

# 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] ~ % 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
```

```
macbook ~ % 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/>

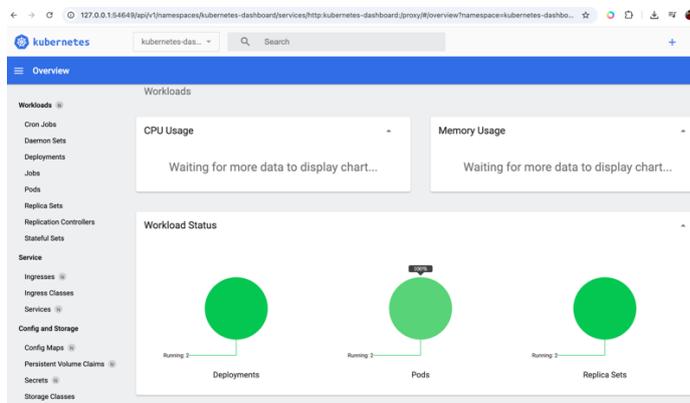


Figure 2. Minikube dashboard

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

```
macbookpro:~$ 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!*
[macbookpro:Majid ~ % kubectl create namespace monitoring
]
namespace/monitoring created
[macbookpro: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
```

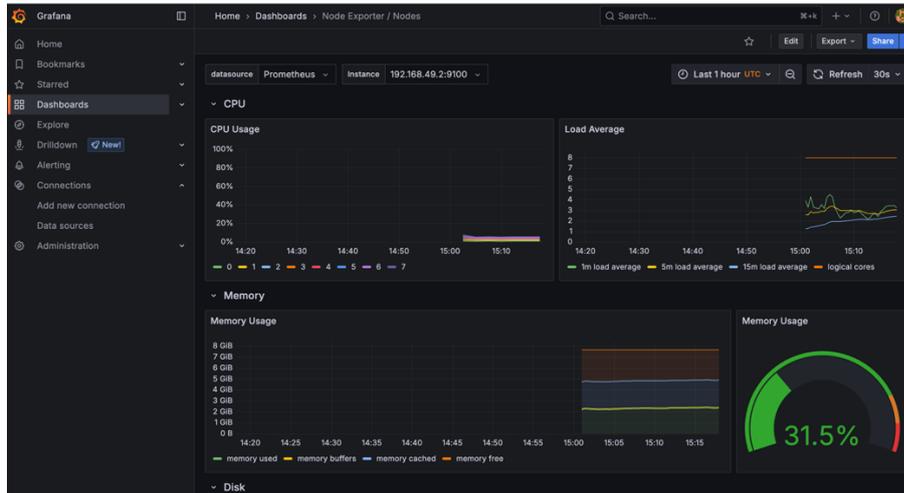


Figure 4. Grafana Dashboard

**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 nginx-api.yaml
```

| Name                         | Namespace | Images                        | Labels                                      | Node     | Status  | Restarts | CPU Usage (cores) | Memory Usage (bytes) | Created       |
|------------------------------|-----------|-------------------------------|---|----------|---------|----------|-------------------|----------------------|---------------|
| nginx-api-6b687d455f-5vnk6   | tenant-c  | nginx:alpine                  | app: nginx<br>pod-template-hash: 6b687d455f | minikube | Running | 0        | 0.00m             | 6.82Mi               | 4 minutes ago |
| mqtt-broker-7f4b5cd5c9-56ngm | tenant-b  | eclipse-mosquitto             | app: mqtt<br>pod-template-hash: 7f4b5cd5c9  | minikube | Running | 0        | 1.00m             | 932.00Ki             | 4 minutes ago |
| ffmpeg-workload              | tenant-a  | jrottenberg/ffmpeg:4.1-alpine | sla: critical<br>type: video                | minikube | Running | 0        | 214.00m           | 52.00Mi              | 5 minutes ago |

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 nginx-api.yaml
```

This will create pods and workloads on EKS cluster.

