CLIENT NAME
Computer Science Department
Swinburne Vietnam Alliance Program HCMC

User Documentation

Secure CI/CD Pipeline for Application Development

Version 1.0

Prepared by: GROUP 1

- Dang Vi Luan - 103802759
- Nguyen Linh Dan – 103488557
- Nguyen Duy Khang - 104056476
- Tran Bao Huy - 103505799

TABLE OF CONTENTS

GLOSSARY OF TERMS 1

USER MANUAL 3

1. INSTALL DOCKER 3

2. INSTALL MINIKUBE 3

3. CONFIGURE THE CI PIPELINE 4

4. CONFIGURE THE CD PIPELINE 5

5. CREATE AWS EKS CLUSTER 8

5.1. Create Role for EKS Cluster 8

5.2. Create Role for EC2 Instances 8

5.3. Create EKS Cluster 9

5.4. Create Compute Resources 9

5.5. Configure Cloud Shell 9

6. INSTALL ARGOCD 10

6.1. Create Namespace for ArgoCD 10

6.2. Apply ArgoCD Manifests 10

6.3. Patch Service Type to Load Balancer 10

6.4. Retrieve Admin Password 10

7. INSTALL GRAFANA AND PROMETHEUS FOR MONITORING 11

7.1. Install Helm 11

7.2. Add Helm Repositories 11

7.3. Install Prometheus 11

7.4. Install Grafana 12

7.5. Edit Prometheus Service 12

7.6. Edit Grafana Service 12

TUTORIAL VIDEOS 13

Video 1: How to make changes to local and remote cluster 13

Video 2: How to create an application on ArgoCD server 13

Video 3: How to import dashboards to track the states of cluster 13

GLOSSARY OF TERMS

Term	Description
ArgoCD	A declarative, GitOps continuous delivery tool for Kubernetes that automates the deployment of applications.
AWS EKS	Amazon Elastic Kubernetes Service is a managed Kubernetes service that makes it easy to run Kubernetes on AWS.
CD Flow	The process or sequence of steps that defines how changes are deployed to the Kubernetes cluster, ensuring the application is always up-to-date.
CD Pipeline	The Continuous Deployment pipeline keeps the application up-to-date by updating Kubernetes deployment manifests and pushing changes to the repository.
CI Flow	The process or sequence of steps that defines how code changes are integrated, built, tested, and pushed to Docker Hub.
CI Pipeline	Continuous Integration pipeline, which automatically builds and pushes Docker images to Docker Hub upon code changes.
Dashboard IDs	Identifiers for pre-built Grafana dashboards that can be imported to visualize different aspects of the Kubernetes cluster. Examples include k8s-addons-prometheus.json (ID: 19105), k8s-system-api-server.json (ID: 15761), k8s-system-coredns.json (ID: 15762), and more.
Docker	A platform used to develop, ship, and run applications inside containers. It includes Docker Daemon (a service to run containers) and Docker Desktop (the interface to manage containers).
Docker Hub	A cloud-based repository where Docker images are stored and shared.
GitHub Actions	A CI/CD service provided by GitHub to automate tasks like building, testing, and deploying code.
Global View	A comprehensive Grafana dashboard that provides an overview of the entire Kubernetes cluster's state and performance.
Grafana	An open-source platform for monitoring and observability, used to visualize metrics from Prometheus and other data sources.
Helm	A package manager for Kubernetes that helps manage Kubernetes applications by packaging them into charts.
Helm Repository	A collection of Helm charts, stored in a repository, used for deploying Kubernetes applications.

IAM Role	AWS Identity and Access Management roles that define permissions for EKS cluster and EC2 instances to interact with other AWS services.
Kubernetes (K8S)	An open-source platform for automating the deployment, scaling, and operations of application containers across clusters of hosts.
LoadBalancer	A Kubernetes service type that exposes applications to the internet by distributing traffic across multiple servers.
Manifest File	YAML configuration files (deployment.yaml and service.yaml) are used to define the desired state and configuration of Kubernetes resources.
Minikube	A tool that runs a single-node Kubernetes cluster locally for development and testing purposes.
NodePort	A Kubernetes service type that exposes applications on each node's IP at a static port.
Prometheus	An open-source monitoring and alerting toolkit for collecting and storing metrics, querying data, and alerting.

