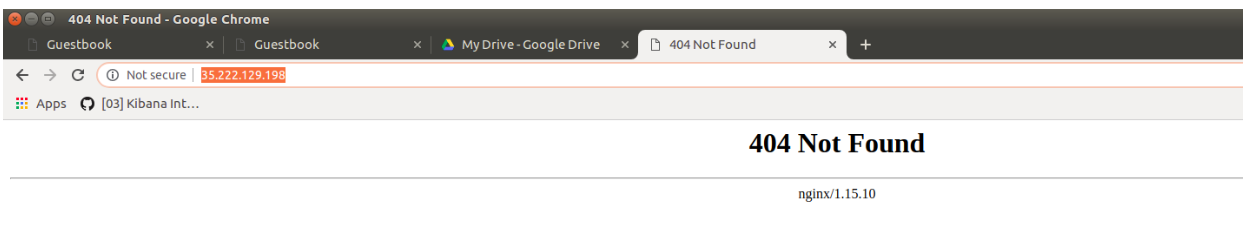
1. **Install nginx ingress controller on the cluster. For now, we consider that the user will add public IP of ingress LoadBalancer to their /etc/hosts file for all hostnames to be used. So do not worry about DNS resolution.**

* 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.](https://github.com/kubernetes/ingress-nginx/tree/master/deploy" \t "_blank)

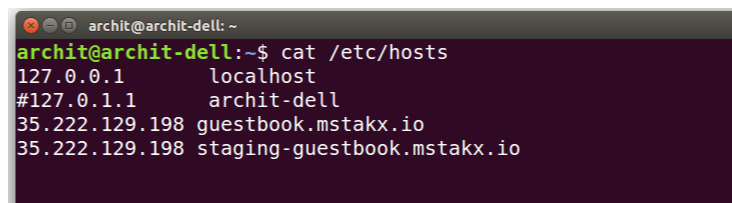
1. $ kubectl apply -f <https://raw.githubusercontent.com/kubernetes/ingress-nginx/master/deploy/static/mandatory.yaml>
2. Deploying nginx-ingress as LoadBalancer

$ kubectl apply –f ingress-nginx\_service.yaml

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



Also updated /etc/hosts file on local machine



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

namespe/staging created

1. $ kubectl create namespace production

namespace/production created

1. **Install** [**guest-book**](https://github.com/kubernetes/examples/tree/master/guestbook) **application on both namespaces.**
2. 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

1. 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

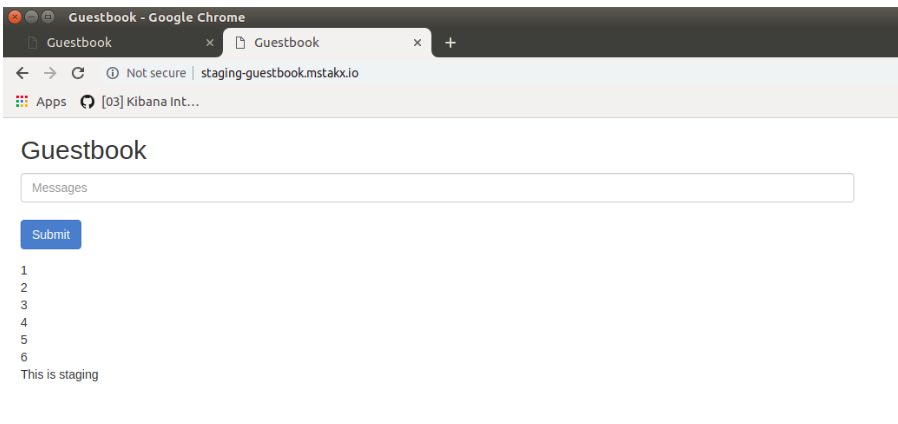
$ kubectl create -f frontend-deployment.yaml -n production

$ 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.

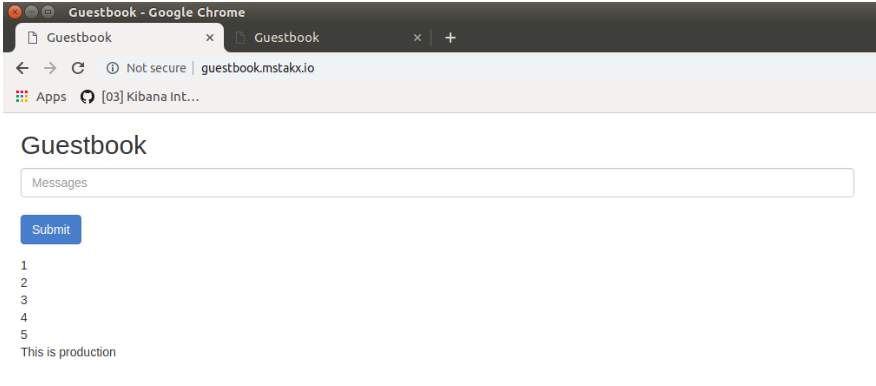
1. **Expose staging application on hostname staging-guestbook.mstakx.io**