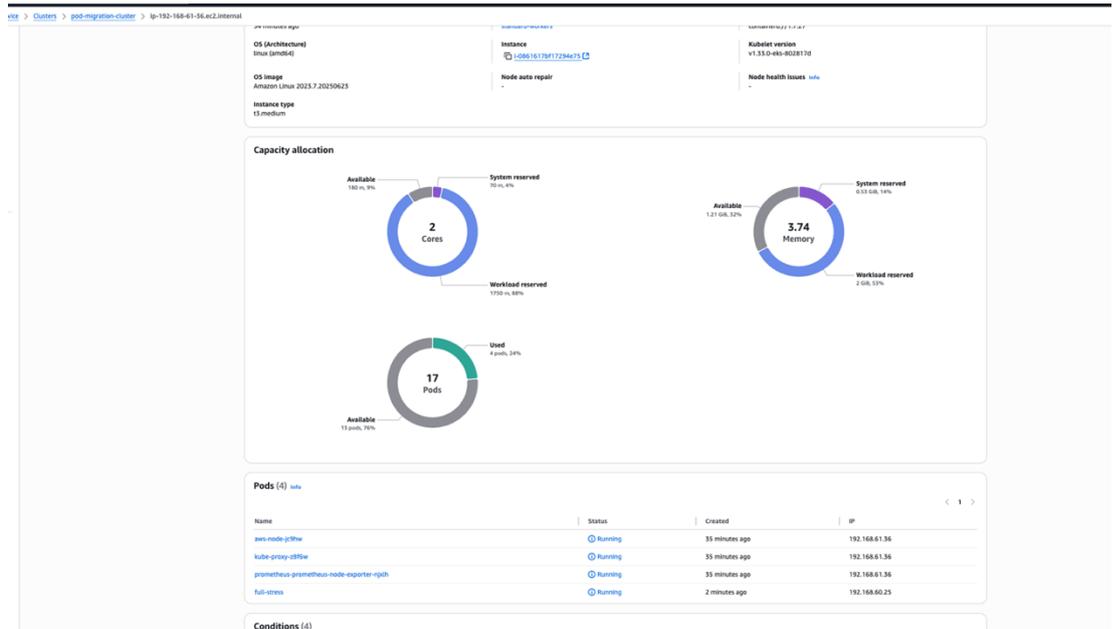


Figure 6. EKS cluster nodes

#### 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:

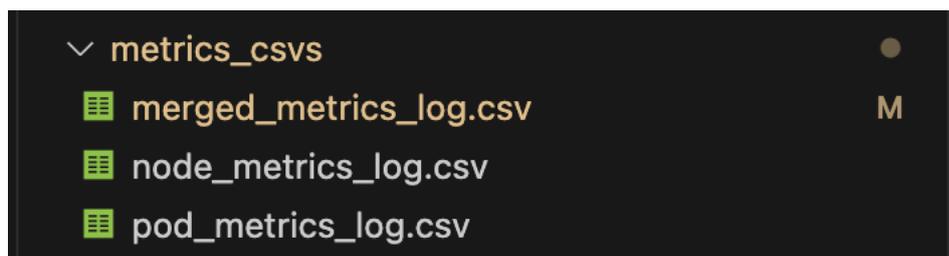


Figure 7. Metrics files



```

source_pressure']
Created episode for pod full-stress with 37 steps
Created episode for pod nginx-api-5c888cb7b-bxbv7 with 44 steps
Created episode for pod ffmpeg-workload-7f7dbfbb9d-mzv44 with 44 steps
Created episode for pod mqtt-broker-9c6c8cb9-xl8pf with 44 steps
Low variation episode for nginx-api-5c888cb7b-bxbv7: CPU range 0.094-0.138
Low variation episode for ffmpeg-workload-7f7dbfbb9d-mzv44: CPU range 0.094-0.138
Low variation episode for mqtt-broker-9c6c8cb9-xl8pf: CPU range 0.094-0.138
Quality episodes: 4 out of 4
Action distribution: MIGRATE=26 (15.4%), STAY=143 (84.6%)

Enhanced preprocessing complete:
Episodes: 4
Avg episode length: 42.25
State vector size: 20
Reward range: -2.50 to 11.00
Reward mean: 3.08 ± 3.08
Workload distribution: {'stress': 1, 'api': 1, 'video': 1, 'sensor': 1}
SLA distribution: {'critical': 2, 'low': 1, 'high': 1}
(.venv) macbookpro@Majid drl_py_copy % python3 ppo_model.py
Loading training data...
Loaded 4 episodes
Training data action distribution:
Migration: 26 (15.4%)
Stay: 143 (84.6%)

Starting PPO training...
Epoch 0 | Reward: 4.15 | Migrate: 1019 (32.6%) | Stress Mig: 246 | Loss: -0.000/5416.919
Epoch 5 | Reward: 4.50 | Migrate: 1003 (165.2%) | Stress Mig: 1128 | Loss: -0.000/6169.535
Epoch 10 | Reward: 4.50 | Migrate: 1003 (165.2%) | Stress Mig: 1459 | Loss: -0.000/5394.386
Epoch 15 | Reward: 3.97 | Migrate: 1029 (161.9%) | Stress Mig: 1423 | Loss: -0.000/4427.086
Epoch 20 | Reward: 3.48 | Migrate: 1051 (157.0%) | Stress Mig: 877 | Loss: -0.000/2976.453
Epoch 25 | Reward: 4.49 | Migrate: 1006 (165.8%) | Stress Mig: 1490 | Loss: -0.000/5412.892
Epoch 30 | Reward: 3.33 | Migrate: 1057 (156.0%) | Stress Mig: 845 | Loss: -0.000/2210.851
Epoch 35 | Reward: 3.97 | Migrate: 1030 (160.7%) | Stress Mig: 1049 | Loss: -0.000/3793.461
Epoch 40 | Reward: 3.48 | Migrate: 1051 (157.1%) | Stress Mig: 910 | Loss: -0.000/2364.926
Epoch 45 | Reward: 5.24 | Migrate: 969 (171.8%) | Stress Mig: 1112 | Loss: -0.000/6544.792
Epoch 50 | Reward: 3.98 | Migrate: 1027 (161.7%) | Stress Mig: 1256 | Loss: -0.000/3322.451
Epoch 55 | Reward: 4.32 | Migrate: 1013 (163.2%) | Stress Mig: 984 | Loss: -0.000/3690.165
Epoch 60 | Reward: 4.16 | Migrate: 1017 (162.5%) | Stress Mig: 978 | Loss: -0.000/3510.721
Epoch 65 | Reward: 3.32 | Migrate: 1059 (157.0%) | Stress Mig: 1323 | Loss: -0.000/1355.528
Epoch 70 | Reward: 3.81 | Migrate: 1035 (159.7%) | Stress Mig: 906 | Loss: -0.000/2409.123
Epoch 75 | Reward: 4.15 | Migrate: 1019 (162.7%) | Stress Mig: 1160 | Loss: -0.000/3297.164
Epoch 80 | Reward: 4.33 | Migrate: 1011 (163.5%) | Stress Mig: 982 | Loss: -0.000/3643.675
Epoch 85 | Reward: 4.33 | Migrate: 1010 (164.3%) | Stress Mig: 1174 | Loss: -0.000/3845.886