Table 1: Glossary of terms

USER MANUAL

The CI/CD pipeline is the key to automating the deployment process of students' projects. In order to implement the pipeline, we will go through several stages to configure the necessary components and ensure smooth data flow. This user manual will explain the pipeline implementation step-by-step to provide the end-user with a better user experience.

1. INSTALL DOCKER

Docker is the fundamental component of this pipeline. You might be confused with **Docker** and **Docker Hub**. They are quite similar, but their functionalities are different:

- **Docker** is the daemon, or the service to run containers, it is the **Docker Desktop** that you will download later.
- **Docker Hub** is a repository to store **Docker images**.

Please follow the guidelines in the URL attached below to install Docker.

https://docs.docker.com/desktop/install/windows-install/

Confirm the installation

Please use **`docker -v`** to confirm the installation is successful.

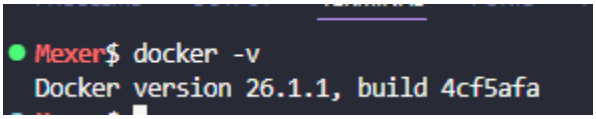

Figure 1.1: Docker version

2. INSTALL MINIKUBE (FOR LOCAL USAGE ONLY)

Kubernetes (K8S) is the service to operate, manage, and scale your Docker application. Therefore, it will also be necessary for our service.

In this section, we will install **Minikube** - a lightweight, single-node K8S cluster for local development.

Please follow the guidelines in the URL attached below to install **Minikube**.

https://minikube.sigs.k8s.io/docs/start/?arch=%2Fwindows%2Fx86-64%2Fstable%2F.exe+download

Confirm the installation

For the first time running **Minikube**, please use the following commands:

```
minikube config set driver docker
minikube start --driver=docker
```

From the second time running minikube, please execute the following command:

```
minikube start
```

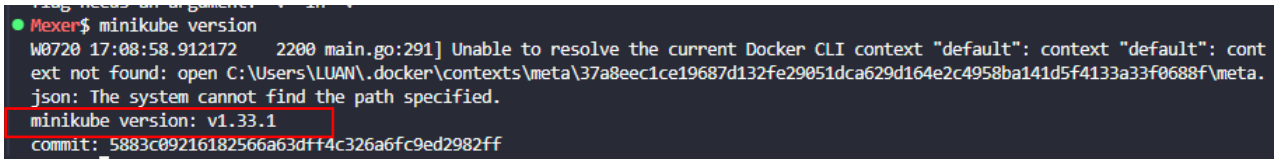

Figure 2.1: Minikube version for local usage

3. CONFIGURE THE CI PIPELINE

First, we will initialize the **GitHub Action** workflow in **.github\workflows** directory as follows:

```
name: Test CI Flow
on:
 push:
  branches:
   - main    Change to wanted branches on remote GitHub repos
 workflow_dispatch: {}
jobs:
 test-image:
  runs-on: ubuntu-latest
  steps:
   - uses: actions/checkout@v4
   - name:  Build and push test Docker image
     working-directory: ./Application/App3/Docker    Change to local directory
     run: |
        docker build . -t ${{ secrets.DOCKER_USERNAME }}/ict30001-test:${{ github.sha }}    Change to wanted DockerHub Repository
       echo "${{ secrets.DOCKER_PASSWORD }}" | docker login -u ${{ secrets.DOCKER_USERNAME }} --password-stdin
      docker push ${{ secrets.DOCKER_USERNAME }}/ict30001-test:${{ github.sha }}
```

From the above script, **GitHub Action** will automatically build and push **Docker Image** onto your **Docker Hub**. We will also include the configurations on **GitHub**.

