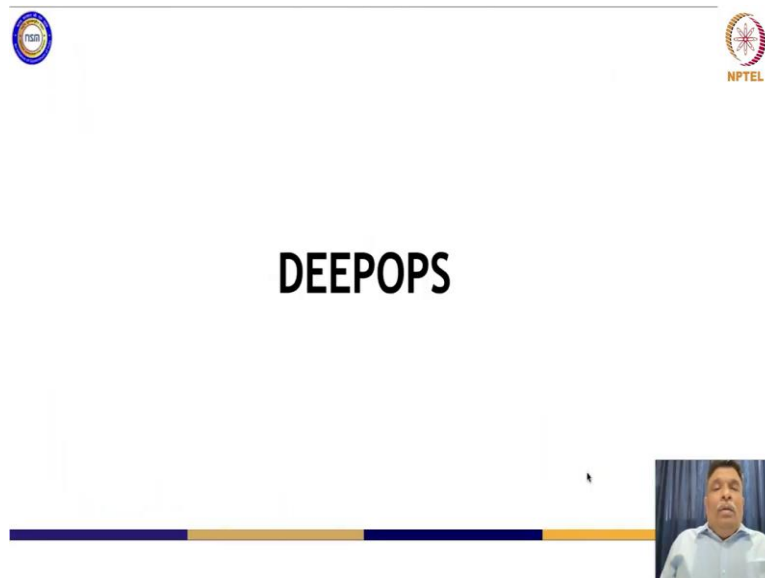


Applied Accelerated Artificial Intelligence
Prof. Satyadhyan Chickerur
School of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 13

DeepOps: Deep Dive into Kubernetes with deployment of various AI based Services
Session II - Kubernetes Part - 2

(Refer Slide Time: 00:15)



So, coming now from the concept of DevOps we reached the concept of MLOps and then now there is next concept of DeepOps which actually involves deep learning. And since deep learning is involved we would be having lot of GPUs with us right. So, those GPUs can be of any make anything.

(Refer Slide Time: 00:40)



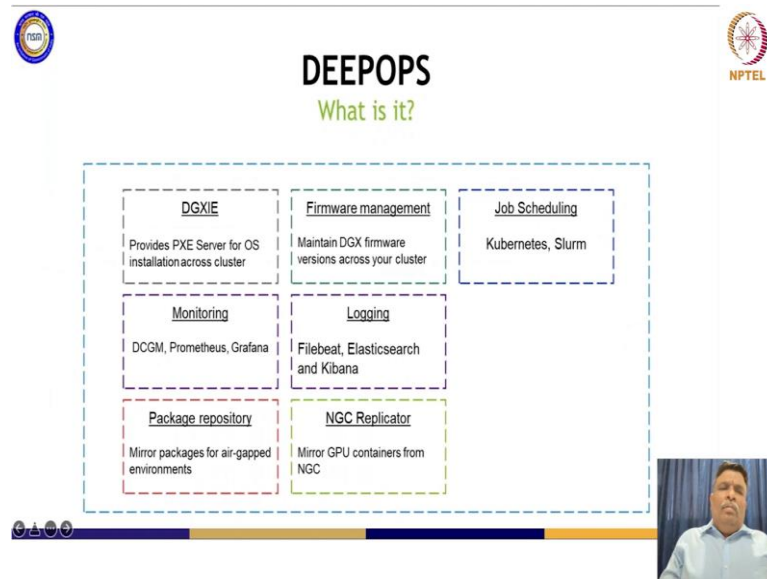
The slide features the IIT Madras logo on the top left and the NPTEL logo on the top right. The title 'DEEPOPS' is centered in a large, bold, black font, with the subtitle 'What is it?' in a smaller, green font below it. A bulleted list is positioned on the left side of the slide, and a small video inset showing a speaker is located in the bottom right corner.

- Opensource project to facilitate deployment of multi-node GPU clusters for Deep Learning and HPC environments, in an on-premise, optionally air-gapped datacenter or in the cloud
- DeepOps is also recognized as the **DGX POD Management Software**
- The **modular** nature of the project also allows more experienced administrators to pick and choose items that may be useful, making the process compatible with their existing software or infrastructure
- GitHub: <https://github.com/NVIDIA/deepops>

But DeepOps is basically a open source project ok from NVIDIA which facilitates deployment of multi node GPU clusters for deep learning and HPC environment in an on-premise optionally air gapped data center or in the cloud. So, the idea is you can have a physical on premise data centre or basically in the cloud. So, this is a open source project.

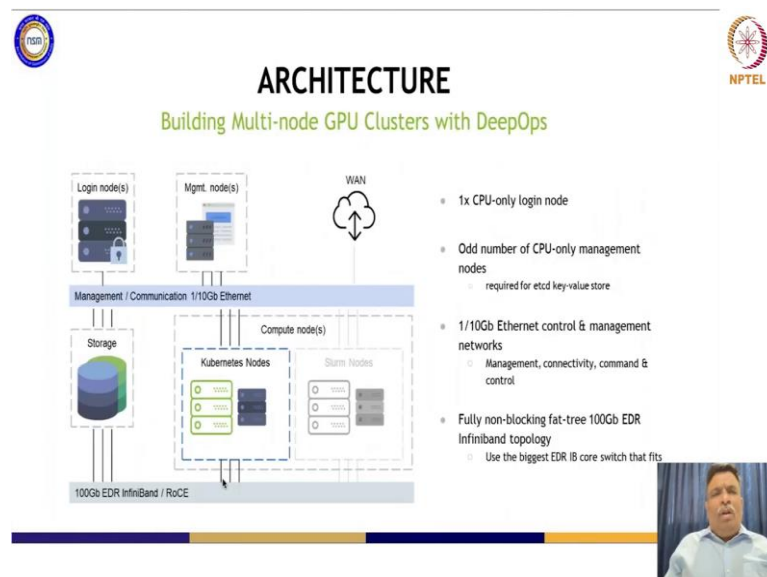
Now this actually is used for managing pods on the DGX. So, at present if you are not able to understand the architecture of DGX cluster, let us not worry too much about it. But let us try to understand that any node which is a worker node has to run some POD on it ok. It may use a GPU it may not use a GPU and that modularity of the project ok will be of lot of help to these administrators who actually maintain right this infrastructure and software. So, this is the GitHub link to DeepOps.

(Refer Slide Time: 02:07)



Now there are so many things which DeepOps can do. So, it can do the job scheduling using Kubernetes Slurm you can maintain the firmware of the DGX you can do monitoring using DCGM and Prometheus and Grafana. So, we will see this actually today monitoring and then you can do Filebeat Elasticsearch and all of this as to how do you log then you can maintain and manage various packages ok so on and so forth.

(Refer Slide Time: 02:44)

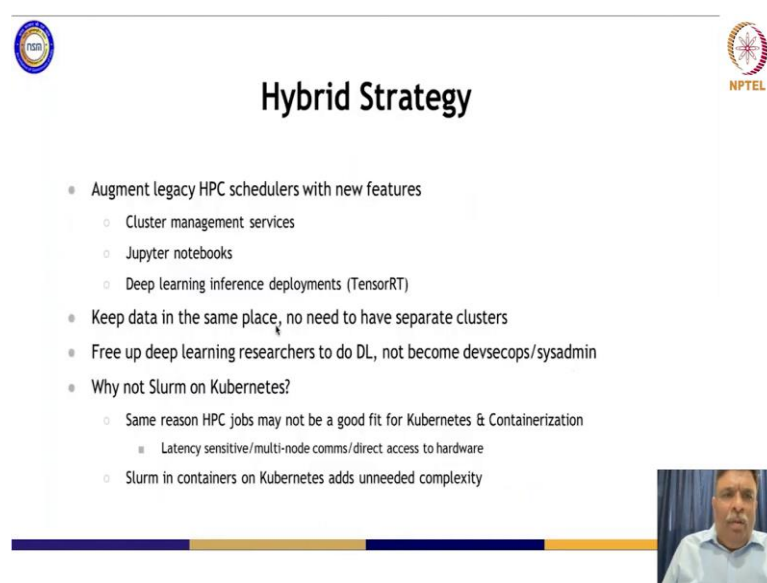


So, now this is the architecture you have got the login node you have got the management node and you have got the Kubernetes node and these Kubernetes node are

actually a part of the compute node. So, you will have the Slurm nodes you will have Kubernetes nodes because we know that any docker thing which is not generated by Kubernetes is not going to be maintained by the Kubernetes cluster.

So, you can actually take a docker image and run it, but if Kubernetes has to manage it it has to be actually what to say started or be done on a Kubernetes through a Kubernetes image right. So, something like that. So, here I am just going into the basic of this as to how DeepOps came into this thing.

(Refer Slide Time: 03:36)



The slide is titled "Hybrid Strategy" and features a list of bullet points. In the top left corner is a circular logo with the text "nptel". In the top right corner is the "NPTEL" logo. A small video inset in the bottom right corner shows a man speaking. The slide has a decorative bar at the bottom with segments of blue, yellow, and blue.

- Augment legacy HPC schedulers with new features
 - Cluster management services
 - Jupyter notebooks
 - Deep learning inference deployments (TensorRT)
- Keep data in the same place, no need to have separate clusters
- Free up deep learning researchers to do DL, not become devsecops/sysadmin
- Why not Slurm on Kubernetes?
 - Same reason HPC jobs may not be a good fit for Kubernetes & Containerization
 - Latency sensitive/multi-node comms/direct access to hardware
 - Slurm in containers on Kubernetes adds unneeded complexity

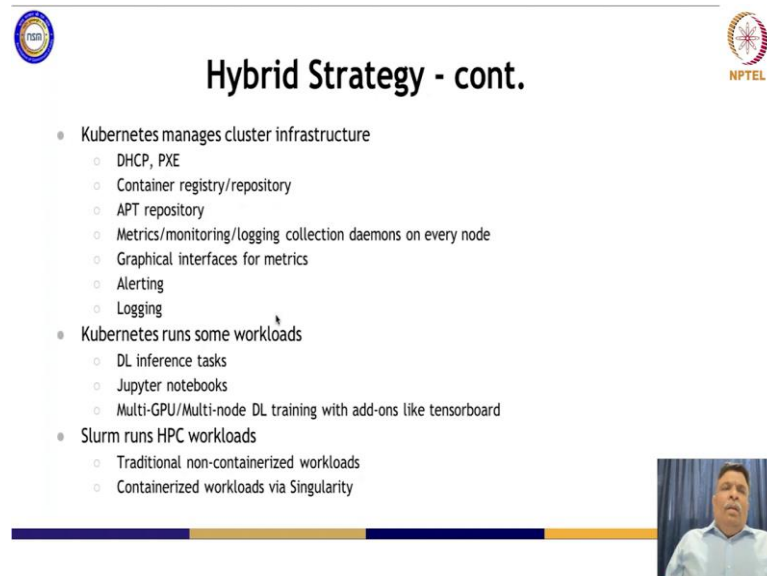
And now there is a idea of hybrid strategy wherein we talk of ok. So, we have this augment legacy of HPC schedulers with new features. So, we have cluster management services we have Jupyter notebooks we have deep learning inference deployments right.

So, you have to do inferencing. Inferencing is nothing, but whatever is being done by the servicing pipeline ok or you need to actually deploy it. So, we use Tensor RT and then. So, that basically means you should ensure how do you deploy. So, we need to also see that how the data has to be kept in the same place no need to have separate clusters free up deep learning researchers to do DL not become devsecops or system admins why not Slurm on kubernetes.

So, there these are all things which are still in discussion or debatable based on the flexibility of the system based on how the user wants to use it and perceive it right. So,

that is how it is, but the idea is that all of this are going to make our lives easy for maintaining things which we need to do ok.

(Refer Slide Time: 04:59)



The slide is titled "Hybrid Strategy - cont." and features two logos at the top: a circular logo on the left and the NPTEL logo on the right. The main content is a bulleted list:

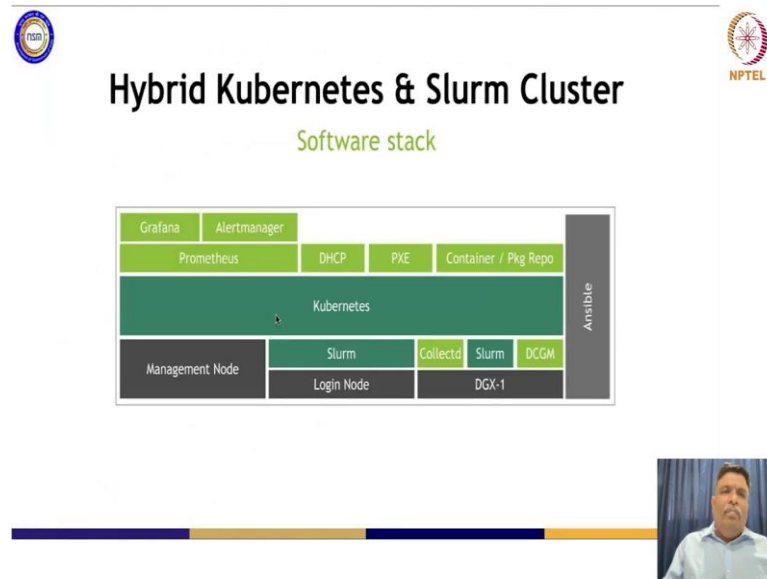
- Kubernetes manages cluster infrastructure
 - DHCP, PXE
 - Container registry/repository
 - APT repository
 - Metrics/monitoring/logging collection daemons on every node
 - Graphical interfaces for metrics
 - Alerting
 - Logging
- Kubernetes runs some workloads
 - DL inference tasks
 - Jupyter notebooks
 - Multi-GPU/Multi-node DL training with add-ons like tensorboard
- Slurm runs HPC workloads
 - Traditional non-containerized workloads
 - Containerized workloads via Singularity

In the bottom right corner, there is a small video inset showing a man in a light blue shirt speaking.

So, we are talking of Kubernetes managing the cluster infrastructure it can run some workloads we will have DL inference tasks we can run Jupyter notebooks we can do multi GPU multi node deep learning training right and we can do tensorboards there can. So, many graphs. So, many charts ok all of this. So, Slurm runs HPC workload right.

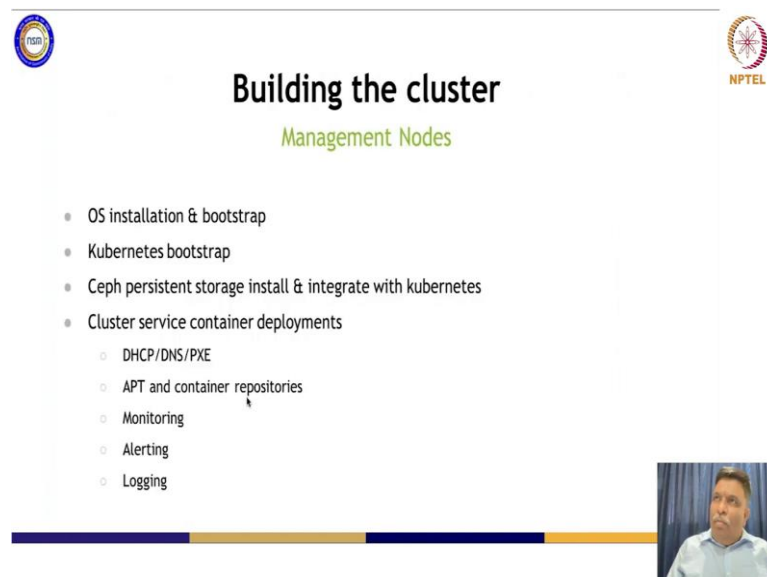
So, these HPC workload traditionally are non containerized workloads. And but nowadays you get lot of HPC containers right. So, whatever people used to do GROMACS this that everything still you have got a lot of containers which you can use for that matter ok. So, now, it is changing a bit. So, no issue on that part actually.

(Refer Slide Time: 05:53)



So, how do you actually stock up the stack up the software stack. You are trying to develop a hybrid Kubernetes and Slurm cluster. So, you have got prometheus here DHCP containers Grafana alert manager Kubernetes then Slurm login node this is your compute node or DGX or whatever then you have this management node. So, something like this right.

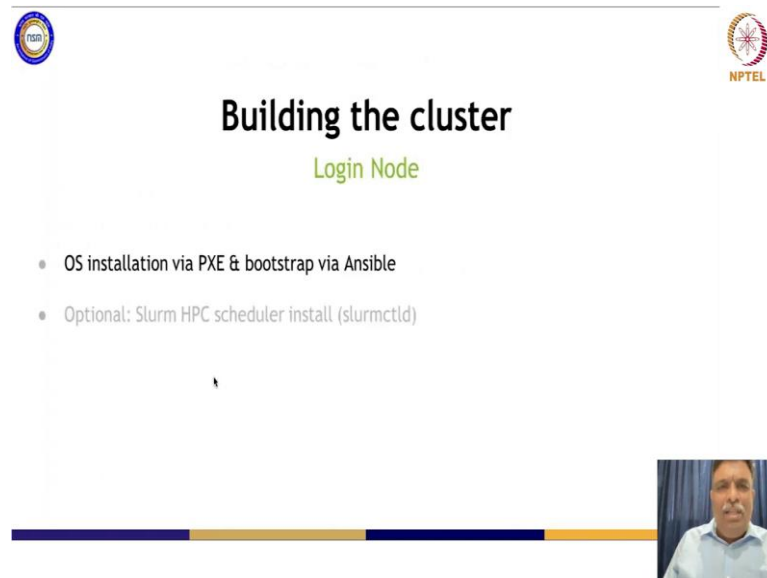
(Refer Slide Time: 06:25)



So, let us try to do some demo and before that just for people who want to understand right how do you actually develop your own cluster. So, I am just going in very brief as

to how do you develop your own cluster you should have a management node which actually has the OS installation and Kubernetes bootstrap you have to install and integrate ceph persistent storage persistent storage is very very important and then there are container deployment.