Epoch 90 | Reward: 4.68 | Migrate: 995 (166.4%) | Stress Mig: 1124 | Loss: -0.000/4584.524
Epoch 95 | Reward: 5.04 | Migrate: 981 (169.8%) | Stress Mig: 1425 | Loss: -0.000/6257.188

Saving models...
Training complete and models saved.
(.venv) macbookpro@Majid drl_py_copy % cd ..
(.venv) macbookpro@Majid drl_algo % cd ..
(.venv) macbookpro@Majid k8s-edge-workloads % cd master
(.venv) macbookpro@Majid master % python3 model_uploader.py
[✓] Model uploaded to s3://ppomodel/models/latest/pod_migration_policy.keras
[✓] Critic uploaded to s3://ppomodel/models/latest/pod_migration_critic.keras

```

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

```

```

=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 620B
=> [internal] load metadata for docker.io/library/ubuntu:22.04
=> [auth] library/ubuntu:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 661B
=> [1/6] FROM docker.io/library/ubuntu:22.04@sha256:1ec65b2719518e27d4d25f104d93f9fac60dc437f81452302406
=> [internal] load build context
=> => transferring context: 801.14kB
=> CACHED [2/6] RUN apt-get update && apt-get install -y python3 python3-pip git curl criu libc
=> CACHED [3/6] RUN pip3 install --no-cache-dir keras tensorflow pandas numpy kubernetes boto3 scikit-learn
=> CACHED [4/6] WORKDIR /app
=> [5/6] COPY . .
=> [6/6] RUN chmod -R 755 /app
=> exporting to image
=> => exporting layers
=> => writing image sha256:b383fdbc7c982f4cac3821c3a484e8eb2681bc11ef972fe6d91c2480eeca4e8
=> => naming to docker.io/majid460/pod-migration-agent:latest
=> => naming to 857174630014.dkr.ecr.us-east-1.amazonaws.com/pod-migration-agent:latest

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/o0v4tpw9sgd104ye1lmdmlpa

What's next:
View a summary of image vulnerabilities and recommendations -> docker scout quickview
Pushing image to Docker Hub...
The push refers to repository [docker.io/majid460/pod-migration-agent]
6bddb6fb20bb: Pushed
24966e376838: Pushed
9130ee7ec09e: Layer already exists
9c16fb78b08c: Layer already exists
b5bf87593072: Layer already exists
3cc982388b71: Layer already exists
latest: digest: sha256:4e4fd840c6857b26a1cf31fba59013464459317e44b670995c11f786f8c6bb43 size: 1581
Pushing image to AWS ECR...
The push refers to repository [857174630014.dkr.ecr.us-east-1.amazonaws.com/pod-migration-agent]
6bddb6fb20bb: Pushed
24966e376838: Pushed
9130ee7ec09e: Layer already exists
9c16fb78b08c: Layer already exists
b5bf87593072: Layer already exists
3cc982388b71: Layer already exists
latest: digest: sha256:4e61f9c6b007282fe74f968eefe2626fbc44fc2231c48dcd6bb24b561ca10c20 size: 1581
Applying Kubernetes deployment...
deployment.apps/pod-migration-agent unchanged
Restarting deployment...
deployment.apps/pod-migration-agent restarted
All done! Image deployed to Docker Hub and AWS ECR

```

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.

