CMS Workshop 2022
Cloud Computing

Setup
Before you begin

1. **Prep-work: Kubernetes Clusters (2021)**

2. **Demo: Creating a cluster (2021)**

Select Cluster in Kubernetes Engine

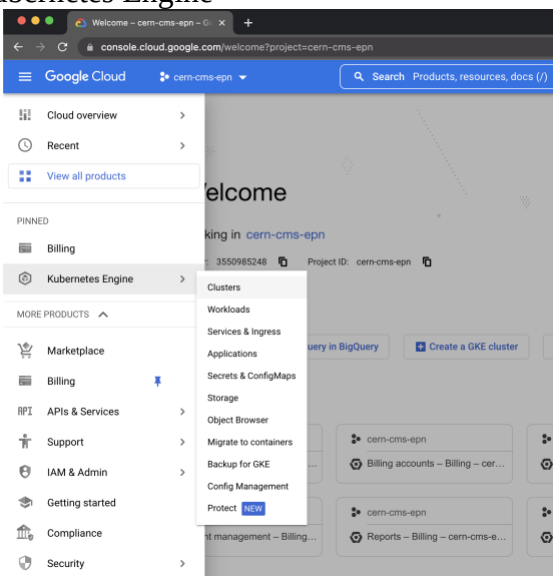


Fig. 1 Select cluster

This interface will appear and then we select CREATE

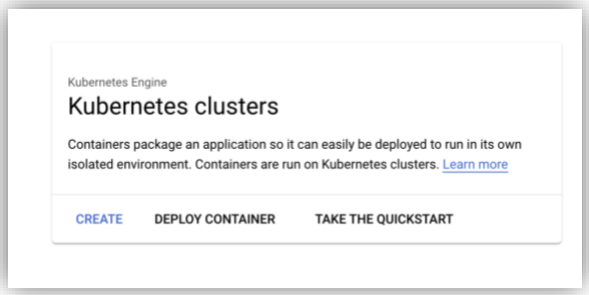


Fig 2. Select Create

Some cluster models are displayed, in our case we will select: GKE STANDARD

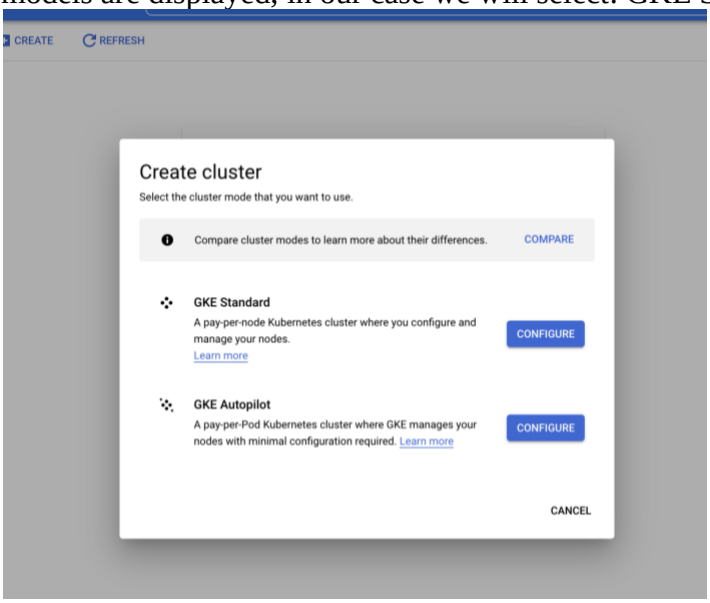


Fig. 3 Select GKE Standard

Give it a name

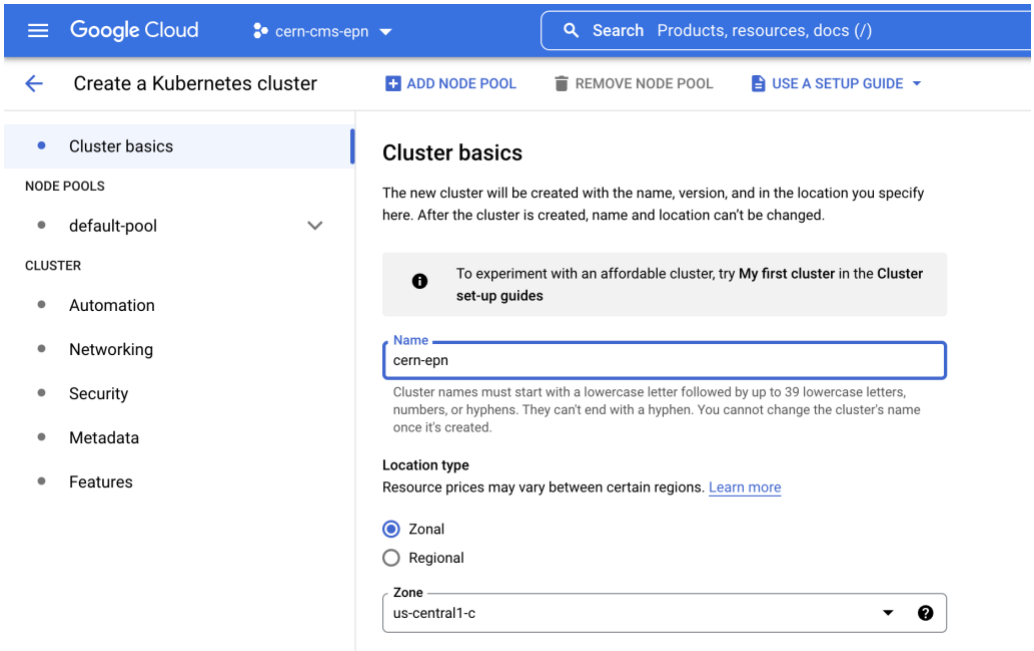


Fig. 4 Basic Cluster Name

- Many ways to configure the cluster, but let's try an efficient one with autoscaling
- Go to default pool
- Choose size: 1 node
- Autoscaling 0 to 4

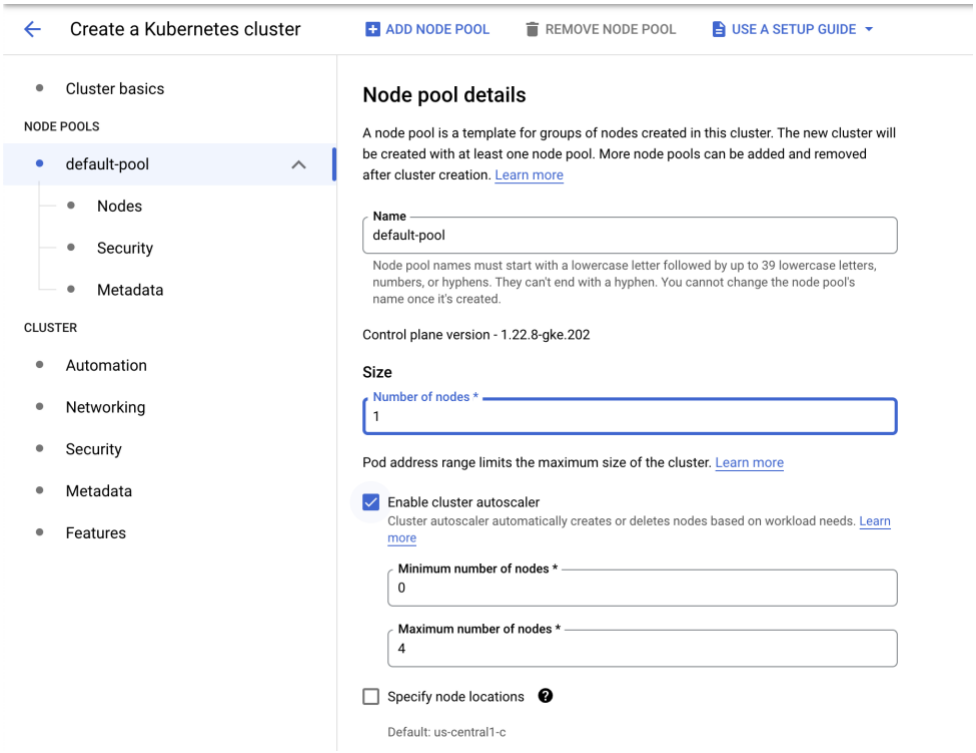


Fig. 5 Cluster details

- Go to Nodes
- Choose a machine e2-standar-4
- Leave the rest as it is
- Hit create

