For this we need to push 2 images v1 and v2 of dockerfile to the container registry which we can use .
Create a simple flask application .

```
from flask import Flask

app = Flask(__name__)


app.route('/')
ef index():
    return 'Welcome to Python Flask World'


if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8080)
```

Add this code to main.py
Create dockerfile for packaging application :
Check docker is installed in cloud shell > docker
Create a new directory and add the main.py file to the directory .
Creating DOCKERFILE :
Update docker file :

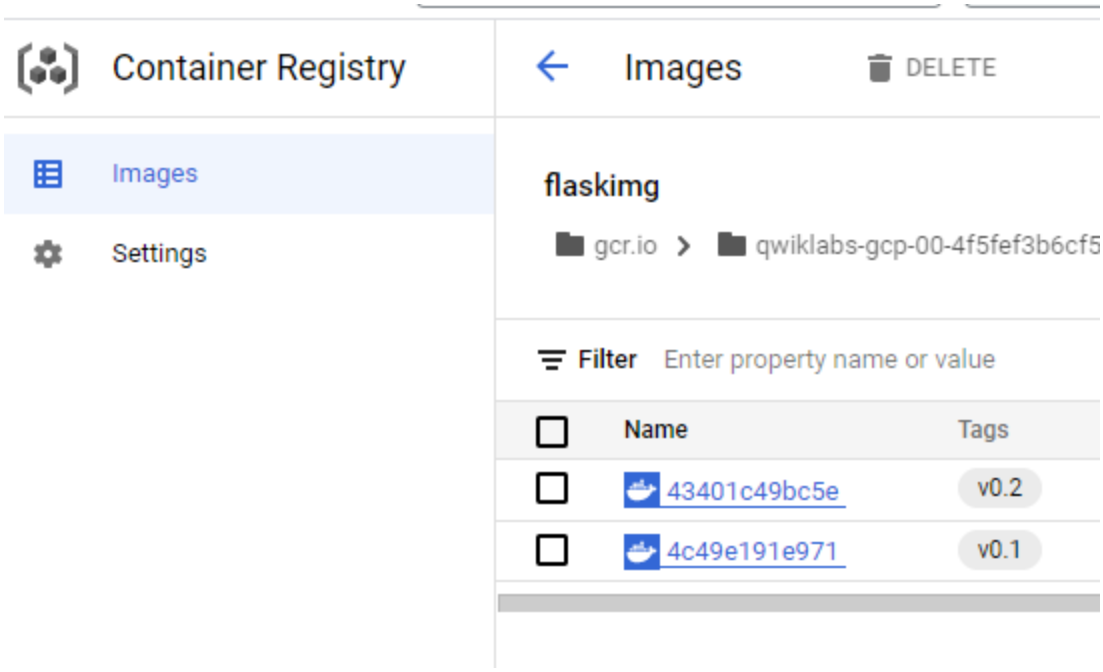
```
student_00_e0786700983b@cloudshell:~ (qwiklabs-gcp-04-1437425201d8)$ cat Dockerfile
FROM python:3.9.16-alpine3.16
RUN pip install flask
WORKDIR /myapp
COPY ./main.py /myapp/
CMD ["python","/myapp/main.py"]
student_00_e0786700983b@cloudshell:~ (qwiklabs-gcp-04-1437425201d8)$
```

Build docker image :
docker build -t gcr.io/project-id/image-name:tag-version . (location of dockerfile)
docker run -p 9090:8080 gcr.io/project-id/image-name:tag
Push docker image to container registry .
We will push the image to google cloud container registry .
docker push image-name:tag
In the container registry image will be added .

```
  1  ls
  2  mkdir deploy-apps
  3  ls
  4  cd deploy-apps/
  5  ls
  6  vi main.py
  7  python3
  8  python3 main.py
  9  vi Dockerfile
 10  docker build -t gcr.io/qwiklabs-gcp-00-4f5fef3b6cf5/flaskimg:v0.1
 11  docker build -t gcr.io/qwiklabs-gcp-00-4f5fef3b6cf5/flaskimg:v0.1 .
 12  vi Dockerfile
 13  docker build -t gcr.io/qwiklabs-gcp-00-4f5fef3b6cf5/flaskimg:v0.1 .
 14  docker images
 15  docker push gcr.io/qwiklabs-gcp-00-4f5fef3b6cf5/flaskimg:v0.1
 16  history
```

