CarTrawler Assessment

GitHub repository : https://github.com/Ganeshru/car_trawler_assessment
Docker hub repository : https://hub.docker.com/repositories/ganeshru

Prerequisite

1. Docker Desktop Installation

- **Install:** Download from docker.com/products/docker-desktop, run installer.
- **Verify:** Open terminal, run `docker --version` and `docker run hello-world`.

2. Minikube Installation

- **Pre-requisites:** Docker Desktop running, `kubectl` installed.
- **Install:** Download from minikube.sigs.k8s.io/docs/start/.
- **Start Minikube:** In terminal, run `minikube start --driver=docker`.
- **Verify:** `minikube status` and `kubectl cluster-info`.

3. Jenkins Setup in Minikube

- **Pre-requisites:** Minikube running, `kubectl` configured.
- **Install Helm:** Follow helm.sh/docs/intro/install/.
- **Install Jenkins via Helm:**

Unset

```
helm repo add jenkins https://charts.jenkins.io
helm repo update
helm install jenkins jenkins/jenkins --namespace jenkins
--create-namespace
```

Get Admin Password:

Shell

```
kubectl exec --namespace jenkins -it $(kubectl get pods --namespace
jenkins -l app.kubernetes.io/component=jenkins-master -o
jsonpath='{.items[0].metadata.name}') -- cat
/run/secrets/additional/jenkins-admin-password
```

Access Jenkins UI:

- In a **new terminal**, run: `kubectl --namespace jenkins port-forward svc/jenkins 8080:8080`
- Open `http://localhost:8080` in your browser and use the retrieved password to log in.

2. Docker commands used in Jenkins

Jenkins Docker Agent Image (`ganeshru/jenkins-one-agent:latest`)

This Docker image is a custom Jenkins agent packed with essential DevOps tools for the pipelines:

- **Tools Included:** Docker CLI, Kubectl, Minikube, Node.js/npm, yamllint, kubeval, and Git.
- **Purpose:** Enables Jenkins to perform tasks like Docker builds, Kubernetes deployments, Node.js application builds, and YAML/Kubernetes config validation directly from the agent.
- **Crucial Setup:** The `DOCKER_GID` build argument (`--build-arg DOCKER_GID=999`) must match your host's Docker group ID for Docker integration to work within the agent.

Dockerfile.oneagent

Shell
```
# Dockerfile for a comprehensive Jenkins Agent
# This agent will contain: Docker CLI, Kubectl, Minikube, Node.js/npm, yamllint,
kubeval, and git.
# Base the agent on the official Jenkins Inbound Agent image
FROM jenkins/inbound-agent:latest
# Set environment variables to prevent interactive prompts during apt installs
ENV DEBIAN_FRONTEND=noninteractive
# Switch to root user for installations
USER root
# 1. Update package lists and install common dependencies
RUN apt-get update \
    && apt-get install -y --no-install-recommends \
        curl \
        gnupg \
        lsb-release \
        ca-certificates \
        build-essential \
        git \
        jq \
    && rm -rf /var/lib/apt/lists/*
# 2. Install Docker CLI
RUN curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --dearmor -o
/usr/share/keyrings/docker-archive-keyring.gpg \
    && echo "deb [arch=$(dpkg --print-architecture)
signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]
```

```
https://download.docker.com/linux/debian $(lsb_release -cs) stable" | tee
/etc/apt/sources.list.d/docker.list > /dev/null \
    && apt-get update \
    && apt-get install -y docker-ce-cli
# 3. Install kubectl
RUN curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl" \
    && install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl \
    && rm kubectl
# 4. Install Minikube CLI (assuming amd64 architecture)
RUN curl -LO
https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64 \
    && install minikube-linux-amd64 /usr/local/bin/minikube \
    && rm minikube-linux-amd64
# 5. Install Node.js and npm (for your 'app' directory tests)
RUN curl -fsSL https://deb.nodesource.com/setup_16.x | bash - \
    && apt-get install -y nodejs
# 6. Install Python and pip (required for yamllint if not available via apt)
RUN apt-get update && apt-get install -y --no-install-recommends \
    python3 \
    python3-pip \
    && rm -rf /var/lib/apt/lists/*
# 7. Install yamllint (and potentially kubeval if it becomes available via apt)
#    Installing yamllint directly via apt as recommended by PEP 668.
#    Note: kubeval is typically a standalone binary, so it remains installed via
curl.
RUN apt-get update \
    && apt-get install -y --no-install-recommends \
        yamllint \
    && rm -rf /var/lib/apt/lists/*
# 8. Install kubeval
#    Downloads the latest kubeval binary and places it in /usr/local/bin.
RUN curl -LO
https://github.com/instrumenta/kubeval/releases/latest/download/kubeval-linux-amd64.
tar.gz \
    && tar -xzf kubeval-linux-amd64.tar.gz \
    && mv kubeval /usr/local/bin/kubeval \
    && rm kubeval-linux-amd64.tar.gz
# Verify installations (optional, but good for debugging)
RUN yamllint --version
RUN kubeval --version
# ... (rest of your Dockerfile) ...
# 9. Configure Docker group permissions for the Jenkins user
#    This is CRUCIAL for the Jenkins agent to interact with the Docker daemon on the
host.
#    You MUST replace 999 with the actual GID of the 'docker' group on your *Jenkins
host machine*.
```

```
#    To find this GID on your host, run: getent group docker | cut -d: -f3
ARG DOCKER_GID=999 # Default placeholder. IMPORTANT: CHANGE THIS!
RUN groupadd -g ${DOCKER_GID} docker || true \
    && usermod -aG docker jenkins
# Clean up apt caches to reduce image size
RUN apt-get clean \
    && rm -rf /var/lib/apt/lists/*
# Switch back to the default jenkins user for security
USER jenkins
# Set default shell
ENV SHELL=/bin/bash
# Define a working directory for consistency, although Jenkins typically sets its
own workspace.
WORKDIR /home/jenkins/agent
```