(Refer Slide Time: 06:57)

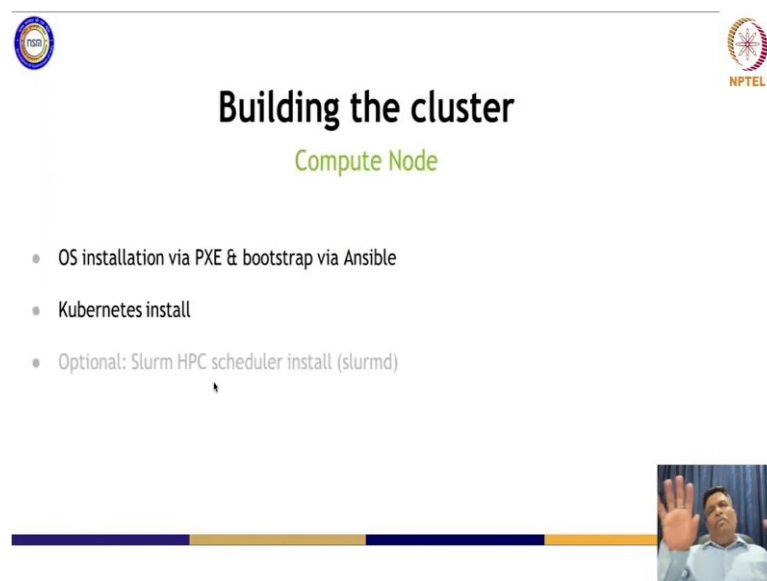


The slide is titled "Building the cluster" and is for the "Login Node". It features a list of bullet points: "OS installation via PXE & bootstrap via Ansible" and "Optional: Slurm HPC scheduler install (slurmctld)". The slide is part of a presentation with logos for TUM and NPTEL in the top corners. A video feed of a man is visible in the bottom right corner.

- OS installation via PXE & bootstrap via Ansible
- Optional: Slurm HPC scheduler install (slurmctld)

So, you have this management node you have the login node there are various ways of trying to actually do OS installation and bootstrapping of the login node.

(Refer Slide Time: 07:07)

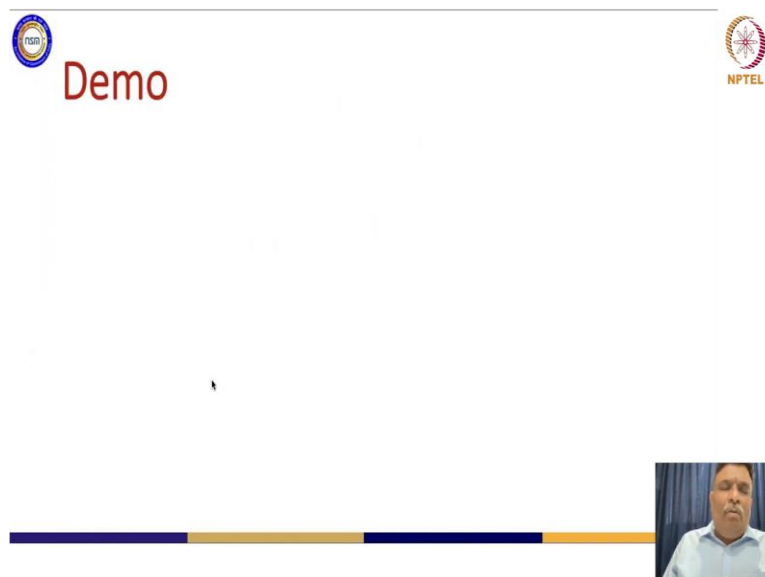


The slide is titled "Building the cluster" and is for the "Compute Node". It features a list of bullet points: "OS installation via PXE & bootstrap via Ansible", "Kubernetes install", and "Optional: Slurm HPC scheduler install (slurmd)". The slide is part of a presentation with logos for TUM and NPTEL in the top corners. A video feed of a man is visible in the bottom right corner.

- OS installation via PXE & bootstrap via Ansible
- Kubernetes install
- Optional: Slurm HPC scheduler install (slurmd)

And then the compute node wherein you need to install Kubernetes again and then bootstrapping it.

(Refer Slide Time: 07:13)

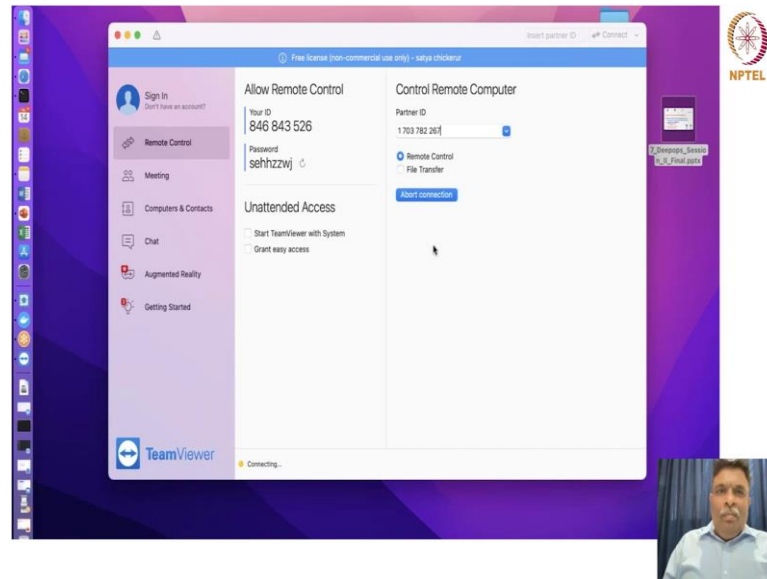


So, this is in brief actually what I thought we will do it in the theory portion. Now, let us go for the demo. So, what we are going to do in the demo now today is I will show you two ways of trying to see the same type of the setup right. We have actually one master node two worker nodes ok.

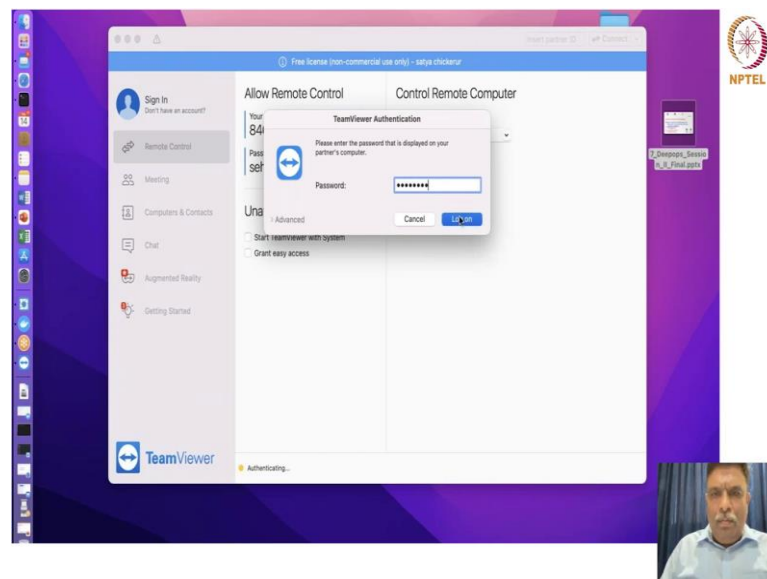
And then we will try to do it from the standard CLI type of thing using the Kubernetes dashboard try to analyze it try to run machine learning application. Wherein we are trying to do machine learning application development wherein by seeing the marks of the students the previous performance of the students we are going to predict as to what is his probability of getting placed right. This is a small ml application which we are going to develop.

And next we will see another type of a dashboard which is far more easier than the previous one. So, that you can appreciate as to how this particular thing of Kubernetes the GUI development the real time analysis all is actually improving ok for us to be better deep learning scientists or programmers.

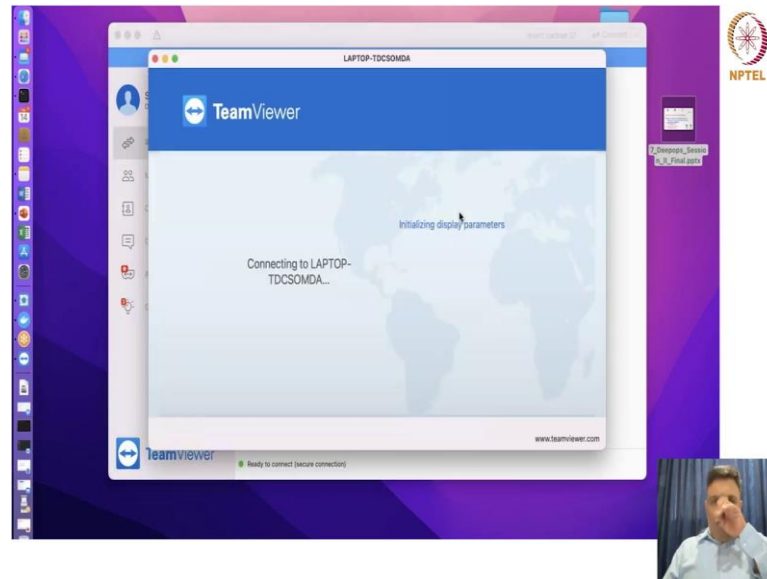
(Refer Slide Time: 08:56)



(Refer Slide Time: 09:12)

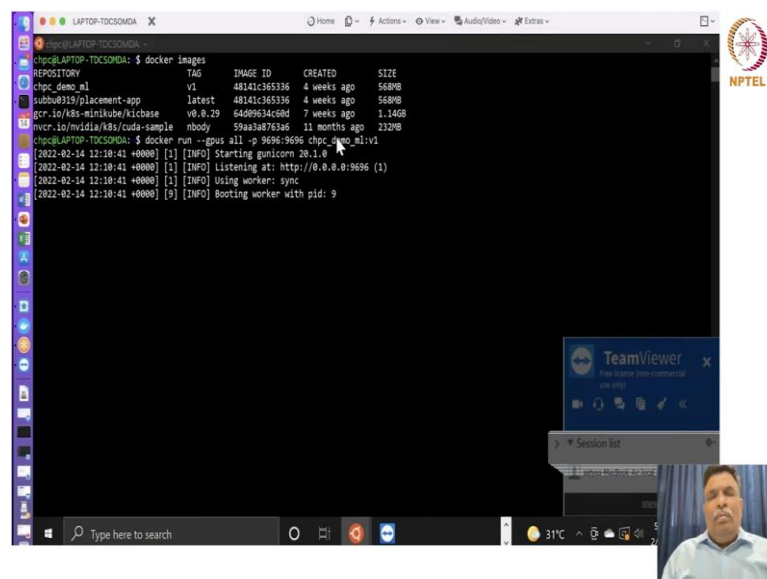


(Refer Slide Time: 09:23)



So, let us try to go to the demo. Let me just log on to it ok.

(Refer Slide Time: 09:24)



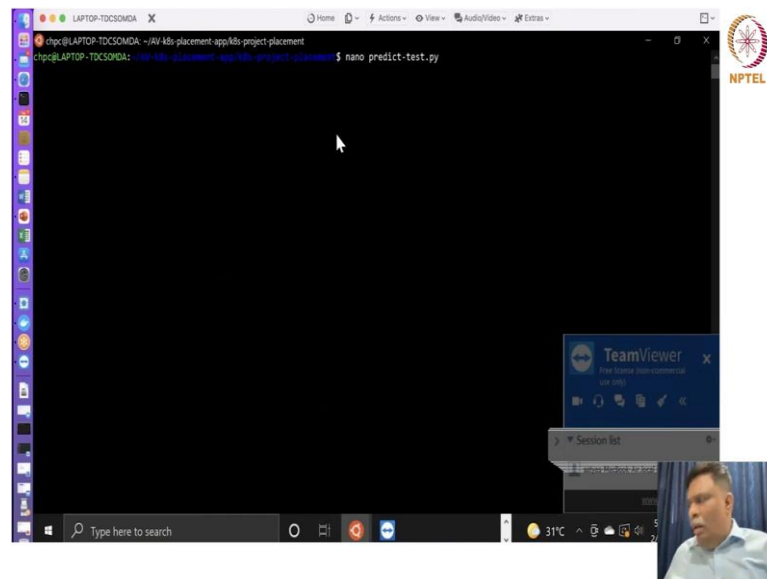
So, now, let us try to see whether what dockers we have ok. What dockers we have? So, yeah so, we have got so many docker images right. So, what we are trying to do is we are trying to use a demo docker which is CHPC demo for machine learning.

It has a tag v1 it has got it is this image id right and it was created 4 weeks ago and this is the size of that particular CHPC demo ml. So, how we have done it? We have accessed

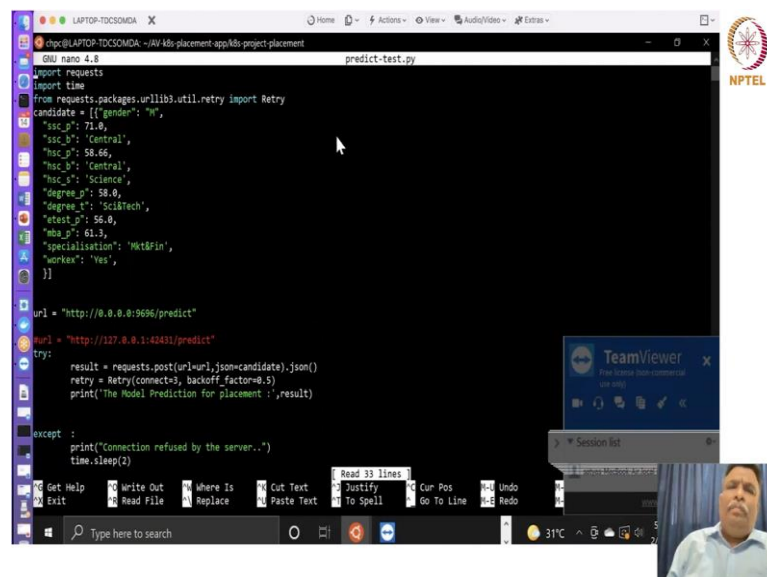
the docker from this we will share the link we did some modification onto it and then tried to rename it and repackage it as a CHPC demo ml docker ok yeah.

So, the next thing is to see and run this docker. So, how do we run? We use this command docker run GPUs all these are the port numbers, local host port numbers then this is forwarding and then this is this particular program ok which we need to run ok.

(Refer Slide Time: 11:08)



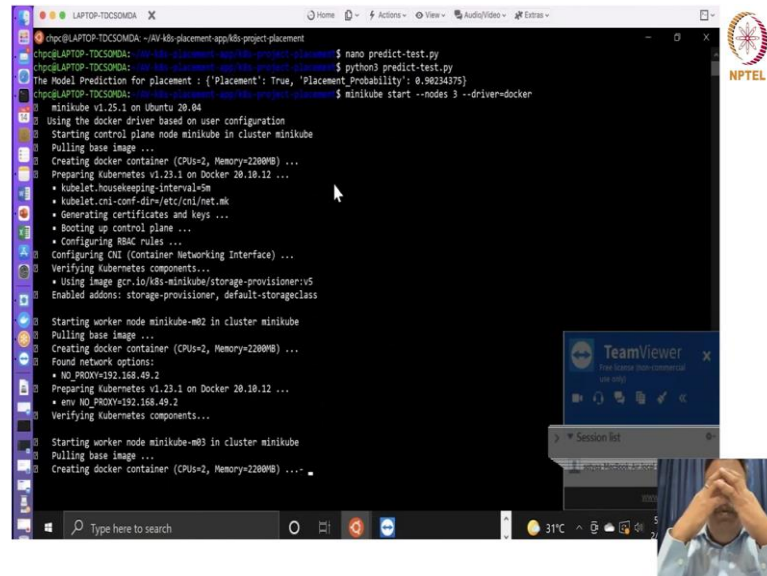
(Refer Slide Time: 11:27)



So, now what we need to do is try to understand and try to see excuse me. Try to see the program to do some prediction which basically is something like this. So, the candidate

gender, his marks ok and what has to be done very simple rudimentary type of stuff just to show you people as to what are we going to do right.

(Refer Slide Time: 12:00)



The screenshot shows a terminal window with the following commands and output:

```
chpc@LAPTOP-TDCSOMDA: ~$ nano predict-test.py
chpc@LAPTOP-TDCSOMDA: ~$ python3 predict-test.py
The Model Prediction for placement : ('Placement': True, 'Placement_Probability': 0.90234375)
chpc@LAPTOP-TDCSOMDA: ~$ minikube start --nodes 3 --driver=docker

minikube v1.25.1 on Ubuntu 20.04
Using the docker driver based on user configuration
Starting control plane node minikube in cluster minikube
Pulling base image ...
Creating docker container (CPUs=2, Memory=2200MB) ...
Preparing Kubernetes v1.23.1 on Docker 20.10.12 ...
  • kubeadm.housekeeping-interval=5m
  • kubeadm.cni-conf-dir=/etc/cni/net.m
  • Generating certificates and keys ...
  • Booting up control plane ...
  • Configuring RBAC rules ...
  • Configuring CNI (Container Networking Interface) ...
  • Verifying Kubernetes components...
  • Using image gcr.io/k8s-minikube/storage-provisioner:v5
Enabled addons: storage-provisioner, default-storageclass

Starting worker node minikube-m02 in cluster minikube
Pulling base image ...
Creating docker container (CPUs=2, Memory=2200MB) ...
Found network options:
  • NO_PROXY=192.168.49.2
Preparing Kubernetes v1.23.1 on Docker 20.10.12 ...
  • new NO_PROXY=192.168.49.2
Verifying Kubernetes components...

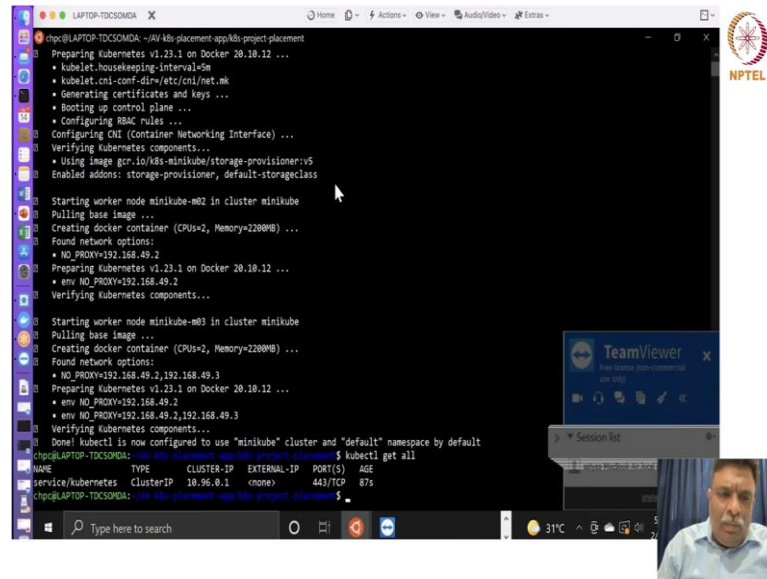
Starting worker node minikube-m03 in cluster minikube
Pulling base image ...
Creating docker container (CPUs=2, Memory=2200MB) ...
```

So, now let us try to run this program. So, once we run this program the probability ok for placement is 0.90234375. Now let us try to see this in the dashboards right. So, the idea is that we are going to start ok the minikube nodes as the three things. So, what we have tried to do here is we have tried to just see ok that the dockers which we have downloaded runs using a python.