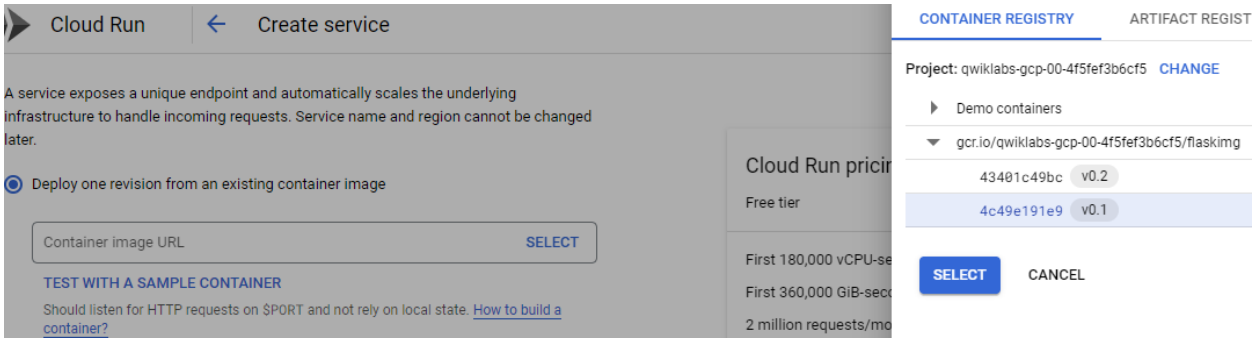
We can update the main.py file and create a new version of the image and push it to the container registry .

```
   16  history
student_00_af4e6b06b202@cloudshell:~/deploy-apps (qwiklabs-gcp-00-4f5fef3b6cf5)$ vi main.py
student_00_af4e6b06b202@cloudshell:~/deploy-apps (qwiklabs-gcp-00-4f5fef3b6cf5)$ docker build -t gcr.io/qwiklabs-gcp-00-4f5fef3b6cf5/flaskimg:v0.2 .
Sending build context to Docker daemon  3.072kB
Step 1/5 : FROM python
 ---> afe5735f16e1
Step 2/5 : RUN pip install flask
 ---> Using cache
 ---> 08aff9fcdd06
Step 3/5 : WORKDIR /myapp
 ---> Using cache
 ---> 1816c0ad52b8
Step 4/5 : COPY ./main.py /myapp/
 ---> 7701be664059
Step 5/5 : CMD ["python","/myapp/main.py"]
 ---> Running in a4ee63697442
Removing intermediate container a4ee63697442
 ---> 47d4b675ca87
Successfully built 47d4b675ca87
Successfully tagged gcr.io/qwiklabs-gcp-00-4f5fef3b6cf5/flaskimg:v0.2
student_00_af4e6b06b202@cloudshell:~/deploy-apps (qwiklabs-gcp-00-4f5fef3b6cf5)$ docker images
REPOSITORY                                          TAG       IMAGE ID       CREATED          SIZE
gcr.io/qwiklabs-gcp-00-4f5fef3b6cf5/flaskimg   v0.2      47d4b675ca87   6 seconds ago    951MB
gcr.io/qwiklabs-gcp-00-4f5fef3b6cf5/flaskimg   v0.1      17d64a85437b   2 minutes ago    951MB
python                                              latest    afe5735f16e1   6 days ago       932MB
student_00_af4e6b06b202@cloudshell:~/deploy-apps (qwiklabs-gcp-00-4f5fef3b6cf5)$ docker push gcr.io/qwiklabs-gcp-00-4f5fef3b6cf5/flaskimg:v0.2
The push refers to repository [gcr.io/qwiklabs-gcp-00-4f5fef3b6cf5/flaskimg]
1f78be18373f: Pushing [==================================================>]   2.56kB
5e9e8d6d9706: Layer already exists
bceda5383f1e: Layer already exists
2127b463e72a: Layer already exists
51dde6a94e51: Layer already exists
eaf866df682c: Layer already exists
```

