**Node.js application with GitLab, Docker, Kubernetes, AWS, Helm charts, Prometheus, and Grafana on an Ubuntu machine**

**Prerequisites**

1. **Ubuntu Machine (or VM)**
2. **GitLab Account**
3. **AWS Account**
4. **Docker** and **Docker Compose**
5. **Kubernetes Cluster (can use EKS, Minikube, or local setup like Docker Desktop)**
6. **Helm** for Kubernetes package management
7. **Prometheus** and **Grafana** for monitoring

**Step 1: Install Prerequisites on Ubuntu**

1. **Install Git**

sudo apt update

sudo apt install git

1. **Install Docker**

sudo apt install apt-transport-https ca-certificates curl software-properties-common

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable"

sudo apt update

sudo apt install docker-ce

sudo systemctl enable docker

sudo systemctl start docker

1. **Install Docker Compose**

sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose

1. **Install kubectl** (Kubernetes CLI)

sudo apt update

sudo apt install -y apt-transport-https

sudo curl -fsSL https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -

sudo echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee -a /etc/apt/sources.list.d/kubernetes.list

sudo apt update

sudo apt install -y kubectl

1. **Install Helm** (Kubernetes package manager)

curl https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 |

1. **Install AWS CLI**

sudo apt install awscli

1. **Install Prometheus & Grafana on Kubernetes** (for monitoring) Prometheus and Grafana will be installed later via Helm charts.

**Step 2: Set Up GitLab**

1. **Create a repository on GitLab**
   * Log in to [GitLab](https://gitlab.com/) and create a new repository.
2. **Clone the GitLab repo to your local machine**

git clone https://gitlab.com/your-username/your-repo.git

cd your-repo

1. **Add your Node.js project**
   * Create a simple Node.js app in the repository folder:

mkdir app

cd app

npm init -y

npm install express

* + Create a simple server file:

// app/server.js

const express = require('express');

const app = express();

const port = process.env.PORT || 3000;

app.get('/', (req, res) => res.send('Hello World!'));

app.listen(port, () => console.log(`Server listening at http://localhost:${port}`));

1. **Push your code to GitLab**

git add .

git commit -m "Initial commit"

git push origin main

**Step 3: Set Up Docker for the Node.js App**

1. **Create a Dockerfile** in the root of your project:

# Dockerfile

FROM node:16

WORKDIR /usr/src/app

COPY package\*.json ./

RUN npm install

COPY . .

EXPOSE 3000

CMD ["node", "app/server.js"]

1. **Build and test the Docker image locally**

docker build -t node-app .

docker run -p 3000:3000 node-app

Your app should be accessible at http://localhost:3000.

1. **Push the Docker image to Docker Hub (or other registry)**

docker login

docker tag node-app your-dockerhub-username/node-app:latest

docker push your-dockerhub-username/node-app:latest

**Step 4: Set Up Kubernetes Cluster (Using AWS EKS)**

1. **Install eksctl**

curl --silent --location "https://github.com/weaveworks/eksctl/releases/download/v0.55.0/eksctl\_Linux\_amd64.tar.gz" | tar xz -C /tmp

sudo mv /tmp/eksctl /usr/local/bin

1. **Create an EKS cluster**

eksctl create cluster --name node-app-cluster --region us-west-2 --nodegroup-name node-group --node-type t2.medium --nodes 2 --nodes-min 1 --nodes-max 3 --managed

1. **Configure kubectl to use the new cluster**

aws eks --region us-west-2 update-kubeconfig --name node-app-cluster

1. **Check if Kubernetes is running**

kubectl get nodes

**Step 5: Deploy the Node.js App on Kubernetes Using Helm**

1. **Create a Helm chart for the Node.js app**
   * Use helm create node-app to scaffold a Helm chart for your application.
   * Modify the values.yaml file to use the Docker image:

image:

repository: your-dockerhub-username/node-app

tag: latest

1. **Install the Helm chart to your Kubernetes cluster**

helm install node-app ./node-app

1. **Verify the deployment**

kubectl get pods

kubectl get svc

1. **Expose the app via a LoadBalancer** You can modify the service.yaml file in your Helm chart to use a LoadBalancer for external access.

apiVersion: v1

kind: Service

metadata:

name: node-app-service

spec:

selector:

app: node-app

ports:

- protocol: TCP

port: 80

targetPort: 3000

type: LoadBalancer

Apply the change:

kubectl apply -f service.yaml

Get the external IP address:

kubectl get svc node-app-service

**Step 6: Install Prometheus and Grafana with Helm**

1. **Add the Prometheus and Grafana Helm repositories**

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

helm repo update

1. **Install Prometheus and Grafana**

helm install prometheus prometheus-community/prometheus

helm install grafana prometheus-community/grafana

1. **Access Grafana dashboard**
   * Port-forward to access the Grafana UI:

kubectl port-forward service/grafana 3000:80

* + Open your browser at http://localhost:3000 and log in using the default credentials (admin/admin).