Commands:

Shell

```shell
# build image

docker build --build-arg DOCKER_GID=999 -t ganeshru/jenkins-one-agent:latest -f
Dockerfile.oneagent .

# login to docker hub

docker login

# push image to docker hub

docker push ganeshru/jenkins-one-agent:latest
```

Kubernetes Configuration for sample-node-app

This document summarizes the Kubernetes YAML configurations (k8s/*) used to deploy a Node.js application. This setup is particularly suited for local testing with Minikube.

Overview

The configuration defines how the sample-node-app is deployed, configured, and exposed within a Kubernetes cluster. It comprises a ConfigMap and Secret for application data, a Deployment for managing the application pods, a Service for internal exposure, and an Ingress for external access.

Key Components

1. ConfigMap (app-config)

- Purpose: Stores non-sensitive configuration data for the application.

- Data: Contains app_message (e.g., "Hello from the Node.js app...") and environment ("development").

2. Secret (app-secret)

- Purpose: Securely stores sensitive application data.
- Data: Includes api_key and db_password, stored in Base64 encoded format.

3. Deployment (sample-node-app-deployment)

- Purpose: Manages the lifecycle of the application's pods.
- Details:
    - Deploys a single replica of the sample-node-app:latest Docker image.
    - Uses imagePullPolicy: Never to prioritize a locally built image, which is common during development with Minikube.
    - Exposes container port 8000.
    - Injects environment variables from both the app-config ConfigMap and app-secret Secret.
    - Includes readinessProbe and livenessProbe to ensure the application is healthy and ready to receive traffic

○ **4. Service (sample-node-app-service)**

- Purpose: Provides a stable network endpoint for the sample-node-app Deployment.
- Details:
    - Type: NodePort, allowing easy access to the application from outside the Kubernetes cluster (ideal for Minikube).
    - Routes traffic from service port 80 to the application's container port 8000.

5. Ingress (sample-node-app-ingress)

- Purpose: Manages external access to the sample-node-app-service based on hostnames.
- Details:
    - Exposes the application via the hostname sample-node-app.info.
    - Includes annotations for NGINX Ingress Controller (nginx.ingress.kubernetes.io/rewrite-target: /).