Config 1: Allow Read and Write permissions for **GitHub Action** workflow.

Config 2: Provide GitHub with necessary the credentials in **DOCKER_USERNAME** and **DOCKER_PASSWORD**.

Confirm the CI pipeline operation

To check the operation of the CI pipeline, please follow the steps:

- Step 1: Write a simple code snippet
- Step 2: Dockerize the code
- Step 3: Run the dockerized code through the CI pipeline

If the image is present on Docker Hub, the CI pipeline has been set up correctly. Otherwise, please troubleshoot with the above instructions or contact us.

Overall flow of the CI flow:

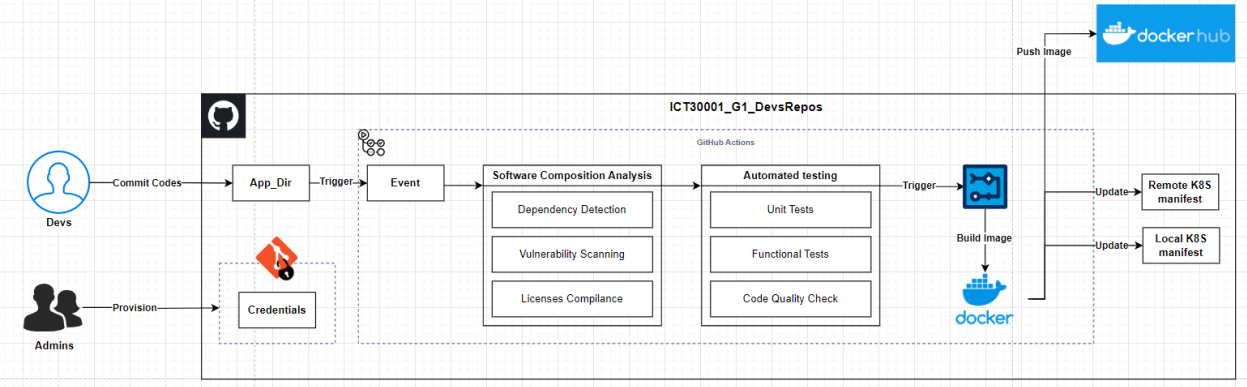

Figure 3.1: CI flow

4. CONFIGURE THE CD PIPELINE

After finishing the CI flow setup, we continue to update the **GitHub Action** workflow with the following scripts to set up the CD pipeline:

```
update-manifest:
  runs-on: ubuntu-latest
  needs: test-image

  steps:
   - name: Check out code
    uses: actions/checkout@v2

   - name: Update Image Tag Values
    run: |
      deployment_file="./k8s_local/deployment.yaml"  Update with the actual path to your deployment.yaml
      new_image_tag=${{ github.sha }}
       Update the deployment.yaml file with the new image tag
```

```
            sed -i "s|image: dixluwn/ict30001-test:.|image: dixluwn/ict30001-test:$new_image_tag|"
"$deployment_file"


  - name: Commit the changes made
    run: |
      git config --global user.name "${{ secrets.GIT_USER_NAME }}"
      git config --global user.email "${{ secrets.GIT_USER_EMAIL }}"
      git commit -am "Updating image tag in deployment.yaml"
      git push
```

Similar to the CI setup, we will also include the following configurations:

Config: Add **GIT_USER_NAME** and **GIT_USER_EMAIL** onto **Git-Secret**.