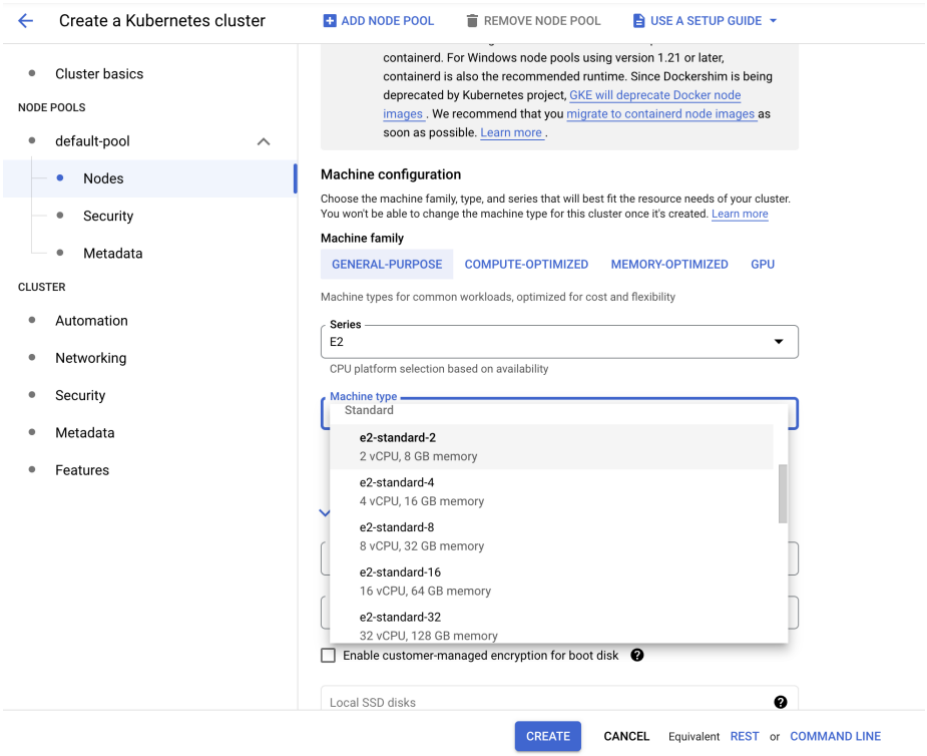


Fig. 6 Machine Configuration

Wait till it’s ready (Around 5 min)