```

k8s-deploy > ! pod-migration-agent-deployment.yaml > {} spec > {} template > {} spec > [ ] containers > {} 0 > {} resources > {} requests
1 id: eks.api.apps.v1.Deployment (v1@deployment.json)
2 apiVersion: apps/v1
3 kind: Deployment
4 metadata:
5   name: pod-migration-agent
6   namespace: pod-migration
7 spec:
8   replicas: 1
9   selector:
10    matchLabels:
11     app: pod-migration-agent
12 template:
13   metadata:
14     labels:
15     app: pod-migration-agent
16   spec:
17     serviceAccountName: migration-agent
18     containers:
19     - name: agent
20       image: 857174630014.dkr.ecr.us-east-1.amazonaws.com/pod-migration-agent:latest
21       imagePullPolicy: Always
22       env:
23       - name: PROMETHEUS_URL
24         value: "http://prometheus-operated.monitoring.svc.cluster.local:9090"
25       - name: MOCK_NODE
26         value: "false"
27     resources:
28     limits:
29       cpu: "500m"
30       memory: "512Mi"
31     requests:
32       cpu: "250m"
33       memory: "256Mi"

```

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
  6/1     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

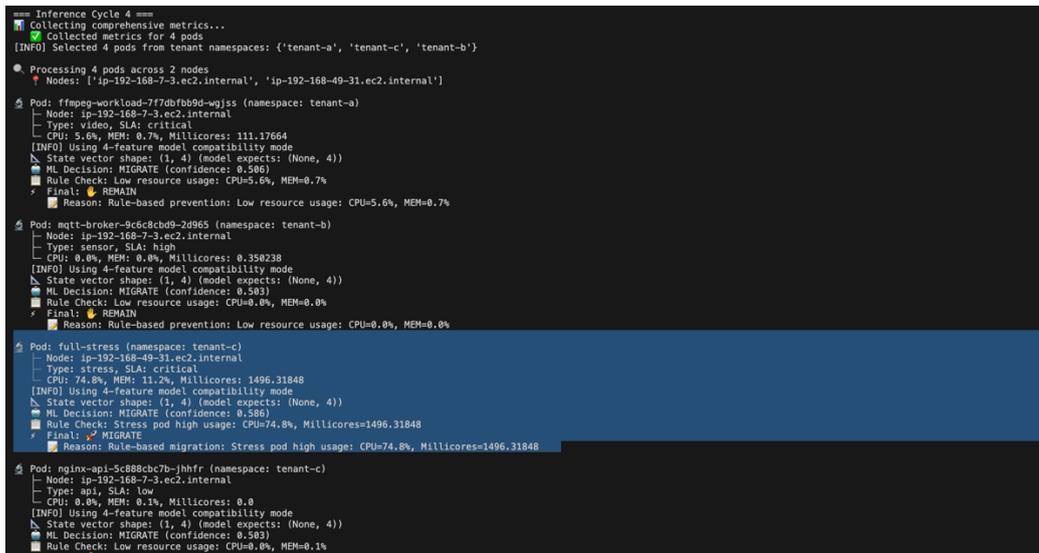
After this to test the agent working about migration apply the stress on cluster by deploying the stress file.

```
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.

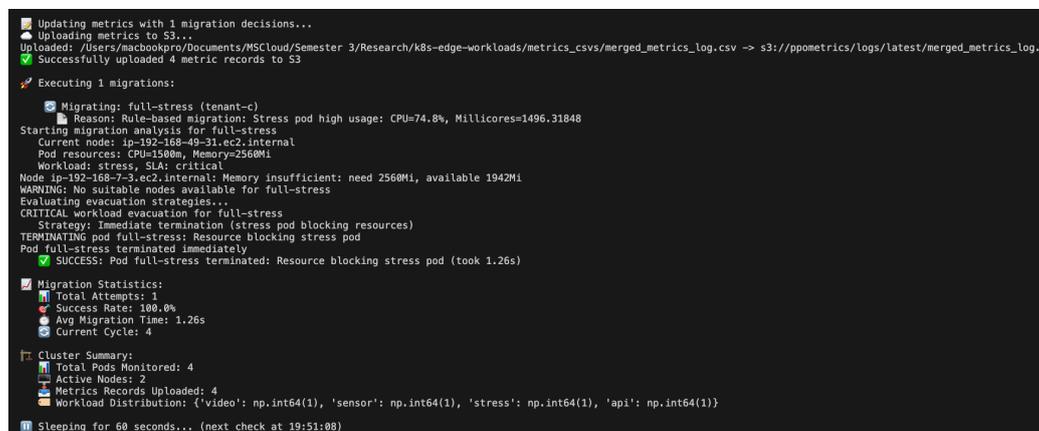
Then system will migrate the pod onto other node with all valid checks.