* $ kubectl create –f ingress\_staging\_guestbook\_routing.yaml –n staging

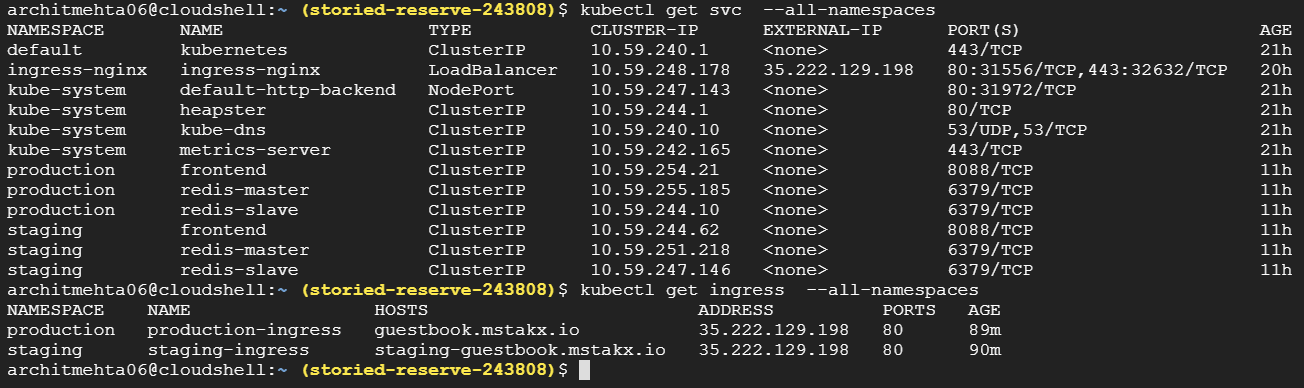


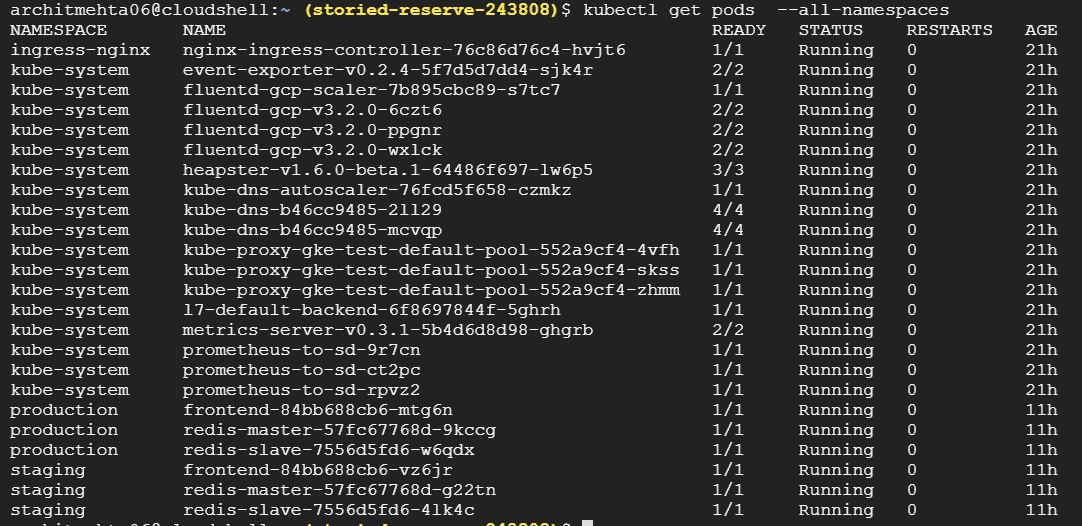
1. **Expose production application on hostname guestbook.mstakx.io**

* $ kubectl create –f ingress\_production\_guestbook\_routing.yaml –n production



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





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

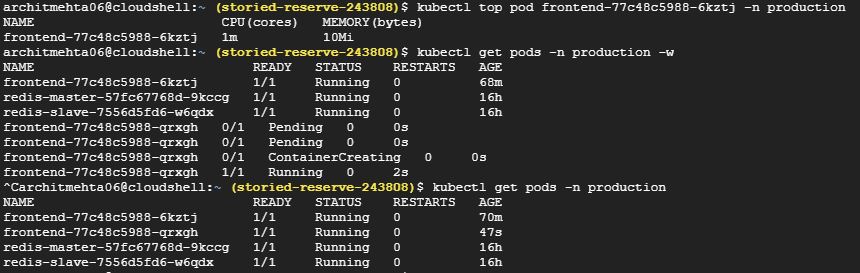
* Added “HorizontalPodAutoscaler” kind with minReplicas: 1 and maxReplicas: 3 with targetCPUUtilizationPercentage: 3 , this component added to “frontend-deployment.yaml”

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

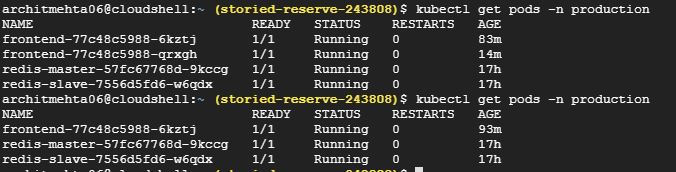
* Wrote shell script to send http request to front end in infinite loop, and started watch on “kubectl get pod” command in other terimnal.

Script: load\_generator.sh (added to github)

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



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



1. 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