Deployment Steps:

Please follow the below steps in order to Deploy this application.

1. ## Create Namespace:

   As shown in the above architecture, we shall deploy the application in two namespace. Namespaces are virtual clusters that can sit on top of the same physical cluster. We shall use a declarative file to create two Namespaces.

   1. Staging
   2. Production

   Please use the below commands to create the above mentioned namespaces.

   ```
   $ kubectl create -f GuestBookApp/Namespaces/Namespace.yml
   ```

   Once executed please use the below command to get all the namespaces. You will be able to see both the Production & Staging Namespaces created.

   ```
   $ kubectl describe namespace
   ```

2. ## Deploy the Guestbook Application in both the Namespace:

   Now, we shall deploy our application in both the Namespaces. Please use the below command to deploy the application in Staging Namespace.

   ```
   $ kubectl create -f GuestBookApp/ApplicationDeployment -n staging
   ```

   Now, lets deploy the application in Production Namespace

   ```
   $ kubectl create -f GuestBookApp/ApplicationDeployment -n production
   ```

   The application is deployed using Kind: Deployment. Please use the below command to get deployments.

   ```
   $ kubectl get deployments -n production
   ```

   ```
   $ kubectl get deployments -n staging
   ```

3. ## Create a Service Account:

   Now, we need to create a Service Account. A service account provides an identity for processes that run in a Pod. When a User access the cluster they are authenticated by the API Server as a particular User Account.

Processes in containers inside pods can also contact the API Server. When they do, they are authenticated as a particular Service Account.
Later, here we will be creating Ingress Controller which will be used to monitor Ingress Resources. In order to monitor all the Ingress Resources we shall need this Service Account with roles and roles binding.

Use the below command to create the Service Account.

```
$ kubectl create -f GuestBook/K8sRoles/ServiceAccount.yml -n staging
```

Please remember that you need to create the Service Account in the same namespace you are creating the Ingress Controller. Here we will be creating the Ingress Controller in the Staging Namespace. If you want you can create the Ingress Controller and Service Account in the default Namespace or any other Namespace as well.

Additionally, you can create required Roles & Roles Binding in the next step and attach it to the default Service Account. Every Namespace that is created does have a default Service Account.

Please use the below command to get the created service account. You shall see an ***nginx-ingress-serviceaccount*** created.

```
$ kubectl get service account -n staging
```

4. Create a Role & Bind the Role to Service Account:

As mentioned above we need to create a Role and Bind that Role to the Service Account that we created in above steps.

Please remember Roles and Rolebindings are specific to namespaces. Also please create the Role and Rolebinding in the same Namespace where you created the Service Account. Here we created the Service Account in Staging Namespace so we shall create the role and rolebinding the Staging Namespace itself.

Please use the below command to create the Role.

```
$ kubectl create -f GuestBook/K8sRoles/Roles.yml -n staging
```

Please use the below command to check whether the role has been created or not.

```
$ kubectl get roles -n staging
```

You shall see a role with name ***nginx-ingress-role.***

Next, we shall create a Rolebinding and bind the above created Role to our Service Account.

Please use the below step to create a Rolebinding.

```
$ kubectl create -f GuestBook/K8sRoles/RolesBInding.yml -n staging
```

Please use the below command to check whether the rolebinding has been created or not.

```
$ kubectl get rolebindings -n staging
```

You shall see an rolebinding with name ***nginx-ingress-role-binding*** created. You can also describe the above created rolebinding to see if the role is bound to Service Account or not. Please use the below command to describe the role binding.

```
$ kubectl describe rolebindings -n staging
```

Please see the below output.

```
ubuntu@ip-172-31-44-53:~$ kubectl describe rolebinding -n staging
Name:         nginx-ingress-role-binding
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  Role
  Name:  nginx-ingress-role
Subjects:
  Kind            Name                          Namespace
  ----            ----                          ---------
  ServiceAccount  nginx-ingress-serviceaccount
```

5. Create a Cluster Role and Bind the Cluster Role to Service Account:

We need to create Cluster Role and Bind that Role to the Service Account. Where Roles are limited to Namespaces, Cluster Roles spread across K8s Clusters and each User or Service Account have the same rights as mentioned in the Cluster Roles.

