National College of Ireland

# Configuration Manual

MSc Research Project

Cloud Computing (MSCCLOUD1_A)

Majid Shahbaz

Student ID: x23332085

School of Computing

National College of Ireland

Supervisor: Yasantha Samarawickrama

# A Proactive Hardware-Aware Pod Migration Approach for Real-Time Applications in Edge Clouds

Configuration Manual

Majid Shahbaz

## 1. Introduction

This guide outlines the step-by-step procedure to set up the experimental environment for proactive, hardware-aware pod migration based on DRL-PPO.

**Scope:**

Includes hardware installation, software installation, Kubernetes configuration, monitoring stack, deployment of DRL agent, and evaluation process.

## 2. Requirements for Hardware and Infrastructure

- **Local Environment:**
  - CPU must have 4 or more cores
  - RAM should be more than 8GB
  - OS: macOS, Linux, Windows or WSL2
  - Docker on local machine

- **Production Environment (Cloud):**

  Create a yaml file with all the required configurations and then apply it on cluster.

  (Cluster config file is present in GitHub repository)

  - AWS Elastic Kubernetes Service Cluster:
    - `apiVersion: eksctl.io/v1alpha5`
    - `kind: ClusterConfig`
    - `Node: t3.medium or t3.larger`
    - `name: standard-workers`
    - `amiFamily: AmazonLinux2023`

If using aws learning lab then mention Role in the cluster config:

```
iam:
  serviceRoleARN: arn:aws:iam::…….
```

- Number of Nodes: 2-3 for testbed

- AWS Services: EC2 instances, Elastic Container Registry (ECR), S3, CloudWatch
- Docker

# 3. Software and Tools

- **Kubernetes Tools**
  - Minikube for local testing
  - kubectl for cluster management
  - eksctl for eks provisioning
- **AWS CLI** for AWS resource and account management
- **Docker** used for building container images and minikube setup
- **Helm** used for packages installation and management.
- **Prometheus and Grafana** for cluster resource monitoring and visualization
- **Python 3.10+** for tensor flow, NumPy, Pandas, Matplotlib, Boto3 (AWS usage)

# 4. Cluster Setup

## 4.1. Local Cluster

### 4.1.1. (MiniKube)

Install minikube if not installed:

**On macOS:**

```
brew install minikube
```

**On Ubuntu/Debian:**

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

**On Windows (PowerShell)**

```
choco install minikube
```

### 4.1.2. Installation Verification

```
minikube version
```

```
minikube version: v1.36.0
commit: f8f52f5de11fc6ad8244afac475e1d0f96841df1
```

**4.1.3. Setup Kubectl** — kubectl is a command-line tool used to interact with Kubernetes clusters.

```
[macbook       id ~ % kubectl version
Client Version: v1.33.1
Kustomize Version: v5.6.0
```

### 4.1.4. Start Minikube

*minikube start --cpus=4 --memory=7000 –addons =metrics-server –driver=docker*

```
[macbo       jid ~ % minikube start --cpus=4 --memory=7000 --addons=metrics-server --driver=docker
😄  minikube v1.36.0 on Darwin 15.4.1 (arm64)
✨  Using the docker driver based on user configuration
📌  Using Docker Desktop driver with root privileges
👍  Starting "minikube" primary control-plane node in "minikube" cluster
🚜  Pulling base image v0.0.47 ...
💾  Downloading Kubernetes v1.33.1 preload ...
    > preloaded-images-k8s-v18-v1...:  327.15 MiB / 327.15 MiB  100.00% 22.31 M
    > gcr.io/k8s-minikube/kicbase...:  463.69 MiB / 463.69 MiB  100.00% 23.13 M
🔥  Creating docker container (CPUs=4, Memory=7000MB) ...
🐳  Preparing Kubernetes v1.33.1 on Docker 28.1.1 ...
    ▪ Generating certificates and keys ...
    ▪ Booting up control plane ...
    ▪ Configuring RBAC rules ...
🔗  Configuring bridge CNI (Container Networking Interface) ...
🔎  Verifying Kubernetes components...
    ▪ Using image registry.k8s.io/metrics-server/metrics-server:v0.7.2
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🌟  Enabled addons: storage-provisioner, default-storageclass, metrics-server

❗  /usr/local/bin/kubectl is version 1.30.5, which may have incompatibilities with Kubernetes 1.33.1.
    ▪ Want kubectl v1.33.1? Try 'minikube kubectl -- get pods -A'
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

Figure 1. Minikube

### 4.1.5. Minikube dashboard:

MiniKube dashboard:

```
minikube dashboard
```

http://127.0.0.1:54649/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/

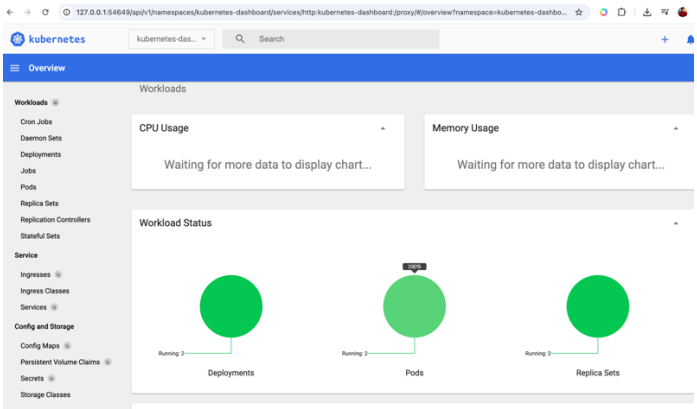


Figure 2. Minikube dashboard

### 4.1.6. Installed **Helm chart repository** for Prometheus