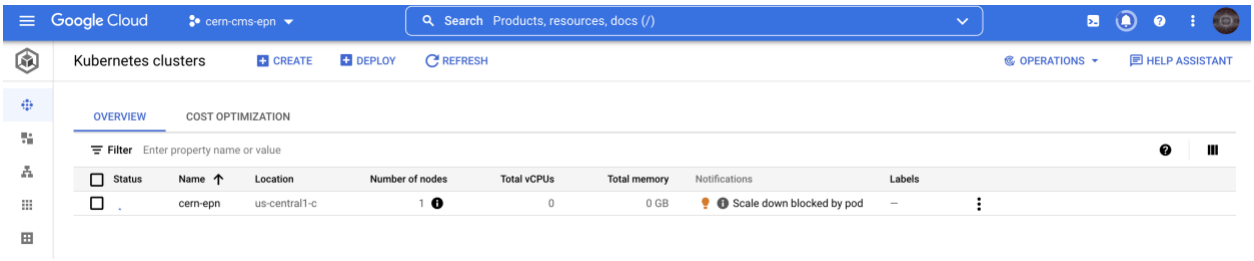


Fig. 7 Creating

Select the connect action

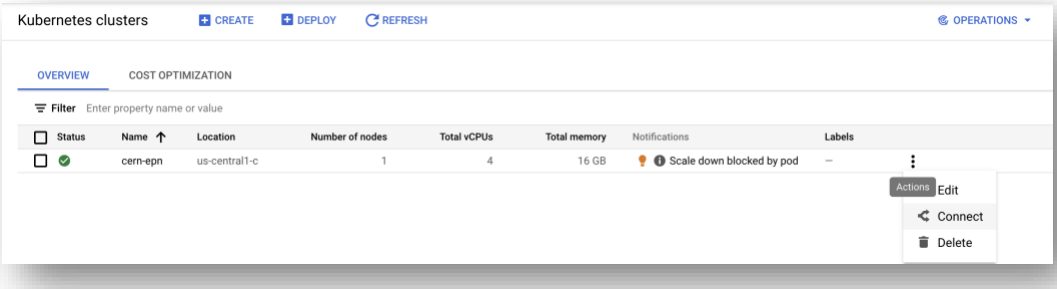


Fig. 8 Connection

Run cloud in shell

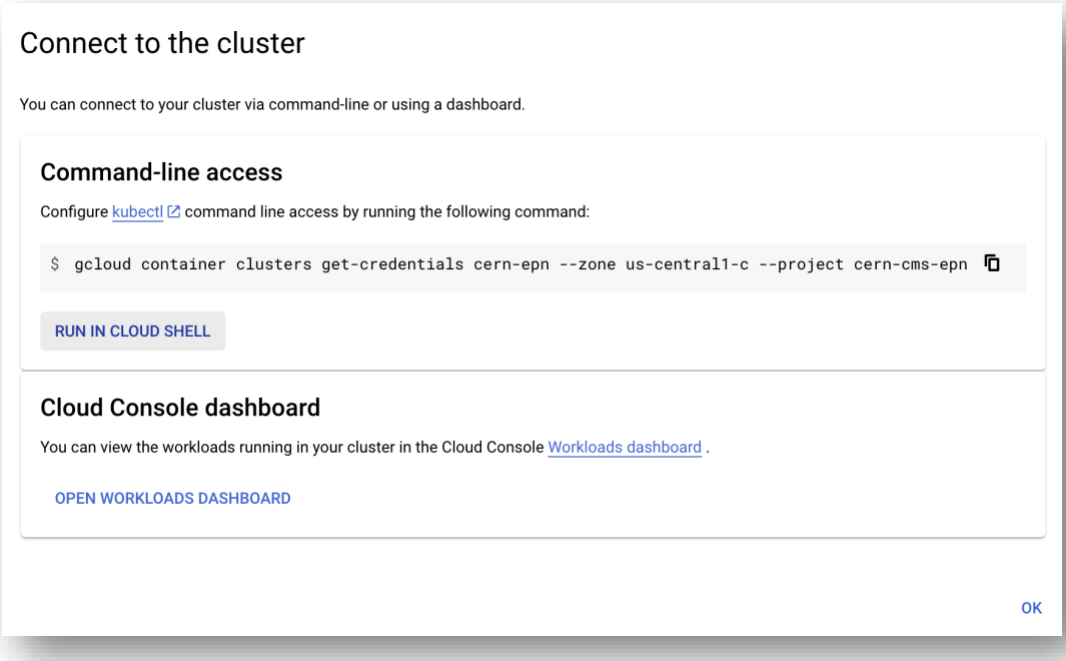


Fig. 9 Command-line

You've connected to your shell, now press enter to link to your GKE cluster

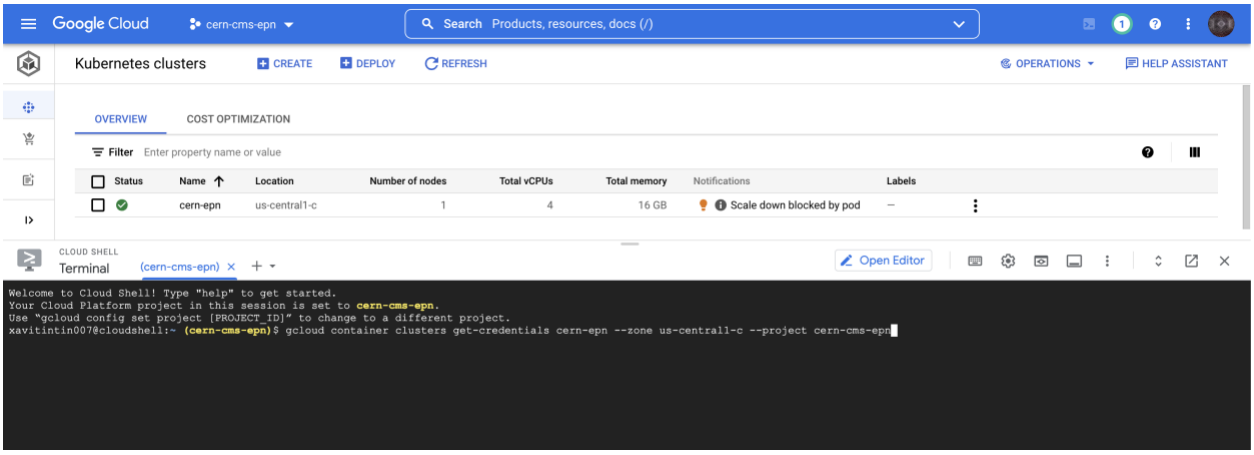


Fig. 10 Load the command

Authorize Shell

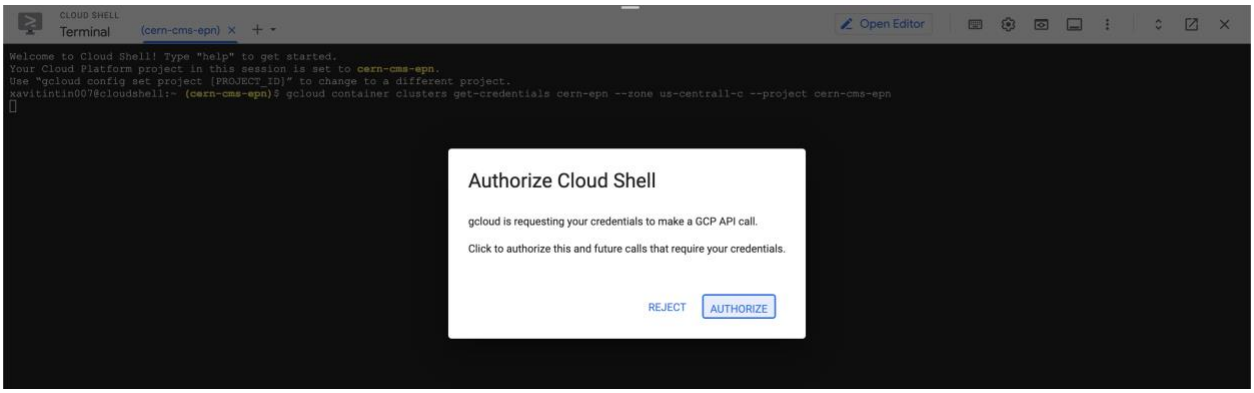


Fig. 11 Some Credentials

3. Kubectl and additional tools and services (2021)

Argo

While jobs can also be run manually, a workflow engine makes defining and submitting jobs easier. In this tutorial, we use [argo quick start] page to install it:

```
kubectl create ns argo
kubectl apply -n argo -f https://raw.githubusercontent.com/argoproj/argo-workflows/master/manifests/quick-start-postgres.yaml
curl -sLO https://github.com/argoproj/argo/releases/download/v2.11.1/argo-linux-amd64.gz
gunzip argo-linux-amd64.gz
chmod +x argo-linux-amd64
sudo mv ./argo-linux-amd64 /usr/local/bin/argo
```

```
kubectl create clusterrolebinding cern-cms-cluster-admin-binding --
clusterrole=cluster-admin --user=cms-gXXX@arkivum.com

argo version
```

Fig. 12 Check Version

Patch a yaml file with the following command:

```
nano patch-workflow-controller-configmap.yaml
```

Paste this inside:

```
data:
  artifactRepository: |
    archiveLogs: false
```

Fig. 13 Inside NANO

To exit press ^X Y Enter

Patch with the following command:

```
kubectl patch configmap workflow-controller-configmap -n argo --patch "$(cat patch-workflow-controller-configmap.yaml)"
```

Run a simple test workflow
To test the setup, run a simple test workflow with

```
argo submit -n argo --watch https://raw.githubusercontent.com/argoproj/argo/master/examples/hello-world.yaml
argo list -n argo
argo get -n argo @latest
argo logs -n argo @latest
```

Wait till the yellow light turns green

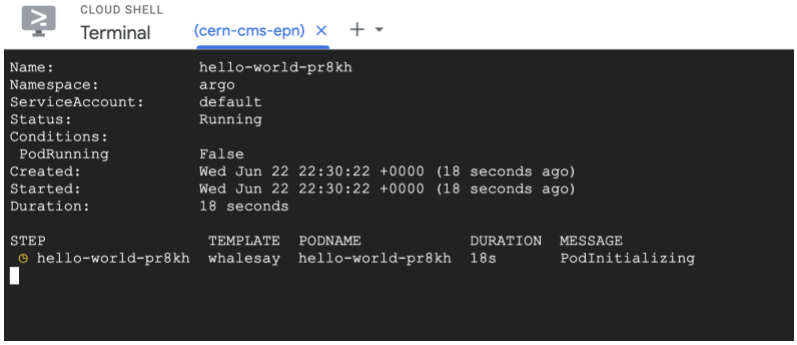


Fig. 14 Starting the service

If argo was installed correctly you will have the following:

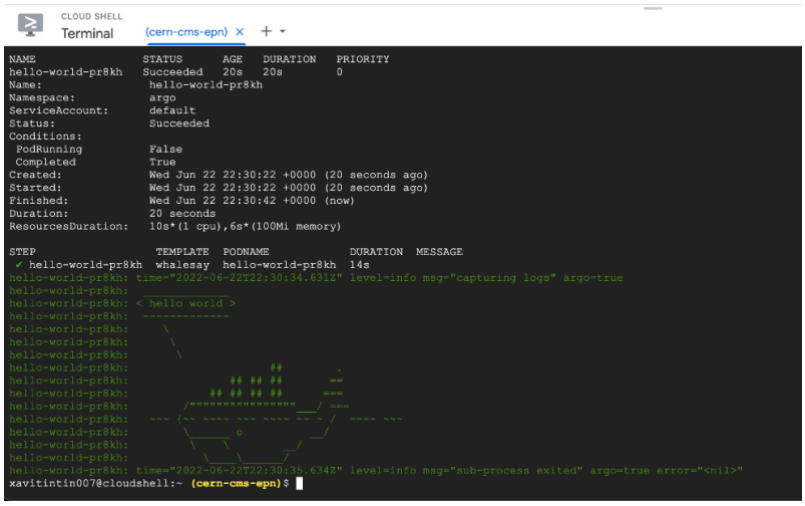


Fig. 15 Hello World

To delete the latest argo workflow use:

```
argo delete -n argo @latest
```

4. Storing workflow output on Google Kubernetes Engine (2020)

We will create the volume disk, in my example my <NUMBER>=1

```
gcloud compute disks create --size=100GB --zone=us-central1-c gce-nfs-disk-
<NUMBER>
```

Set up an nfs server for this disk:

NOTE: If you did use a different <NUMBER> the you have to change this number in every yaml configuration file, you can do this with the following command:
vim (file you want to change), for the first file it would look like this: *vim 001-nfs-server.yaml*
Press the letter "*i"* in your keyboard to edit text, and to exit press ^C and the text: *":wq"*.
Remember to run the kubectl apply command afterwards.

```
wget https://cms-opendata-workshop.github.io/workshop2021-lesson-
cloud/files/001-nfs-server.yaml
kubectl apply -n argo -f 001-nfs-server.yaml
```