Now, we are trying to link it up to cluster these three nodes which we are running ok. So, I hope it is clear that minikube is a local Kubernetes which gives me the option of generating a master and two slaves or workers before that I have my python program which can do this prediction right.

So, now I am going to take this python program take it as a docker ok and then try to run it. So, that it is serviced for people or people can access it when it is running ok on one of these worker nodes right.

(Refer Slide Time: 14:33)



The terminal window displays the following commands and output:

```
chpc@LAPTOP-TDCSOMDA: ~/k8s-placement-app/k8s-project/placement
Preparing Kubernetes v1.23.1 on Docker 20.10.12 ...
  • kubeadm.housekeeping-interval=5m
  • kubeadm.conf-dir=/etc/kubernetes
  • Generating certificates and keys ...
  • Booting up control plane ...
  • Configuring RBAC rules ...
Configuring CNI (Container Networking Interface) ...
Verifying Kubernetes components...
  • Using image gcr.io/k8s-minikube/storage-provisioner:v5
Enabled addons: storage-provisioner, default-storageclass

Starting worker node minikube-m02 in cluster minikube
Pulling base image ...
Creating docker container (CPUs=2, Memory=2200MB) ...
Found network options:
  • NO_PROXY=192.168.49.2
Preparing Kubernetes v1.23.1 on Docker 20.10.12 ...
  • env NO_PROXY=192.168.49.2
Verifying Kubernetes components...

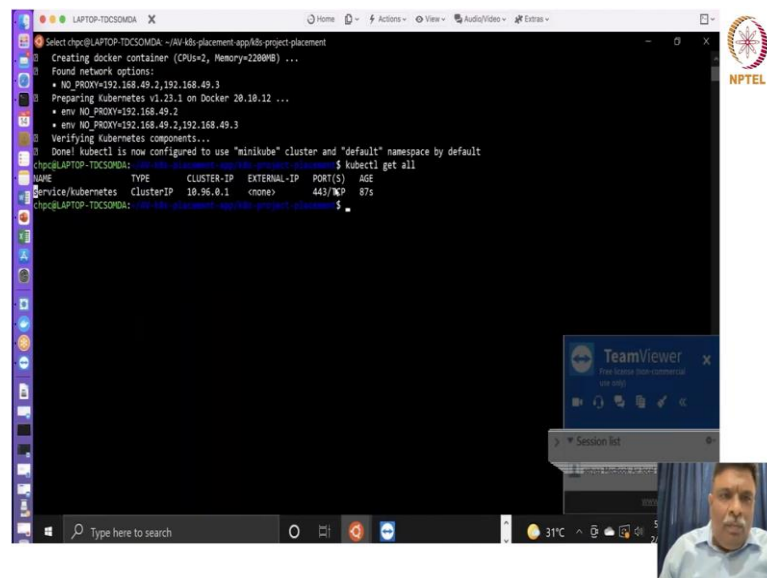
Starting worker node minikube-m03 in cluster minikube
Pulling base image ...
Creating docker container (CPUs=2, Memory=2200MB) ...
Found network options:
  • NO_PROXY=192.168.49.2,192.168.49.3
Preparing Kubernetes v1.23.1 on Docker 20.10.12 ...
  • env NO_PROXY=192.168.49.2
  • env NO_PROXY=192.168.49.2,192.168.49.3
Verifying Kubernetes components...
Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default

chpc@LAPTOP-TDCSOMDA: ~/k8s-placement-app/k8s-project/placement$ kubectl get all
NAME                                TYPE                CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
service/kubernetes                  ClusterIP           10.96.0.1        <none>            443/TCP          87s
```

A TeamViewer overlay is visible in the bottom right corner of the terminal window.

So, the idea is that when you work on a desktop, you can just run your program see it and then you tell that you will test it. So, you give the input it gives you the prediction. So, it all ok done. But when you are giving it as a service to people in a sense that when you are trying to develop application trying to sell it or then trying to develop it for the use of people then basically you need to actually give it as a service. So, we will have to deploy it somewhere right. Now how are we going to deploy it we are going to deploy it using kubectl see here.

(Refer Slide Time: 15:20)



The terminal window displays the following commands and output:

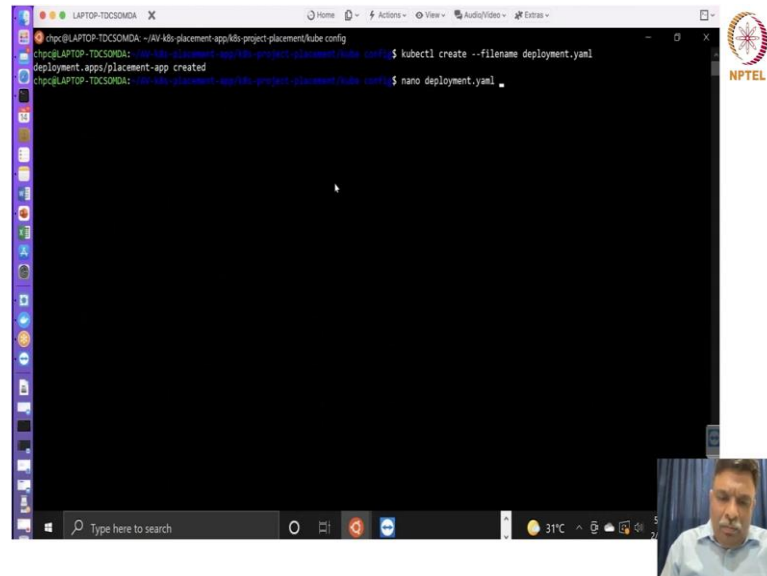
```
chpc@LAPTOP-TDCSOMDA: ~/k8s-placement-app/k8s-project/placement
Creating docker container (CPUs=2, Memory=2200MB) ...
Found network options:
  • NO_PROXY=192.168.49.2,192.168.49.3
Preparing Kubernetes v1.23.1 on Docker 20.10.12 ...
  • env NO_PROXY=192.168.49.2
Verifying Kubernetes components...
Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default

chpc@LAPTOP-TDCSOMDA: ~/k8s-placement-app/k8s-project/placement$ kubectl get all
NAME                                TYPE                CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
service/kubernetes                  ClusterIP           10.96.0.1        <none>            443/TCP          87s
```

A TeamViewer overlay is visible in the bottom right corner of the terminal window.

So, this is actually the service being provided by the Kubernetes on that particular IP cluster IP right.

(Refer Slide Time: 15:38)



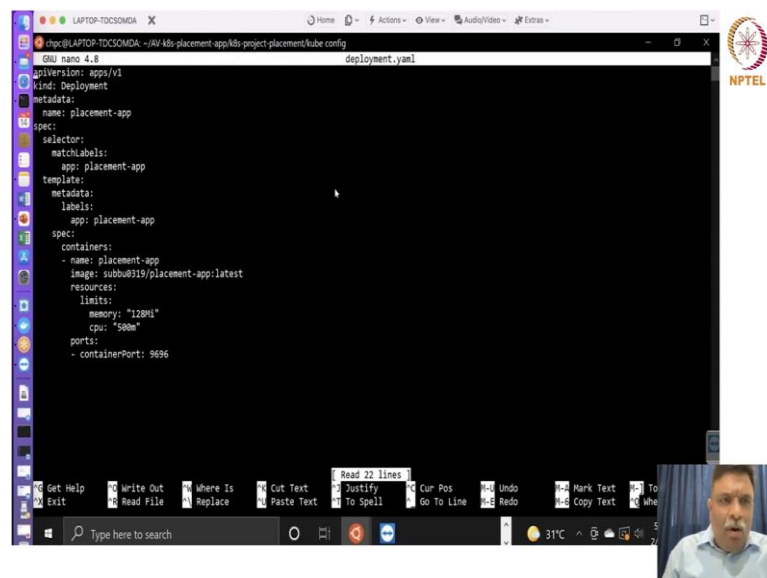
The screenshot shows a terminal window with the following commands and output:

```
chpc@LAPTOP-TDCSOMDA: ~/k8s-placement-app/k8s-project-placement/kube.config
chpc@LAPTOP-TDCSOMDA: ~/k8s-placement-app/k8s-project-placement/kube.config $ kubectl create --filename deployment.yaml
deployment.apps/placement-app created
chpc@LAPTOP-TDCSOMDA: ~/k8s-placement-app/k8s-project-placement/kube.config $ nano deployment.yaml
```

The terminal window is titled "LAPTOP-TDCSOMDA" and shows the file explorer on the left. The nano editor is open, showing the beginning of the deployment file.

So, now we will run this and then try to deploy it right. So, we have created we will try to see the deployment file now.

(Refer Slide Time: 15:59)



The screenshot shows the nano editor displaying the contents of the deployment file:

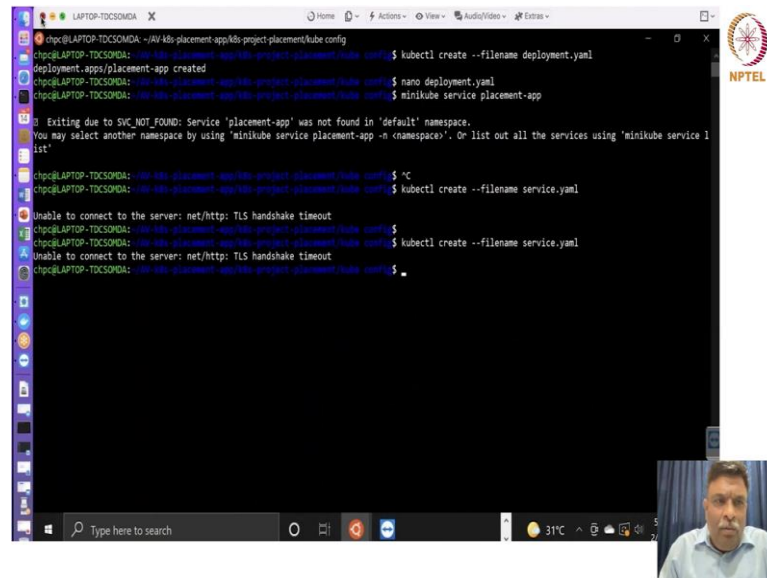
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: placement-app
spec:
  selector:
    matchLabels:
      app: placement-app
  template:
    metadata:
      labels:
        app: placement-app
    spec:
      containers:
        - name: placement-app
          image: subbu0319/placement-app:latest
          resources:
            limits:
              memory: "128Mi"
              cpu: "500m"
          ports:
            - containerPort: 9090
```

The terminal window is titled "LAPTOP-TDCSOMDA" and shows the file explorer on the left. The nano editor is open, showing the full contents of the deployment file.

So, what does this deployment file mean? We are trying to deploy this is a kind of deployment metadata tells that it is some name by name placement app application is a placement app the containers name is placement app, but this is the container which we

have tried to take the image from. I have this limit of my memory decided the CPU this much ok and then the container pod on which this particular pod actually is supposed to be deployed right. So, this is what we have tried to do it.

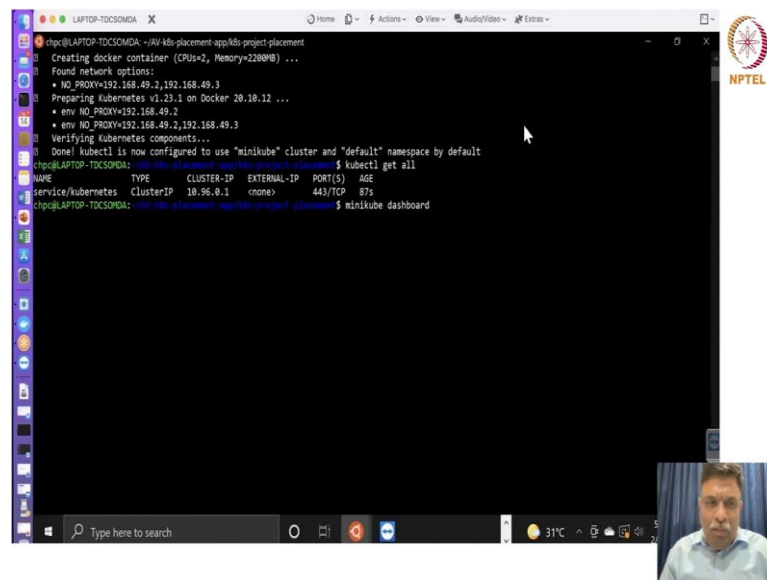
(Refer Slide Time: 16:43)



```
chpc@LAPTOP-TDCSOMDA: ~/k8s-placement-app/k8s-project-placement/kube-config
chpc@LAPTOP-TDCSOMDA: ~/k8s-placement-app/k8s-project-placement/kube-config$ kubectl create --filename deployment.yaml
deployment.apps/placement-app created
chpc@LAPTOP-TDCSOMDA: ~/k8s-placement-app/k8s-project-placement/kube-config$ nano deployment.yaml
chpc@LAPTOP-TDCSOMDA: ~/k8s-placement-app/k8s-project-placement/kube-config$ minikube service placement-app
Exiting due to SVC_NOT_FOUND: Service 'placement-app' was not found in 'default' namespace.
You may select another namespace by using 'minikube service placement-app -n <namespace>'. Or list out all the services using 'minikube service list'
chpc@LAPTOP-TDCSOMDA: ~/k8s-placement-app/k8s-project-placement/kube-config$ ^C
chpc@LAPTOP-TDCSOMDA: ~/k8s-placement-app/k8s-project-placement/kube-config$ kubectl create --filename service.yaml
Unable to connect to the server: net/http: TLS handshake timeout
chpc@LAPTOP-TDCSOMDA: ~/k8s-placement-app/k8s-project-placement/kube-config$
chpc@LAPTOP-TDCSOMDA: ~/k8s-placement-app/k8s-project-placement/kube-config$ kubectl create --filename service.yaml
Unable to connect to the server: net/http: TLS handshake timeout
chpc@LAPTOP-TDCSOMDA: ~/k8s-placement-app/k8s-project-placement/kube-config$
```

Now, once we deploy it we will actually make it available as a service correct. So, then we run this minikube service placement app right. So, now, it is running ok. Let us wait for some time. 1 minute, we got some error it seems we have not been able to, yes. I do not know why this has become so slow. We will see the Kubernetes dashboard and see whether the previous one is running.

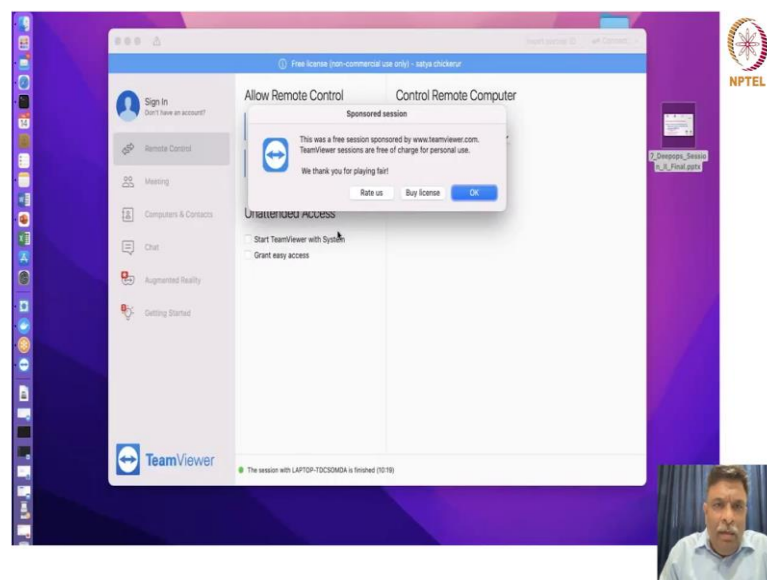
(Refer Slide Time: 19:01)