Please use the below command to create the Cluster Role.

```
$ kubectl create -f GuestBook/K8sRoles/ClusterRole.yml
```

Here we are not providing the namespace as this Cluster Role will be used across the Cluster. We shall bind the Cluster Role to our Service Account. Please use the below command to check the created cluster role.

```
$ kubectl get clusterroles
```

Here you shall see all the clusterroles that are created by Kubernetes as well. You shall see a cluster role with name ***nginx-ingress-clusterrole.***

```
ubuntu@ip-172-31-44-53:~$ kubectl get clusterroles
NAME                                                                   AGE
admin                                                                  7d19h
cluster-admin                                                          7d19h
edit                                                                   7d19h
nginx-ingress-clusterrole                                              2d3h
system:aggregate-to-admin                                              7d19h
system:aggregate-to-edit                                               7d19h
system:aggregate-to-view                                               7d19h
system:aggregated-metrics-reader                                       4d3h
system:auth-delegator                                                  7d19h
system:aws-cloud-provider                                              7d19h
system:basic-user                                                      7d19h
system:certificates.k8s.io:certificatesigningrequests:nodeclient       7d19h
system:certificates.k8s.io:certificatesigningrequests:selfnodeclient   7d19h
system:controller:attachdetach-controller                              7d19h
system:controller:certificate-controller                               7d19h
system:controller:clusterrole-aggregation-controller                   7d19h
system:controller:cronjob-controller                                   7d19h
system:controller:daemon-set-controller                                7d19h
```

Next, we need to create a rolebinding and bind the role that we created above to the Service Account that we created previously.

Please use the below command to create the Rolebinding.

```
$ kubectl create -f GuestBook/K8sRoles/ClusterRoleBinding.yml
```

Please use the below command to check whether the clusterbinding has been created or not and bound to the service account.

```
$ kubectl create -f GuestBook/K8sRoles/ClusterRoleBinding.yml
```

You shall see a binding with name ***nginx-ingress-clusterrole-binidng.*** Please see the below image.

```
ubuntu@ip-172-31-44-53:~$ kubectl get clusterrolebinding
NAME                                                   AGE
cluster-admin                                          7d19h
kubeadm:kubelet-bootstrap                              7d19h
kubeadm:node-autoapprove-bootstrap                     7d19h
kubeadm:node-autoapprove-certificate-rotation          7d19h
kubeadm:node-proxier                                   7d19h
metrics-server:system:auth-delegator                   4d3h
nginx-ingress-clusterrole--binding                     2d3h
nginx-ingress-clusterrole-nisa-binding                 6d1h
system:aws-cloud-provider                              7d19h
system:basic-user                                      7d19h
system:controller:attachdetach-controller              7d19h
system:controller:certificate-controller               7d19h
system:controller:clusterrole-aggregation-controller   7d19h
system:controller:cronjob-controller                   7d19h
system:controller:daemon-set-controller                7d19h
system:controller:deployment-controller                7d19h
system:controller:disruption-controller                7d19h
system:controller:endpoint-controller                  7d19h
system:controller:expand-controller                    7d19h
system:controller:generic-garbage-collector            7d19h
system:controller:horizontal-pod-autoscaler            7d19h
system:controller:job-controller                       7d19h
system:controller:namespace-controller                 7d19h
system:controller:node-controller                      7d19h
system:controller:persistent-volume-binder             7d19h
system:controller:pod-garbage-collector                7d19h
```

You can use the below command to check if the cluster role is bound to Service Account or not.

```
$ kubectl describe clusterrolebinding nginx-ingress-clusterrole-binidng
```

Please see the below screenshot that mentions the Cluster Role is bounded to the Service Account.

```
ubuntu@ip-172-31-44-53:~$ kubectl describe clusterrolebinding nginx-ingress-clusterrole--binding
Name:         nginx-ingress-clusterrole--binding
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  ClusterRole
  Name:  nginx-ingress-clusterrole
Subjects:
  Kind            Name                          Namespace
  ----            ----                          ---------
  ServiceAccount  nginx-ingress-serviceaccount  staging
```

6. Create an Ingress Controller:

Now that all our required resources are created we shall now be creating the Ingress Controller. We will be creating the Ingress Controller in the Staging Namespace. Ingress Controller will be Pod created using Kind Deployment.