1. **Configure Prometheus as a data source in Grafana**
   * In Grafana, go to **Configuration > Data Sources > Add Data Source** and select **Prometheus**. Set the URL as http://prometheus-server:80.

**Step 7: Automate Deployment with GitLab CI/CD**

1. **Create a .gitlab-ci.yml file in your repository**

stages:

- build

- deploy

build:

stage: build

script:

- docker build -t $CI\_REGISTRY\_IMAGE .

- docker push $CI\_REGISTRY\_IMAGE

deploy:

stage: deploy

script:

- kubectl apply -f k8s/

1. **Set up GitLab CI/CD environment variables** in your GitLab repository to manage credentials for Docker and Kubernetes.

**Step 8: Monitor and Scale Your App**

1. **Use Prometheus to monitor your application** and set up alerts.
2. **Use Grafana dashboards** to visualize the data collected by Prometheus.

**Step-by-Step Setup of a Kubernetes Cluster Using AWS EKS**

**1. Prerequisites**

* **AWS Account:** Create an AWS account if you don't already have one: [AWS Signup](https://aws.amazon.com/)
* **IAM User:** Create an IAM user with AdministratorAccess. Generate and download Access Key ID and Secret Access Key.
* **Ubuntu Machine:** Local machine or an EC2 instance with:
  + **AWS CLI** (aws)
  + **eksctl** (eksctl)
  + **kubectl** (kubectl)

**1.1. Install Required Tools**

1. **Install AWS CLI**

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

unzip awscliv2.zip

sudo ./aws/install

aws --version

1. **Configure AWS CLI**

aws configure

Provide your Access Key ID, Secret Access Key, Region, and Output format (json).

1. **Install eksctl**

curl -LO "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl\_$(uname -s)\_amd64.tar.gz"

tar -xzf eksctl\_\*.tar.gz -C /tmp

sudo mv /tmp/eksctl /usr/local/bin

eksctl version

1. **Install kubectl**

curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

kubectl version --client

**2. Create an EKS Cluster**

**2.1. Create a Cluster Using eksctl**

eksctl create cluster \

--name my-eks-cluster \

--region us-west-2 \

--nodegroup-name my-node-group \

--node-type t3.medium \

--nodes 2 \

--nodes-min 1 \

--nodes-max 3 \

--managed

* **Cluster Name:** my-eks-cluster
* **Region:** us-west-2 (Change to your preferred region)
* **Node Group:** my-node-group
* **Instance Type:** t3.medium
* **Node Count:** Minimum 1, Desired 2, Maximum 3

**2.2. Verify the EKS Cluster**

kubectl get nodes

You should see your worker nodes in a Ready state.

**3. Configure kubectl for EKS**

aws eks --region us-west-2 update-kubeconfig --name my-eks-cluster

* Check the connection:

kubectl get svc

**4. Test the Kubernetes Cluster**

**4.1. Deploy a Test Application**

# nginx-deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

spec:

replicas: 2

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:1.21.6

ports:

- containerPort: 80

kubectl apply -f nginx-deployment.yaml

**4.2. Expose the Deployment as a Service**

# nginx-service.yaml

apiVersion: v1

kind: Service

metadata:

name: nginx-service

spec:

type: LoadBalancer

ports:

- port: 80

targetPort: 80

selector:

app: nginx

kubectl apply -f nginx-service.yaml

* **Get External IP:**

kubectl get services

* Access the application in the browser using the External IP.

**5. Manage EKS Cluster**

**5.1. Scale the Deployment**

kubectl scale deployment nginx-deployment --replicas=4

kubectl get pods

**5.2. Delete the Test Resources**

kubectl delete -f nginx-service.yaml

kubectl delete -f nginx-deployment.yaml

**6. Clean Up Resources**

**6.1. Delete the EKS Cluster**

eksctl delete cluster --name my-eks-cluster

**7. Additional Configurations (Optional)**

* **Set Up Autoscaling:** Integrate with the Kubernetes Cluster Autoscaler.
* **Enable Monitoring:** Use AWS CloudWatch or install Prometheus & Grafana.
* **Add Persistent Storage:** Configure EBS, EFS, or S3 as storage backends.

**create an Access Key ID, Secret Access Key, and set the Region in AWS**

**1. Create an IAM User with Programmatic Access**

**Step 1: Log in to AWS Management Console**

* Go to [AWS Management Console](https://aws.amazon.com/).
* Sign in with your credentials.

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

* In the AWS Console, search for and select **IAM**.