```
chpc@LAPTOP-TDCSOMDA: ~/AI4-K8s-placement-app/k8s-project-placement
Creating docker container (CPUs=2, Memory=2200MB) ...
Found network options:
  * NO_PROXY=192.168.49.2,192.168.49.3
  * env NO_PROXY=192.168.49.2,192.168.49.3
  * env NO_PROXY=192.168.49.2,192.168.49.3
Preparing Kubernetes v1.23.1 on Docker 20.10.12 ...
Verifying Kubernetes components...
Done! kubect1 is now configured to use "minikube" cluster and "default" namespace by default
chpc@LAPTOP-TDCSOMDA: ~$ kubectl get all
NAME                                TYPE                CLUSTER-IP    EXTERNAL-IP    PORT(S)    AGE
service/kubernetes                  ClusterIP           10.96.0.1     <none>          443/TCP    87s
chpc@LAPTOP-TDCSOMDA: ~$ minikube dashboard
```

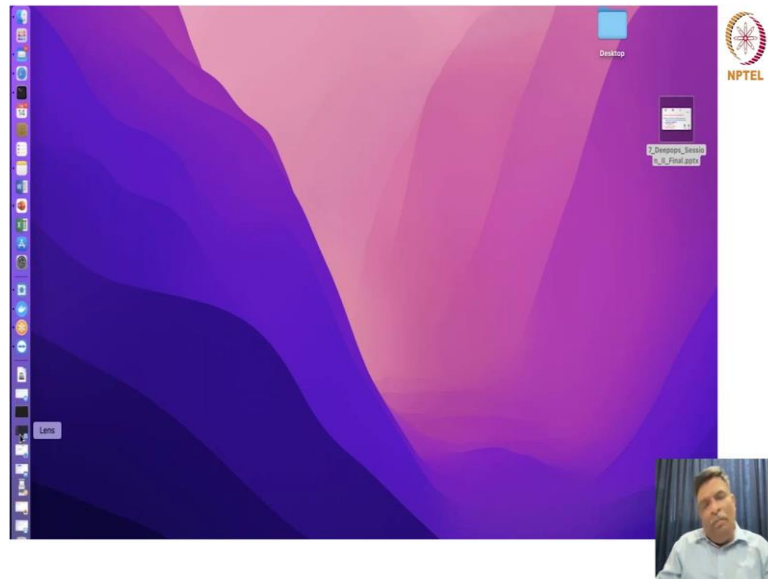
So, there is some issue with the TLS handshake. So, let me just go to the next thing in the meantime we will try to debug it.

(Refer Slide Time: 19:42)



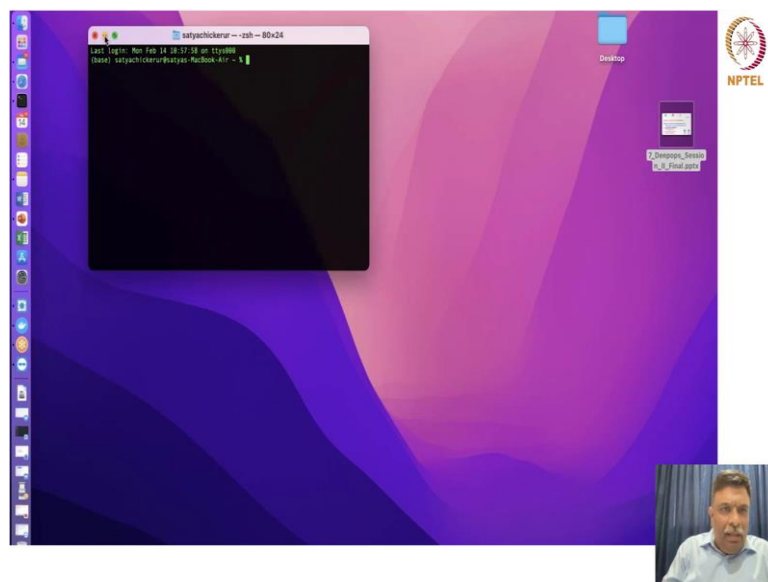
But let us try to actually go and see a situation here.

(Refer Slide Time: 19:51)



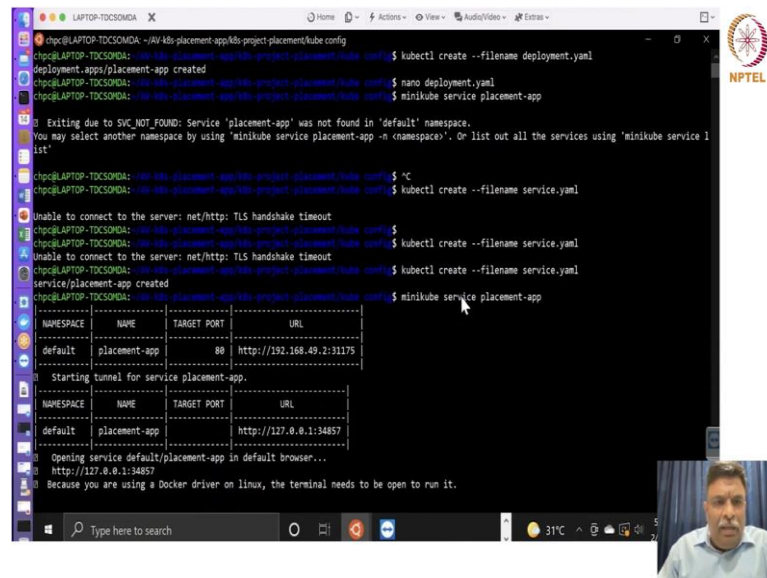
See once you do it using this right there will be certain issues like this popping up what is now actually happening is this kubectl has got some issues right. So, these type of issues to avoid yes.

(Refer Slide Time: 20:11)



So, we will try to actually see that it is done up and running it takes time. So, sometimes what happens is you are and again connect it where is this r x 3 4 0 6.

(Refer Slide Time: 20:45)



The terminal shows the following commands and output:

```
chpc@LAPTOP-TDCSOMDA: ~/k8s-placement-app/k8s-project-placement/kube config
chpc@LAPTOP-TDCSOMDA: ~/k8s-placement-app/k8s-project-placement/kube config $ kubectl create --filename deployment.yaml
deployment.apps/placement-app created
chpc@LAPTOP-TDCSOMDA: ~/k8s-placement-app/k8s-project-placement/kube config $ nano deployment.yaml
chpc@LAPTOP-TDCSOMDA: ~/k8s-placement-app/k8s-project-placement/kube config $ minikube service placement-app
Exiting due to SVC_NOT_FOUND: Service 'placement-app' was not found in 'default' namespace.
You may select another namespace by using 'minikube service placement-app -n <namespace>'. Or list out all the services using 'minikube service list'
chpc@LAPTOP-TDCSOMDA: ~/k8s-placement-app/k8s-project-placement/kube config $ ^C
chpc@LAPTOP-TDCSOMDA: ~/k8s-placement-app/k8s-project-placement/kube config $ kubectl create --filename service.yaml
Unable to connect to the server: net/http: TLS handshake timeout
chpc@LAPTOP-TDCSOMDA: ~/k8s-placement-app/k8s-project-placement/kube config $ kubectl create --filename service.yaml
Unable to connect to the server: net/http: TLS handshake timeout
chpc@LAPTOP-TDCSOMDA: ~/k8s-placement-app/k8s-project-placement/kube config $ kubectl create --filename service.yaml
service/placement-app created
chpc@LAPTOP-TDCSOMDA: ~/k8s-placement-app/k8s-project-placement/kube config $ minikube service placement-app
```

NAMESPACE	NAME	TARGET PORT	URL
default	placement-app	80	http://192.168.49.2:31175

Starting tunnel for service placement-app.

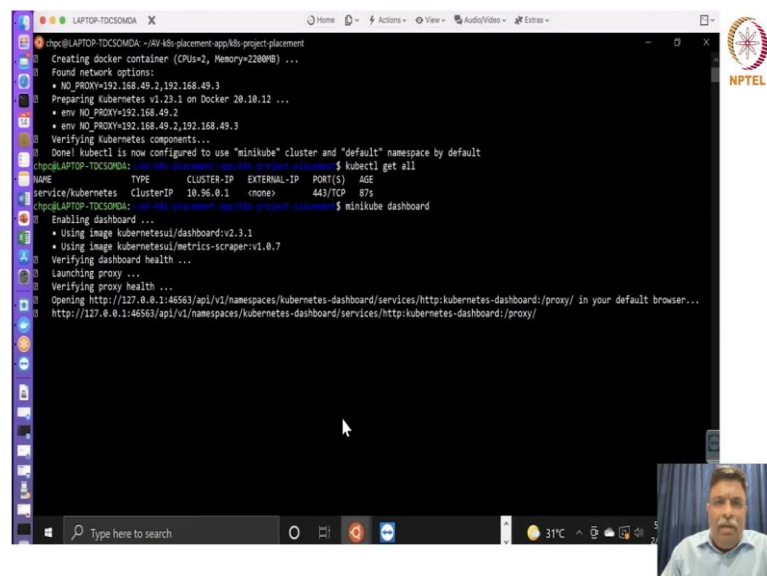
NAMESPACE	NAME	TARGET PORT	URL
default	placement-app	80	http://127.0.0.1:34857

Opening service default/placement-app in default browser...
http://127.0.0.1:34857

Because you are using a Docker driver on linux, the terminal needs to be open to run it.

Yeah so, the service placement app got created we were running the same thing, but it was giving us the TLS handshake timeout. So, sometimes it happens, but anyway once we are able to do it. Now yes so, if you see this we are able to run it in the default name placement space right the default name space this placement app with this particular target port using this URL and then we have to actually do a tunnelling for this service placement app to this URL right.

(Refer Slide Time: 21:33)

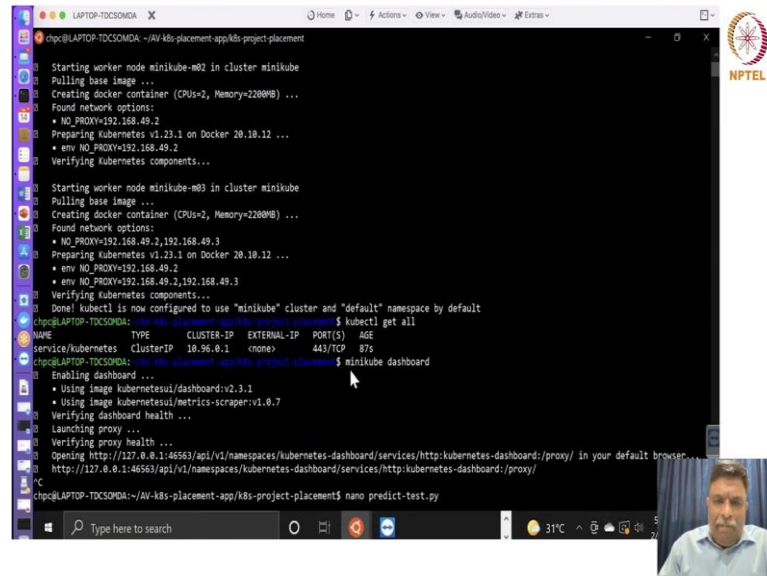


The terminal shows the following commands and output:

```
chpc@LAPTOP-TDCSOMDA: ~/k8s-placement-app/k8s-project-placement
chpc@LAPTOP-TDCSOMDA: ~/k8s-placement-app/k8s-project-placement $ kubectl get all
NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1     <none>         443/TCP   87s
chpc@LAPTOP-TDCSOMDA: ~/k8s-placement-app/k8s-project-placement $ minikube dashboard
Enabling dashboard ...
  * Using image kubernetes/dashboard:v2.3.1
  * Using image kubernetes/metrics-scraper:v1.0.7
Verifying dashboard health ...
Launching proxy ...
Verifying proxy health ...
Opening http://127.0.0.1:46563/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/ in your default browser...
http://127.0.0.1:46563/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/
```

So, we will now try to open this dashboard ok and see as to what we get and then we will discuss.

(Refer Slide Time: 21:44)



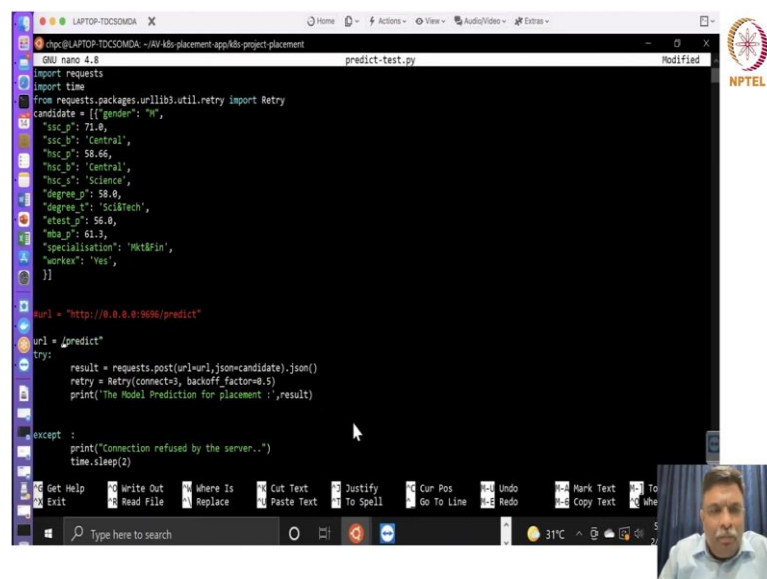
A terminal window titled 'LAPTOP-TDCSOMDA' showing the setup of a minikube cluster. The output includes commands for starting worker nodes, pulling base images, creating docker containers, and preparing Kubernetes components. A table shows the cluster status:

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/kubernet	ClusterIP	10.96.0.1	<none>	443/TCP	87s

The user then runs 'minikube dashboard' and the terminal shows instructions for enabling the dashboard and opening the URL in a browser.

So, what effectively happens is we are trying to do two things; one is the old URL one is the new URL and we will have to do the tunnelling. So, we will now try to run the prediction ok using that URL because it will be serviced there now ok. So, we will we have changed this and then we will now try to run it yeah.

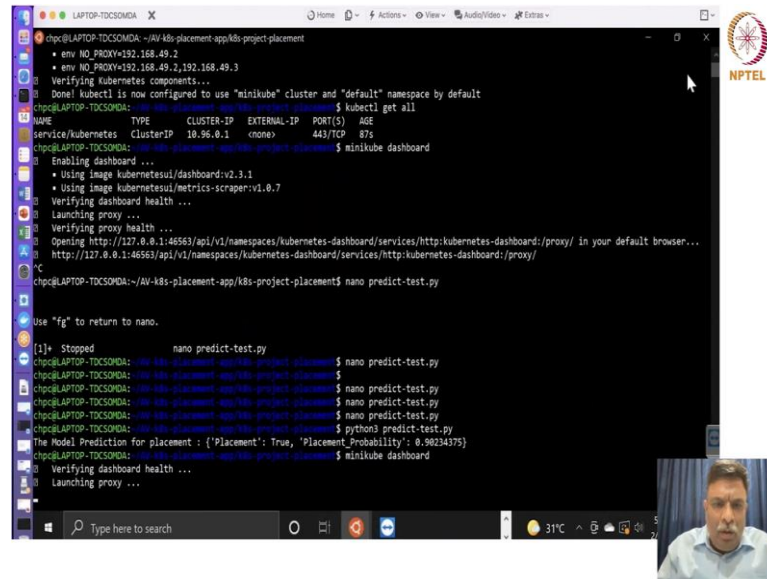
(Refer Slide Time: 22:17)



A terminal window titled 'LAPTOP-TDCSOMDA' showing a Python script named 'predict-test.py'. The script imports requests and time, defines a candidate dictionary, and makes a POST request to a prediction endpoint. The output shows the candidate dictionary and the prediction result.

We will have to put that IP.

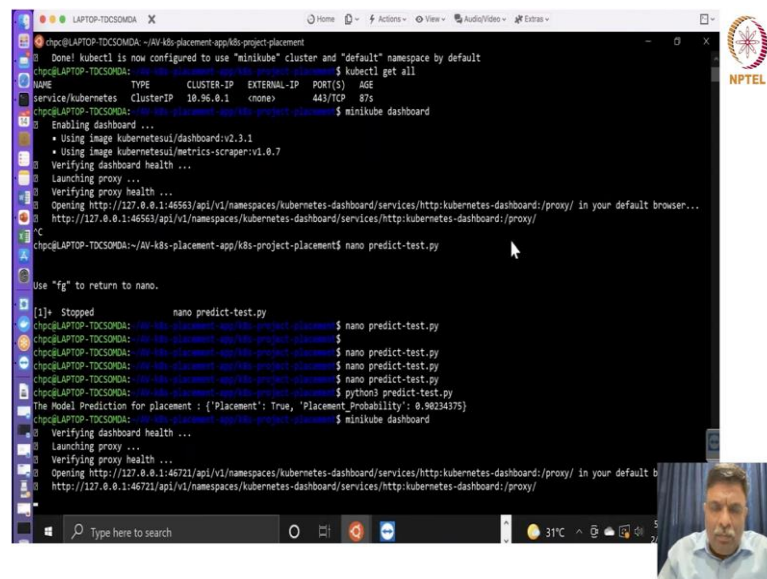
(Refer Slide Time: 23:06)



A terminal window titled 'LAPTOP-TDCSONDA' showing the setup of a Kubernetes cluster. The user runs 'kubectl get all' and receives a table of resources. Then, they run 'minikube dashboard' and are prompted to enable the dashboard. After enabling it, they run 'python3 predict-test.py' and receive a model prediction output. The terminal output is as follows:

```
chpc@LAPTOP-TDCSONDA: ~/AV/k8s-placement-app/k8s-project-placement
• env NO_PROXY=192.168.49.2
• env NO_PROXY=192.168.49.2,192.168.49.3
Verifying Kubernetes components...
Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
chpc@LAPTOP-TDCSONDA: ~/AV/k8s-placement-app/k8s-project-placement$ kubectl get all
NAME                TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes  ClusterIP   10.96.0.1    <none>         443/TCP   87s
chpc@LAPTOP-TDCSONDA: ~/AV/k8s-placement-app/k8s-project-placement$ minikube dashboard
Enabling dashboard ...
• Using image kubernetes/dashboard:v2.3.1
• Using image kubernetes/metrics-scraper:v1.0.7
Verifying dashboard health ...
Launching proxy ...
Verifying proxy health ...
Opening http://127.0.0.1:46563/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/ in your default browser...
http://127.0.0.1:46563/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/
chpc@LAPTOP-TDCSONDA: ~/AV/k8s-placement-app/k8s-project-placement$ nano predict-test.py
Use "fg" to return to nano.
[1] Stopped nano predict-test.py
chpc@LAPTOP-TDCSONDA: ~/AV/k8s-placement-app/k8s-project-placement$ nano predict-test.py
chpc@LAPTOP-TDCSONDA: ~/AV/k8s-placement-app/k8s-project-placement$ nano predict-test.py
chpc@LAPTOP-TDCSONDA: ~/AV/k8s-placement-app/k8s-project-placement$ nano predict-test.py
chpc@LAPTOP-TDCSONDA: ~/AV/k8s-placement-app/k8s-project-placement$ python3 predict-test.py
The Model Prediction for placement : {"Placement": True, "Placement Probability": 0.98234375}
chpc@LAPTOP-TDCSONDA: ~/AV/k8s-placement-app/k8s-project-placement$ minikube dashboard
Verifying dashboard health ...
Launching proxy ...
```

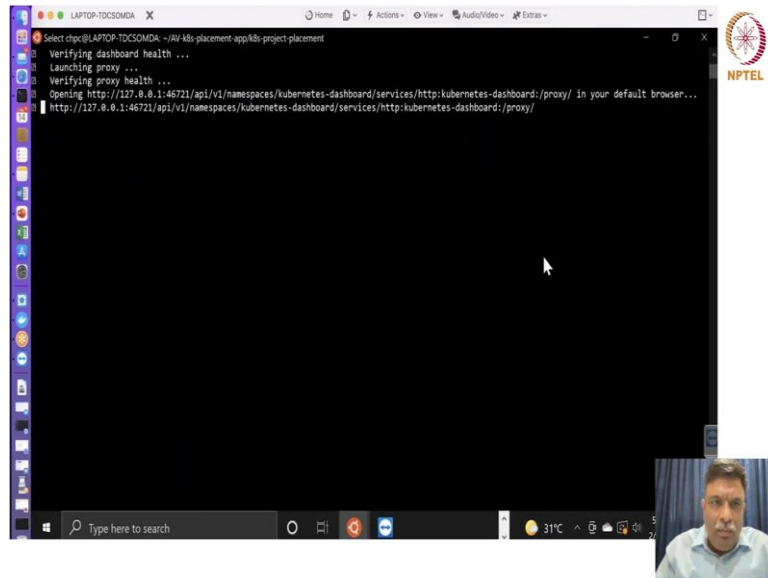
(Refer Slide Time: 23:34)



A terminal window titled 'LAPTOP-TDCSONDA' showing the setup of a Kubernetes cluster. The user runs 'kubectl get all' and receives a table of resources. Then, they run 'minikube dashboard' and are prompted to enable the dashboard. After enabling it, they run 'python3 predict-test.py' and receive a model prediction output. The terminal output is as follows:

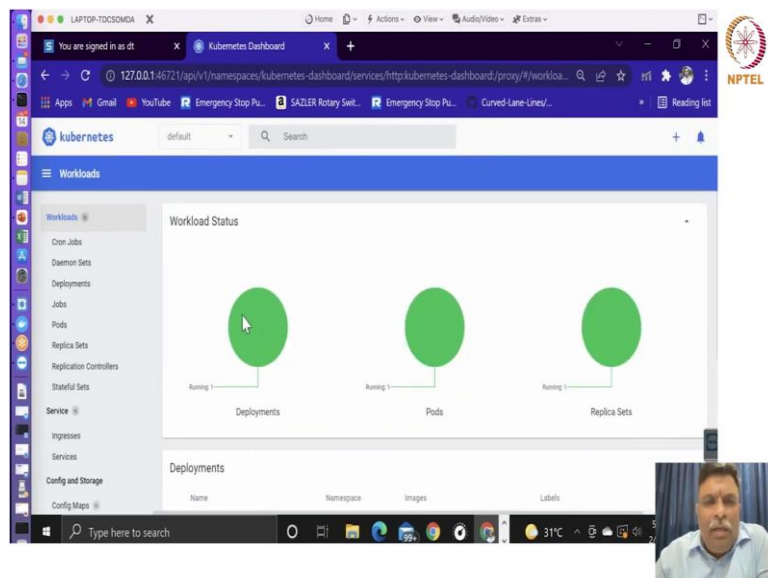
```
chpc@LAPTOP-TDCSONDA: ~/AV/k8s-placement-app/k8s-project-placement
Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
chpc@LAPTOP-TDCSONDA: ~/AV/k8s-placement-app/k8s-project-placement$ kubectl get all
NAME                TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes  ClusterIP   10.96.0.1    <none>         443/TCP   87s
chpc@LAPTOP-TDCSONDA: ~/AV/k8s-placement-app/k8s-project-placement$ minikube dashboard
Enabling dashboard ...
• Using image kubernetes/dashboard:v2.3.1
• Using image kubernetes/metrics-scraper:v1.0.7
Verifying dashboard health ...
Launching proxy ...
Verifying proxy health ...
Opening http://127.0.0.1:46563/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/ in your default browser...
http://127.0.0.1:46563/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/
chpc@LAPTOP-TDCSONDA: ~/AV/k8s-placement-app/k8s-project-placement$ nano predict-test.py
Use "fg" to return to nano.
[1] Stopped nano predict-test.py
chpc@LAPTOP-TDCSONDA: ~/AV/k8s-placement-app/k8s-project-placement$ nano predict-test.py
chpc@LAPTOP-TDCSONDA: ~/AV/k8s-placement-app/k8s-project-placement$ nano predict-test.py
chpc@LAPTOP-TDCSONDA: ~/AV/k8s-placement-app/k8s-project-placement$ nano predict-test.py
chpc@LAPTOP-TDCSONDA: ~/AV/k8s-placement-app/k8s-project-placement$ python3 predict-test.py
The Model Prediction for placement : {"Placement": True, "Placement Probability": 0.98234375}
chpc@LAPTOP-TDCSONDA: ~/AV/k8s-placement-app/k8s-project-placement$ minikube dashboard
Verifying dashboard health ...
Launching proxy ...
Verifying proxy health ...
Opening http://127.0.0.1:46721/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/ in your default b
http://127.0.0.1:46721/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/
```

(Refer Slide Time: 23:35)



Yeah So, now, we will go to the dashboard and see as to the same result right just running on dockers and then running on Kubernetes right as a pod now how will we get it right. So, we will try to see this yeah.

(Refer Slide Time: 23:58)



(Refer Slide Time: 23:59)

The screenshot shows the Kubernetes Dashboard interface. The 'Workloads' section is active, displaying a table of workloads. The workload 'placement-app-594b74f668-d9rmd' is selected, and its details are shown in the 'Replica Sets' section. The dashboard includes a sidebar with navigation options like Cron Jobs, Daemon Sets, Deployments, Jobs, Pods, Replica Sets, Replication Controllers, and Stateful Sets. The main content area displays a table of workloads and a detailed view of the selected workload's replica sets.

Name	Namespace	Images	Labels	Node
placement-app-594b74f668-d9rmd	default	subbu0319/placement-app:latest	app: placement-app pod-template-hash: 594b74f668	minikube-m03

Name	Namespace	Images	Labels
placement-app-594b74f668	default	subbu0319/placement-app:latest	

So, see here if you see what happened if you see the workload status there is one deployment, there is one pod, there is one replica set and then if you see what are those right, the deployments this is the placement app.

(Refer Slide Time: 24:19)

The screenshot shows the Kubernetes Dashboard interface. The 'Workloads' section is active, displaying a table of workloads. The workload 'placement-app' is selected, and its details are shown in the 'Pods' section. The dashboard includes a sidebar with navigation options like Cron Jobs, Daemon Sets, Deployments, Jobs, Pods, Replica Sets, Replication Controllers, and Stateful Sets. The main content area displays a table of workloads and a detailed view of the selected workload's pods.