Please use the below command to create the Ingress Controller.

```
$ kubectl create -f GuestBookApp/Ingress/Ingress-Controller-Deployment.yml -n staging
```

Once the Ingress Controller is created we shall create a Service of Type Node Port. This shall expose the Ingress Controller on Port 80 on the node for handling requests coming to the Ingress Controller.

Please use the below command to create the Ingress Service.

```
$ kubectl create -f GuestBookApp/Ingress/Ingress-Service.yml -n staging
```

We need to create a services where all the requests will be routed if the rules defined in the Ingress Resources does not match. You shall create the service in the default namespace as well or any other namespace as well. You just need to change the below parameter in the Ingress Controller Deployment file.

```
--default-backend-service=tutum-namespace/tutum-hello-world-service
```

Here, we will be creating a service ***tutum-hello-world-service*** in ***tutum-namespace*** Namespace and all the request that does not matches the Rules in Ingress will be redirected to this service. This service will be attached to a pod. Please use the below steps to implement this.

```
$ kubectl create ns tutum-namespace && kubectl create -f TutumHelloWorldApp/
```

7. Create Ingress Resources:

Now we need to create Ingress Resources which will redirect the request coming to the Ingress Controller from the external world to the Guest Book Application. We shall create the Ingress Resources in both the Namespaces so that the request can be redirected to the desired Namespace based on the Hostname.

Please use the below command to create the Ingress Resources in Staging Namespace.

```
$ kubectl create -f GuestBookApp/Ingress/Ingress-Rules.yml -n staging
```

You can create the same resources in the Production Namespace using below command.

```
$ kubectl create -f GuestBookApp/Ingress/Ingress-Rules.yml -n production
```

Please check whether the Ingress resources has been created or not using below command.

```
$ kubectl get ing –all-namespaces
```

Here you can see the Ingress Resources that we created in the above step. You shall see an Ingress Resource with name ingress-wear created in both the namespaces.

```
ubuntu@ip-172-31-44-53:~$ kubectl get ing --all-namespaces
NAMESPACE          NAME           HOSTS                          ADDRESS   PORTS
 AGE
production         ingress-wear   guestbook.mstakx.io                      80
 2d4h
staging            ingress-wear   staging-guestbook.mstakx.io              80
 2d4h
```

You can check the details of these resources using below command. You can see the resources as well that are defined while creating them.

```
$ kubectl describe ing  ingress-wear –n staging
```

8. Deploy the Metrics Server for gathering the Resource Info:

We will be creating Metrics Server to monitor the CPU on the Guestbook Application Pod and Ingress Controller Pod. In the later step we will be creating Autoscaler which will be use the metrics provided by Metrics Sever to scale in and scale out the Pods.

Please use the below command to deploy the Metrics Server.

```
$ git clone https://github.com/kubernetes-incubator/metrics-server.git && kubectl create -f metrics-server/deploy/1.8+/
```

Please use the below command to verify if the metrics server is running or not in the cluster.

```
$ kubectl top pods –all-namespaces
```