```
macbo      ajid ~ % brew install helm
==> Auto-updating Homebrew...
Adjust how often this is run with HOMEBREW_AUTO_UPDATE_SECS or disable with
HOMEBREW_NO_AUTO_UPDATE. Hide these hints with HOMEBREW_NO_ENV_HINTS (see `man brew`).
```

### 4.1.7. Installed **kube-prometheus-stack** via Helm:

helm repo add prometheus-community https://prometheus-community.github.io/helm-charts

helm install prometheus prometheus-community/kube-prometheus-stack

```
macbookpro@Majid ~ % helm repo add prometheus-community https://prometheus-community.github.io/helm-
charts
"prometheus-community" has been added to your repositories
macbookpro@Majid ~ % helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "prometheus-community" chart repository
Update Complete. ⎈Happy Helming!⎈
  acbookpro@Majid ~ % kubectl create namespace monitoring
namespace/monitoring created
  acbookpro@Majid ~ % helm install prometheus \
   prometheus-community/kube-prometheus-stack \
[  --namespace monitoring
      prometheus
LAST DEPLOYED: Wed Jun  4 15:59:53 2025
NAMESPACE: monitoring
STATUS: deployed
REVISION: 1
NOTES:
kube-prometheus-stack has been installed. Check its status by running:
  kubectl --namespace monitoring get pods -l "release=prometheus"

Get Grafana 'admin' user password by running:

  kubectl --namespace monitoring get secrets prometheus-grafana -o jsonpath="{.data.admin-password}"
 | base64 -d ; echo

Access Grafana local instance:
```

Figure 3. Installation of Prometheus and Grafana

This will install **Prometheus**, **Node Exporter**, **Grafana**, and more into the monitoring namespace.

Grafana is a **data visualization and dashboarding tool**.

For port forwarding run these:

Grafana -> kubectl port-forward -n monitoring svc/prometheus-grafana 3000:80

Prometheus -> kubectl port-forward -n monitoring svc/prometheus-kube-prometheus-prometheus 9090

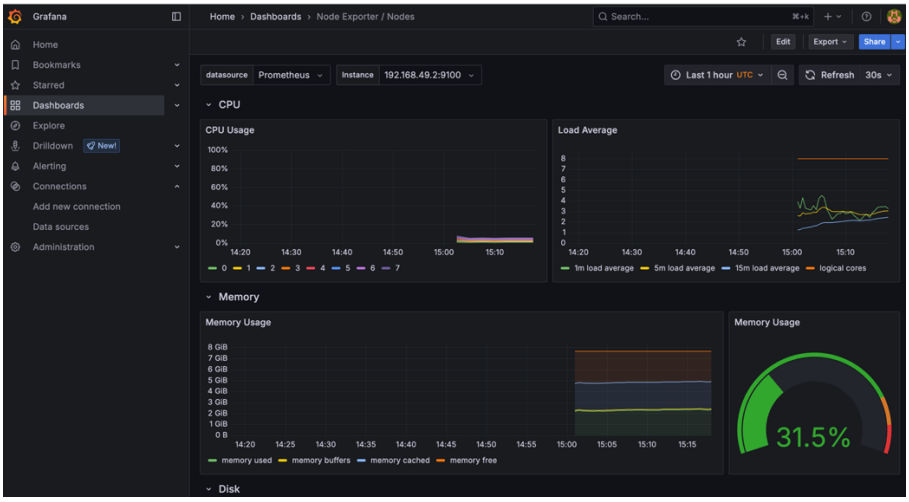


Figure 4. Grafana Dashboard

### 4.1.8. Then create three tenants:

```
kubectl create namespace tenant-a
kubectl create namespace tenant-b
kubectl create namespace tenant-c
```

### 4.1.9. Apply the workload on tenants

```
kubectl apply -f ffmpeg-pod.yaml
kubectl apply -f mqtt-broker.yaml
kubectl apply -f ngnix-api.yaml
```

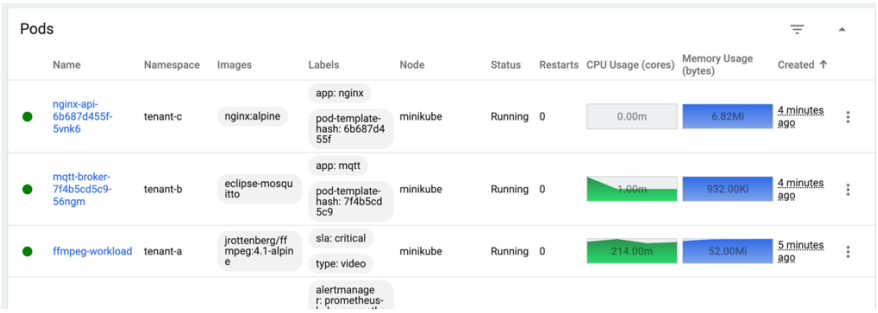


Figure 5. Minikube pods

Then start the data scraping by running the master file. It will start collecting the data from pods and then will train the model and then run docker pipeline.

## 4.2. Cloud EKS Cluster

Connect AWS with machine by credentials. If using AWS learner lab then connect with:

[default]
aws_access_key_id=……
aws_secret_access_key=…….
aws_session_token=……

#### 4.2.1. EKS cluster setup

Create cluster by running this in project workspace:

```
cd eks  eksctl create cluster -f cluster-config.yaml
```

**Connect to Cluster:**

```
aws eks update-kubeconfig --region us-east-1 --name pod-migration-cluster
```

**Create Namespace Monitoring**:

```
kubectl create namespace monitoring
```

**Install Prometheus:**

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts

helm install prometheus prometheus-community/kube-prometheus-stack
```

#### 4.2.2. Then create three tenants:

```
kubectl create namespace tenant-a

kubectl create namespace tenant-b

kubectl create namespace tenant-c
```

#### 4.2.3. Install Workload Pods

```
kubectl apply -f ffmpeg-pod.yaml

kubectl apply -f mqtt-broker.yaml

kubectl apply -f ngnix-api.yaml
```

This will create pods and workloads on EKS cluster.