Name	Namespace	Images	Labels	Pods
placement-app	default	subbu0319/placement-app:latest		1/1

Name	Namespace	Images	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)
placement-app-594b74f668-d9rmd	default	subbu0319/placement-app:latest	app: placement-app pod-template-hash: 594b74f668	minikube-m03	Running	0		

Name	Namespace	Images	Labels	Pods
placement-app-594b74f668	default	subbu0319/placement-app:latest	app: placement-app pod-template-hash: 594b74f668	

(Refer Slide Time: 24:22)

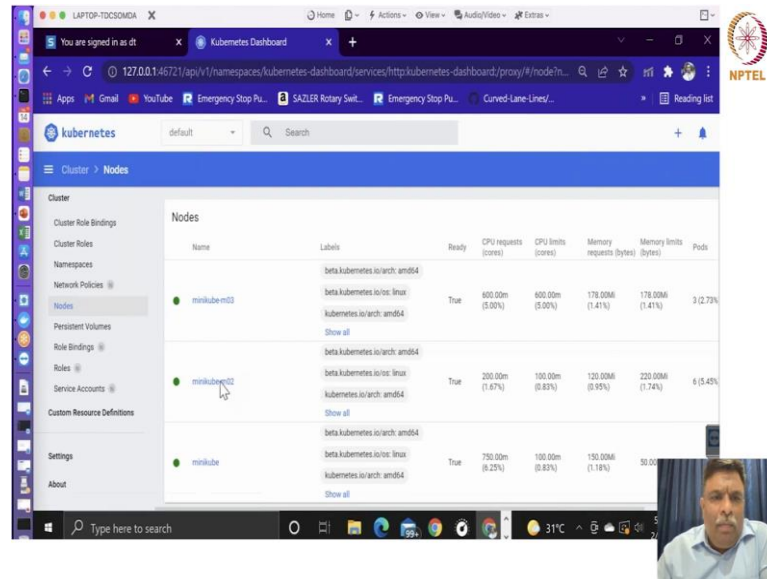
The screenshot shows the Kubernetes Dashboard for the 'placement-app' deployment. The 'Rolling update strategy' section indicates a 'Min surge' of 25% and 'Min unavailable' of 25%. The 'Pods status' table shows 1 updated, 1 total, and 1 available pod. The 'Conditions' table shows 'Progressing' as 'True' and 'Available' as 'True'. A video feed of a man is visible in the bottom right corner.

(Refer Slide Time: 24:24)

The screenshot shows the Kubernetes Dashboard for the 'placement-app' deployment. The 'Name' field shows 'placement-app-594b74f668'. The 'Namespace' is 'default'. The 'Age' is '8 minutes ago'. The 'Pods' column shows '1 / 1'. The 'Labels' section shows 'app: placement-app' and 'pod-template-hash: 594b74f668'. The 'Images' section shows 'subbu0319/placement-app:latest'. The 'Old Replica Sets' section shows 'There is nothing to display here'. The 'Horizontal Pod Autoscalers' section shows 'Items: 0'. The 'Events' section is empty. A video feed of a man is visible in the bottom right corner.

What is the status, when did it start right? So, what type of label it uses? Which is the image you are trying to run on it right? All these details you get.

(Refer Slide Time: 24:41)

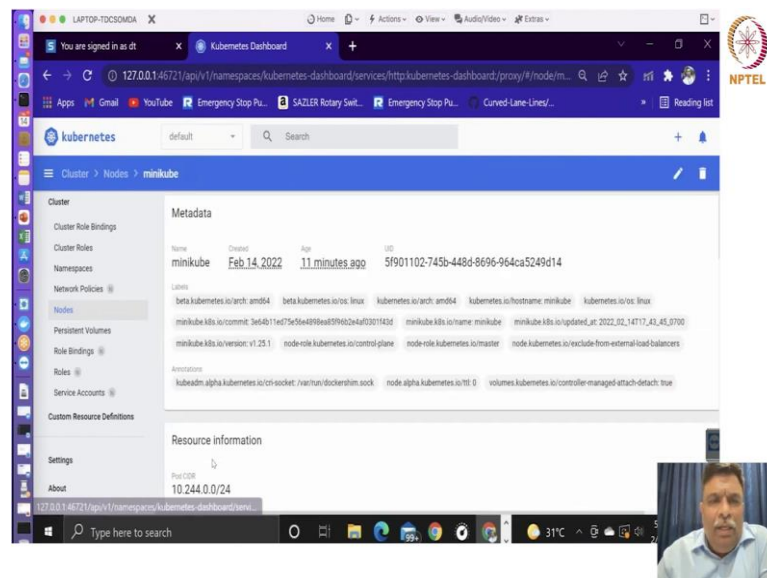


The screenshot shows the Kubernetes Dashboard interface. The left sidebar contains navigation links for Cluster, Nodes, Namespaces, Network Policies, Persistent Volumes, Role Bindings, Roles, Service Accounts, Custom Resource Definitions, Settings, and About. The main content area displays the 'Nodes' page, which lists three nodes: minikube02, minikube01, and minikube. Each node entry includes its name, labels, status (Ready), and resource usage (CPU requests/limits, memory requests/limits, and pods). The minikube node is highlighted as the master node.

Name	Labels	Ready	CPU requests (cores)	CPU limits (cores)	Memory requests (bytes)	Memory limits (bytes)	Pods
minikube02	beta.kubernetes.io/arch: amd64 beta.kubernetes.io/os: linux kubernetes.io/arch: amd64	True	600.00m (5.00%)	600.00m (5.00%)	178.00Mi (1.41%)	178.00Mi (1.41%)	3 (2.73%)
minikube01	beta.kubernetes.io/arch: amd64 beta.kubernetes.io/os: linux kubernetes.io/arch: amd64	True	200.00m (1.67%)	100.00m (0.83%)	120.00Mi (0.95%)	220.00Mi (1.74%)	4 (5.45%)
minikube	beta.kubernetes.io/arch: amd64 beta.kubernetes.io/os: linux kubernetes.io/arch: amd64	True	750.00m (6.25%)	100.00m (0.83%)	150.00Mi (1.18%)	50.00Mi (0.39%)	0

Similarly, if you see the nodes right. So, yesterday we saw that there were three nodes right; the one is the master there are two worker nodes.

(Refer Slide Time: 24:52)



The screenshot shows the Kubernetes Dashboard interface, specifically the 'Metadata' page for the minikube node. The page displays the node's name, creation time, age, and UID. It also lists the node's labels and annotations.

Name	Created	Age	UID
minikube	Feb 14, 2022	11 minutes ago	5f901102-745b-448d-8696-964ca5249d14

Labels:

- beta.kubernetes.io/arch: amd64
- beta.kubernetes.io/os: linux
- kubernetes.io/arch: amd64
- kubernetes.io/hostname: minikube
- kubernetes.io/os: linux

Annotations:

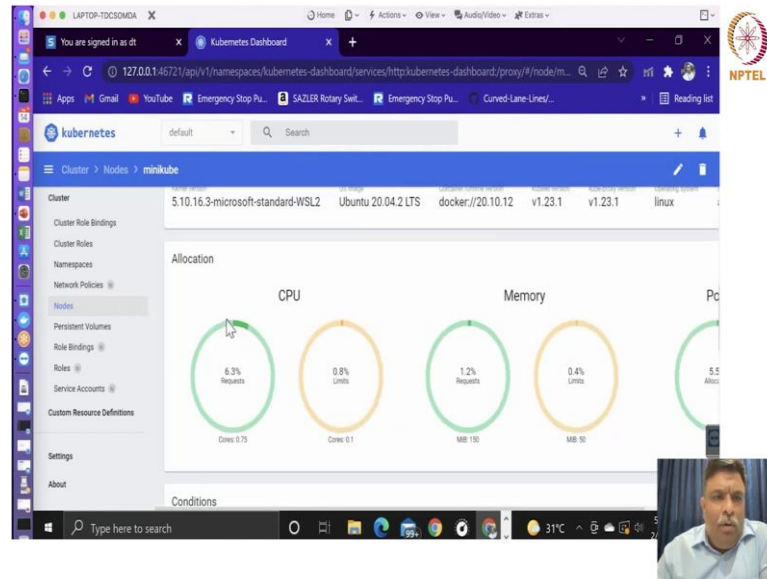
- minikube.k8s.io/commit: 3e54b11e775e54a895e859f62e4f0301143d
- minikube.k8s.io/name: minikube
- minikube.k8s.io/updated_at: 2022-02-14T17:43:45.0700
- minikube.k8s.io/version: v1.25.1
- node-role.kubernetes.io/control-plane: node-role.kubernetes.io/master
- node.kubernetes.io/exclude-from-external-load-balancers

Resource information:

- Pod CIDR: 10.244.0.0/24

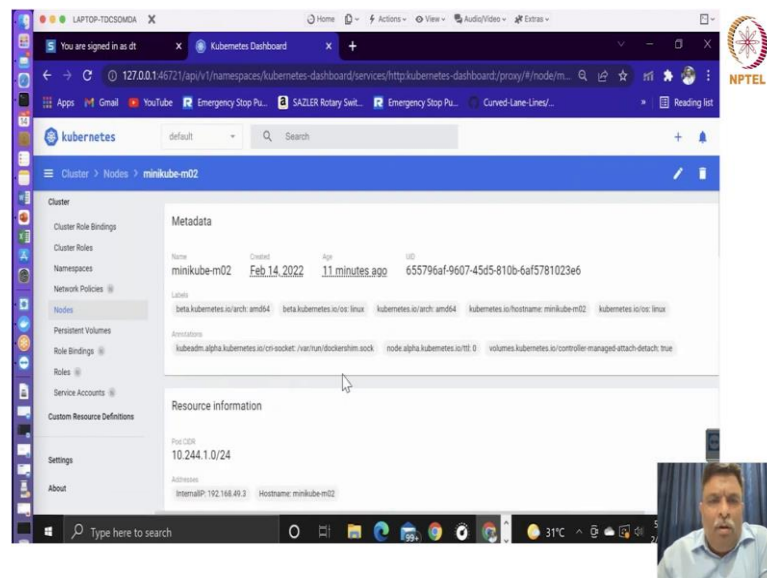
So, if you see now this minikube which is the master node.

(Refer Slide Time: 24:56)



You have got details about this right the CPU the memory and how many pods it is running. Now if you go to this worker node ok.

(Refer Slide Time: 25:12)



Minikube m 02 again it has got two CPUs right so many pods running ok.

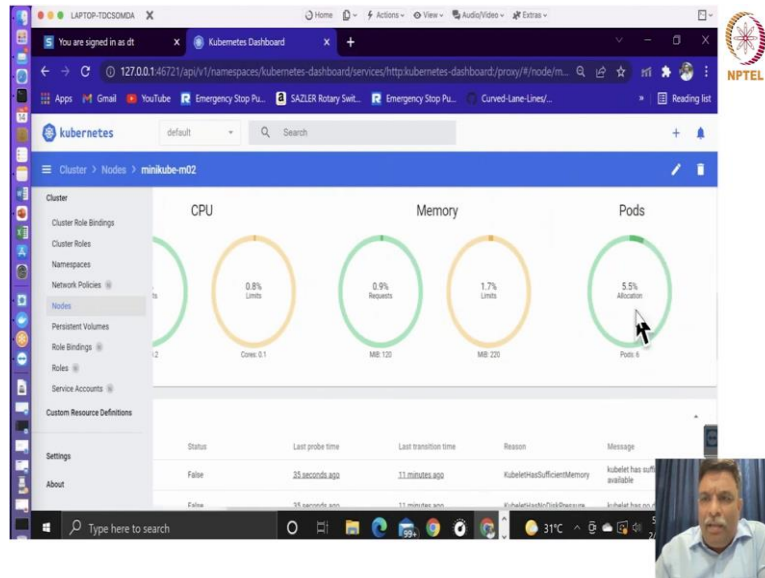
(Refer Slide Time: 25:16)

The screenshot shows the Kubernetes Dashboard interface. The main content area displays a table of pods for the 'minikube-m02' cluster. The pods listed are 'dashboard-metrics-scraper' and 'kubernetes-dashboard', both in a 'Running' state. The dashboard also shows a sidebar with navigation options and a top bar with the cluster name and status.

(Refer Slide Time: 25:18)

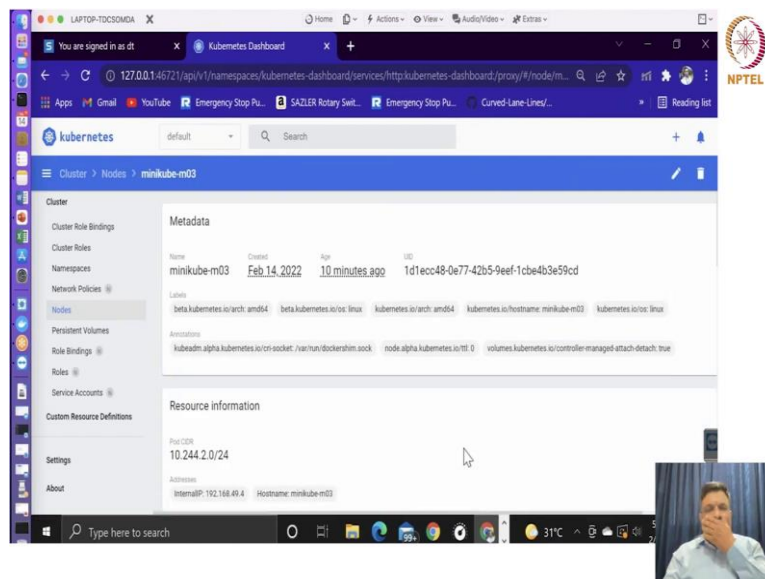
The screenshot shows the Kubernetes Dashboard interface. The main content area displays a table of conditions for the 'minikube-m02' cluster. The conditions listed are 'MemoryPressure' and 'PodSandbox', both in a 'False' state. The dashboard also shows a sidebar with navigation options and a top bar with the cluster name and status.