Set up a nfs service, so we can access the server:

```
wget https://cms-opendata-workshop.github.io/workshop2021-lesson-
cloud/files/002-nfs-server-service.yaml
kubectl apply -n argo -f 002-nfs-server-service.yaml
```

Let's find out the IP of the nfs server:

```
kubectl get -n argo svc nfs-server |grep ClusterIP | awk '{ print $3; }'
```

Save this ip address or open a new terminal window

We need to write that IP number above into the appropriate place in this file:

```
wget https://cms-opendata-workshop.github.io/workshop2021-lesson-
cloud/files/003-pv.yaml
vim 003-pv.yaml
```

To edit you have to press the letter “I” and to save and quit press ^C and enter the string: “:wq”

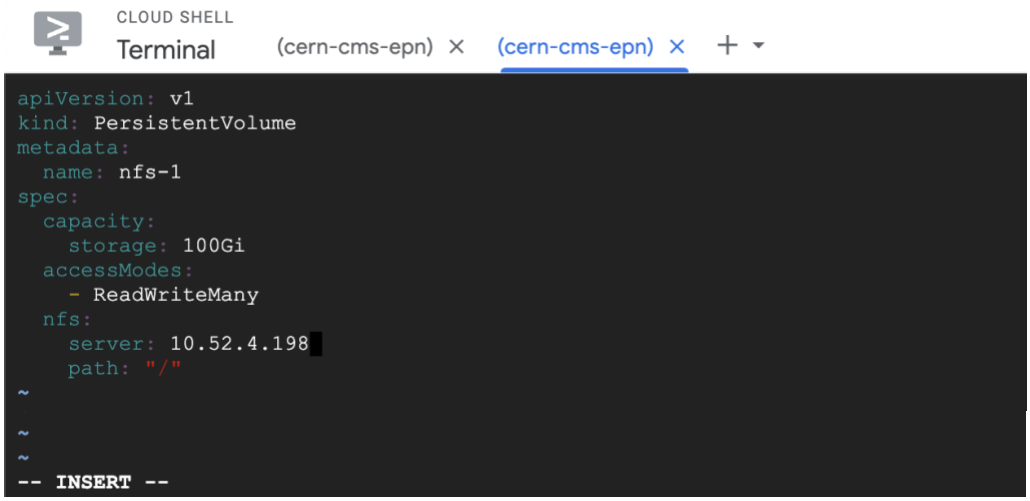


Fig. 16 Load our IP

Deploy

```
kubectl apply -f 003-pv.yaml
```

Check:

```
kubectl get pv
```

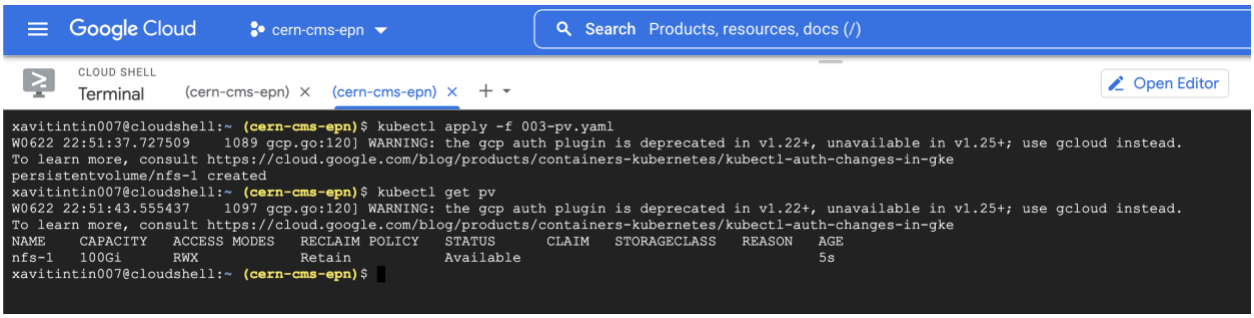


Fig. 17 Commands

Apps can claim persistent volumes through *persistent volume claims* (pvc). Let’s create a pvc:

```
wget https://cms-opendata-workshop.github.io/workshop2021-lesson-cloud/files/003-pvc.yaml
kubectl apply -n argo -f 003-pvc.yaml
```

Check:

```
kubectl get pvc -n argo
```

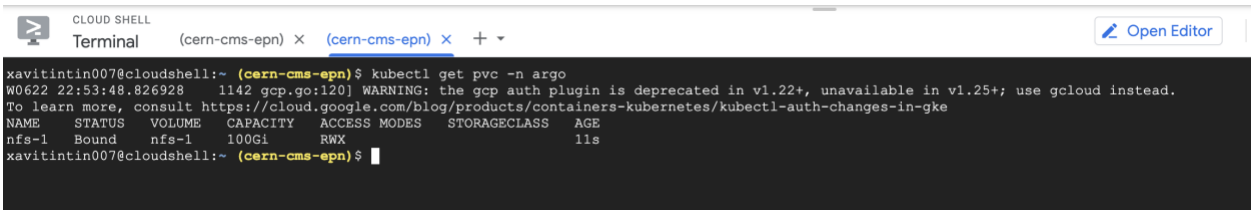


Fig. 18 Get Argo

Now an argo workflow coul claim and access this volume with a configuration like:

```
nano argo-wf-volume.yaml
```

Paste the following and change <NUMBER>, exit with ^X, Y, Enter

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: test-hostpath-
spec:
  entrypoint: test-hostpath
  volumes:
    - name: task-pv-storage
      persistentVolumeClaim:
        claimName: nfs-<NUMBER>
  templates:
  - name: test-hostpath
    script:
      image: alpine:latest
      command: [sh]
      source: |
        echo "This is the ouput" > /mnt/vol/test.txt
        echo ls -l /mnt/vol: `ls -l /mnt/vol`
      volumeMounts:
      - name: task-pv-storage
        mountPath: /mnt/vol
```

Submit and check this workflow with

```
argo submit -n argo argo-wf-volume.yaml
argo list -n argo
```

Take the name of the workflow from the output (replace XXXXX in the following command) and check the logs:

```
kubectl logs pod/test-hostpath-XXXXX  -n argo main
```

Once the job is done, you will see something like:

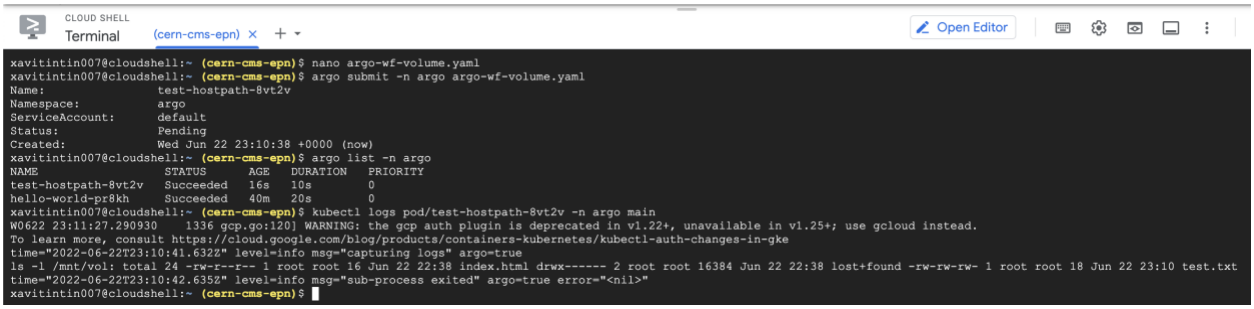


Fig. 19 Logs and status review

Run a CMS open data workflow

If the steps above are successful, we are now ready to run a workflow to process CMS open data.

Create a workflow file argo-wf-cms.yaml with the following content:

nano argo-wf-cms.yaml

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: nanoaod-argo-
spec:
  entrypoint: nanoaod-argo
  volumes:
  - name: task-pv-storage
    persistentVolumeClaim:
      claimName: nfs-<NUMBER>
  templates:
  - name: nanoaod-argo
    script:
      image: cmsopendata/cmssw_5_3_32
      command: [sh]
      source: |
        cat /etc/redhat-release
        git clone https://github.com/cms-opendata-analyses/AOD2NanoAODOutreachTool  AOD2NanoAOD
        source /opt/cms/entrypoint.sh
        sudo chown $USER /mnt/vol
        mkdir workspace
        cd workspace
        mv /home/cmsusr/AOD2NanoAOD home/cmsusr/CMSSW_5_3_32/src/workspace
        cd AOD2NanoAOD
        scram b -j8
        nevents=100
        eventline=$(grep maxEvents configs/data_cfg.py)
        sed -i "s/$eventline/process.maxEvents = cms.untracked.PSet( input = cms.untracked.int32($nevents) )/g"
configs/data_cfg.py
        cmsRun configs/data_cfg.py
        cp output.root /mnt/vol/
      volumeMounts:
      - name: task-pv-storage
```