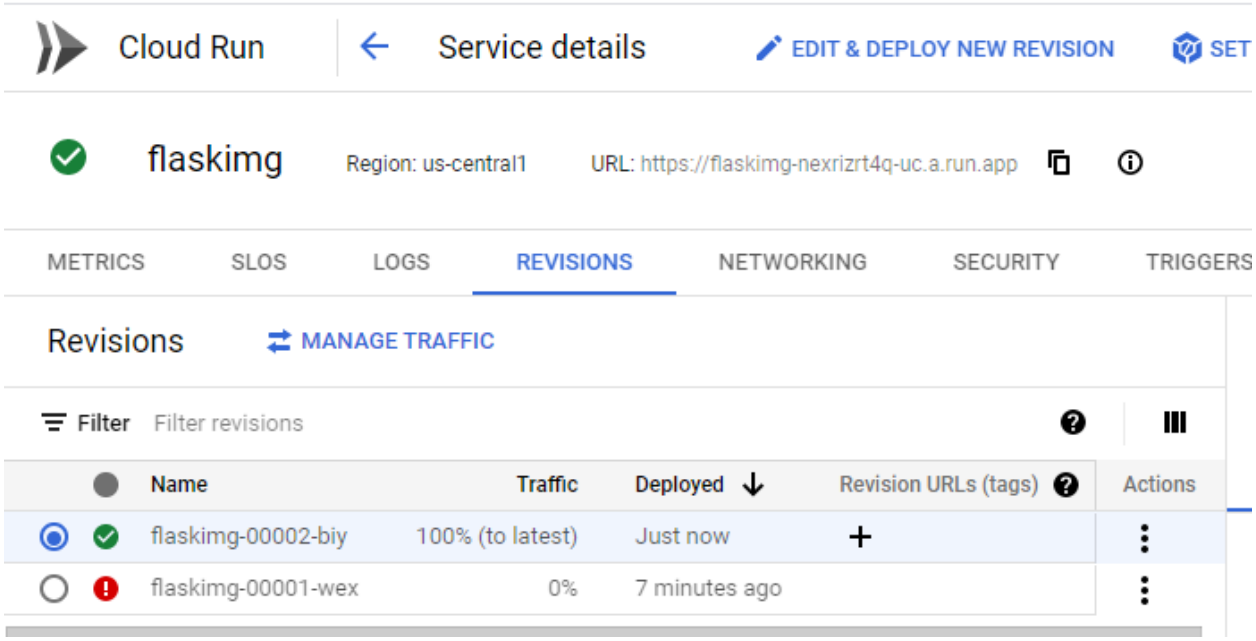


DEPLOY to cloud run :

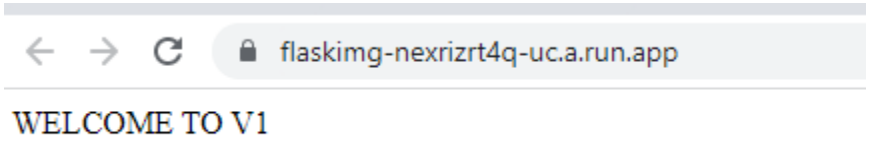
it's completely serverless, just like an app engine. So you can scale it from zero to infinity, you can say theoretically for your compute resource. It's completely serverless. So you do not need to manage anything. And here you can deploy your containerized application. So just like an app engine standard environment, which we deploy, where you need to worry about specific run time, which is being predefined by Google.
Apart from that, you just cannot go ahead. Here, it's a containerized application.
So you build your own docker image with your custom run time and anything you can deploy it.
In the container registry images will be stored of the app engine and cloud function also by default .
CLoud run > we have 2 options
1) deploy 1 revision from the existing container image .
2) Continuously deploy new revisions from the source repository . - we have to set up a cloud build .
> deploy 1 revision from the existing container image . > select image for v1 .



> region - us central1> cpu during request > Autoscale - 0 to 10 > Ingress - allow all traffic > authentication - allow unauthenticated invocations > 256 mb memory > Timeout - 300 > Max request per container - 80 > execution env - default . > create .

