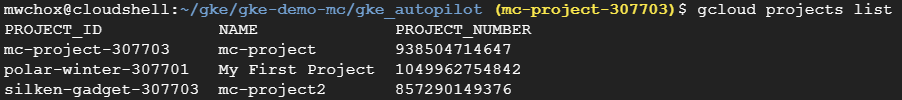
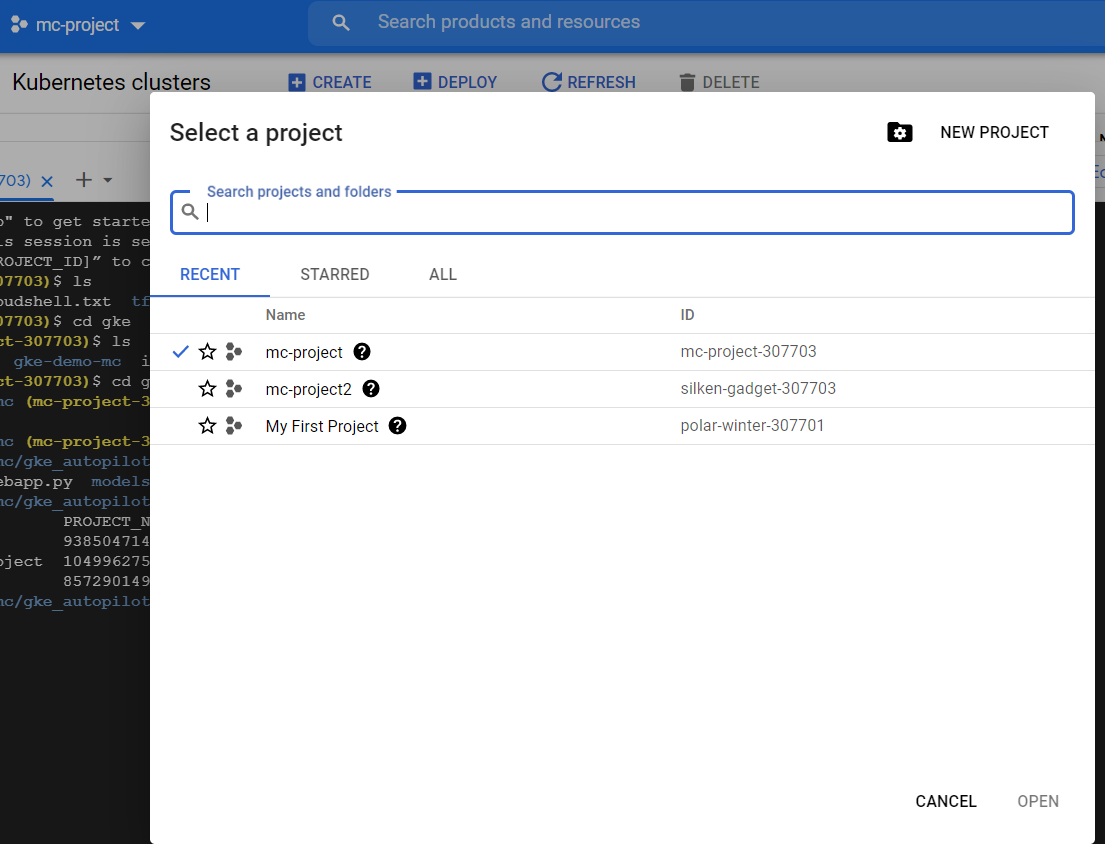
1. Access the cloud shell in top right. You should see it default to a project highlighted in yellow. In my case its mc-project-307703. You can list different projects via the following command. Remember to use the project-id instead of the project name for successive commands. You can also create new projects from the top left.