This shall show all the pods and their CPU Usage & Memory Usage. Please see the below sample output.

```
ubuntu@ip-172-31-44-53:~$ kubectl top pods --all-namespaces
NAMESPACE          NAME                                       CPU(cores)   MEMORY(bytes)
default            frontend-74b4665db5-ljgld                  1m           10Mi
default            frontend-74b4665db5-pg4cx                  1m           14Mi
default            frontend-74b4665db5-t57tz                  1m           7Mi
default            redis-master-6fbbc44567-wggcz              1m           9Mi
default            redis-slave-74ccb764fc-2x79m               1m           8Mi
default            redis-slave-74ccb764fc-qlx5c               1m           8Mi
kube-system        coredns-54ff9cd656-gbwtx                   3m           19Mi
kube-system        coredns-54ff9cd656-tfzfk                   3m           19Mi
kube-system        etcd-ip-172-31-44-53                       18m          343Mi
kube-system        kube-apiserver-ip-172-31-44-53             46m          434Mi
kube-system        kube-controller-manager-ip-172-31-44-53    52m          57Mi
kube-system        kube-proxy-hjthz                           3m           21Mi
kube-system        kube-proxy-m5qnx                           6m           27Mi
kube-system        kube-proxy-vrtxb                           3m           19Mi
kube-system        kube-scheduler-ip-172-31-44-53             16m          22Mi
kube-system        metrics-server-7f74c46cb6-9v47h            1m           20Mi
kube-system        weave-net-dfvq2                            2m           113Mi
kube-system        weave-net-mvtnr                            4m           92Mi
kube-system        weave-net-xlmr6                            4m           100Mi
production         frontend-74b4665db5-6478p                  1m           8Mi
production         redis-master-6fbbc44567-hgk2p              1m           9Mi
production         redis-slave-74ccb764fc-htt8l               1m           7Mi
production         redis-slave-74ccb764fc-wqp9l               1m           7Mi
staging            frontend-74b4665db5-gptmd                  1m           8Mi
staging            ingress-controller-b66d8dd57-78rjw         2m           67Mi
staging            redis-master-6fbbc44567-59984              1m           9Mi
staging            redis-slave-74ccb764fc-npsft               1m           8Mi
staging            redis-slave-74ccb764fc-pmql6               1m           7Mi
tutum-namespace    tutum-hello-world-pod                      1m           7Mi
ubuntu@ip-172-31-44-53:~$
```

9. Create an Horizontal Pod Autoscaler for Guestbook Application:

Now since the metrics server is up and running and it has started collecting the Pod metrics, we shall now create an Autoscaler for our Guestbook Application. We shall create the Autoscaler in both the namespaces so that pods in both the namespace can scale in and scale out as per the load.

Please use the below step to create an Autoscaler for Guestbook Application in Staging & Production Namespaces.

```
$ kubectl create -f HorizontalPodScaler/FrontEndAutoScaler.yml -n staging
```

```
$ kubectl create -f HorizontalPodScaler/FrontEndAutoScaler.yml -n prodcution
```

Please use the below command to check if the Autoscaler is created or not.

```
$ kubectl get hpa –all-namespaces
```

```
ubuntu@ip-172-31-44-53:~$ kubectl get hpa --all-namespaces
NAMESPACE    NAME                     REFERENCE                 TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
production   fronted-app-autoscaler   Deployment/frontend       1%/50%    1         5         1          2d4h
staging      fronted-app-autoscaler   Deployment/frontend       1%/50%    1         5         1          2d4h
```

10. Create an Horizontal Pod Autoscaler for Ingress Controller:

Now we shall create an Autoscaler for Ingress Controller as well. This will help to scale the Ingress Controller as well if the load on the Ingress Controller Increases.

Please use the below command to create an Autoscaler for Ingress Controller. Please make sure to deploy the Ingress Autoscaler in the same namespace where the Ingress Controller is deployed. Here we shall deploy the Ingress Auto Scaler in the Staging Namespace.

```
$ kubectl create -f HorizontalPodScaler/IngressAutoScaler.yml -n staging
```

11. Validate the Deployment:

You have now deployed all the required resources for the application to work. Now we shall test the application.

You can use the below curl command to test the application.

```
$ curl -I 172.31.36.224 -H 'Host: guestbook.mstakx.io'
```

You shall see a 200 Status Code as shown in the below fig.

```
ubuntu@ip-172-31-44-53:~$ curl -I 172.31.36.224 -H 'Host: guestbook.mstakx.io'
HTTP/1.1 200 OK
Server: openresty/1.15.8.1
Date: Sat, 20 Jul 2019 10:26:11 GMT
Content-Type: text/html
Content-Length: 921
Connection: keep-alive
Vary: Accept-Encoding
Last-Modified: Wed, 09 Sep 2015 18:35:04 GMT
ETag: "399-51f54bdb4a600"
Accept-Ranges: bytes
Vary: Accept-Encoding
```