mountPath: /mnt/vol

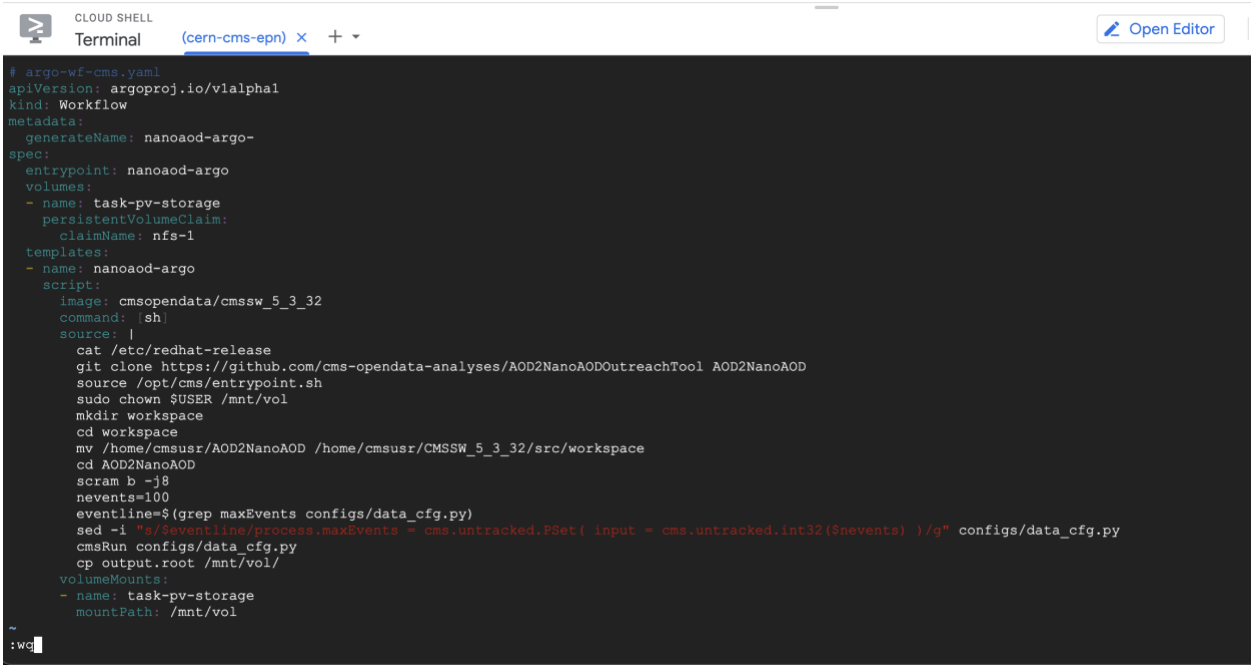


Fig. 20 Workflow file

Exit. Submit the job with:

argo submit ***-n*** *argo argo-wf-cms.yaml* ***--watch***

Wait till the yellow light turns green so it is available to launch

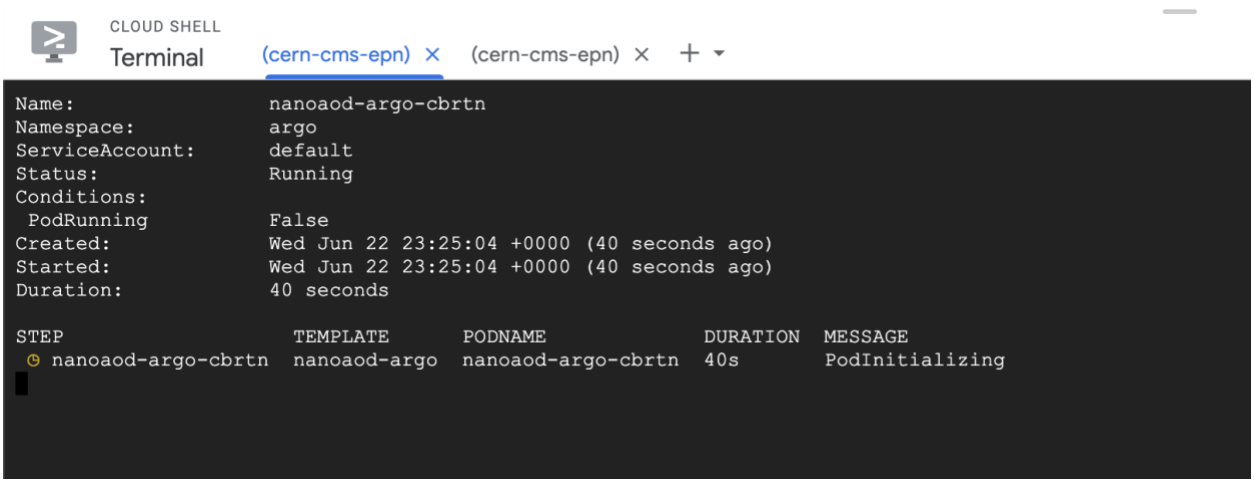


Fig. 21 Connecting

The option --watch gives a continuous follow-up of the progress. To get the logs of the job, use the process name (nanoaod-argo-XXXXX) which you can also find with

argo get ***-n*** *argo @latest*

and follow the container logs with

kubectl logs pod/nanoaod-argo-XXXXX ***-n*** *argo main*

You will have to get the following:

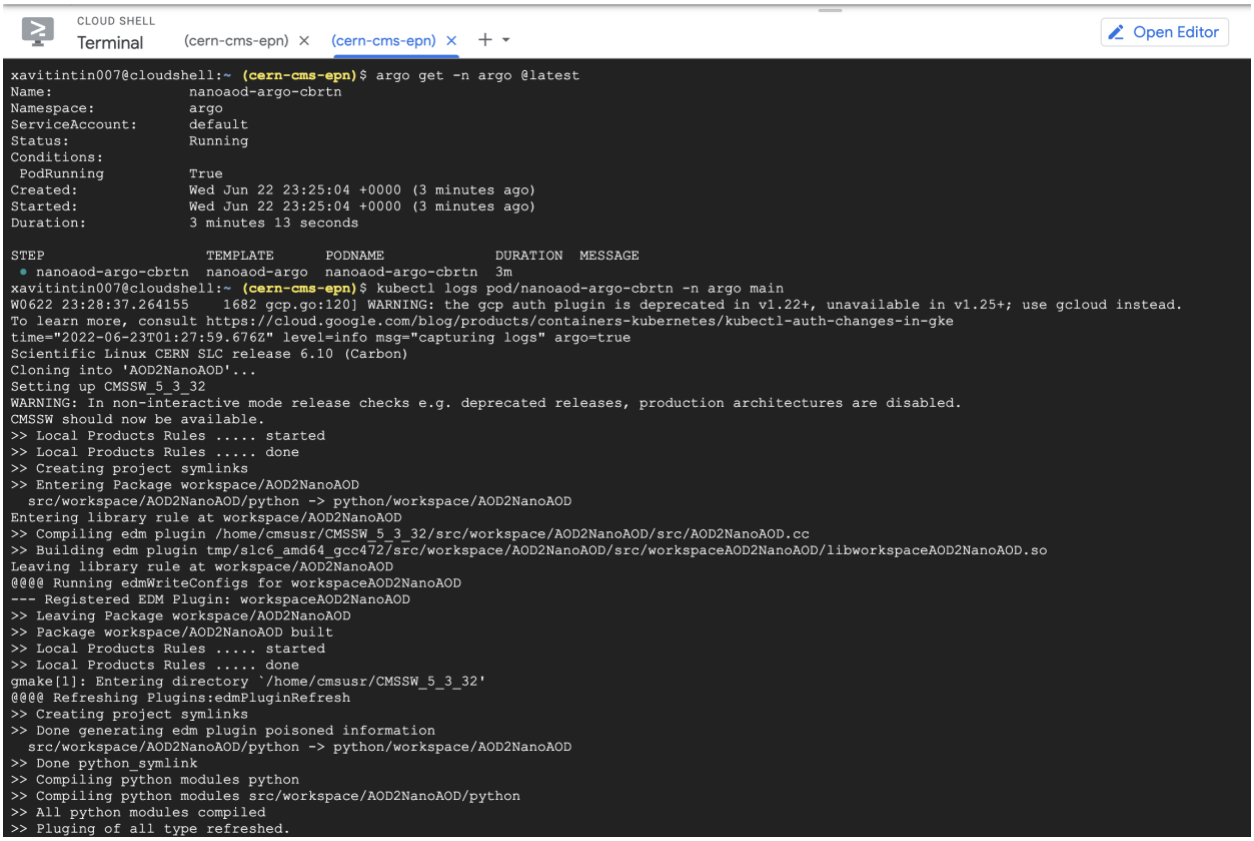


Fig. 21 Connecting

Note: There is a problem connecting with the CERN servers, it will take around 5 minutes for it to finally run.

To see the container logs re-enter the latest command:

kubectl logs pod/nanoaod-argo-XXXXX ***-n*** *argo main*

Finally, success!

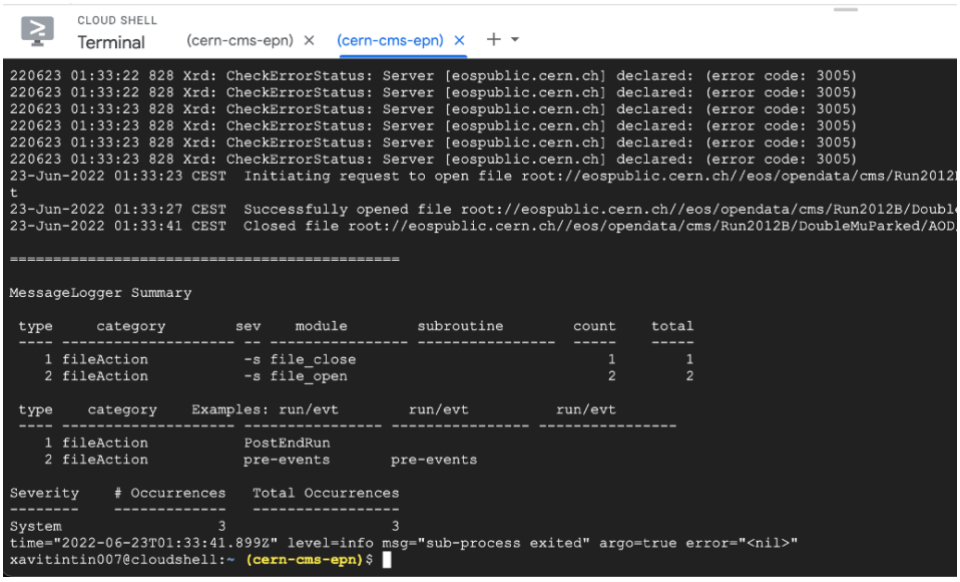


Fig. 22 Output File

Get the output file

Create a file pv-pod.yaml with the following contents:

```
apiVersion: v1
kind: Pod
metadata:
  name: pv-pod
spec:
  volumes:
    - name: task-pv-storage
      persistentVolumeClaim:
        claimName: nfs-<NUMBER>
  containers:
    - name: pv-container
      image: busybox
      command: ["tail", "-f", "/dev/null"]
      volumeMounts:
        - mountPath: /mnt/data
          name: task-pv-storage
```

```
kubectl apply -f pv-pod.yaml -n argo
kubectl cp  pv-pod:/mnt/data /tmp/poddata -n argo
```

and you will get the file created by the job in /tmp/poddata/test.txt.

ARGO

Check the services running and the associated IP addresses:

kubectl get svc -n argo

kubectl -n argo port-forward deployment/argo-server 2746:2746

It will start fowarding port, to not inturrupt this open a new window, after a couple minutes it will be handling connection.

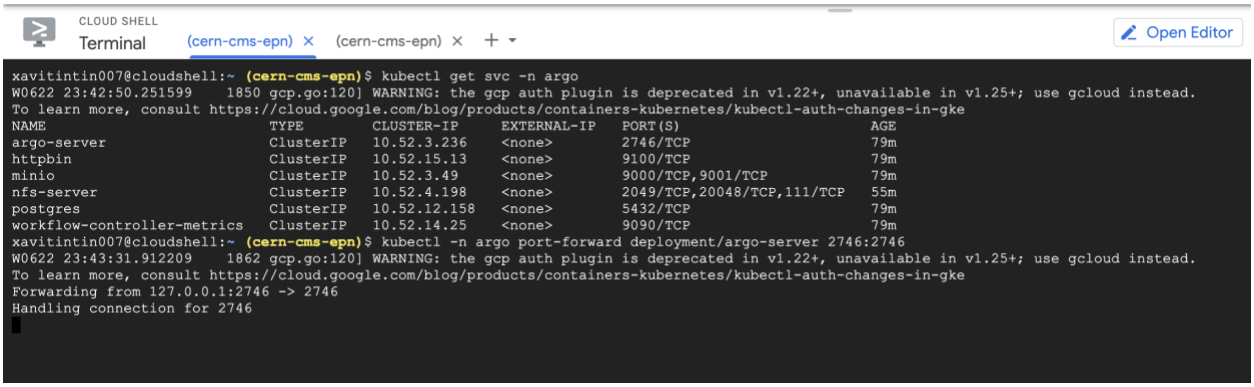


Fig. 23 Connection Stablished

If you want to see the process in depth, in new terminal window run: *lynx* [*https://localhost:2746*](https://localhost:2746)

kubectl patch svc argo-server -n argo -p '{"spec": {"type": "LoadBalancer"}}'

kubectl get svc -n argo

Creating an external ip, wait a couple minutes.

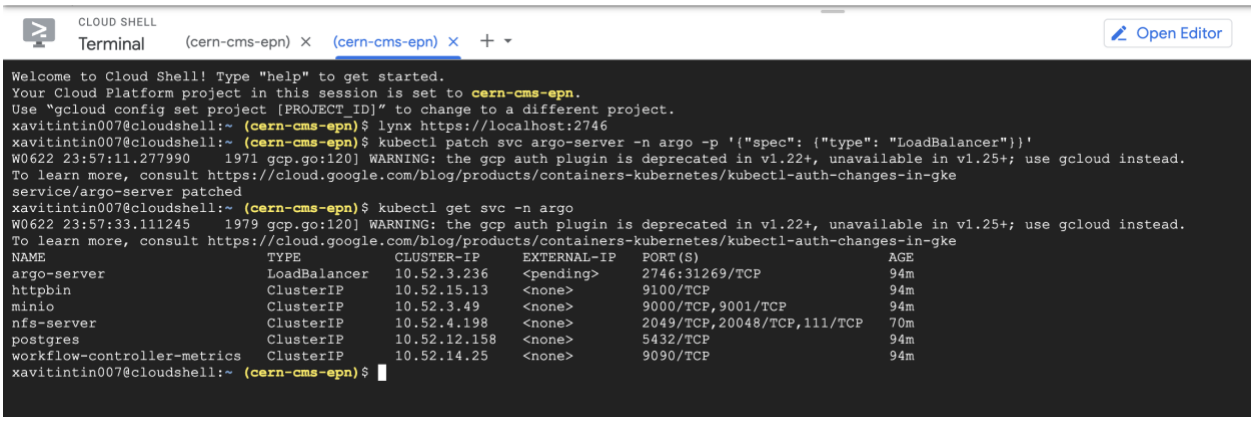


Fig. 24 Patch argo

Finally, you can access this address in your localhost. Do not forget to add “https://” and “:2746”