### Migrations Results:



```
==== Inference Cycle 4 ====
Collecting comprehensive metrics...
[INFO] Collected metrics for 4 pods
[INFO] Selected 4 pods from tenant namespaces: ('tenant-a', 'tenant-c', 'tenant-b')
Processing 4 pods across 2 nodes
  Nodes: ['ip-192-168-7-3.ec2.internal', 'ip-192-168-49-31.ec2.internal']
Pod: ffmpeg-workload-7f7dbfb9d-wgjs (namespace: tenant-a)
  Node: ip-192-168-7-3.ec2.internal
  Type: video, SLA: critical
  CPU: 5.6%, MEM: 0.7%, Millicores: 111.17664
  [INFO] Using 4-feature model compatibility mode
  State vector shape: (1, 4) (model expects: (None, 4))
  ML Decision: MIGRATE (confidence: 0.586)
  Rule Check: Low resource usage: CPU=5.6%, MEM=0.7%
  Final: REMAIN
  Reason: Rule-based prevention: Low resource usage: CPU=5.6%, MEM=0.7%
Pod: mqtt-broker-9c6c8cb9d-2d965 (namespace: tenant-b)
  Node: ip-192-168-7-3.ec2.internal
  Type: sensor, SLA: high
  CPU: 0.0%, MEM: 0.0%, Millicores: 0.350238
  [INFO] Using 4-feature model compatibility mode
  State vector shape: (1, 4) (model expects: (None, 4))
  ML Decision: MIGRATE (confidence: 0.583)
  Rule Check: Low resource usage: CPU=0.0%, MEM=0.0%
  Final: REMAIN
  Reason: Rule-based prevention: Low resource usage: CPU=0.0%, MEM=0.0%
Pod: full-stress (namespace: tenant-c)
  Node: ip-192-168-49-31.ec2.internal
  Type: stress, SLA: critical
  CPU: 74.8%, MEM: 11.2%, Millicores: 1496.31848
  [INFO] Using 4-feature model compatibility mode
  State vector shape: (1, 4) (model expects: (None, 4))
  ML Decision: MIGRATE (confidence: 0.586)
  Rule Check: Stress pod high usage: CPU=74.8%, Millicores=1496.31848
  Final: MIGRATE
  Reason: Rule-based migration: Stress pod high usage: CPU=74.8%, Millicores=1496.31848
Pod: nginx-api-5c888cb7b-jhfr (namespace: tenant-c)
  Node: ip-192-168-7-3.ec2.internal
  Type: api, SLA: low
  CPU: 0.0%, MEM: 0.1%, Millicores: 0.0
  [INFO] Using 4-feature model compatibility mode
  State vector shape: (1, 4) (model expects: (None, 4))
  ML Decision: MIGRATE (confidence: 0.583)
  Rule Check: Low resource usage: CPU=0.0%, MEM=0.1%
  Final: REMAIN
```

Figure 14. Cluster pods monitoring



```
Updating metrics with 1 migration decisions...
Uploading metrics to S3...
Uploads: /Users/mcbokoro/Documents/MSCloud/Semester 3/Research/k8s-edge-workloads/metrics_csvs/merged_metrics_log.csv -> s3://ppometrics/logs/latest/merged_metrics_log.c
Successfully uploaded 4 metric records to S3
Executing 1 migrations:
Migrating: full-stress (tenant-c)
  Reason: Rule-based migration: Stress pod high usage: CPU=74.8%, Millicores=1496.31848
Starting migration analysis for full-stress
Current node: ip-192-168-49-31.ec2.internal
Pod resources: CPU=1500m, Memory=2560Mi
Workload: stress, SLA: critical
Node ip-192-168-7-3.ec2.internal: Memory insufficient: need 2560Mi, available 1942Mi
WARNING: No suitable nodes available for full-stress
Evaluating evacuation strategies...
CRITICAL workload evacuation for full-stress
Strategy: Immediate termination (stress pod blocking resources)
TERMINATING pod full-stress: Resource blocking stress pod
Pod full-stress terminated immediately
SUCCESS: Pod full-stress terminated: Resource blocking stress pod (took 1.26s)
Migration Statistics:
  Total Attempts: 1
  Success Rate: 100.0%
  Avg Migration Time: 1.26s
  Current Cycle: 4
Cluster Summary:
  Total Pods Monitored: 4
  Active Nodes: 2
  Metrics Records Uploaded: 4
  Workload Distribution: ('video': np.int64(1), 'sensor': np.int64(1), 'stress': np.int64(1), 'api': np.int64(1))
Sleeping for 60 seconds... (next check at 19:51:08)
```