- Create Manifest files for Kubernetes

This is a sub-section of the CD pipeline setup process. Please execute the 02 provided **YAML** scripts to create the necessary Manifest files for **Kubernetes.**

Script 1: deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: app-server-deployment
spec:
 replicas: 3
 selector:
  matchLabels:
   app: app-server
 template:
  metadata:
   name: app-server
   labels:
    app: app-server
```

```
spec:
  containers:
   - name: app-server
     image: dixluwn/ict30001-test:be391509d4355ab985862f500712b7b8f371fdee
     ports:
      - containerPort: 80
```

Script 2: service.yaml

```
apiVersion: v1
kind: Service
metadata:
 name: app-server-service
 labels:
  app: app-server
spec:
 type: NodePort
 selector:
  app: app-server
 ports:
  - nodePort: 30007
    port: 80
    targetPort: 80
```

*Please note that the provided YAML scripts are for **reference only**. You may need to modify some lines based on your configuration.*

Overall flow of the CD pipeline

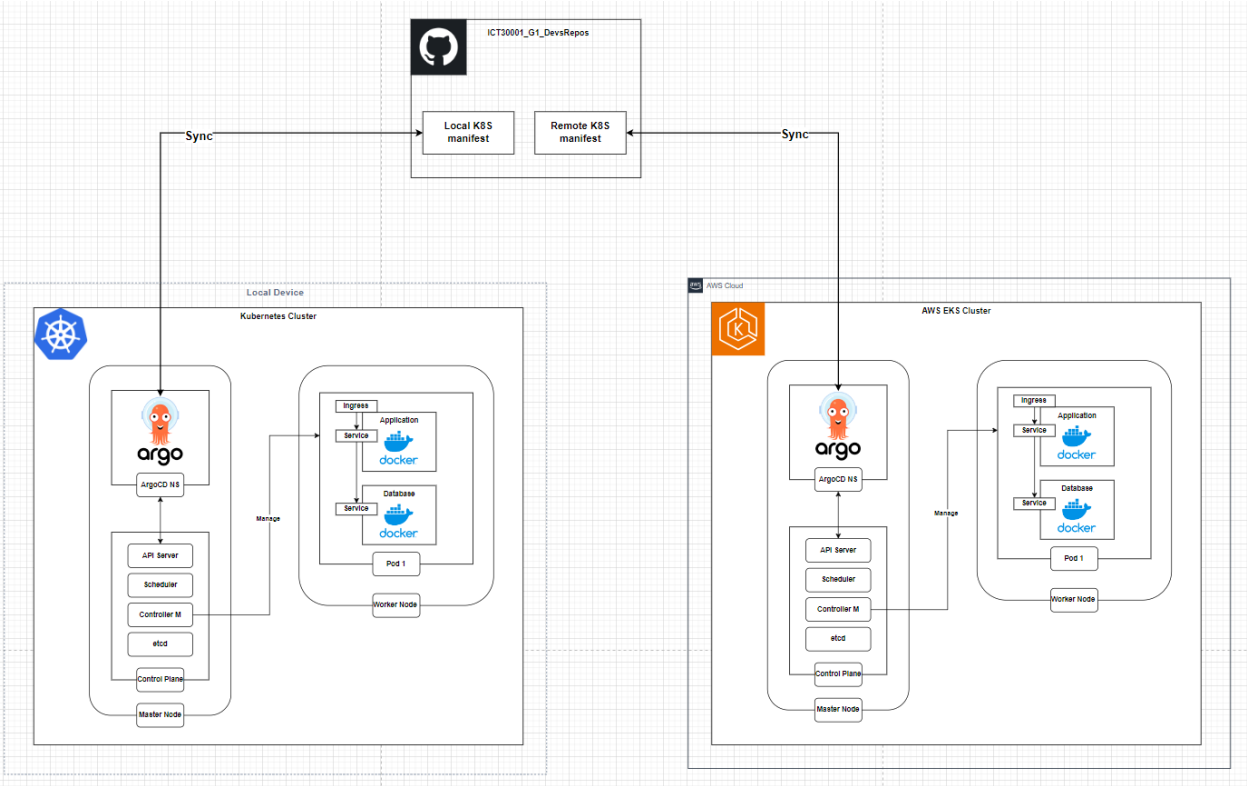

Figure 4.1: CD flow

5. CREATE AWS EKS CLUSTER

In this section, we will set up the EKS cluster with all the necessary roles and compute resources.

5.1. Create Role for EKS Cluster

Step 1: Go to **AWS Management Console**.

Step 2: Navigate to **IAM (Identity and Access Management)**.

Step 3: Click on **Roles** and then select **Create role.**

Step 4: Choose **AWS Service** as the trusted entity.

Step 5: Choose **EKS-cluster** as the use case.

Step 6: Click **Next** and provide a name for the role.