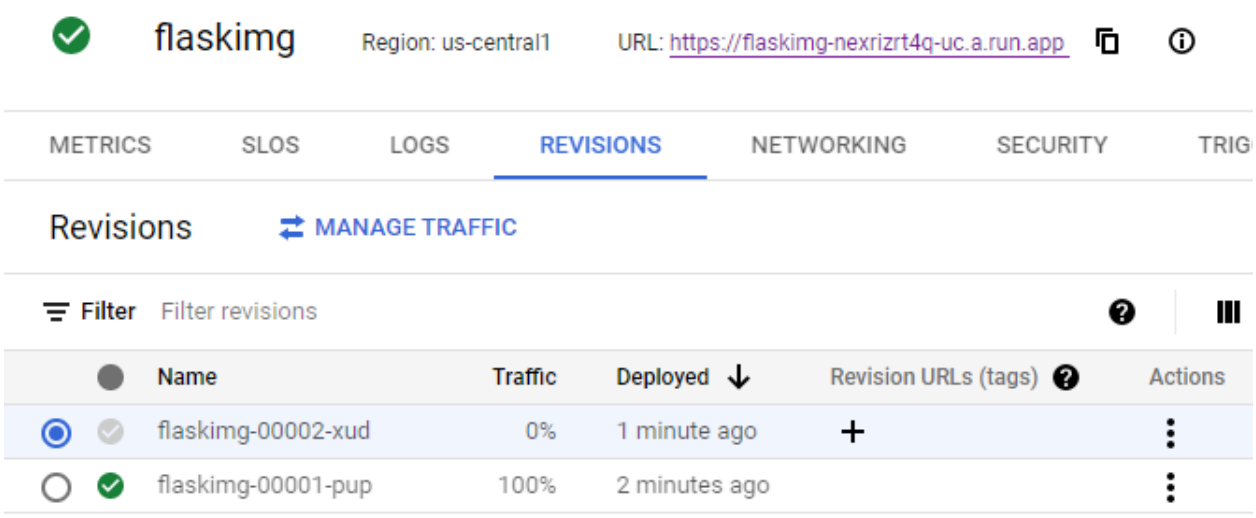


Cloud run is faster from the app engine.
Now 1 version of docker is deployed . Try accessing it .

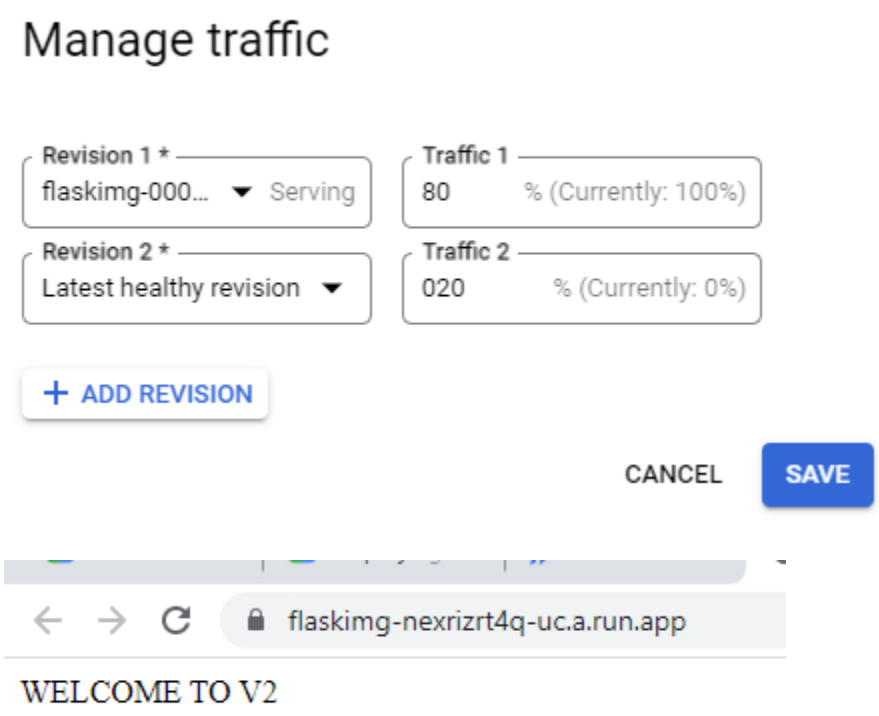


WELCOME TO V1

Lets create a new version of docker file and push it to the container registry .
Once the image is pushed .
Cloud run > edit > container image url - select v2 > Uncheck - server this revision immediately (it will slit 100% traffic to new version)> Deploy .

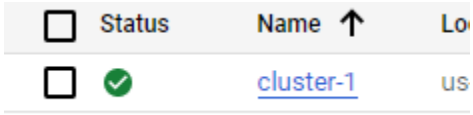


Once deployed - > manage traffic > 20% for v2 > new version will reflect in url > Once tested we will migrate all traffic to v2 .
This is how we can deploy docker images to cloud run .