gcloud projects list



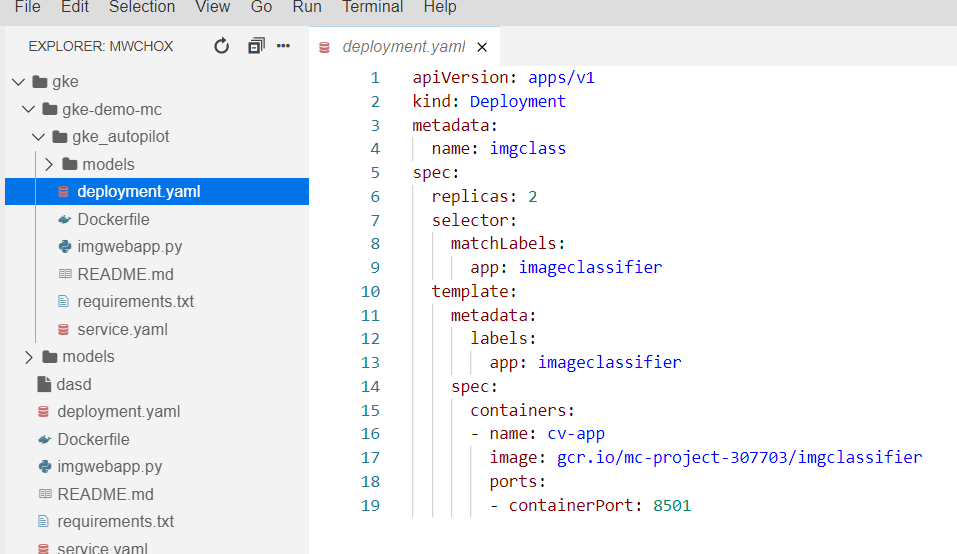
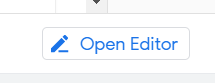






1. At the top, navigate to cloud storage, create a bucket, and then upload the gke\_autopilot folder to that bucket. The storage can now be accessed like hdfs via gsutil command. You have to copy from hdfs to your remote local via the command gsutil cp -r gs://<bucket>/gke\_autopilot . . A folder named gke\_autopilot should now be there. Change directory to there via cd gke\_autopilot.
2. Now, we need to create a Kubernetes cluster. Search for Kubernetes cluster at the top and create. You’ll see two options, one for Standard and Autopilot. Pick Standard. This will take some time to finish. You can also create a Kubernetes cluster from the cloud shell using the command below.

gcloud container clusters create <name of cluster> --zone <zone> --enable-basic-auth --machine-type n1-standard-1 --num-nodes <number of nodes> --scopes https://www.googleapis.com/auth/logging.write,https://www.googleapis.com/auth/devstorage.read\_only

1. Now, you can see the files more in depth and also edit by clicking on open editor button. You’ll notice certain files in there.
   1. Requirements.txt – list of python libraries to download
   2. Dockerfile – definition of the docker image
   3. Imgwebapp.py – contains the streamlit and tensorflow code that underlies the app for detecting beanrust
   4. Deployment.yml – definition for the deployment configuration, notice the docker image name as well as the number of replicas
   5. Service.yml – definition for the service configuration
   6. Models – contains the actual model saved in saved\_model.pb
2. Now it times to create the docker image. You can check current images via gcloud containers images list. Use the following command to create the image. This will create the docker file in the google container registry.

gcloud builds submit --tag gcr.io/<project id>/imgclassifier

1. Finally its time to deploy. Make sure that you are referring to the right Kubernetes cluster via gcloud container clusters get-credentials <Kubernetes cluster> --zone <zone of Kubernetes cluster>.
   1. Create the deployment via kubectl apply -f deployment.yaml. This will launch your container images in pods. You can check the status of your deployment via the following commands.
      1. kubectl get deployments
      2. kubectl get pods – number of pods should correspond to number of replicas and when they done Status will show Running
      3. kubectl describe pod <pod>
      4. kubectl logs -f <pod> - check why a pod failed to create
   2. Next, we need to connect the pods to the public internet for the app, so type in kubectl apply -f service.yaml.
2. You should now be able to see the service by navigating to the Kubernetes Engine -> Services & Ingress -> click on Endpoint

Other Kubectl commands –

1. Autoscale a min and max number of nodes when a pod hits a certain cpu percentage - kubectl autoscale deployment complaints-gke --max 6 --min 3 --cpu-percent 50
2. Continuous deployment by updating a container image for pods- kubectl rollout restart deployment <deployment name>
3. Add more replicas - kubectl scale deployment complaints --replicas 2
4. Check your autoscaling policy - kubectl get hpa