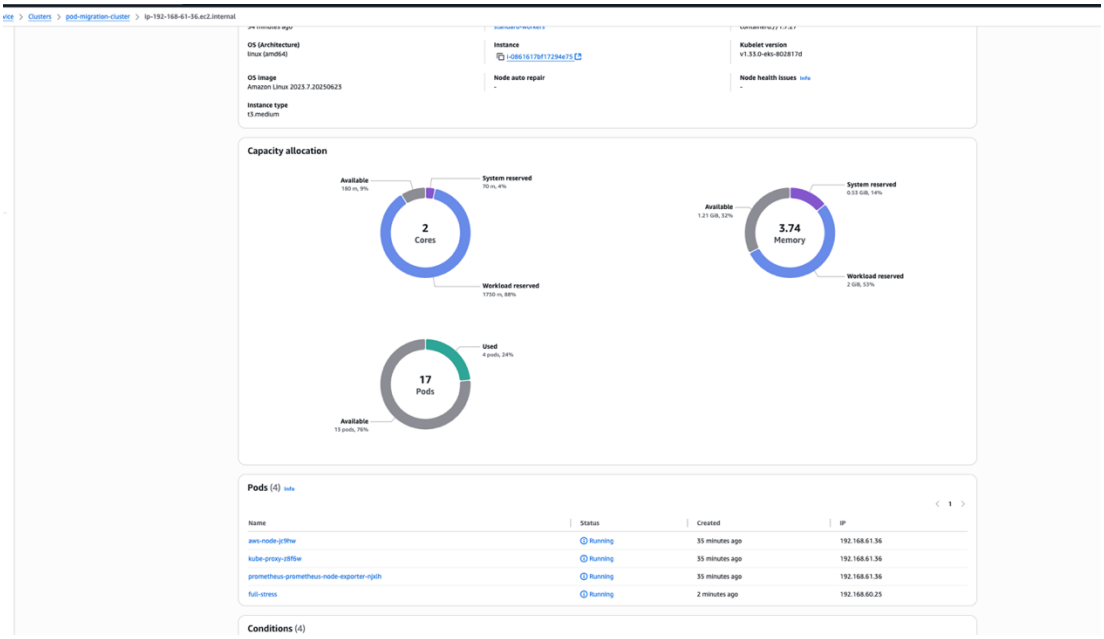


Figure 6. EKS cluster nodes

### 4.2.4. Master Loop for training and data collection

Then run these commands in project directory to start collecting the logs and train the model.

```
cd master

./master_test_train.sh
```

This will start a loop to collect metrics from the eks cluster and it will store them in files such as:

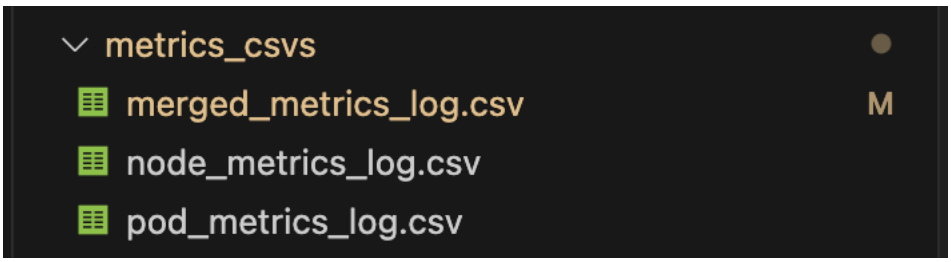


Figure 7. Metrics files

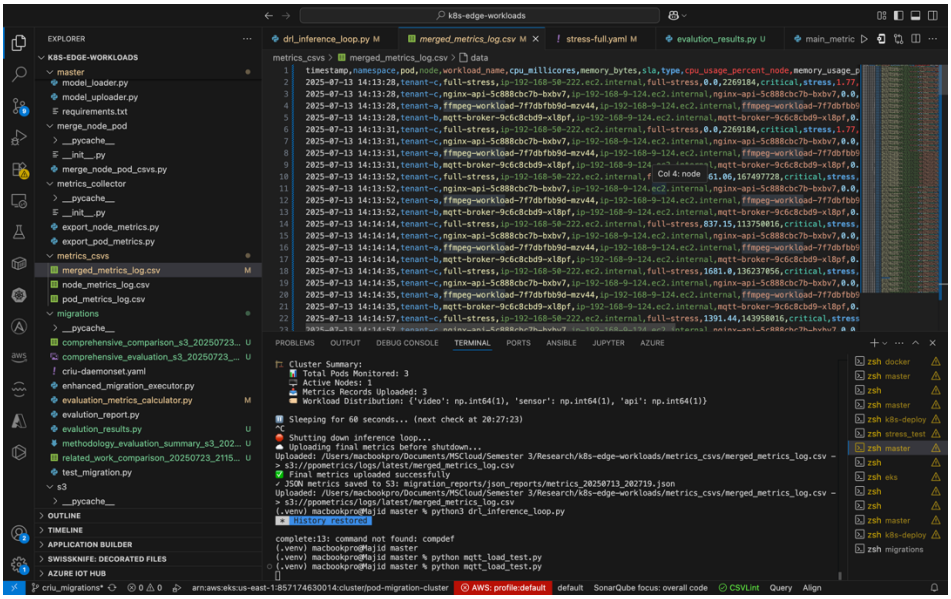

Figure 8. Collected Metrics

I have defined intervals in config.py file for the loop. These are in seconds.

```
# Configurable parameters
COLLECT_INTERVAL = 20
MERGE_INTERVAL = 30
TRAIN_EPOCHS = 5
DRL_STATE_DIM = 5
ACTION_SPACE = 2
UPLOAD_INTERVAL_METRICS = 120
```

Figure 9. Configurable Parameters for Automated Loop

Model will get train after some time in loop based on the epoch length. Then after training it auto uploads the model on the s3 bucket.

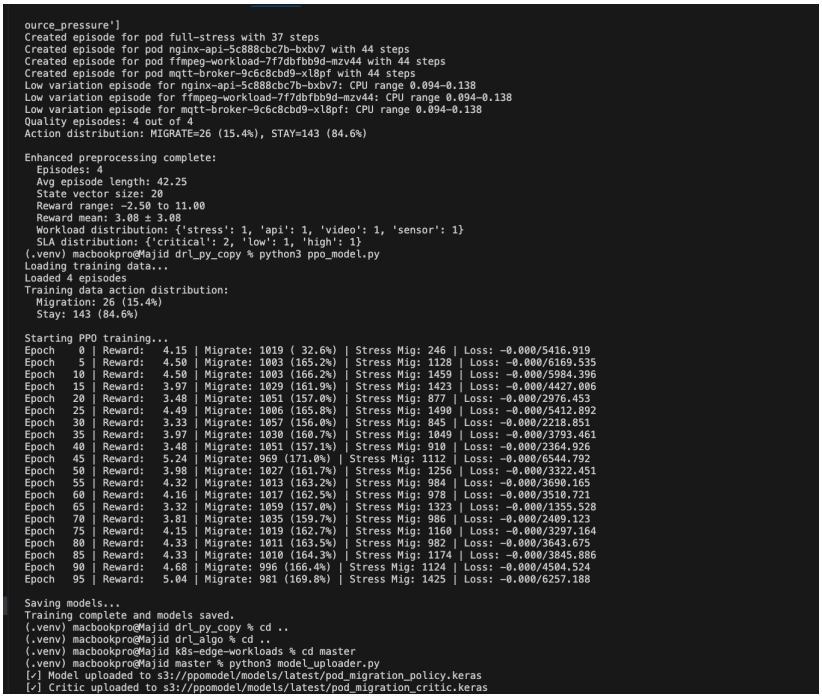


Figure 10. Model training and s3 uploading

### 4.2.5. Docker Hub and ECR

Navigate to docker directory in files:

```
cd docker

./build_docker.sh
```

This will make the docker image and then upload it to docker hub and ECR.

There is another option to run automatic pipeline to make and upload the docker image for that run this command inside the docker folder. It runs after every 60 minutes.

```
cd docker

python3 docker_rebuilder.py
```

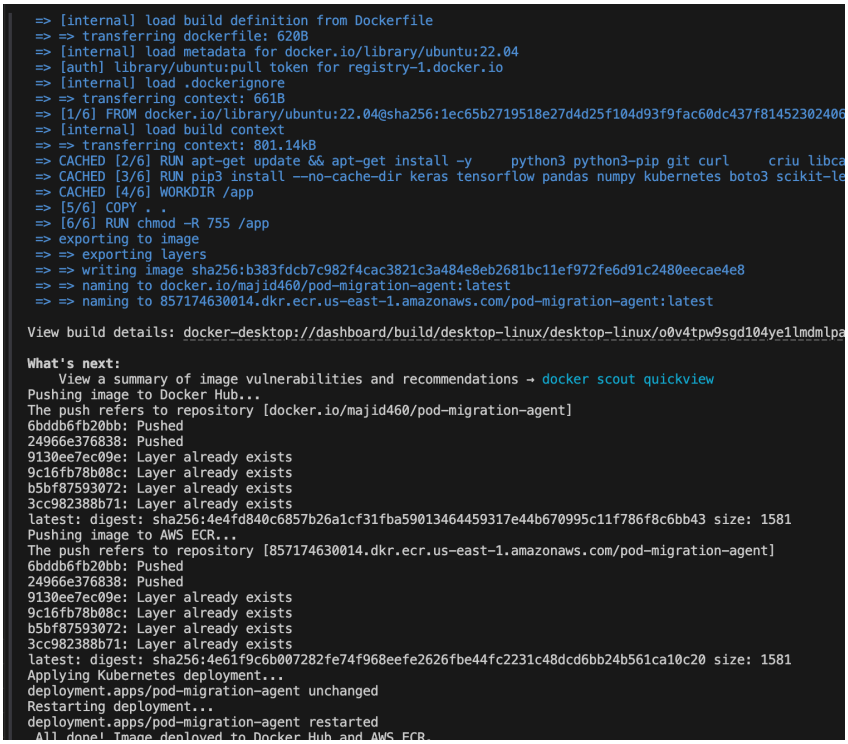


Figure 11. Dockerization of files

### 4.2.6. Agent Deployment

For agent deployment create a namespace within eks cluster.

```
kubectl create namespace pod-migration
```

Then deploy the agent deployment file on this namespace.

