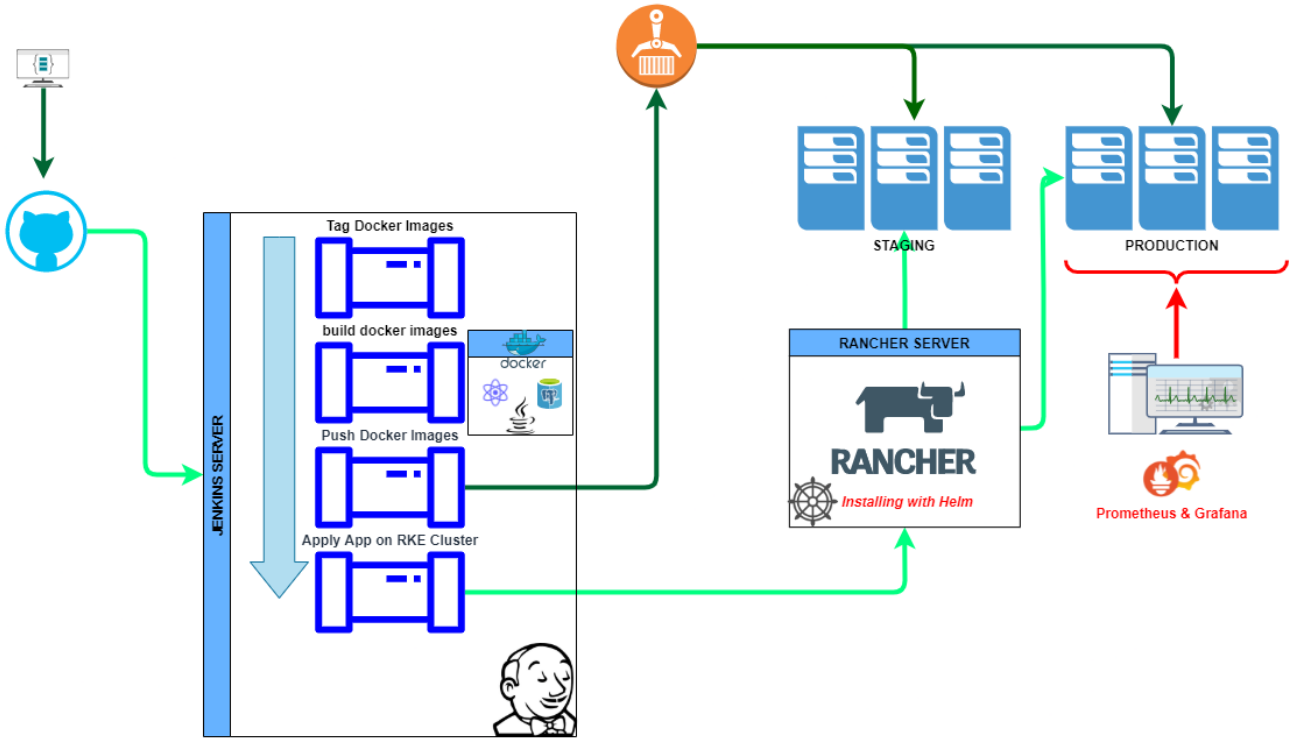
Project Web Application: CI/CD Pipeline

Description

This project aims to create full CI/CD Pipeline for Car-Rental Web Application using a React in frontend, Java Application with Spring Boot framework in backend and PostgreSQL Database, Car-Rental Web Application. Jenkins Server deployed on Elastic Compute Cloud (EC2) Instance is used as CI/CD Server to build pipelines.

DevOps Pipelines



Flow of Tasks for Project Realization

Task No.	Task Definition
1	Prepare Kubernetes YAML Files
2	Prepare High-availability RKE Kubernetes Cluster on AWS EC2
3	Install Rancher App on RKE Kubernetes Cluster
4	Create Production Environment with Rancher
5	Prepare a Production Pipeline
6	Setting Domain Name and TLS for Production Pipeline with Route 53.