Overall Objective

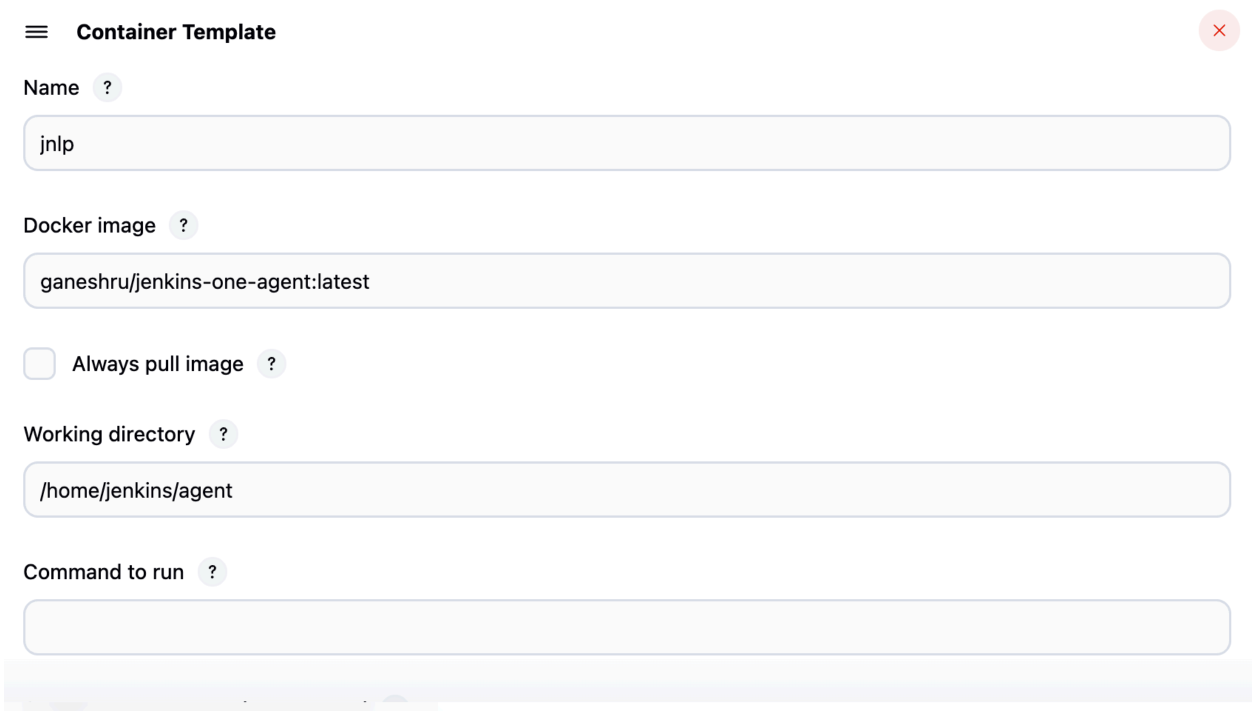
These configurations streamline the deployment and management of a Node.js application within a Kubernetes cluster, demonstrating best practices for handling configuration, secrets, and external access in a development or local environment.

Jenkinsfile Usage:

Configure Jenkins one-agent

Key Change: I have successfully updated the `Docker image` field from its default value (likely `jenkins/inbound-agent`) to your custom-built image: `ganeshru/jenkins-one-agent:latest`.

This means that any Jenkins pipeline or job configured to use this "jnlp" container template will now run inside your specialized Docker image, giving it access to all the tools (Docker CLI, Kubectl, Node.js, etc.) you bundled within it.



Container Template

Name: jnlp

Docker image: ganeshru/jenkins-one-agent:latest

Always pull image: ☐

Working directory: /home/jenkins/agent

Command to run:

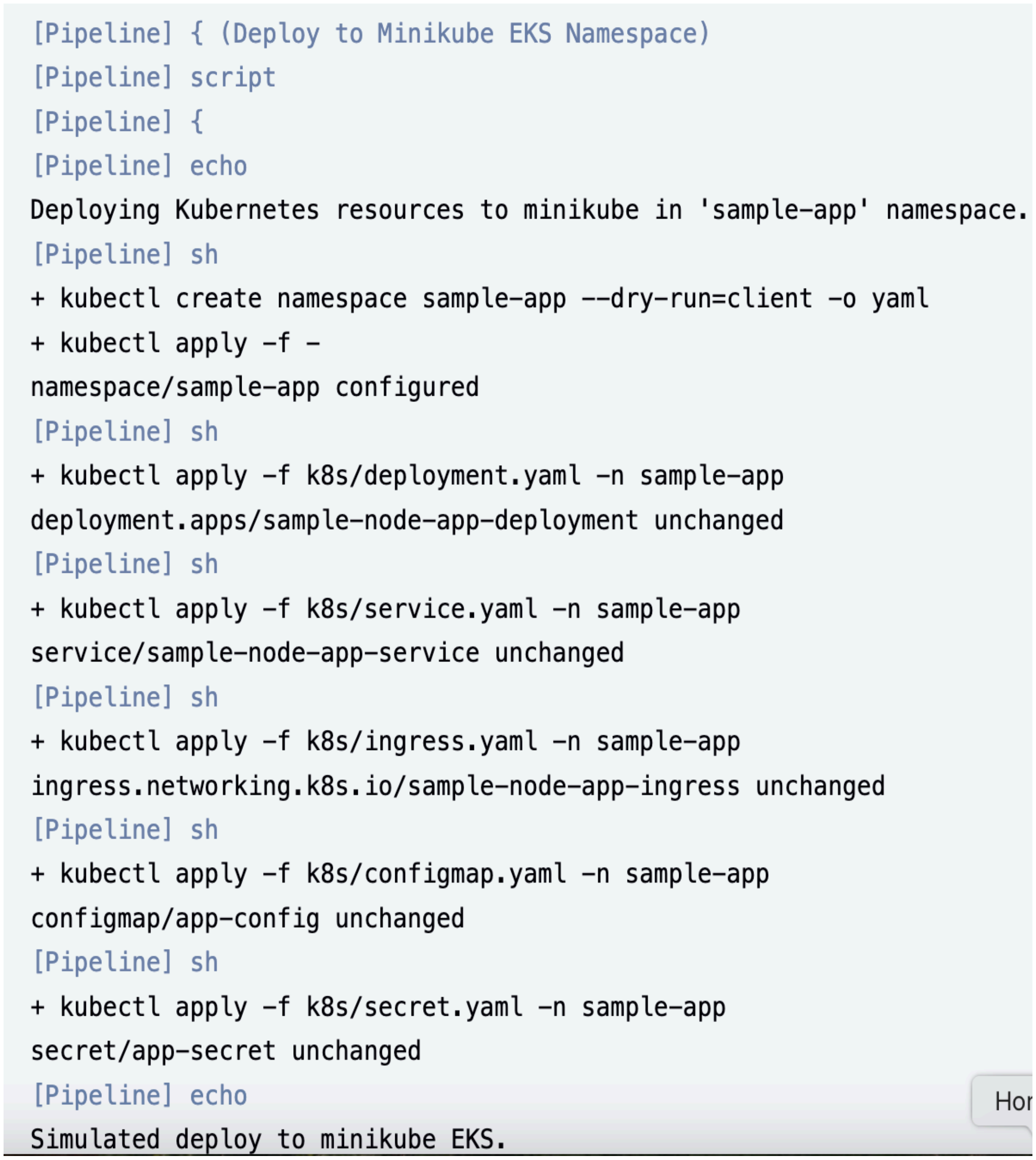
Jenkinsfile Summary (jenkins/Jenkinsfile)

This Jenkinsfile defines a CI/CD pipeline for a Node.js application, designed to run on a custom Jenkins agent with Docker, Kubectl, and Minikube installed.