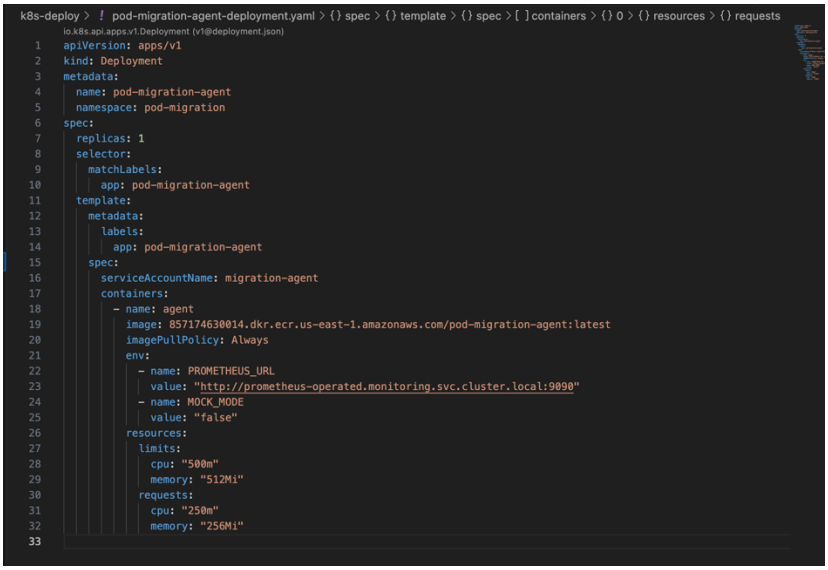


Figure 12. Agent deployment

It will use the uploaded image on ECR and then it will deploy the image in form of container on EKS cluster.

Before running the inference for agent you should deploy the agent roles and role binding on EKS cluster.

```
orkloads % cd k8s-deploy
 % kubectl apply -f migration-agent-sa.yaml
.yaml" does not exist
 % kubectl apply -f pod-migration-agent-sa.yaml
anged
 % kubectl apply -f pod-migration-agent-role.yaml
io/pod-eviction-role unchanged
 % kubectl apply -f pod-migration-agent-rolebinding.yaml
on.k8s.io/pod-eviction-binding unchanged
 % kubectl apply -f pod-migration-agent-deployment.yaml
 configured
 % kubectl get pods -n pod-migration
   READY   STATUS    RESTARTS      AGE
   1/1     Running   2 (16s ago)   36s
 % kubectl get pods -n pod-migration
    READY   STATUS   RESTARTS      AGE
6   0/1     Error    4 (65s ago)   118s
```

Figure 13. Agent Roles deployment

### 4.2.7. Inference

To check the agent is working and monitoring the EKS cluster run this command:

```
kubectl logs -f deployment/pod-migration-agent -n pod-migration
```

This will collect the logs and we can see the agent is monitoring the cluster.

```
📊 Collecting ORIGINAL metrics only...
   ✅ Collected ORIGINAL metrics for 3 pods
   🔒 Fields: ['timestamp', 'namespace', 'pod', 'node', 'workload_name', 'cpu_millicores', 'memory_bytes',
'cpu_usage_percent_node', 'memory_usage_percent_node', 'load_1min']
[INFO] Selected 3 pods from tenant namespaces: {'tenant-c', 'tenant-a', 'tenant-b'}

🔍 Processing 3 filtered pods across 1 nodes
   📍 Nodes: ['ip-192-168-36-137.ec2.internal']

🔬 Pod: ffmpeg-workload-7f7dbfbb9d-gzjjl (namespace: tenant-a)
   ├─ Node: ip-192-168-36-137.ec2.internal
   ├─ Type: video, SLA: critical
   └─ CPU: 10.3%, MEM: 20.7%, Millicores: 119.362002
   [INFO] Using 4-feature model compatibility mode
   📐 State vector shape: (1, 4) (model expects: (None, 4))
   🤖 ML Decision: MIGRATE (confidence: 0.516)
   📋 Rule Check: Within normal range: CPU=10.3%, MEM=20.7%
   ⚡ Final: ✋ REMAIN
      📝 Reason: ML decision kept as remain or low confidence: 0.516

🔬 Pod: mqtt-broker-9c6c8cbd9-wxjp7 (namespace: tenant-b)
   ├─ Node: ip-192-168-36-137.ec2.internal
   ├─ Type: sensor, SLA: high
   └─ CPU: 10.3%, MEM: 20.7%, Millicores: 0.367025
   [INFO] Using 4-feature model compatibility mode
   📐 State vector shape: (1, 4) (model expects: (None, 4))
   🤖 ML Decision: MIGRATE (confidence: 0.513)
   📋 Rule Check: Within normal range: CPU=10.3%, MEM=20.7%
   ⚡ Final: ✋ REMAIN
      📝 Reason: ML decision kept as remain or low confidence: 0.513

🔬 Pod: nginx-api-5c888cbc7b-fn8j9 (namespace: tenant-c)
   ├─ Node: ip-192-168-36-137.ec2.internal
   ├─ Type: api, SLA: low
   └─ CPU: 10.3%, MEM: 20.7%, Millicores: 0.0
   [INFO] Using 4-feature model compatibility mode
   📐 State vector shape: (1, 4) (model expects: (None, 4))
   🤖 ML Decision: MIGRATE (confidence: 0.513)
   📋 Rule Check: Within normal range: CPU=10.3%, MEM=20.7%
   ⚡ Final: ✋ REMAIN
      📝 Reason: ML decision kept as remain or low confidence: 0.513
```

Figure 13. Inference logs

# 5. Stress Test

After this to test the agent working about migration apply the stress on cluster by deploying the stress file.

```
cd stress_test

kubectl apply -f stress-full.yaml
```

It will deploy the stress pod on a node to see the migration will going to happen or not.