Figure 15. Migration of pod

```

Pod: full-stress (namespace: tenant-c)
├─ Node: ip-192-168-62-139.ec2.internal
├─ Type: stress, SLA: critical
├─ CPU: 80.0%, MEM: 10.6%, Millicores: 1599.18861
└─ [INFO] Using 4-feature model compatibility mode
  └─ State vector shape: (1, 4) (model expects: (None, 4))
    └─ ML Decision: MIGRATE (confidence: 0.592)
      └─ Rule Check: Stress pod high usage: CPU=80.0%, Millicores=1599.18861
        └─ Final: ⚡ MIGRATE
          └─ Reason: Rule-based migration: Stress pod high usage: CPU=80.0%, Millicores=1599.18861

Pod: nginx-api-5c888cbc7b-5sclz (namespace: tenant-c)
├─ Node: ip-192-168-24-97.ec2.internal
├─ Type: api, SLA: low
├─ CPU: 0.0%, MEM: 0.1%, Millicores: 0.0
└─ [INFO] Using 4-feature model compatibility mode
  └─ State vector shape: (1, 4) (model expects: (None, 4))
    └─ ML Decision: MIGRATE (confidence: 0.503)
      └─ Rule Check: Low resource usage: CPU=0.0%, MEM=0.1%
        └─ Final: 🛑 REMAIN
          └─ Reason: Rule-based prevention: Low resource usage: CPU=0.0%, MEM=0.1%

Updating metrics with 1 migration decisions...
Uploading metrics to S3...
Uploaded: /app/metrics_csvs/merged_metrics_log.csv -> s3://ppometrics/logs/latest/merged_metrics_log.csv
Successfully uploaded 4 metric records to S3

Executing 1 migrations:

Migrating: full-stress (tenant-c)
Reason: Rule-based migration: Stress pod high usage: CPU=80.0%, Millicores=1599.18861
Starting migration analysis for full-stress
Current node: ip-192-168-62-139.ec2.internal
Pod resources: CPU=1600m, Memory=3072Mi
Workload: stress, SLA: critical
Node ip-192-168-24-97.ec2.internal: CPU insufficient: need 1600m, available 1527m

```

Figure 16. Migrating the pod

```

Updating metrics with 1 migration decisions...
Uploading metrics to S3...
Uploaded: /app/metrics_csvs/merged_metrics_log.csv -> s3://ppometrics/logs/latest/merged_metrics_log.csv
Successfully uploaded 4 metric records to S3

Executing 1 migrations:

Migrating: full-stress (tenant-c)
Reason: Rule-based migration: Stress pod high usage: CPU=80.0%, Millicores=1599.18861
Starting migration analysis for full-stress
Current node: ip-192-168-62-139.ec2.internal
Pod resources: CPU=1600m, Memory=3072Mi
Workload: stress, SLA: critical
Node ip-192-168-24-97.ec2.internal: CPU insufficient: need 1600m, available 1527m
Node ip-192-168-61-36.ec2.internal: Memory insufficient: need 3072Mi, available 3004Mi
WARNING: No suitable nodes available for full-stress
Evaluating evacuation strategies...
CRITICAL workload evacuation for full-stress
Strategy: Immediate termination (stress pod blocking resources)
TERMINATING pod full-stress: Resource blocking stress pod
Pod full-stress terminated immediately
SUCCESS: Pod full-stress terminated: Resource blocking stress pod (took 0.36s)

Migration Statistics:
Total Attempts: 4
Success Rate: 100.0%
Avg Migration Time: 2.97s
Current Cycle: 23

Cluster Summary:
Total Pods Monitored: 4
Active Nodes: 2
Metrics Records Uploaded: 4
Workload Distribution: {'video': np.int64(1), 'sensor': np.int64(1), 'stress': np.int64(1), 'api': np.int64(1)}