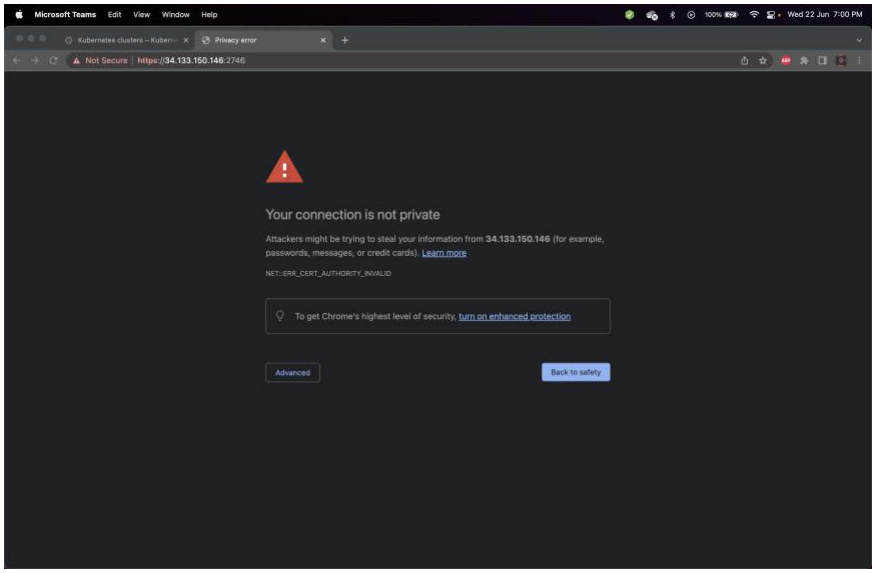


Fig. 25 Enter Argo

Click on Advanced, proceed to <ip>(unsafe) and voilà

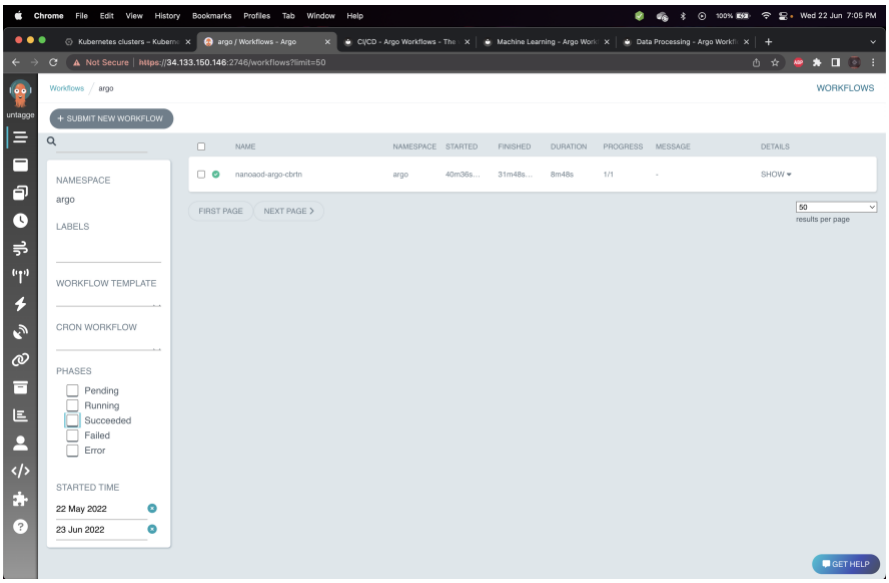


Fig. 26 Argo Interface

Accessing files via http

We first patch the config of the webserver to be created as follows:

```
mkdir conf.d
cd conf.d
curl -sLO https://raw.githubusercontent.com/cms-opendata-workshop/workshop-payload-kubernetes/master/conf.d/nginx-basic.conf
cd ..
kubectl create configmap basic-config --from-file=conf.d -n argo
```

Then prepare to deploy the fileserver by downloading the manifest:

```
curl -sLO https://github.com/cms-opendata-workshop/workshop-payload-kubernetes/raw/master/deployment-http-fileserver.yaml
```

Open this file and again adjust the <NUMBER> and the apiVersion to apps/v1 as follows:

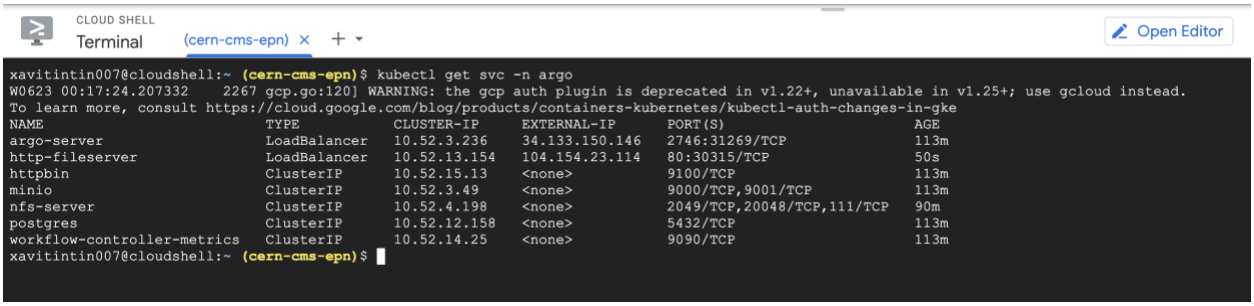
```
vim deployment-http-fileserver.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    service: http-fileserver
  name: http-fileserver
spec:
  replicas: 1
  strategy: {}
  selector:
    matchLabels:
      service: http-fileserver
  template:
    metadata:
      labels:
        service: http-fileserver
    spec:
      volumes:
      - name: volume-output
        persistentVolumeClaim:
          claimName: nfs-<NUMBER>
      - name: basic-config
        configMap:
          name: basic-config
      containers:
      - name: file-storage-container
        image: nginx
```

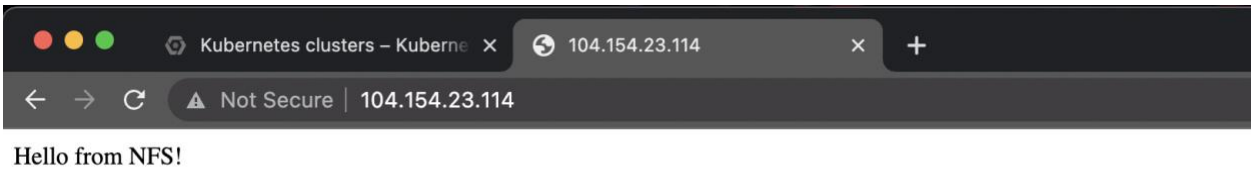
```
ports:
- containerPort: 80
volumeMounts:
  - mountPath: "/usr/share/nginx/html"
    name: volume-output
  - name: basic-config
    mountPath: /etc/nginx/conf.d
```

Apply and expose the port as a LoadBalancer:

```
kubectl create -n argo -f deployment-http-fileserver.yaml
kubectl expose deployment http-fileserver -n argo --type LoadBalancer --port
80 --target-port 80

kubectl get svc -n argo
```