Then system will migrate the pod onto other node with all valid checks.

**Migrations Results:**

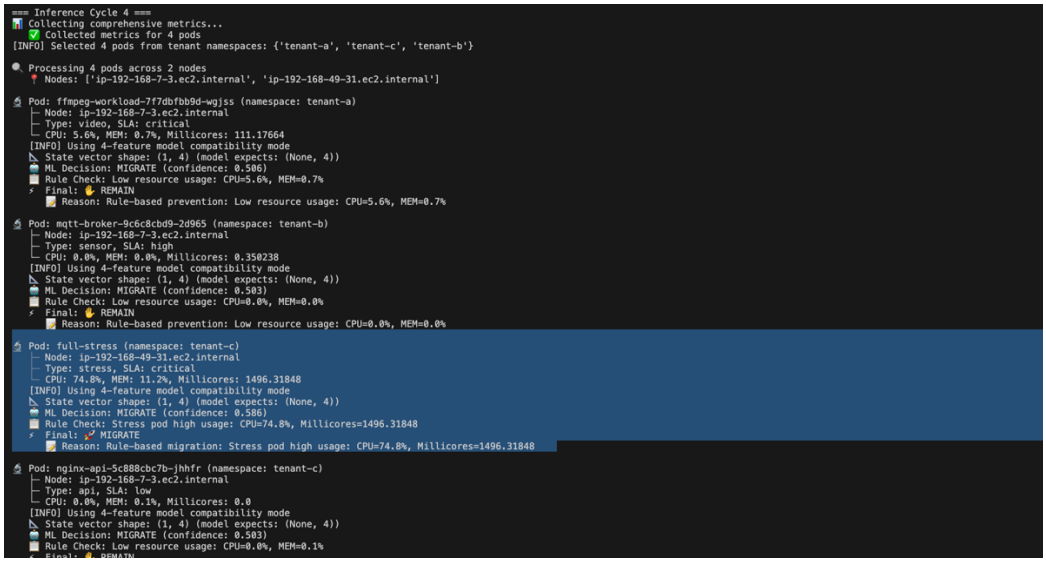


Figure 14. Cluster pods monitoring

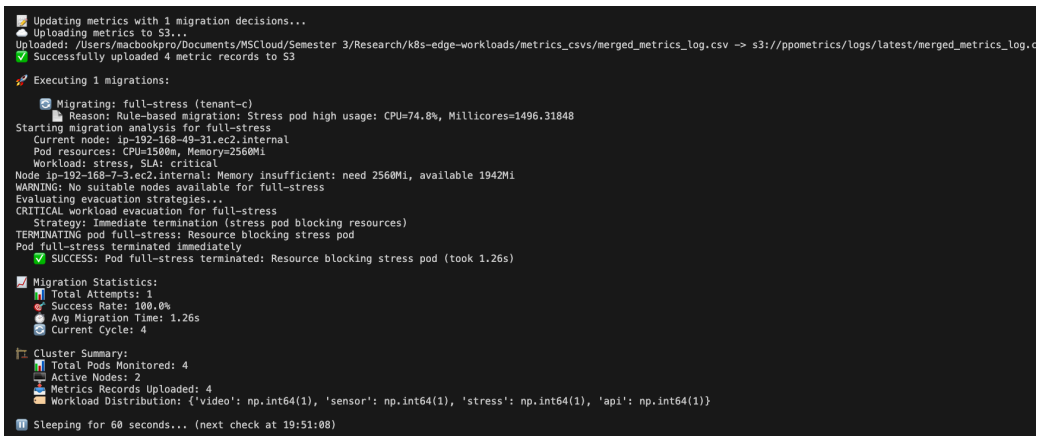