Task No.	Task Definition
7	Monitoring with Prometheus and Grafana.

Task 1 - Prepare Kubernetes YAML Files

- Create a folder with name of `k8s` for keeping the deployment files of Car-rental App on Kubernetes cluster.
- Create a `docker-compose.yml` under `k8s` folder with the following content as to be used in conversion the k8s files.

```
version: '3'
services:
  ui:
    image: IMAGE_TAG_UI
    deploy:
      replicas: 1
    depends_on:
      - app
      - db
    ports:
      - "80:3000"
    labels:
      kompose.image-pull-secret: "regcred"
      kompose.service.expose: "carrental.littlepricing.com"
      kompose.service.type: "nodeport"
    environment:
      - REACT_APP_API_URL=http://carrental.littlepricing.com:8080/car-rental/api/

  app:
    image: IMAGE_TAG_API
    deploy:
      replicas: 3
    depends_on:
      - db
    ports:
      - "8080:8080"
    labels:
      kompose.image-pull-secret: "regcred"
    restart: "always"
    environment:
      - DATABASE_URL=jdbc:postgresql://db:5432/carrental


  grafana-server:
    image: IMAGE_TAG_GRAFANA_SERVICE
    ports:
    - 3000:3000
    labels:
      kompose.image-pull-secret: "regcred"
```

```
  prometheus-server:
    image: IMAGE_TAG_PROMETHEUS_SERVICE
    ports:
    - 9090:9090
    labels:
      kompose.image-pull-secret: "regcred"

  db:
    image: 'postgres:13.1-alpine'
    container_name: postgres
    environment:
      - POSTGRES_USER=techprodb_user
      - POSTGRES_PASSWORD=password
      - POSTGRES_DB=carrental
    ports:
      - "5432:5432"
```

- Install conversion tool named `Kompose` on your Jenkins Server. User Guide

```
curl -L https://github.com/kubernetes/kompose/releases/download/v1.22.0/kompose-
linux-amd64 -o kompose
chmod +x kompose
sudo mv ./kompose /usr/local/bin/kompose
kompose version
```

- Create folders named `base`, `prod` under `k8s` folder.
- Convert the `docker-compose.yml` into K8s objects and save under `k8s/base` folder.

```
kompose convert -f K8S/docker-compose.yml -o K8S/base
```

- Create `kustomization-template.yml` file with following content and save under `k8s/base` folder.

```
resources:
- app-deployment.yaml
- app-service.yaml
- db-deployment.yaml
- db-service.yaml
- grafana-server-deployment.yaml
- grafana-server-service.yaml
- prometheus-server-deployment.yaml
- prometheus-server-service.yaml
- ui-deployment.yaml
- ui-ingress.yaml
- ui-service.yaml

images:
```

```
- name: IMAGE_TAG_UI
  newName: "${IMAGE_TAG_UI}"
- name: IMAGE_TAG_API
  newName: "${IMAGE_TAG_API}"
- name: IMAGE_TAG_GRAFANA_SERVICE
  newName: "${IMAGE_TAG_GRAFANA_SERVICE}"
- name: IMAGE_TAG_PROMETHEUS_SERVICE
  newName: "${IMAGE_TAG_PROMETHEUS_SERVICE}"
```

- Create `kustomization.yml` and `replica-count.yml` files for production envrionment and save them under `k8s/prod` folder.

```
# kustomization.yml
namespace: car-rental-prod-ns"
bases:
- ../base
patches:
- replica-count.yml
```

```
# replica-count.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ui
spec:
  replicas: 3
```

- Install `kubectl` on Jenkins Server. Install and Set up kubectl

```
curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.18.9/2020-11-
02/bin/linux/amd64/kubectl
sudo mv kubectl /usr/local/bin/kubectl
chmod +x /usr/local/bin/kubectl
kubectl version --short --client
```