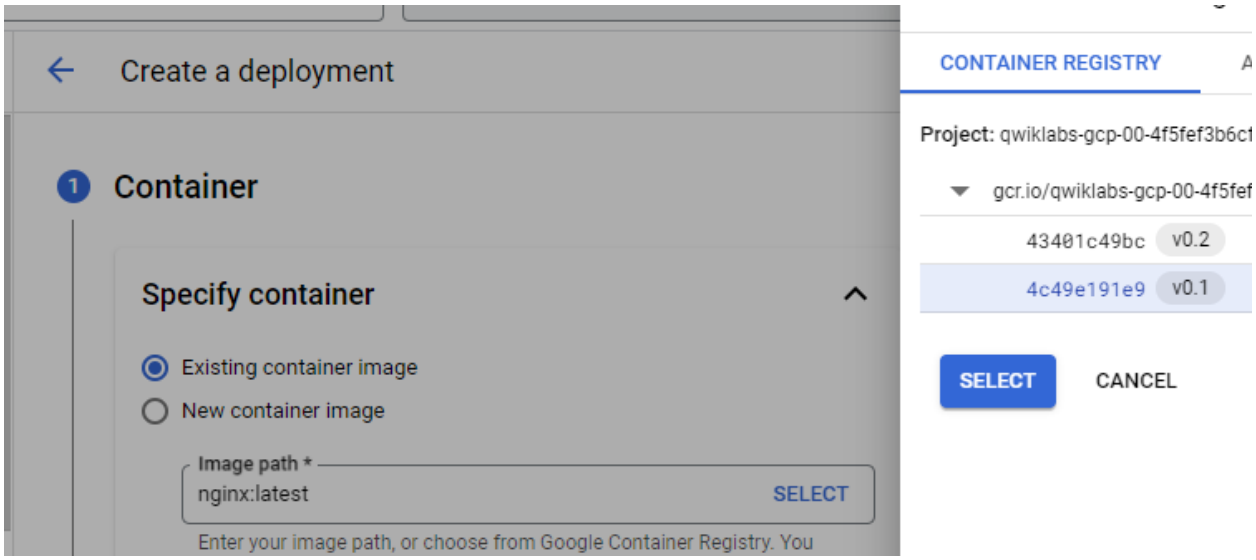


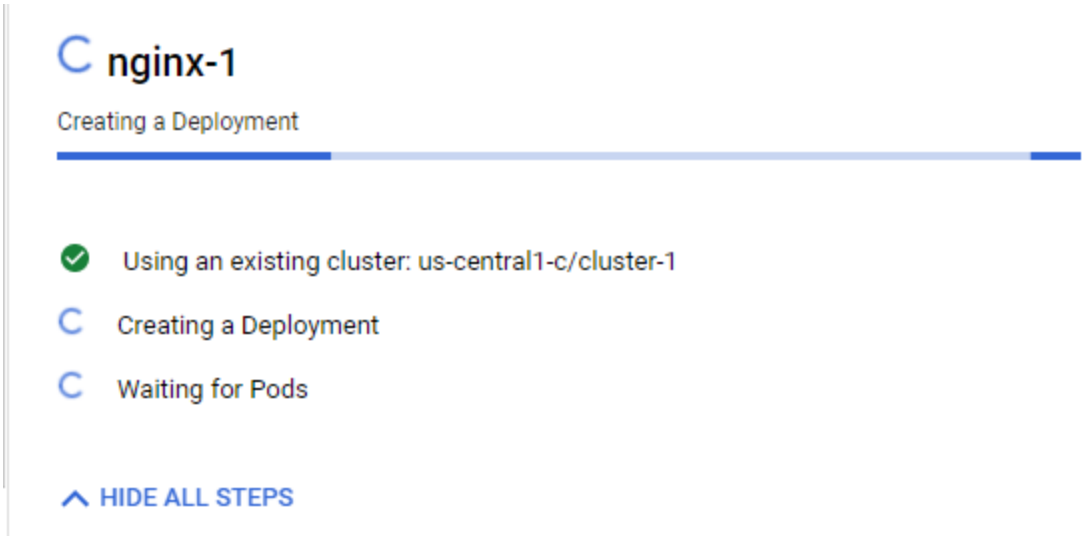
DEPLOY to GKE :
Kubernetes is kind of a container orchestration engine and Google has provided a complete online version inside their Google cloud with GKE service, Google Kubernetes Engine and that's where we are gonna deploy our application now. So this is kind of an