5.2. Create Role for EC2 Instances

Step 1: Go to **AWS Management Console**.

Step 2: Navigate to **IAM (Identity and Access Management)**.

Step 3: Click on **Roles** and then select **Create role**.

Step 4: Choose **AWS Service** as the trusted entity.

Step 5: Choose **EC2** as the use case.

Step 6: Click on **Next**.

Step 7: Add the following policies:

- AmazonEC2ContainerRegistryReadOnly
- AmazonEKS_CNI_Policy
- AmazonEBSCSIDriverPolicy
- AmazonEKSWorkerNodePolicy

Step 8: Provide a name for the role. For example: "myNodeGroupPolicy".

Note: You should give clear and concise names for everything you create, as you will need to use them in the next parts of the implementation.

5.3. Create EKS Cluster

Step 1: Go to **AWS Management Console**.

Step 2: Navigate to **Amazon EKS** service.

Step 3: Select **Create cluster**.

Step 4: Enter your desired name, select version, and specify the role created in section 5.1.

Step 5: Configure the **Security Group, Cluster Endpoint,** etc. based on your demands.

Step 6: Click **Next and Proceed** to create the cluster.

5.4. Create Compute Resources

Step 1: Go to **AWS Management Console.**

Step 2: Navigate to **Amazon EKS** service.

Step 3: Select **Compute** or **Node groups**.

Step 4: Provide a name for the compute resource.

Step 5: Select the role created in section 5.2.

Step 6: Select **Node Type & Size**.

Step 7: Click **Next and Proceed** to create the compute resource.

5.5. Configure Cloud Shell

Step 1: Open **AWS CloudShell** or **AWS CLI**.

Step 2: Execute the command:

```
aws eks update-kubeconfig --name shack-eks --region ap-south-1
```

Step 3: Remember to replace **shack-eks** with the name of your EKS cluster and **ap-south-1** with your region.

6. INSTALL ARGOCD

In this section, all commands provided below will install **ArgoCD** into the specified namespace, set up the service as a **Load Balancer**, and retrieve the admin password for you to access the **ArgoCD UI**.

6.1. Create Namespace for ArgoCD

Execute the following command to create a namespace for **ArgoCD**:

```
bash
kubectl create namespace argocd
```

6.2. Apply ArgoCD Manifests

Execute the following command to apply the manifest file to **ArgoCD**:

```
bash
kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/v2.4.7/manifests/install.yaml
```

6.3. Patch Service Type to Load Balancer

Execute the following command to patch the **Service Type** to the **Load Balancer**:

```
bash
kubectl patch svc argocd-server -n argocd -p '{"spec":{"type":"LoadBalancer"}}'
```

6.4. Retrieve Admin Password

Execute the following command to retrieve the admin password to access the interface of **ArgoCD**:

```
bash
kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}" | base64 -d
```

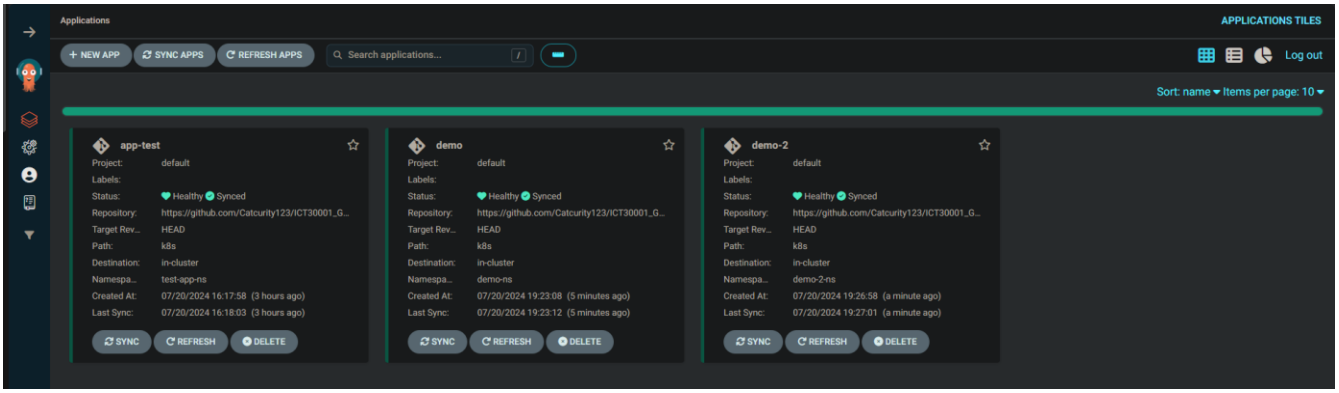

Figure 6.1: The user interface of ArogCD

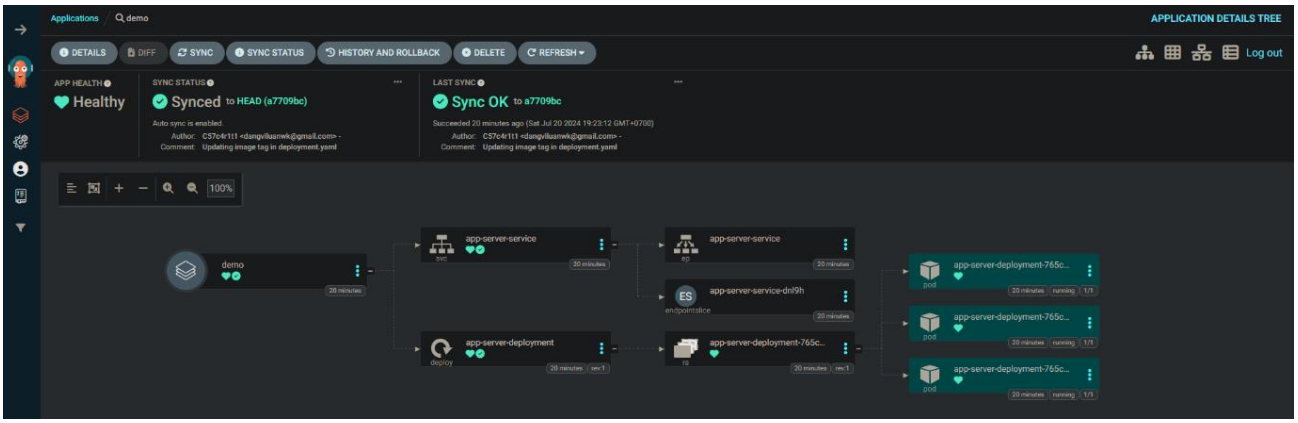

Figure 6.2: An application created in ArgoCD

7. INSTALL GRAFANA AND PROMETHEUS FOR MONITORING

In this section, we will install **Grafana** and **Prometheus** on AWS CloudShell for monitoring purposes. The following scripts are used to set up **Helm**, add necessary **Helm** repositories, and deploy **Grafana** and **Prometheus**.

7.1. Install Helm

Helm is a package manager for Kubernetes that helps you manage Kubernetes applications. We will install Helm using the following command:

```
curl -fsSL https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash
```

7.2. Add Helm Repositories

We need to add the repositories for **Prometheus, Grafana**, and **Ingress-Nginx** to **Helm**. These repositories contain the **Helm** charts for the respective applications.

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

helm repo add grafana https://grafana.github.io/helm-charts

helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx

helm repo update
```