**Step 3: Create a New IAM User**

1. Go to **Users** > **Add users**.
2. Enter a **User name**, e.g., my-eks-user.
3. **Select Access Type:**
   * Check **Access key - Programmatic access** to enable CLI, SDK, and API access.

**Step 4: Set Permissions**

**Option 1: Attach Policies Directly**

* Choose **AdministratorAccess** for full permissions (for testing/dev purposes).

AdministratorAccess

**Option 2: Use Groups (Recommended)**

1. Create a **Group** with **AdministratorAccess** or specific required policies.
2. Add the new user to this group.

**Step 5: Review and Create User**

* Skip tags (optional).
* Review the user and click **Create user**.

**Step 6: Download Access Key ID & Secret Access Key**

* **Important:** Copy or download the .csv file containing:
  + **Access Key ID** (e.g., AKIAIOSFODNN7EXAMPLE)
  + **Secret Access Key** (e.g., wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY)

**2. Configure AWS CLI with Access Key and Region**

**Step 1: Install AWS CLI (If not already installed)**

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

unzip awscliv2.zip

sudo ./aws/install

**Step 2: Configure AWS CLI**

aws configure

* **Enter Access Key ID:**

AKIAIOSFODNN7EXAMPLE

* **Enter Secret Access Key:**

wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

* **Enter Default Region Name:** (e.g., us-west-2)

us-west-2

* **Enter Default Output Format:** (e.g., json)

json

**Step 3: Verify the Configuration**

aws sts get-caller-identity

This should return information about your AWS account if the credentials are correct.

**3. Manually Set or Change Default Region**

You can manually set the default region if needed:

aws configure set region us-west-2

Or modify the ~/.aws/config file manually:

[default]

region = us-west-2

**4**

**To Delete an Access Key:**

aws iam delete-access-key --access-key-id <ACCESS\_KEY\_ID>

**To expose your application via a LoadBalancer in Kubernetes using a Helm chart, follow these steps:**

**1. Modify the service.yaml File in the Helm Chart**

**Update service.yaml Configuration**

Edit the service.yaml file located in your Helm chart under the templates directory:

# templates/service.yaml

apiVersion: v1

kind: Service

metadata:

name: {{ .Release.Name }}-service

labels:

app: {{ .Chart.Name }}

spec:

type: LoadBalancer # Set the service type to LoadBalancer

ports:

- port: 80 # External port

targetPort: 3000 # Internal port where your app runs

protocol: TCP

selector:

app: {{ .Release.Name }}

**Key Configuration:**

* **type: LoadBalancer**: Creates an external load balancer in your cloud provider (e.g., AWS, GCP, Azure).
* **port**: The port that will be exposed externally (e.g., 80).
* **targetPort**: The port your application is listening on inside the container (e.g., 3000).

**2. Apply the Helm Chart**

**Install or Upgrade the Helm Release**

# Install a new release

helm install my-app ./my-app-chart

# OR Upgrade an existing release

helm upgrade my-app ./my-app-chart

* Replace my-app with your release name and ./my-app-chart with the path to your Helm chart.

**Verify the Deployment**

kubectl get all

Check that the service type is **LoadBalancer** and look for the **EXTERNAL-IP**:

kubectl get services

**3. Access the Application**

* Copy the **EXTERNAL-IP** or the **DNS name** of the LoadBalancer.
* Open the app in your browser, e.g., http://a1b2c3d4e5f6.us-west-2.elb.amazonaws.com.

**4. For AWS EKS: Ensure LoadBalancer Permissions**

When using **AWS EKS**, your worker nodes need the correct IAM policies for the LoadBalancer:

1. **Attach the Policy to the EKS Worker Role**:

{

"Effect": "Allow",

"Action": [

"ec2:DescribeInstances",

"ec2:DescribeRegions",

"ec2:DescribeSecurityGroups",

"ec2:DescribeSubnets",

"ec2:DescribeVpcs",

"elasticloadbalancing:\*"

],

"Resource": "\*"

}

1. **Verify the LoadBalancer Creation:**

kubectl describe service my-app-service

**5. Update Values.yaml for Helm Customization (Optional)**

Instead of directly editing service.yaml, you can manage configuration through values.yaml:

# values.yaml

service:

type: LoadBalancer

port: 80

targetPort: 3000

Update the service.yaml template:

type: {{ .Values.service.type }}

ports:

- port: {{ .Values.service.port }}

targetPort: {{ .Values.service.targetPort }}

This method allows you to override values during the Helm install/upgrade:

helm upgrade my-app ./my-app-chart --set service.type=LoadBalancer

**6. Troubleshooting**

**Check Service Status:**

kubectl get service my-app-service

**Describe the Service:**

kubectl describe service my-app-service

**Check Pod Logs:**

kubectl logs <pod-name>