open source version of Kubernetes inside the Google Cloud platform or Google Kubernetes Engine. So what we are gonna do is, the same image which we have deployed to Google Cloud, run version one and version two, subsequently we'll deploy to Kubernetes engine. So let's just first create a cluster. So first step is cluster creation, second step here, workload deployment. So, there are three steps and the last step is to expose your service. So all three steps, lets just see how to perform via Cloud Shell.
KUBERNETES ENGINE > CLUSTER> CREATE > AUTOPILOT / STANDARD - STANDARD > ZONAL / REGIONAL - ZONAL - us central1 c > CONTROL PLANE VERSION - default > NODE POOl - > number of node - 2 > machine size - 2 cpu 4 gb ram > persistent disk - size 20 gb > Security - allow default access > Metadata - Create

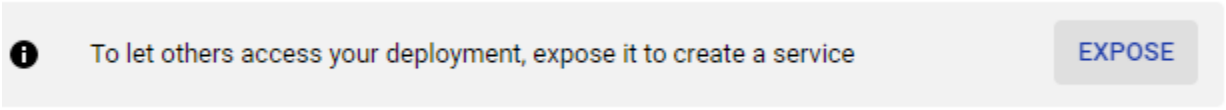


CLuster creation will take time .
Deploy workload - CLuster > deploy - existing container image / new container image - existing image - select version of image . > CLUSTR - select cluster created > Deploy .

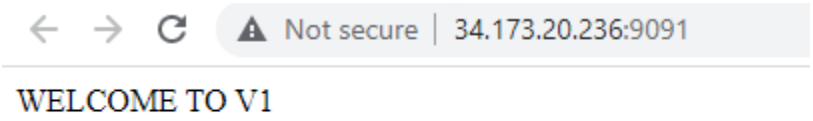




Pod is the smallest unit inside the kubernetes cluster . Once deployed , we can't use applications unless services are exposed in kubernetes .



Expose (from outside world any1 can access server)> Port mapping (from which port we want to do mapping)> from outside world we will access from 9091 and target port will be 8080 as our container app is running on 8080> service type (There are 3 types : lb , node port , cluster ip) - lb - with external ip we can access our app > Services name > expose .



WELCOME TO V1

Once exposed we can access from lb ip .

Load Balancer

Cluster IP	10.8.7.102
Load balancer IP	34.173.20.236
Load balancer	ad86ea754c4da43e7924d8d4bff06f58