(Refer Slide Time: 25:21)



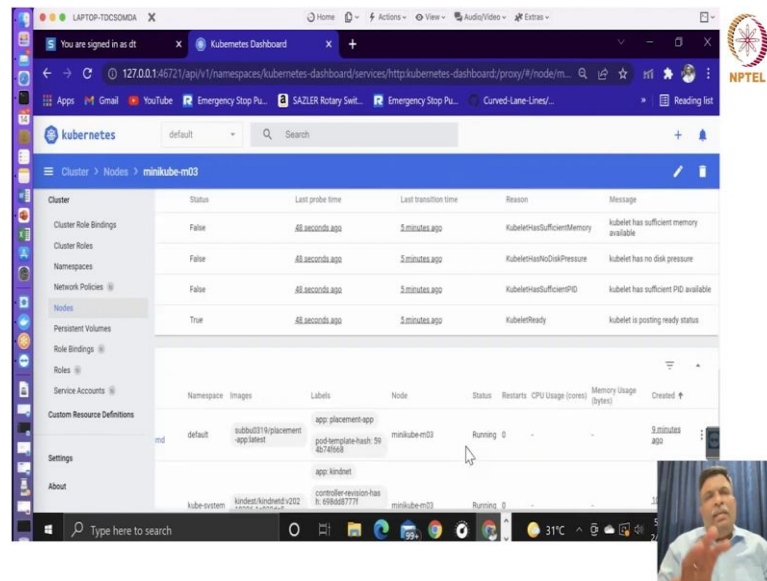
So, much memory allocated and stuffs like that.

(Refer Slide Time: 25:30)



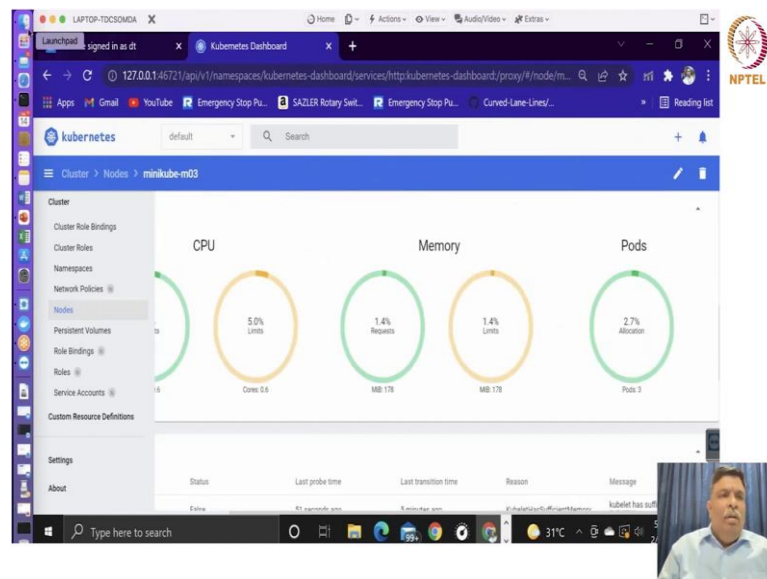
And then if you see this minikube m3 it again has this ok.

(Refer Slide Time: 25:36)



So, this is the Kubernetes dashboard.

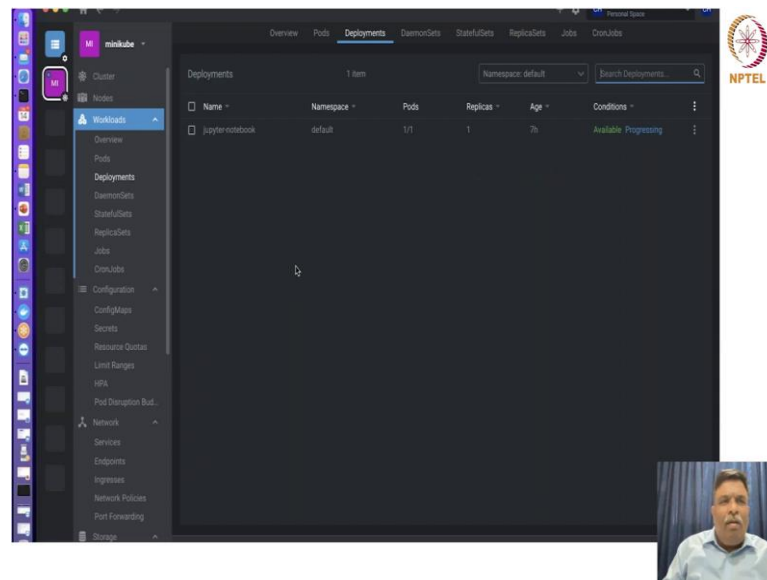
(Refer Slide Time: 25:38)



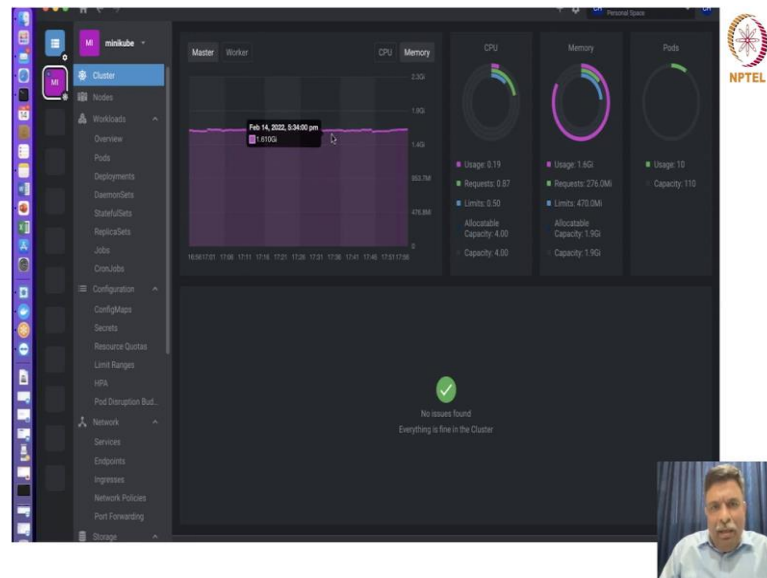
Now, before we end today's lecture maybe I will take 5 to 10 minutes more I will show you another very good GUI right. So, even if you are not understood it totally at least you should be in a position to understand that there is a node on that node you are running the pod that pod runs a docker it can be any application. So, for the time being it is a machine learning application which we saw.

Now, when you see the dashboard it is not that user friendly in a sense that you will have to scroll you will have to see everything. Now I am going to show you another way of seeing the same thing right which is one of the very very newest and the latest things which is called as lens.

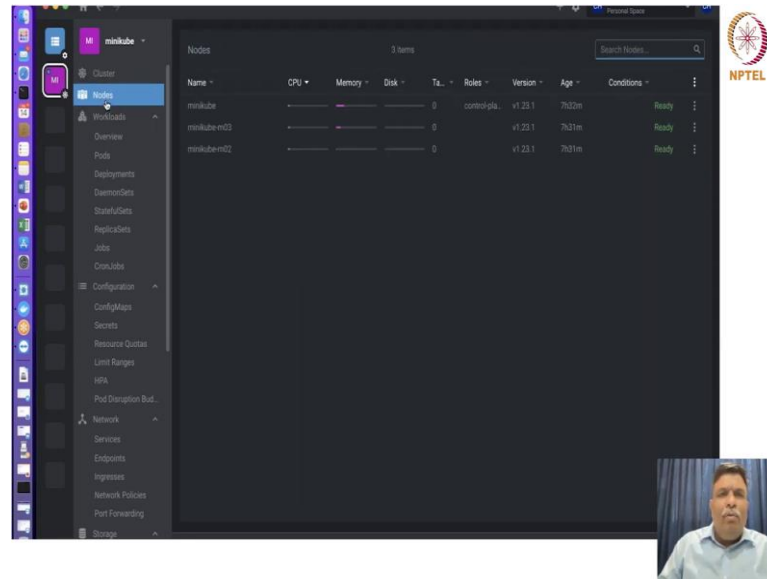
(Refer Slide Time: 26:25)



(Refer Slide Time: 26:33)

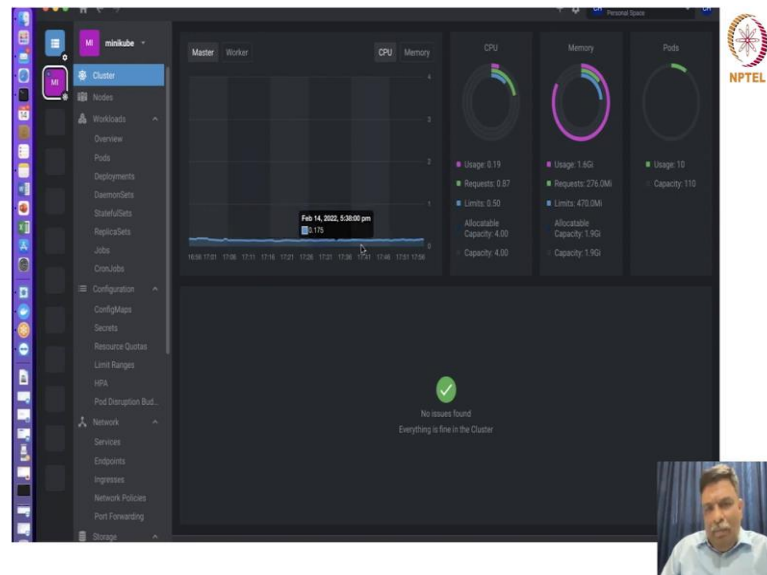


(Refer Slide Time: 26:33)



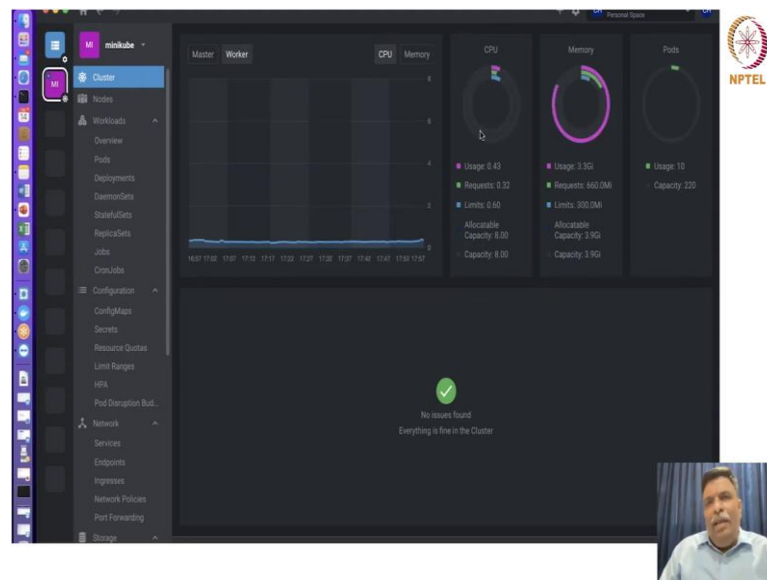
Now this helps me to actually see my cluster information in a very very user friendly way. For example, it shows me the master the master what is the CPU utilization, how much memory is being utilized, how many pods are running right and how much of the memory in real time actually how is it being utilized.

(Refer Slide Time: 27:01)



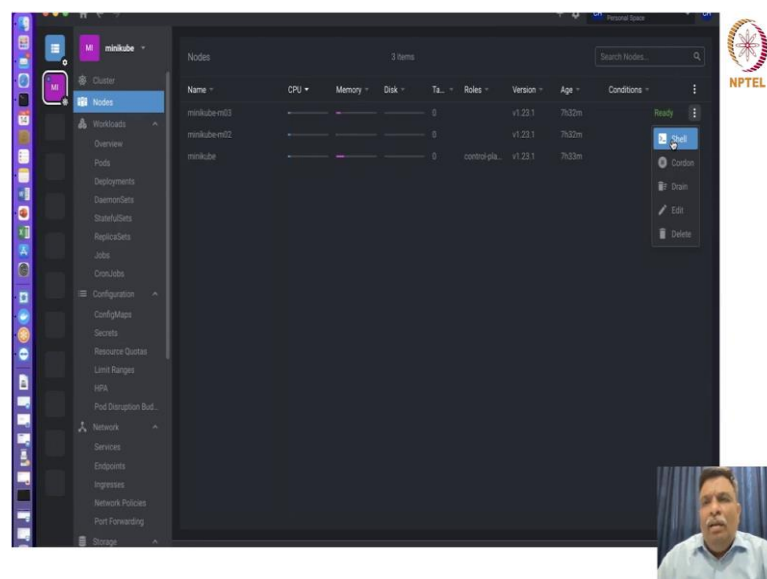
CPU performance or how is CPU actually being used right. So, usage wise on the master.

(Refer Slide Time: 27:10)



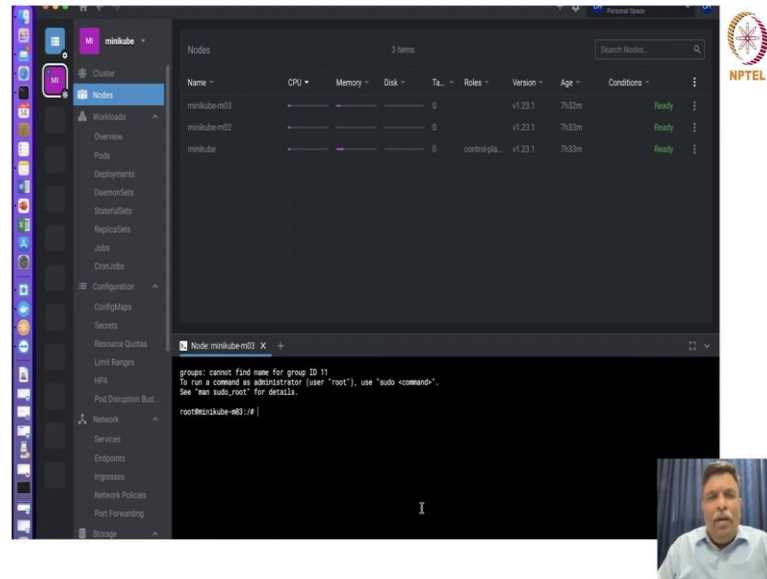
If you go to worker again you get the information about how what is the utilization of the CPU, how much memory is being utilized, how many pods are running right. So, if you see this is a better way of trying to understand it in a better way right.

(Refer Slide Time: 27:31)



Here also you can see three nodes; one is the master node, one is the worker node, another one is the other worker node. And if you see this you can work at the shell prompt of that particular node right.

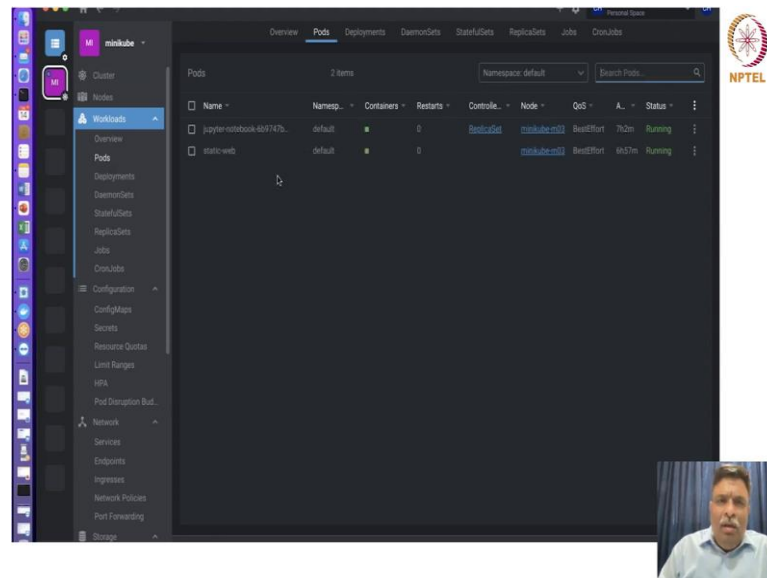
(Refer Slide Time: 27:49)



The screenshot shows the Kubernetes dashboard interface. The 'Nodes' tab is selected in the left sidebar. The main panel displays a table of nodes with columns: Name, CPU, Memory, Disk, Taints, Roles, Version, Age, and Conditions. The table lists three nodes: minikube-m03, minikube-m02, and minikube. The 'minikube-m03' node is highlighted. Below the table, a terminal window for the selected node shows the command 'root@minikube-m03: /# |'.

So, let us try to understand this how it will be useful ok in your days to come when you are trying to use it right. So, you have actually entered the node minikube 3 now right. I am not using it at present, but I am just trying to tell you the facilities available in this. Now you see the jobs ok.

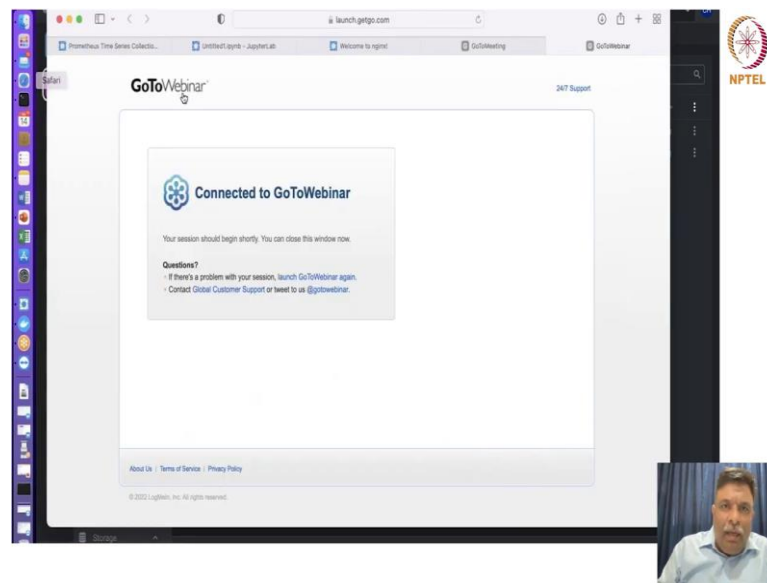
(Refer Slide Time: 28:18)



The screenshot shows the Kubernetes dashboard interface. The 'Pods' tab is selected in the left sidebar. The main panel displays a table of pods with columns: Name, Namespace, Containers, Restarts, Controller, Node, QoS, Age, and Status. The table lists two pods: jupyter-notebook-4671475 and static-web. The 'static-web' pod is highlighted. Below the table, a terminal window for the selected pod shows the command 'root@minikube-m03: /# |'.

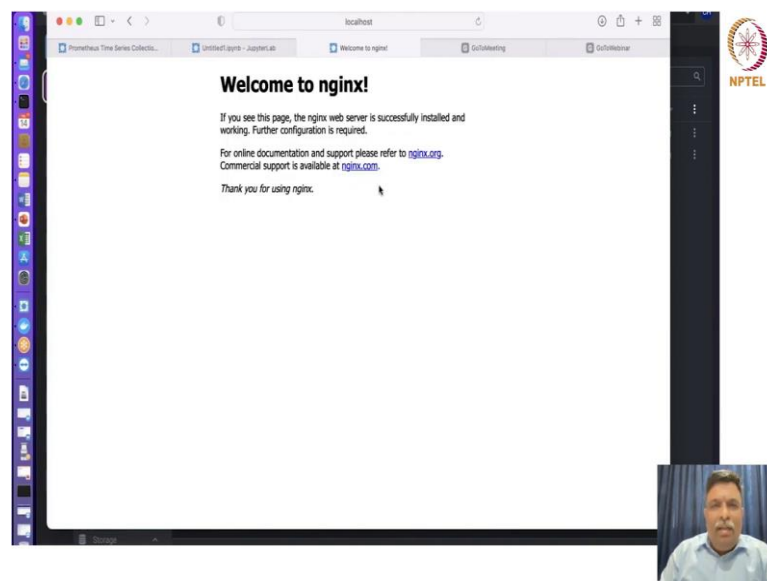
Let us see the pods we have got two pods running I will tell you what are those pods actually ok. So, one is a static web which is this nginx thing which we saw in the previous this thing. So, if you see the nginx thing.

(Refer Slide Time: 28:40)



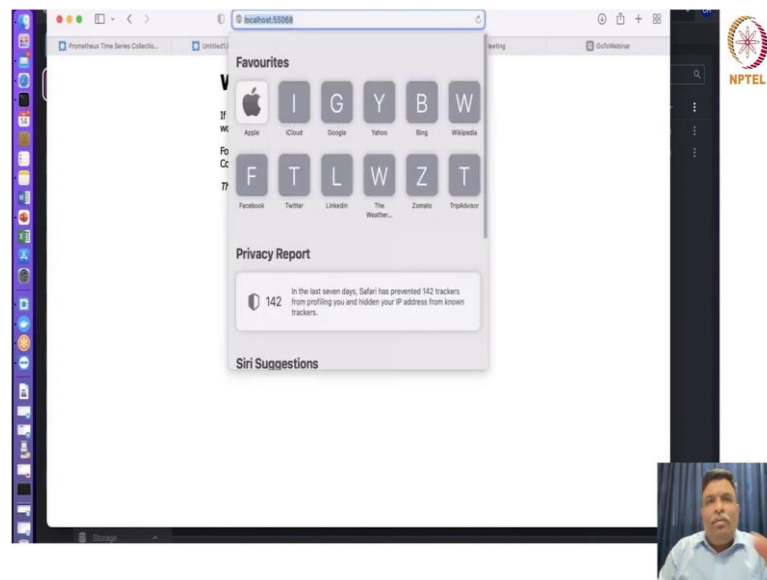
Let me just show you that yeah.

(Refer Slide Time: 28:41)



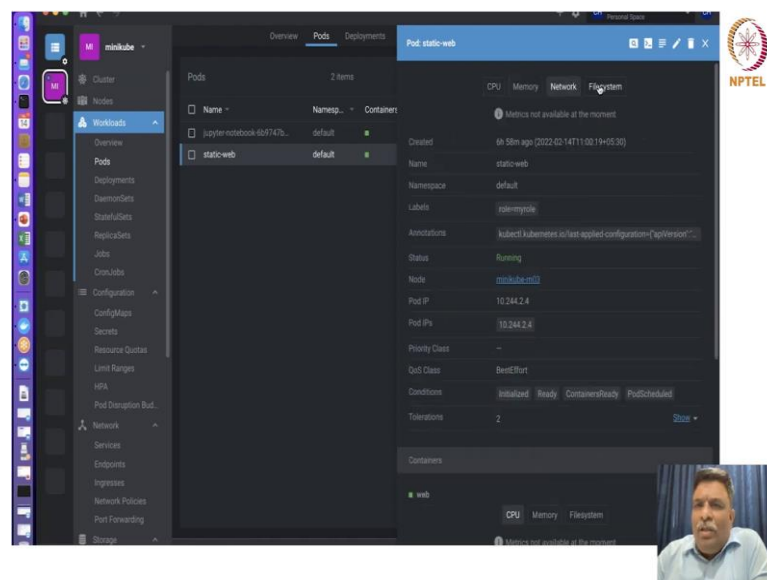
So, see this is running on that particular pod right. So, I will tell you what; that means, ok.

(Refer Slide Time: 28:50)



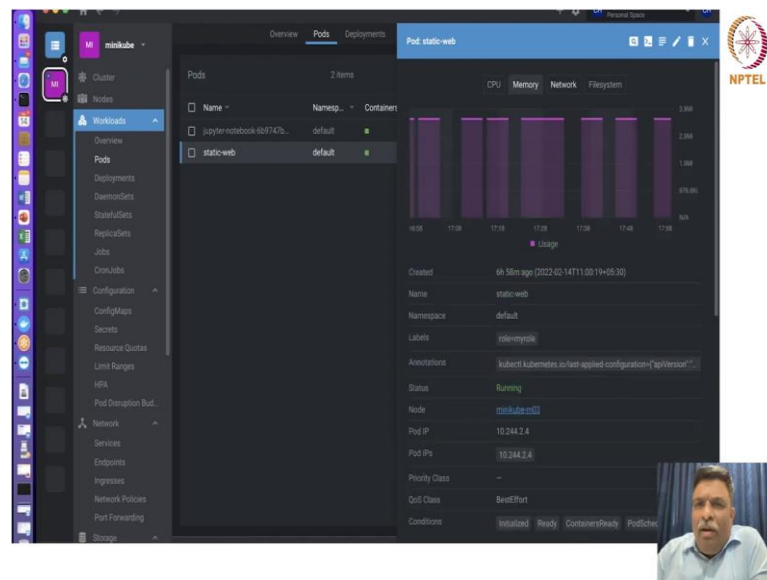
So, this is actually getting serviced on the local host it is a web server installed ok and it is running in this pod this is this pod ok this particular pod.

(Refer Slide Time: 29:00)



If you see this how much CPU utilization, what is the network it uses.

(Refer Slide Time: 29:08)



The screenshot shows the Kubernetes dashboard interface. On the left, the 'minikube' cluster is selected. The 'Pods' tab is active in the left sidebar. The main panel displays the 'Pod: static-web' details. The 'Overview' tab is selected, showing a table of pods and a detailed view of the 'static-web' pod. The pod is in a 'Running' state, created 6h 58m ago, and is running on the 'minikube-m02' node. The 'Containers' section shows a single container named 'web'.

Name	Namespace	Container
static-web	default	web

Pod: static-web

Created: 6h 58m ago (2022-02-14T11:00:19+05:30)

Name: static-web

Namespace: default

Labels: kube-system

Annotations: kubernet.kubernetes.io/last-applied-configuration: {"apiVersion": "v1", "kind": "Pod", "metadata": {"name": "static-web", "namespace": "default"}, "spec": {"containers": [{"name": "web", "image": "nginx"}]}}

Status: Running

Node: minikube-m02

Pod IP: 10.244.2.4