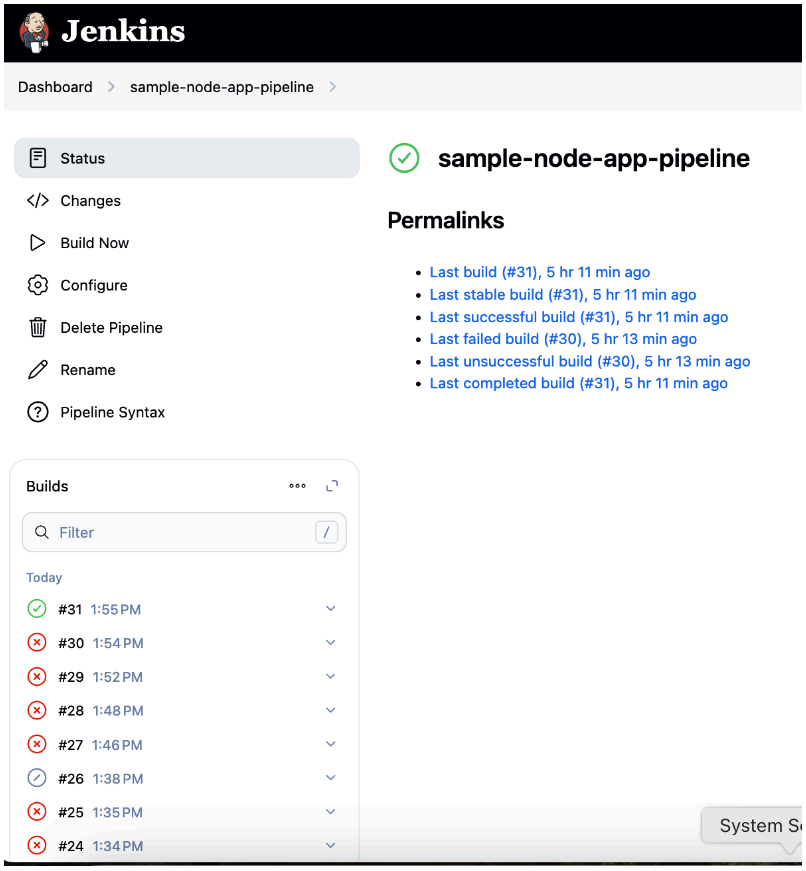
Key Stages:

1. Checkout: Retrieves the source code from version control.
2. Lint: A placeholder stage.
3. Test: Runs npm install and npm test within the application directory to validate the Node.js code.
4. Build Docker Image: Builds the Node.js application into a Docker image (e.g., sample-node-app:BUILD_NUMBER) using the Docker client available on the agent.

5. Deploy to Minikube:
    - Creates a Kubernetes namespace called sample-app.
    - Applies the Kubernetes YAML files (Deployment, Service, Ingress, ConfigMap, Secret) from the k8s/ directory to the Minikube cluster within the sample-app namespace. This effectively deploys the application.
6. Helm Deploy (Optional): A placeholder stage indicating where a Helm deployment to a staging/production environment would occur.
7. Notify: Simulates a notification (e.g., Slack/email) upon pipeline completion.



```
[Pipeline] { (Deploy to Minikube EKS Namespace)
[Pipeline] script
[Pipeline] {
[Pipeline] echo
Deploying Kubernetes resources to minikube in 'sample-app' namespace.
[Pipeline] sh
+ kubectl create namespace sample-app --dry-run=client -o yaml
+ kubectl apply -f -
namespace/sample-app configured
[Pipeline] sh
+ kubectl apply -f k8s/deployment.yaml -n sample-app
deployment.apps/sample-node-app-deployment unchanged
[Pipeline] sh
+ kubectl apply -f k8s/service.yaml -n sample-app
service/sample-node-app-service unchanged
[Pipeline] sh
+ kubectl apply -f k8s/ingress.yaml -n sample-app
ingress.networking.k8s.io/sample-node-app-ingress unchanged
[Pipeline] sh
+ kubectl apply -f k8s/configmap.yaml -n sample-app
configmap/app-config unchanged
[Pipeline] sh
+ kubectl apply -f k8s/secret.yaml -n sample-app
secret/app-secret unchanged
[Pipeline] echo
Simulated deploy to minikube EKS.
```