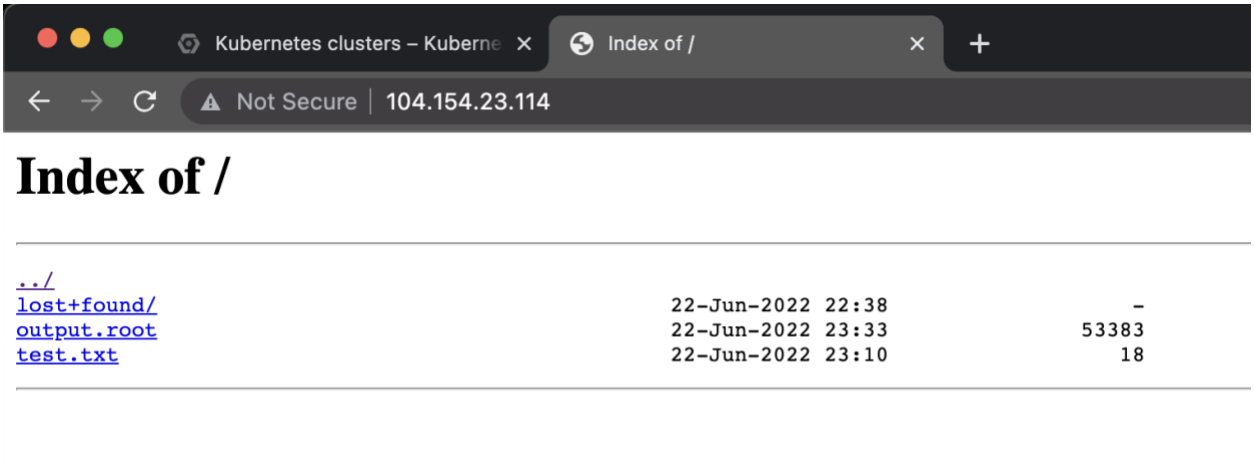
Test it in your browser



The <pending> EXTERNAL-IP will update after a few minutes (run the command again to check). Once it's there, copy the IP and paste it into a new browser tab. This should welcome you with a "Hello from NFS" message. In order to enable file browsing, we need to delete the index.html file in the pod. Determine the pod name using the first command listed below and adjust the second command accordingly.

```
kubectl get pods -n argo
kubectl exec http-fileserver-XXXXXXXX-YYYYY -n argo -- rm
/usr/share/nginx/html/index.html
```

Reload your localhost



NOTE: This IP is now accessible from anywhere in the world, and therefore also your files (mind: there are charges for outgoing bandwidth). Please delete the service again once you have finished downloading your files.

```
kubectl delete svc/http-fileserver -n argo
```

Remember to delete your workflow again to avoid additional charges.

```
argo delete -n argo @latest
```

Get the output file

Create a file pv-pod.yaml with the following contents:

```
nano pv-pod.yaml
```

Paste configuration and change <NUMBER>:

```
apiVersion: v1
kind: Pod
metadata:
  name: pv-pod
spec:
  volumes:
    - name: task-pv-storage
      persistentVolumeClaim:
        claimName: nfs-<NUMBER>
```

```
  containers:
    - name: pv-container
      image: busybox
      command: ["tail", "-f", "/dev/null"]
      volumeMounts:
        - mountPath: /mnt/data
          name: task-pv-storage
```

Create the storage pod and copy the files from there

```
kubectl apply -f pv-pod.yaml -n argo
kubectl cp  pv-pod:/mnt/data /tmp/poddata -n argo
```

and you will get the file created by the job in /tmp/poddata/test.txt.

Deleting Workspace

Commands:

```
kubectl delete ns argo
rm *
gcloud compute disks list
```

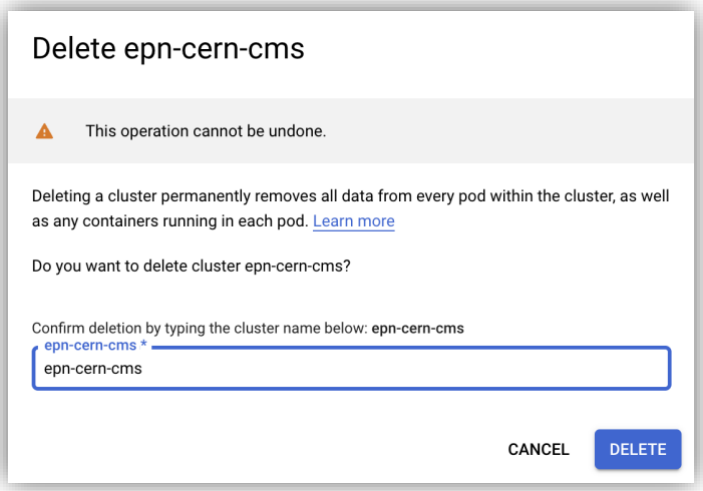
Only delete gce-nfs-disk-…….. with

```
xavitintin007@cloudshell:~ (cern-cms-epn)$ gcloud compute disks list
NAME: gce-nfs-disk-1
LOCATION: us-central1-c
LOCATION_SCOPE: zone
SIZE_GB: 100
TYPE: pd-standard
STATUS: READY

NAME: gke-epn-cern-cms-default-pool-d23d8bc4-svf4
LOCATION: us-central1-c
LOCATION_SCOPE: zone
SIZE_GB: 100
TYPE: pd-standard
STATUS: READY

NAME: gke-epn-cern-cms-default-pool-d23d8bc4-wtzj
LOCATION: us-central1-c
LOCATION_SCOPE: zone
SIZE_GB: 100
TYPE: pd-standard
STATUS: READY
xavitintin007@cloudshell:~ (cern-cms-epn)$
```

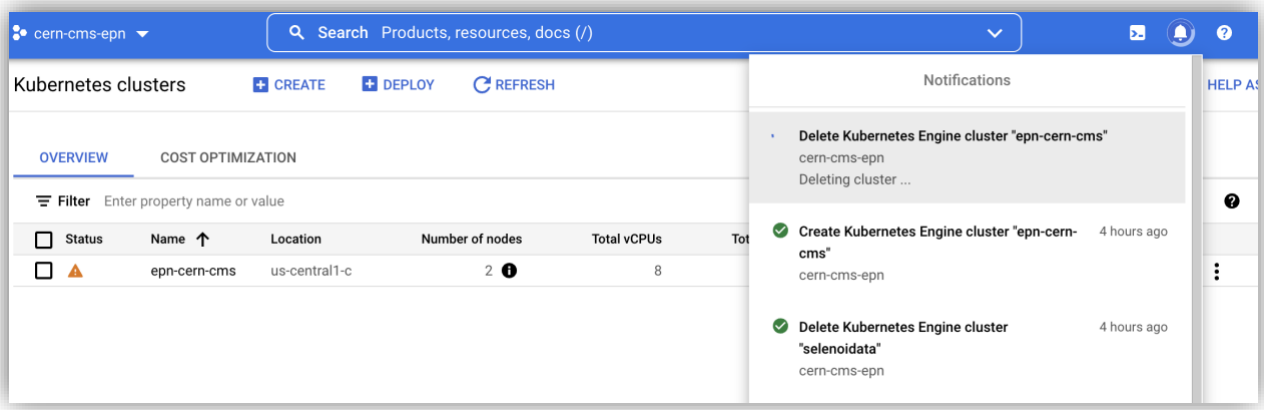
```
gcloud compute disks delete gce-nfs-disk-1 --zone=us-central1-c
```

```
xavitintin007@cloudshell:~ (cern-cms-epn)$ gcloud compute disks delete gce-nfs-disk-1 --zone=us-central1-c
The following disks will be deleted:
 - [gce-nfs-disk-1] in [us-central1-c]

Do you want to continue (Y/n)?  y

Deleted [https://www.googleapis.com/compute/v1/projects/cern-cms-epn/zones/us-central1-c/disks/gce-nfs-disk-1].
xavitintin007@cloudshell:~ (cern-cms-epn)$
```

Delete Kubernetes Cluster:



Standby to see the complete deletion of the cluster



Perfect you’re ready to start over

References:

Workshop 2020: https://cms-opendata-workshop.github.io/workshop-lesson-kubernetes/08-cleaning-up/index.html

Workshop 2021: https://cms-opendata-workshop.github.io/workshop2021-lesson-cloud/

Argo Getting Started: https://argo-cd.readthedocs.io/en/stable/getting_started/

CMS Repository: https://github.com/cms-opendata-analyses/AOD2NanoAODOutreachTool