Deployments

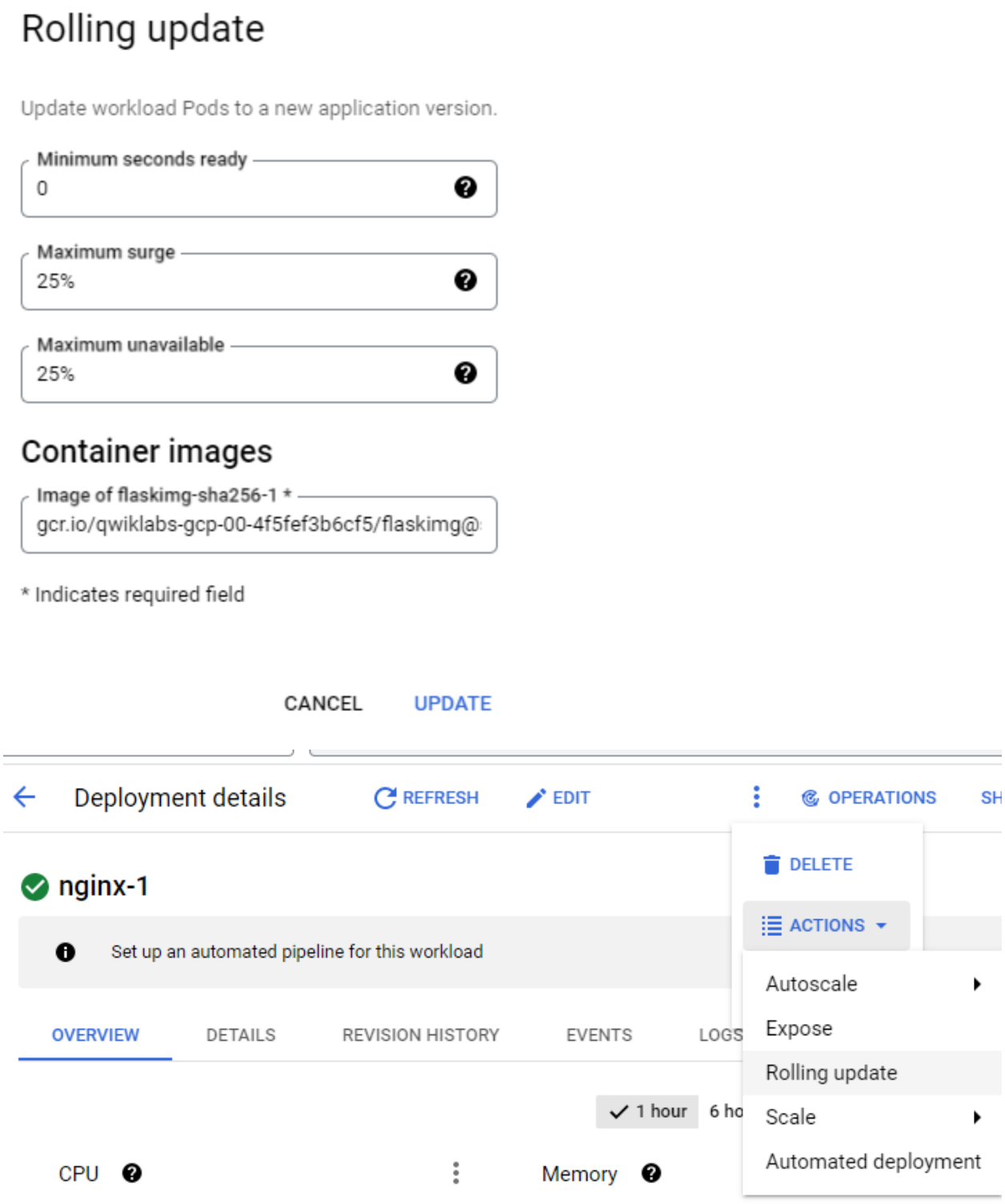
Name	Status	Pods
nginx-1	OK	3/3

Serving pods

Sort

Name	Status	Endpoints
nginx-1-768cbfd695-79fd7	Running	10.4.1.8
nginx-1-768cbfd695-5hgbw	Running	10.4.0.7
nginx-1-768cbfd695-sd9ms	Running	10.4.0.6

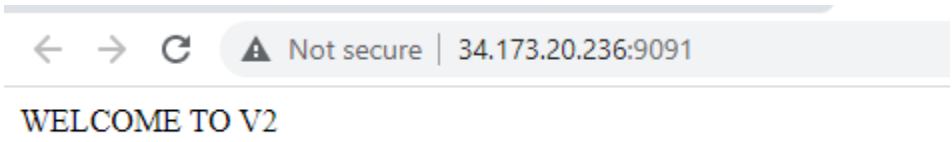
We can see 3 pods are present .
For each task in kubernetes yaml file is present which we can use to automate tasks .
LB will also be created to check external ip it will work .
Now deploying v2 from workload > V1 workload > actions > rolling update > min second ready - 0 (min tym in which new pod will be ready) > Max surge - 25% > 25% > images - select v2 url .> update



So, here we are just dealing with the three ports. But when you're dealing with hundreds, even thousands, of ports, what it says that maximum surge means. The maximum number of ports that can be created over the desired number of ports. So let's give a

nice example. The value can be an absolute number, five, or a percentage of desire. So you can provide, let's say, 25 percent. So maximum 25 percentage of new ports will be created while destroying new. Eventually, it is going to destroy your earlier one, and keep creating new ones. But that is what the strategy being allowed, or I would say, possible in case of Kubernetes deployment, or rolling update. That means, one resource at a time will be updated. And slowly, slowly all those resources will be updated by version two.
Once done, new pods with v2 will be started getting created and old pods will be destroyed and new pods will get created . We will get v2 in lb ip .

Revision	Name	Status
1	nginx-1-768cbfd695-79fd7	Running
1	nginx-1-768cbfd695-5hgbw	Terminating
1	nginx-1-768cbfd695-sd9ms	Running
2	nginx-1-54c9d888c8-pcwfw	Running
2	nginx-1-54c9d888c8-vjj5w	ContainerCreating



V2 deployed using a rolling update in the kubernetes engine .
We can see the yaml file also > yaml .