Figure 15. Migration of pod

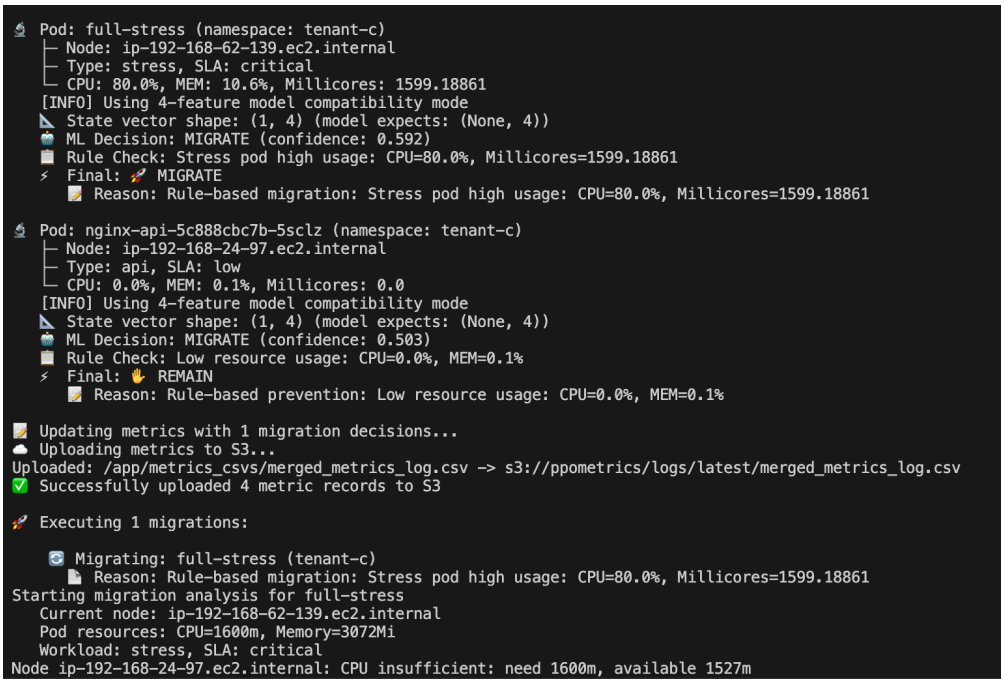


Figure 16. Migrating the pod

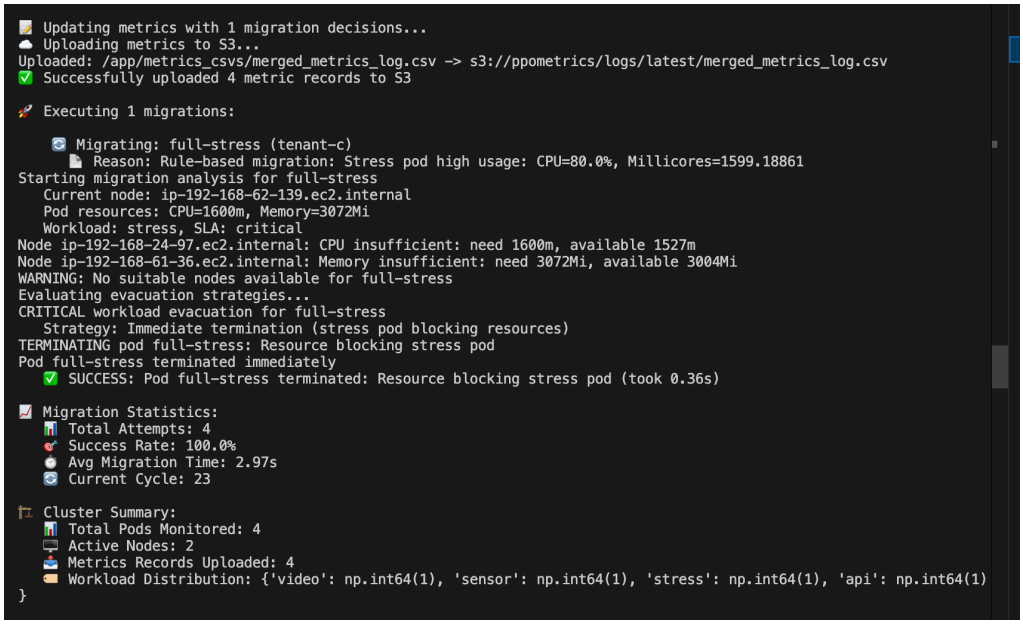


Figure 17. Migration Results

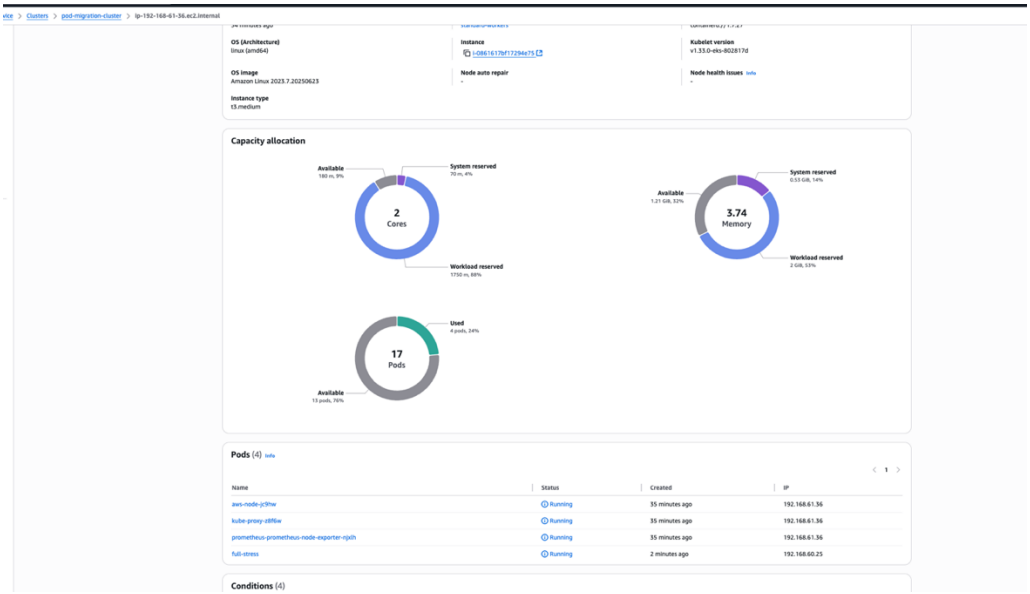


Figure 18. Node visualization

