In this example show the practical concepts:

- pod scaling front end scale up and down easily,
- rolling update when changing the front end image version, have function to check the status of the rolling updates
- Concept of stateless, in memory caching
    - Notice in yaml file the resources requested are all memory and cpu related not persistent storage (so your session will not be saved)
- Load Balancer for front end, different types Cluster IP (No external), Node Port means all containers have an external IP, Load Balancer

Launch Deployment and service together

Practical Start
connect your kubernetes cluster to your cloud console. Create another project and connect your cloud console see if it needs to be recreated again.
kubectl apply -f (all three files)
check the deployment and services after the creation of each and check the pods

To just see the front end pods
```
kubectl get pods -l tier=frontend
```

To just see the backend pods
```
kubectl get pods -l tier=backend
```

Go to your GCP Console and see the workloads which is your deployments and see your discovery and load balancing. You can see a good overview of your kubernetes platform from the GCP Console.

Currently its load balancer
Open up the Guestbook url by entering in the external ip address of your front end service
Enter in some names. Open up other sessions. You see the names are still there. Now we will delete the pods and see if the name is still there.

Stateless and Desired State Example

Note that when you delete the pods with tier of front end it will regenerate the lost three pods because it is the desired state.
Open up another window and type in
kubectl get pods --watch

any changes you do to the pods will be seen here now delete the pods and see the changes in the watch terminal
kubectl delete pods -l tier=backend

You should see that the backend pods are being terminated

kubectl get pods -l tier=backend
Wait until all pods have finished terminating and are running.
Now open up the webpage again is the guest list still there?
Nope, this is because the cluster was designed to be stateless. If the backend pods gets recreated or terminated by some reason you have lost your data.

Load Balancer Example

The load balancer distributes requests evenly between all the pods in a deployment so we will test this by sending a few requests to the front end end point to see if different pods respond or is it just the one.

The response can be seen in the pods logs.
Open up the webpage and enter in a name
Sam and check the logs for the first pod.
Then open up a new browser and enter in another name Paul. Check the second pod logs.

kubectl logs <pod name>

Then open up yet another browser this time enter in Lilly, check the third pod's logs it should be there and not in the other pod's logs. You can actually see this in GCP Console's interface as well. It look nicer there. But its good to know the command to view the logs. This shows load balancing in action. For every active browser or get request the workload for the request is distributed between all the available server pods.

Node Port

Edit the front end deployment yaml file where the service config says load balancer and change it to NodePort and add in nodePort option. Now run
kubectl apply -f frontend-deployment.yaml
kubectl get svc -l tier=frontend
kubectl describe svc -l tier=frontend

Now since we have moved away from a load balancer which google would create for us automatically and will expose an end point to the public. We have to manually allow this end point and port which we have specified to be available. To do this we need to create a new firewall rule using this command:

```
gcloud compute firewall-rules create guestbook --allow tcp:30080
```

This will allow all nodes to expose the port 30080 to the public.

Now we need to find our nodes external IP address by using:

```
gcloud compute instances list
```

And since we only have one node in our cluster we just take the external IP of that one node and use the port in the browser eg:
http://35.189.48.21:30080
and you should see your guestbook

ClusterIP

Now commend out the node port and name and change it to Cluster IP then:
kubectl apply -f frontend-deployment.yaml
kubectl get svc -l tier=frontend
kubectl describe svc -l tier=frontend

From the description you can see there is no nodeport and you cannot access this from your browser. You can try to copy the clusterIP and paste it in your browser it doesn't work..
However if you do a curl request in your local environment you will get a result: The below command executes a piece of code inside of the pod name specified:
get a pod name by
kubectl get pod -l tier=frontend
`kubectl exec -ti <POD Name> curl localhost:80`

As you can see you can see the guesbook title and form etc which is what you would expect the html document to look like.

Changing Replication Scale your Nodes

There are two ways of doing this the preferred way is to always edit your yaml file because usually you will use git and it keeps track of the changes you have made.
Its pretty simple you can just edit the replicas from 3 to 5 if you want 5 front end pods.
kubectl apply -f frontend-deployment.yaml
kubectl get pod -l tier=frontend

You can see there are now 5 pods. The other way of doing this is to just issue a command
kubectl scale deployment frontend --replicas=3
kubectl get pod -l tier=frontend

Again you see the number of pods have changed.

Rolling Update

The last exercise that I want to showcase is a rolling update. Often a team want to update a new frontend to their production environment but don't want to disrupt users so a rolling update is a perfect choice. Always make sure there are available resources to meet demand while updating the pods one by one.
Lets update the docker image to version 5 and check out the rollout status
edit the yaml file to v5.

kubectl apply -f frontend-deployment.yaml

kubectl rollout status deployments/frontend
kubectl describe deployment frontend
you can see that the image has been changed to v5

//Undo rollout if there are some bugs for example
kubectl rollout undo deployments/frontend
kubectl describe deployment frontend
you can see that the image has been changed back to v4