```

Figure 17. Migration Results

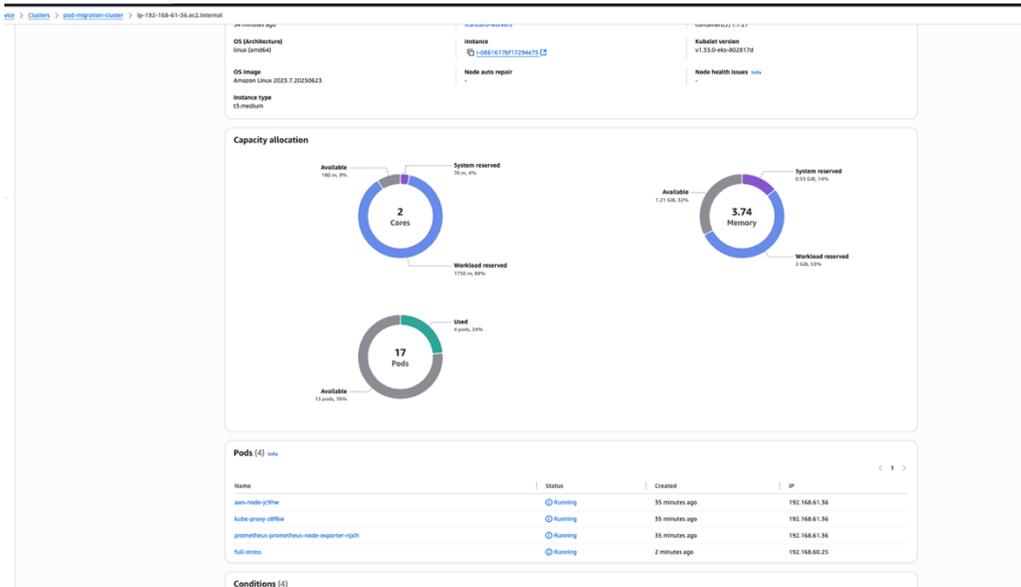


Figure 18. Node visualization

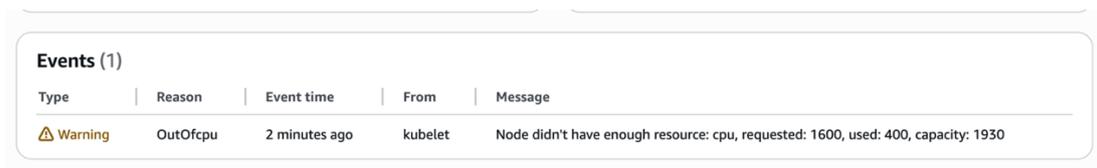


Figure 19. Node validation before migration

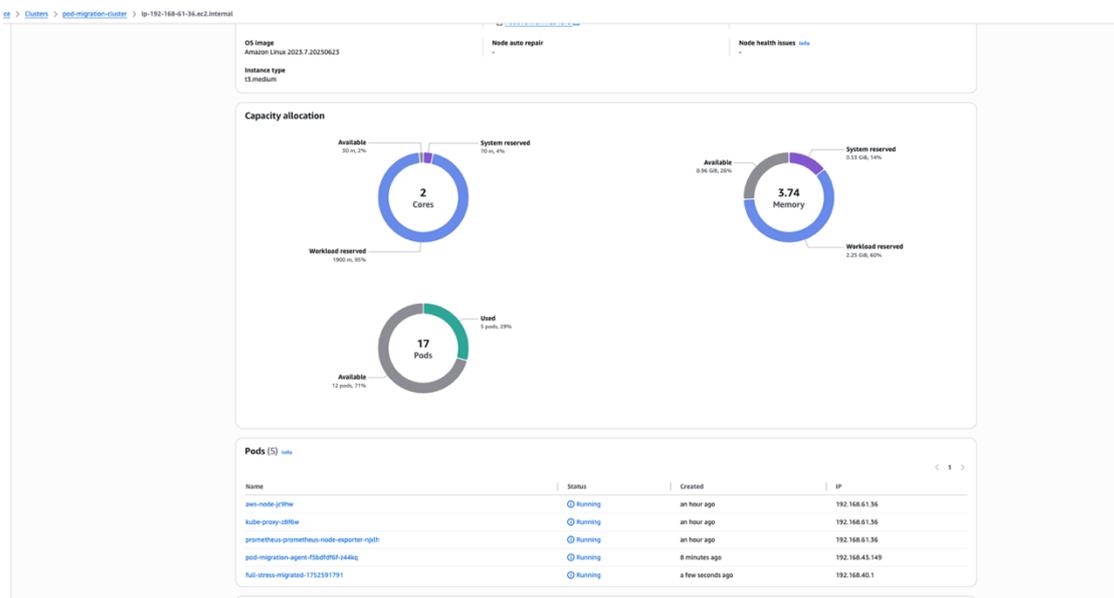


Figure 20. Migration results

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS ANSIBLE JUPYTER AZURE
└─ CPU: 0.0%, MEM: 0.0%, Millicores: 0.337809
  [INFO] Using 4-feature model compatibility mode
  ↳ State vector shape: (1, 4) (model outputs: (None, 4))
  ── ML Decision: MIGRATE (confidence: 0.583)
  ── Rule Check: Low resource usage: CPU=0.0%, MEM=0.0%
  ↳ Final: 0.583MIX
  ── Reason: Rule-based prevention: Low resource usage: CPU=0.0%, MEM=0.0%
5 Pod: full-stress (namespace: tenant-c)
└─ Node: ip-192-168-62-139.ec2.internal
  └─ Type: stress, SLA: critical
  └─ CPU: 74.9%, MEM: 9.0%, Millicores: 1498.756239
  [INFO] Using 4-feature model compatibility mode
  ↳ State vector shape: (1, 4) (model outputs: (None, 4))
  ── ML Decision: MIGRATE (confidence: 0.586)
  ── Rule Check: Stress pod high usage: CPU=74.9%, Millicores=1498.756239
  ↳ Final: 0.586MIX
  ── Reason: Rule-based migration: Stress pod high usage: CPU=74.9%, Millicores=1498.756239
5 Pod: nginx-api-5c888cb7b-5sc2z (namespace: tenant-c)
└─ Node: ip-192-168-24-97.ec2.internal
  └─ Type: api, SLA: low
  └─ CPU: 0.0%, MEM: 0.1%, Millicores: 0.0
  [INFO] Using 4-feature model compatibility mode
  ↳ State vector shape: (1, 4) (model outputs: (None, 4))
  ── ML Decision: MIGRATE (confidence: 0.583)
  ── Rule Check: Low resource usage: CPU=0.0%, MEM=0.1%
  ↳ Final: 0.583MIX
  ── Reason: Rule-based prevention: Low resource usage: CPU=0.0%, MEM=0.1%
  ── Updating metrics with 1 migration decisions...
  ── Uploading metrics to S3...
  ↳ Uploaded: /app/metrics.csv/merged_metrics_log.csv -> s3://ppometrics/logs/latest/merged_metrics_log.csv
  ── Successfully uploaded 4 metric records to S3