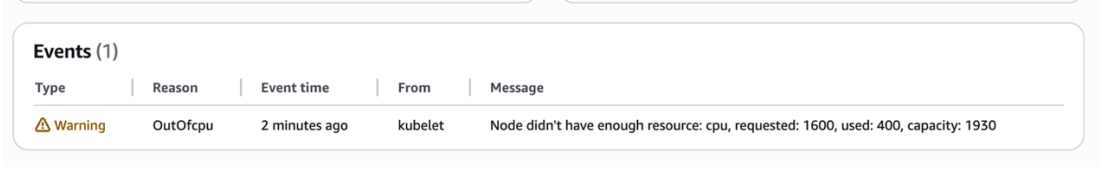


Figure 19. Node validation before migration

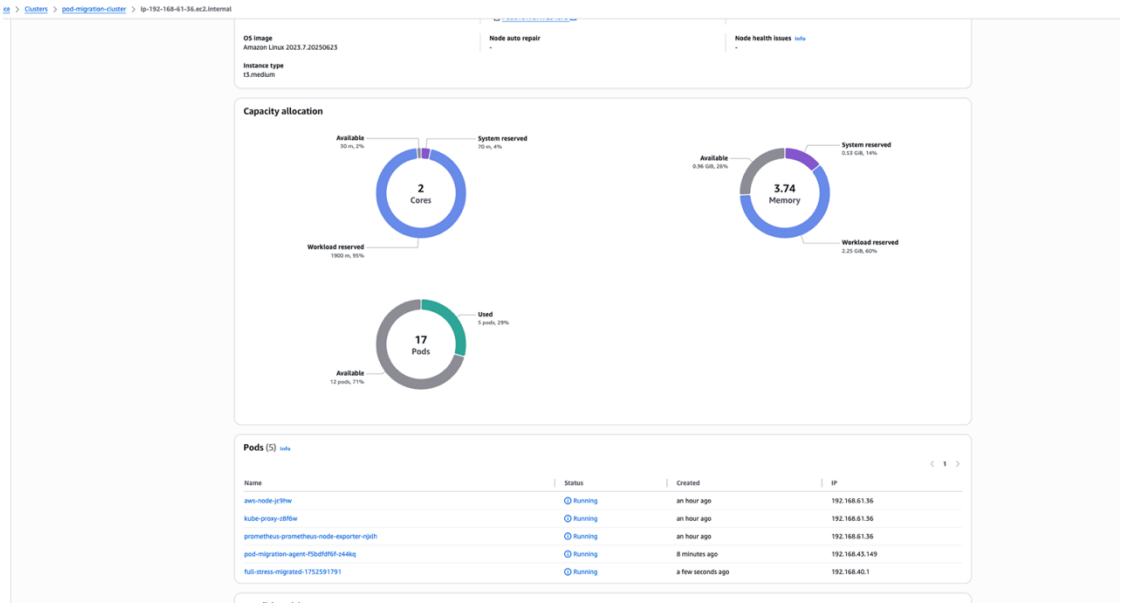


Figure 20. Migration results

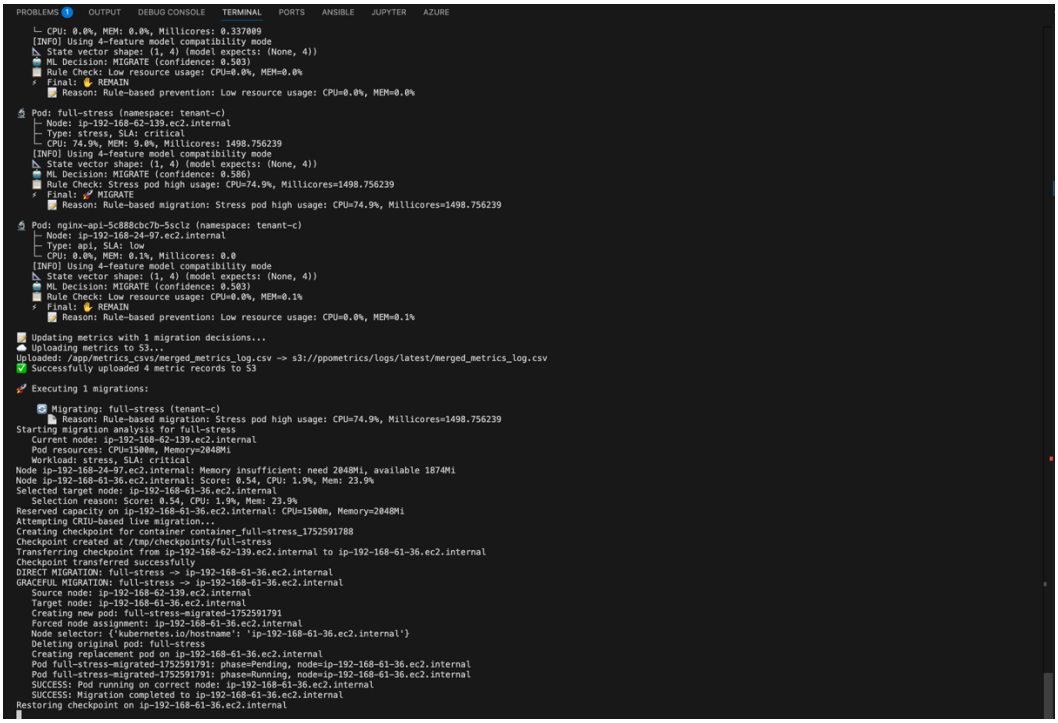


Figure 21. Final logs