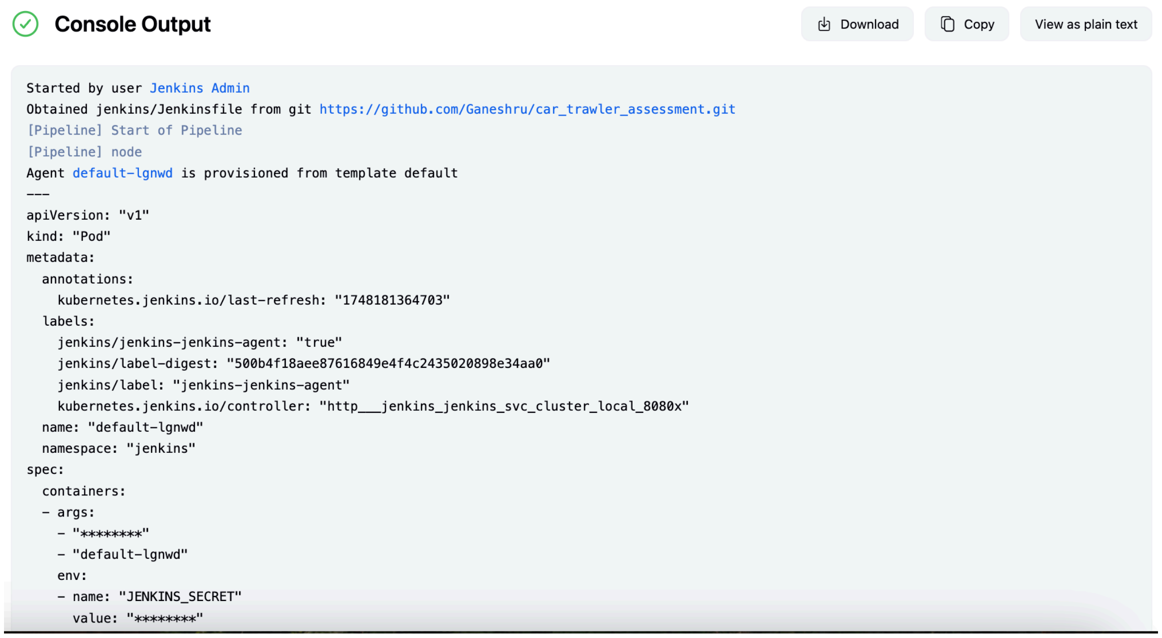


sample-node-app-pipeline

Permalinks

- Last build (#31), 5 hr 11 min ago
- Last stable build (#31), 5 hr 11 min ago
- Last successful build (#31), 5 hr 11 min ago
- Last failed build (#30), 5 hr 13 min ago
- Last unsuccessful build (#30), 5 hr 13 min ago
- Last completed build (#31), 5 hr 11 min ago

```
#31 1:55 PM
#30 1:54 PM
#29 1:52 PM
#28 1:48 PM
#27 1:46 PM
#26 1:38 PM
#25 1:35 PM
#24 1:34 PM
```



Console Output

```
Started by user Jenkins Admin
Obtained jenkins/Jenkinsfile from git https://github.com/Ganeshru/car_trawler_assessment.git
[Pipeline] Start of Pipeline
[Pipeline] node
Agent default-lgnwd is provisioned from template default
---
apiVersion: "v1"
kind: "Pod"
metadata:
  annotations:
    kubernetes.jenkins.io/last-refresh: "1748181364703"
  labels:
    jenkins/jenkins-jenkins-agent: "true"
    jenkins/label-digest: "500b4f18aee87616849e4f4c2435020898e34aa0"
    jenkins/label: "jenkins-jenkins-agent"
    kubernetes.jenkins.io/controller: "http___jenkins_jenkins_svc_cluster_local_8080x"
  name: "default-lgnwd"
  namespace: "jenkins"
spec:
  containers:
  - args:
    - "********"
    - "default-lgnwd"
    env:
    - name: "JENKINS_SECRET"
      value: "********"
```

Environment Variables: Defines variables like AWS_REGION, ECR_REPO (placeholder), IMAGE_TAG (based on build number), and KUBECONFIG.

Agent Configuration:

- agent any: Specifies that the pipeline will run on any available agent. It implicitly relies on the custom Jenkins agent configured to have Docker CLI, kubectl, minikube, and access to the Docker socket.

Post-build Actions:

- always: Executes actions regardless of pipeline status (e.g., "Pipeline completed.").
- failure: Executes actions only if the pipeline fails (e.g., "Initiating rollback (if implemented).").