  ── Executing 1 migrations:
  ── Migrating: full-stress (tenant-c)
  ── Reason: Rule-based migration: Stress pod high usage: CPU=74.9%, Millicores=1498.756239
  Starting migration analysis for full-stress
  Current node: ip-192-168-62-139.ec2.internal
  Pod resources: CPU=1500m, Memory=2048Mi
  Workload: stress, SLA: critical
  Node ip-192-168-24-97.ec2.internal: Memory insufficient: need 2048Mi, available 1874Mi
  Node ip-192-168-61-36.ec2.internal: Score: 0.54, CPU: 1.9%, Mem: 23.5%
  Selected target node: ip-192-168-61-36.ec2.internal
  Selection reason: Score: 0.54, CPU: 1.9%, Mem: 23.5%
  Reserved capacity on ip-192-168-61-36.ec2.internal: CPU=1500m, Memory=2048Mi
  Attempting CRU-based Live migration...
  Creating checkpoint for container container_full-stress_1752591788
  Checkpoint created at /tmp/checkpoints/full-stress
  Transferring checkpoint from ip-192-168-62-139.ec2.internal to ip-192-168-61-36.ec2.internal
  Checkpoint transferred successfully
  DIRECT MIGRATION: full-stress -> ip-192-168-61-36.ec2.internal
  GRACEFUL MIGRATION: full-stress -> ip-192-168-61-36.ec2.internal
  Source node: ip-192-168-62-139.ec2.internal
  Target node: ip-192-168-61-36.ec2.internal
  Creating new node: full-stress-migrated-1752591791
  Forced node assignment: ip-192-168-61-36.ec2.internal
  Node selector: {'Kubernetes.io/hostname': 'ip-192-168-61-36.ec2.internal'}
  Deleting original pod: full-stress
  Creating replacement pod on ip-192-168-61-36.ec2.internal
  Pod full-stress-migrated-1752591791: phase=Pending, node=ip-192-168-61-36.ec2.internal
  Pod full-stress-migrated-1752591791: phase=Running, node=ip-192-168-61-36.ec2.internal
  SUCCESS: Pod running on correct node: ip-192-168-61-36.ec2.internal
  SUCCESS: Migration completed to ip-192-168-61-36.ec2.internal
  Restoring checkpoint on ip-192-168-61-36.ec2.internal
```

Figure 21. Final logs