7.3. Install Prometheus

Prometheus is a monitoring and alerting toolkit. We will install it using the Helm chart from the Prometheus Community repository.

```
helm install prometheus prometheus-community/kube-prometheus-stack -n monitoring --
create-namespace
```

7.4. Install Grafana

Grafana is an open-source platform for monitoring and observability. We will install it using the Helm chart from the Grafana repository.

```
helm install grafana grafana/grafana -n monitoring --create-namespace
```

Confirm the Installation

After installing **Prometheus** and **Grafana**, we can confirm that the services are running by checking the services in the monitoring namespace.

```
kubectl get svc -n monitoring
```

7.5. Edit Prometheus Service

By default, the **Prometheus** server service is of type **NodePort**. We need to change it to **LoadBalancer** to make it accessible from outside the cluster.

```
kubectl edit svc prometheus-kube-prometheus-prometheus -n monitoring
```

7.6. Edit Grafana Service

Similarly, the **Grafana** service type needs to be changed from **NodePort** to **LoadBalancer**.

```
kubectl edit svc grafana -n monitoring
```

By following these steps, we set up **Prometheus** and **Grafana** for monitoring our **Kubernetes** cluster using **AWS CloudShell**.

Dashboard IDs to try:

k8s-addons-prometheus.json 19105

k8s-system-api-server.json 15761

k8s-system-coredns.json 15762

k8s-views-global.json 15757

k8s-views-namespaces.json 15758

k8s-views-nodes.json 15759

k8s-views-pods.json 15760

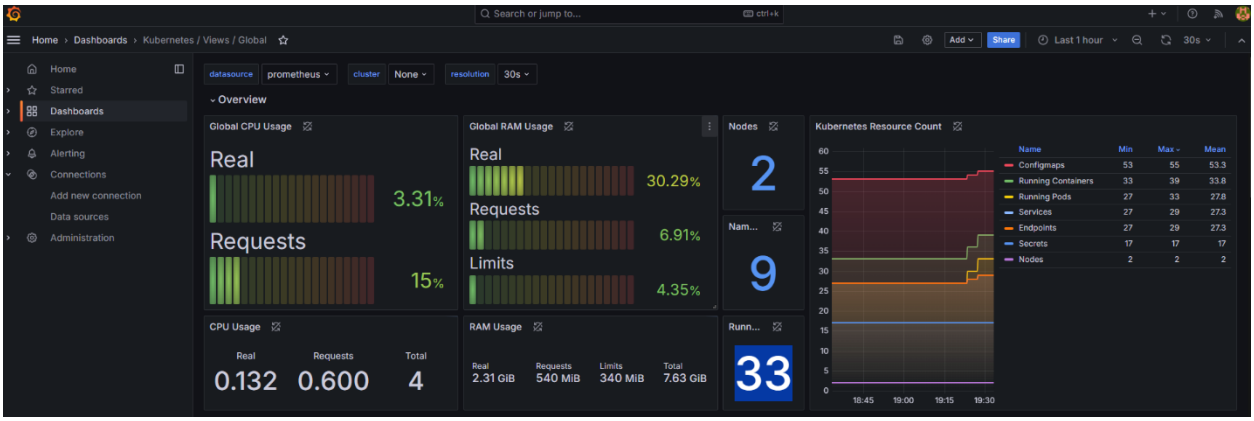

Figure 7.1: Global view of Grafana

TUTORIAL VIDEOS

Now you have set up the CI/CD pipeline. In the next part, we will explore how to complete basic tasks with the CI/CD pipeline. In the tutorial videos attached below, we will guide you through step-by-step instructions on various aspects of using the pipeline to create an application, manage deployments, and monitor pipeline health.

Video 1: How to make changes to local and remote cluster

Link to video

Video 2: How to create an application on ArgoCD server

Link to video

Video 3: How to import dashboards to track the states of cluster

Link to video