Overall: This pipeline automates the steps from code checkout to building a Docker image and deploying the application to a local Minikube Kubernetes cluster.

Terraform configuration files:

Terraform tf file : This Terraform script sets up a VPC and an EKS cluster in AWS. The VPC is created with custom CIDR, subnets, NAT gateway, and DNS settings using terraform-aws-modules/vpc/aws. The EKS cluster is deployed using terraform-aws-modules/eks/aws, with private subnets, public API access (0.0.0.0/0), and a managed node group. It attaches necessary IAM policies and installs core addons like CoreDNS, kube-proxy, and VPC CNI. Tags are applied to all resources.

Output file : This Terraform script defines output values to expose key information after deploying the EKS and VPC resources.

cluster_name: Outputs the name of the EKS cluster.
kubeconfig: Provides kubeconfig data for accessing the cluster (sensitive).
cluster_endpoint: Outputs the API endpoint URL of the EKS cluster.
cluster_security_group_id: Outputs the security group ID associated with the cluster.
node_group_roles_arns: Lists the IAM role ARNs used by the EKS managed node groups.
vpc_id: Outputs the ID of the created VPC.
private_subnet_ids: Outputs the IDs of the private subnets.
public_subnet_ids: Outputs the IDs of the public subnets.

Provider file : This Terraform snippet configures the required AWS provider and sets it up for use.

- It specifies the AWS provider from HashiCorp with a version constraint around 5.0 to ensure compatibility
- The AWS provider block configures the provider to use a region defined by the variable `aws_region`, making the region configurable.

Variables file : This Terraform code defines variables for AWS region, EKS cluster name and version, VPC CIDR, subnet CIDRs, availability zones, EKS node instance type and scaling sizes, plus resource tags with default values.

Scripts/deploy.sh

This `deploy.sh` script provides a robust and repeatable way to deploy Kubernetes applications locally, making it particularly useful for development and testing purposes on platforms like Minikube.

It handles prerequisite checks, performs linting on both the script itself and your Kubernetes YAML files, and then safely applies your Kubernetes resources, complete with error handling and verification steps.

Sample Output:

```shell
bash scripts/deploy.sh

INFO: Starting pre-deployment checks...
INFO: Linting deploy.sh script with shellcheck...
INFO: Linting Kubernetes YAML files in k8s/ with kubeval...
PASS - k8s/deployment.yaml contains a valid Deployment (sample-node-app-deployment)
PASS - k8s/service.yaml contains a valid Service (sample-node-app-service)
PASS - k8s/configmap.yaml contains a valid ConfigMap (app-config)
PASS - k8s/secret.yaml contains a valid Secret (app-secret)
PASS - k8s/ingress.yaml contains a valid Ingress (sample-node-app-ingress)
INFO: Kubeval passed for Kubernetes YAML files.
INFO: Starting deployment to Minikube EKS namespace: sample-app
INFO: Ensuring Kubernetes namespace 'sample-app' exists.
namespace/sample-app configured
INFO: Applying Kubernetes Deployment from k8s/deployment.yaml
deployment.apps/sample-node-app-deployment unchanged
INFO: Applying Kubernetes Service from k8s/service.yaml
service/sample-node-app-service unchanged
INFO: Applying Kubernetes Ingress from k8s/ingress.yaml
ingress/sample-node-app-ingress unchanged
INFO: Applying Kubernetes ConfigMap from k8s/configmap.yaml
configmap/app-config unchanged
INFO: Applying Kubernetes Secret from k8s/secret.yaml
secret/app-secret unchanged
INFO: Deployment to Minikube EKS namespace 'sample-app' completed successfully.
INFO: Simulated deploy to minikube EKS.
INFO: Verifying deployment status (optional, but recommended):
NAME                                          READY   STATUS    RESTARTS   AGE
sample-node-app-deployment-579dd4cd5d-j27tw   1/1     Running   0          38h
NAME                      TYPE       CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
sample-node-app-service   NodePort   10.110.146.154   <none>        80:31364/TCP
38h
NAME                      CLASS   HOSTS                  ADDRESS        PORTS   AGE
sample-node-app-ingress   nginx   sample-node-app.info   192.168.49.2   80      38h
```