Pod IPs: 10.244.2.4

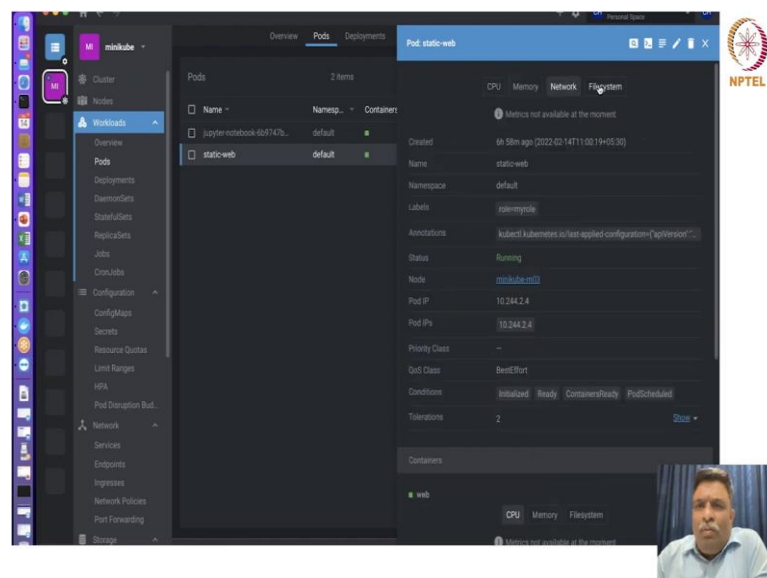
Priority Class: --

QoS Class: BestEffort

Conditions: Initialized Ready ContainersReady PodScheduled

Containers: web

(Refer Slide Time: 29:08)



The screenshot shows the Kubernetes dashboard interface. On the left, the 'minikube' cluster is selected. The 'Pods' tab is active in the left sidebar. The main panel displays the 'Pod: static-web' details. The 'Overview' tab is selected, showing a table of pods and a detailed view of the 'static-web' pod. The pod is in a 'Running' state, created 6h 58m ago, and is running on the 'minikube-m02' node. The 'Containers' section shows a single container named 'web'.

Name	Namespace	Container
static-web	default	web

Pod: static-web

Created: 6h 58m ago (2022-02-14T11:00:19+05:30)

Name: static-web

Namespace: default

Labels: kube-system

Annotations: kubernet.kubernetes.io/last-applied-configuration: {"apiVersion": "v1", "kind": "Pod", "metadata": {"name": "static-web", "namespace": "default"}, "spec": {"containers": [{"name": "web", "image": "nginx"}]}}

Status: Running

Node: minikube-m02

Pod IP: 10.244.2.4

Pod IPs: 10.244.2.4

Priority Class: --

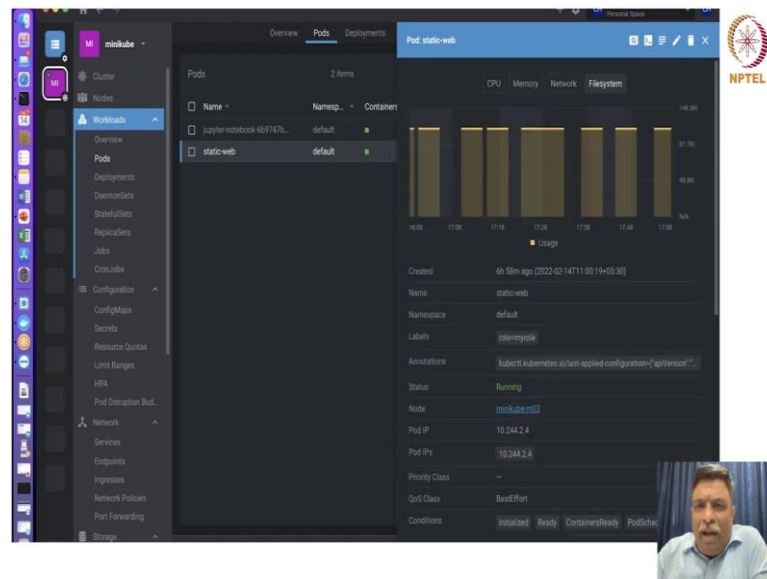
QoS Class: BestEffort

Conditions: Initialized Ready ContainersReady PodScheduled

Tolerations: 2

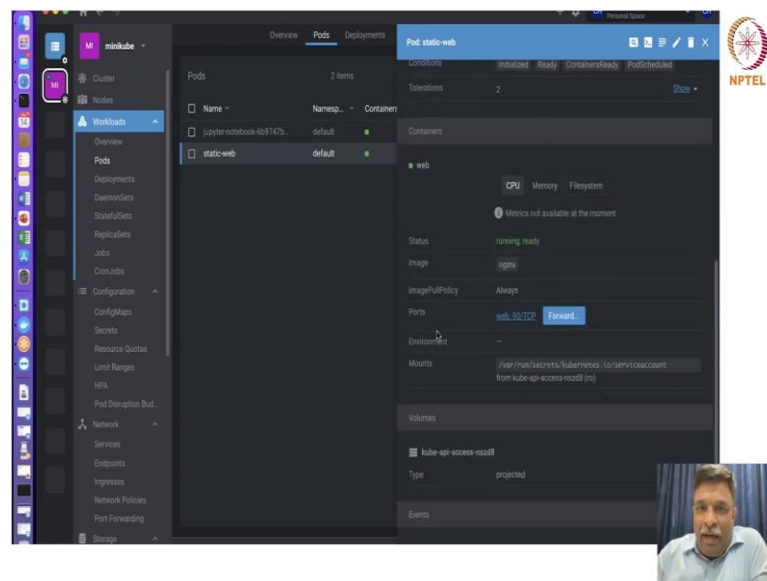
Containers: web

(Refer Slide Time: 29:09)



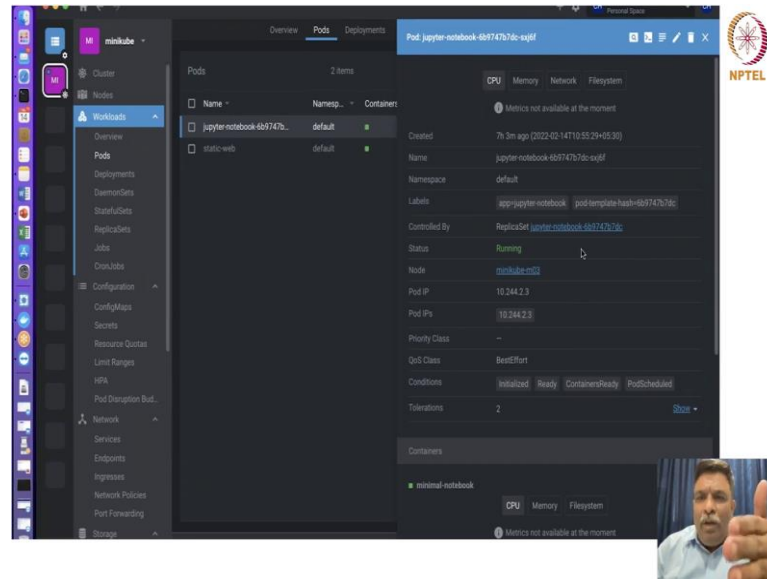
What is the file system ok. Where, what are the pods on which you can view it what is the status right.

(Refer Slide Time: 29:11)

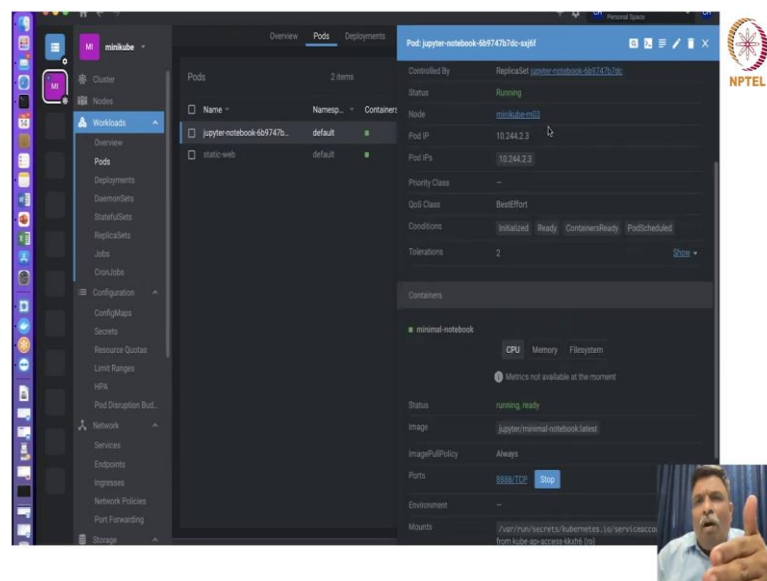


All of this is available to you ok. Similarly this is Jupyter Notebook.

(Refer Slide Time: 29:27)

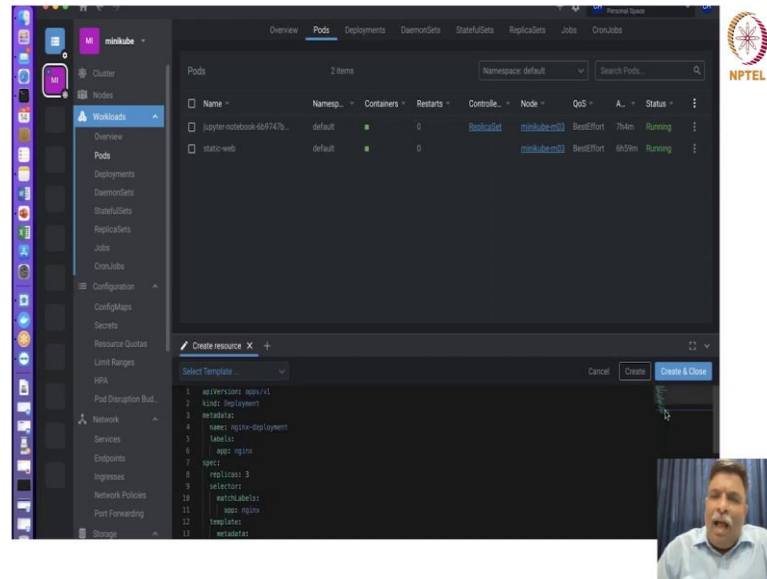


(Refer Slide Time: 29:28)



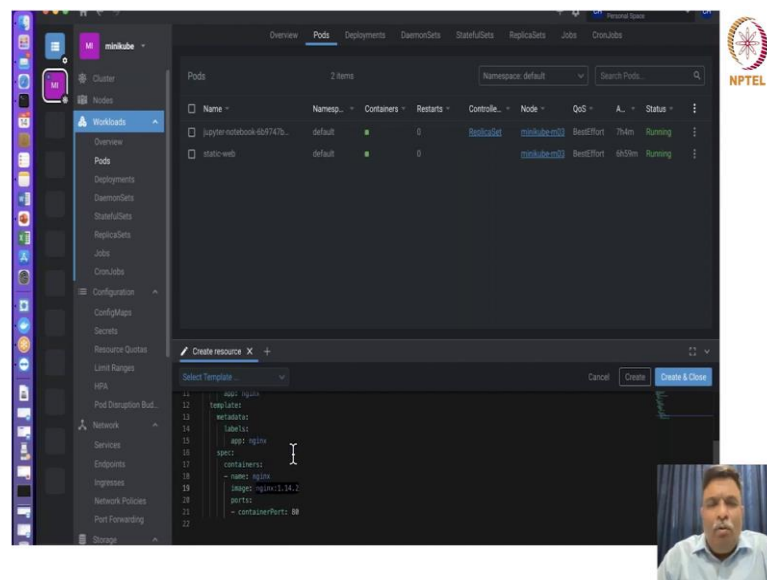
So, this Jupyter Notebook is running on node 3 whereas, this is also running on node 3 it is its not necessary that it has to run on node 2. It has actually allocated right based on some decision which it makes ok. This is also running on node 3. So, if you see this memory file system the CPU ok how its running ready at what pod. So, you will see that also. But before that let me just tell you if you want to create another pod right.

(Refer Slide Time: 30:04)



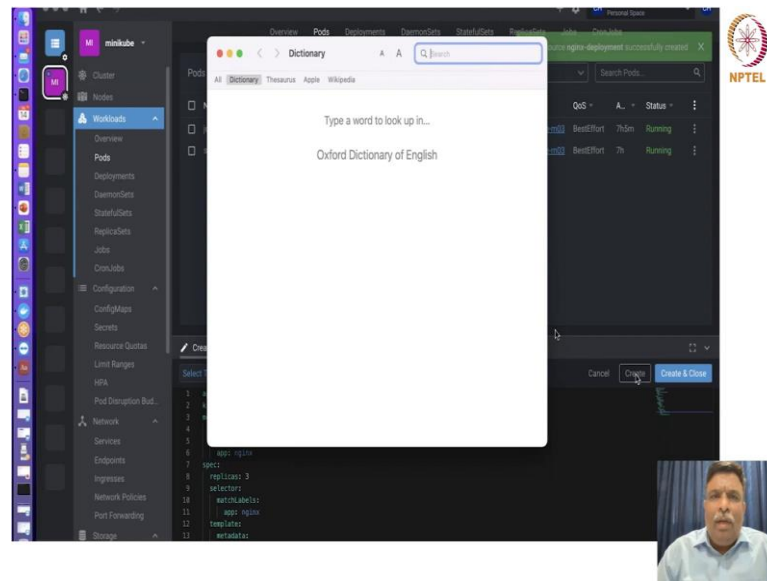
You have this resource right creating a pod and then you can use deployment, you can use something. So, I am just telling to showing to I am just showing you this that this is the API version right YAML file this what type of deployment name I have given.

(Refer Slide Time: 30:24)



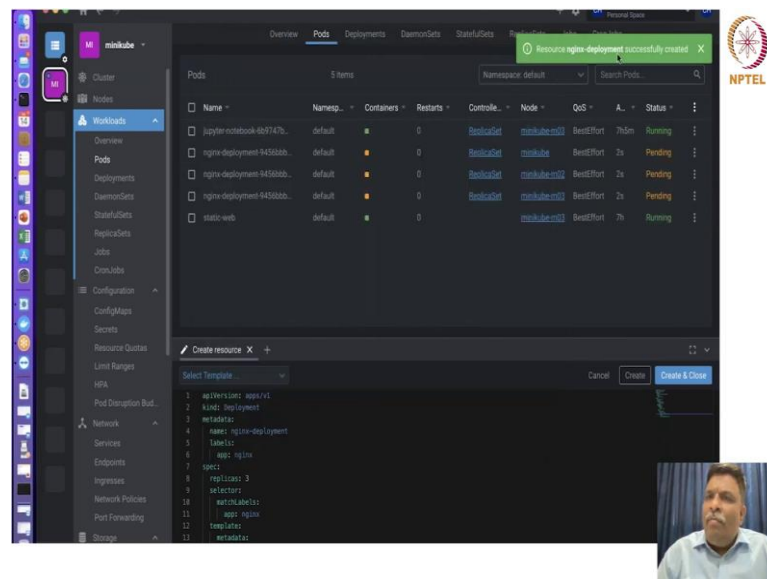
Which is the image it is running see here. So, this is a docker image right. So, I can actually when I am trying to run the Jupyter Notebook ok I will put the docker image which actually has a Jupyter Notebook and then I actually create it.

(Refer Slide Time: 30:46)



I can create another nginx now ok. I do not know why it is coming.

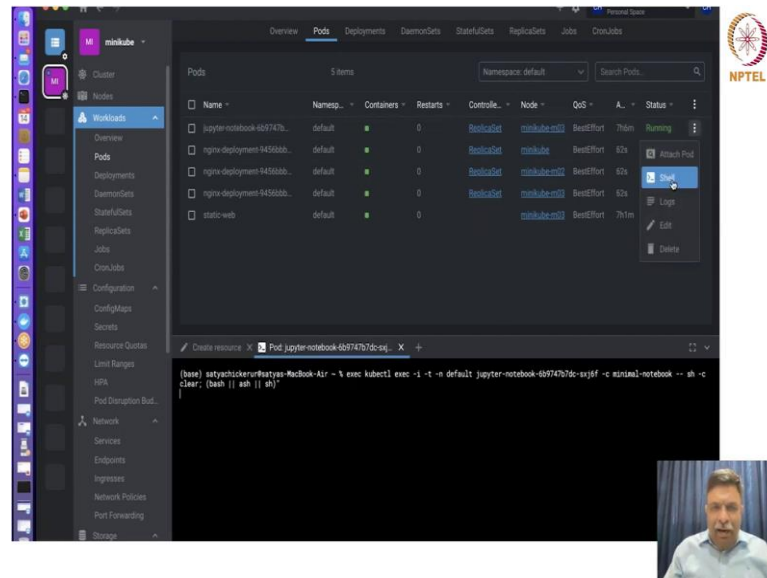
(Refer Slide Time: 30:49)



See resource nginx development deployment successfully created. So, see here now I suppose by mistake I clicked it three times maybe I do not know. So, it has created three instances right and its creating a replica set also of it and its pending you will wait for some two three minutes then we will see that it will actually all the three of them will be running. And if you see here this minikube sorry this particular deployment is running on node 2 and this is running on node 1.

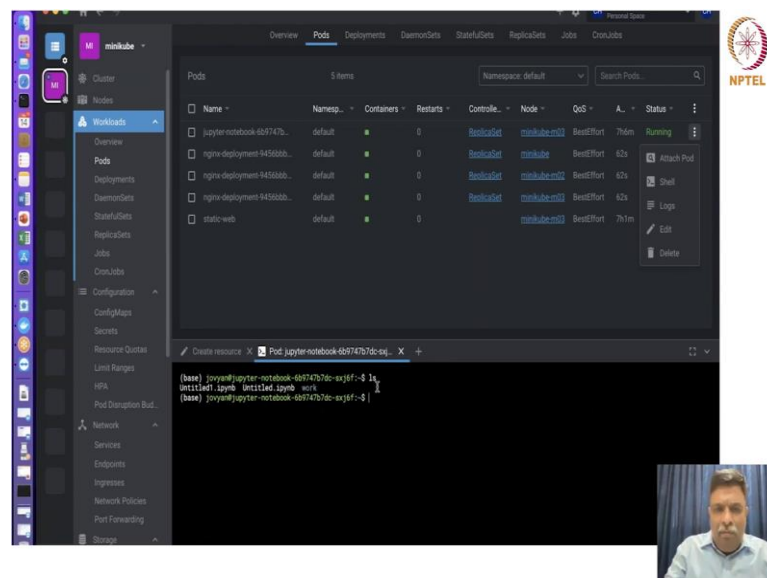
Now, ok or the master node ok and this was running on node 3. So, this also is now trying to run on node 3 and if you see all of them have the status now running. So, you can actually see that Jupyter Notebook is running.

(Refer Slide Time: 31:47)



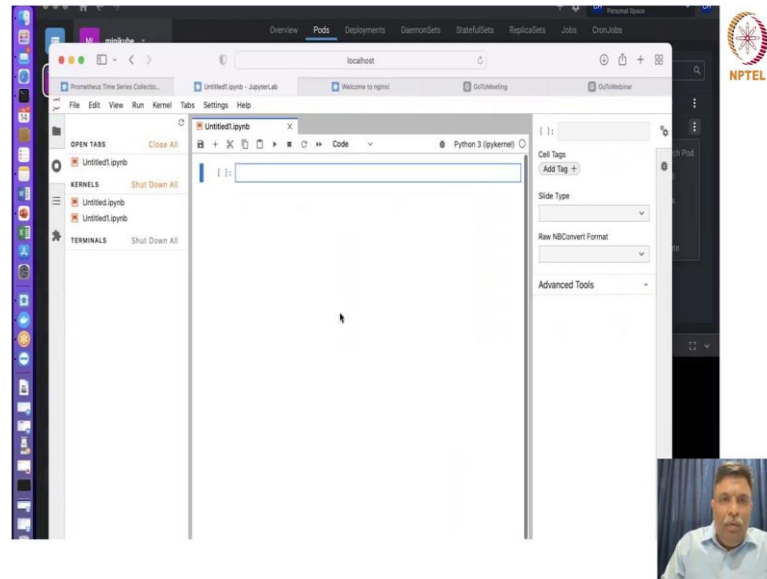
You can go to the shell.

(Refer Slide Time: 31:50)

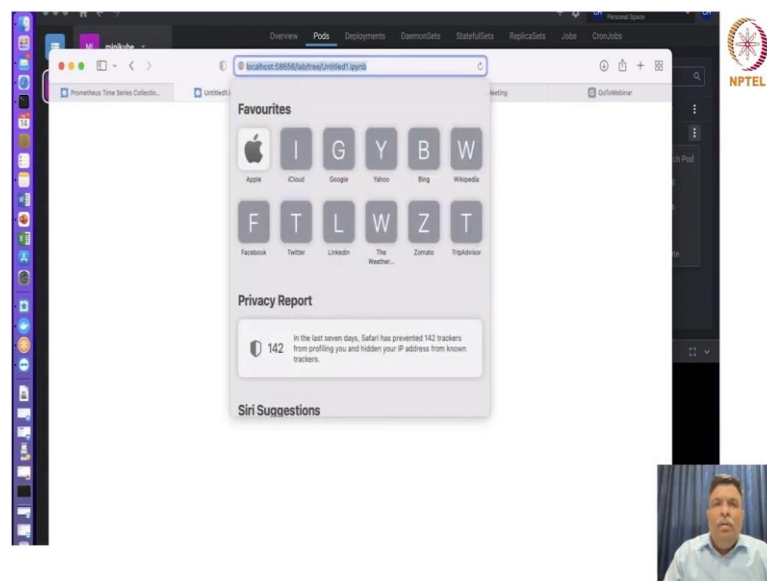


So, you can go to the root of that particular container you have got all of this files and folders there and if you see this Jupyter Notebook it is here its running its running on local host ok with this.

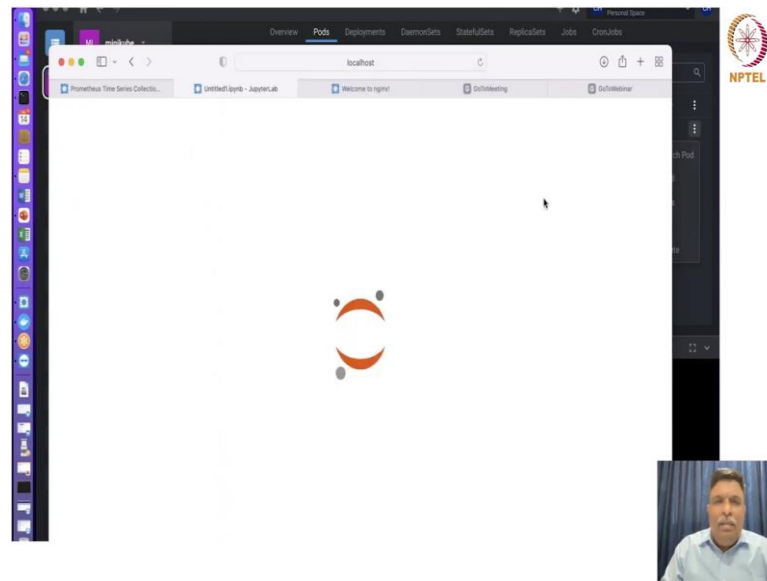
(Refer Slide Time: 32:05)



(Refer Slide Time: 32:09)

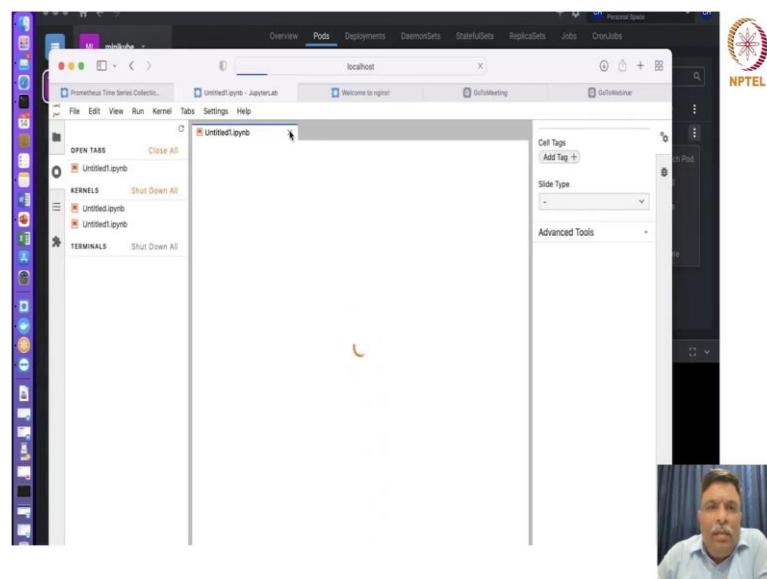


(Refer Slide Time: 32:13)



So, let me just try to connect it, yes.

(Refer Slide Time: 32:21)

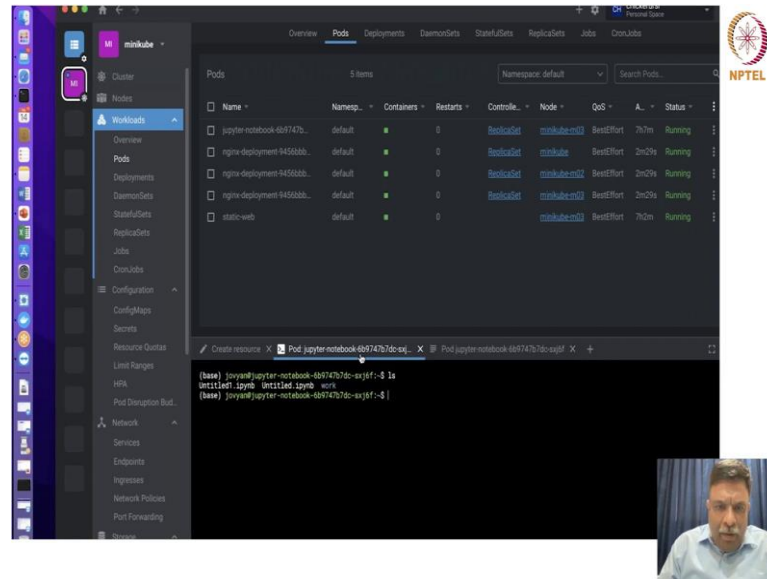


So, it will open its initializing. So, this is how actually it is going to happen. So, my Jupyter Notebook is running on node three you can attach a pod to a different this thing you can shift it you can see the logs.

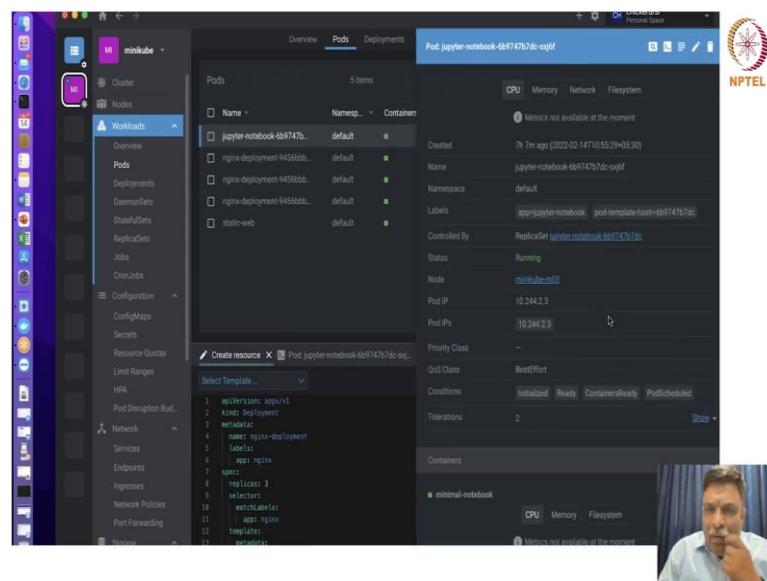


own.

(Refer Slide Time: 33:17)



(Refer Slide Time: 33:17)



Resource ok by a template and if you want to see how we did it. So, you can see the image which we pulled we pulled the Jupyter Notebook minimal to latest from that ok that is how actually it has been done. So, I hope this is a bit of idea to you. Because from tomorrow onwards we will be developing your own applications right.

Deep learning and machine learning applications using pytorch using Tensorflow and all. So, ultimately once you develop those type of applications using dockers then you can

actually put into the pods and do the things very easily you can actually do such use such templates and do things very easily ok. So, that is it.

Thank you so much.