- Check if the customization tool working as expected.

```
export IMAGE_TAG_UI="testing-image-1"
export IMAGE_TAG_API="testing-image-2"
export IMAGE_TAG_GRAFANA_SERVICE="testing-image-3"
export IMAGE_TAG_PROMETHEUS_SERVICE="testing-image-4"
# create base kustomization file from template by updating with environments
variables
envsubst < k8s/base/kustomization-template.yml > k8s/base/kustomization.yml
```

```
# test customization for production
kubectl kustomize k8s/prod/
```

Tasks 2 - Prepare High-availability RKE Kubernetes Cluster on AWS EC2

- Explain Rancher Container Management Tool.
- Create an IAM Policy with name of `rke-controlplane-policy` and also save it under `templates` for `Control Plane` node to enable Rancher to create or remove EC2 resources.

```
{
"Version": "2012-10-17",
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "autoscaling:DescribeAutoScalingGroups",
      "autoscaling:DescribeLaunchConfigurations",
      "autoscaling:DescribeTags",
      "ec2:DescribeInstances",
      "ec2:DescribeRegions",
      "ec2:DescribeRouteTables",
      "ec2:DescribeSecurityGroups",
      "ec2:DescribeSubnets",
      "ec2:DescribeVolumes",
      "ec2:CreateSecurityGroup",
      "ec2:CreateTags",
      "ec2:CreateVolume",
      "ec2:ModifyInstanceAttribute",
      "ec2:ModifyVolume",
      "ec2:AttachVolume",
      "ec2:AuthorizeSecurityGroupIngress",
      "ec2:CreateRoute",
      "ec2:DeleteRoute",
      "ec2:DeleteSecurityGroup",
      "ec2:DeleteVolume",
      "ec2:DetachVolume",
      "ec2:RevokeSecurityGroupIngress",
      "ec2:DescribeVpcs",
      "elasticloadbalancing:AddTags",
      "elasticloadbalancing:AttachLoadBalancerToSubnets",
      "elasticloadbalancing:ApplySecurityGroupsToLoadBalancer",
      "elasticloadbalancing:CreateLoadBalancer",
      "elasticloadbalancing:CreateLoadBalancerPolicy",
      "elasticloadbalancing:CreateLoadBalancerListeners",
      "elasticloadbalancing:ConfigureHealthCheck",
      "elasticloadbalancing:DeleteLoadBalancer",
      "elasticloadbalancing:DeleteLoadBalancerListeners",
      "elasticloadbalancing:DescribeLoadBalancers",
      "elasticloadbalancing:DescribeLoadBalancerAttributes",
      "elasticloadbalancing:DetachLoadBalancerFromSubnets",
```

```
        "elasticloadbalancing:DeregisterInstancesFromLoadBalancer",
        "elasticloadbalancing:ModifyLoadBalancerAttributes",
        "elasticloadbalancing:RegisterInstancesWithLoadBalancer",
        "elasticloadbalancing:SetLoadBalancerPoliciesForBackendServer",
        "elasticloadbalancing:AddTags",
        "elasticloadbalancing:CreateListener",
        "elasticloadbalancing:CreateTargetGroup",
        "elasticloadbalancing:DeleteListener",
        "elasticloadbalancing:DeleteTargetGroup",
        "elasticloadbalancing:DescribeListeners",
        "elasticloadbalancing:DescribeLoadBalancerPolicies",
        "elasticloadbalancing:DescribeTargetGroups",
        "elasticloadbalancing:DescribeTargetHealth",
        "elasticloadbalancing:ModifyListener",
        "elasticloadbalancing:ModifyTargetGroup",
        "elasticloadbalancing:RegisterTargets",
        "elasticloadbalancing:SetLoadBalancerPoliciesOfListener",
        "iam:CreateServiceLinkedRole",
        "kms:DescribeKey"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

- Create an IAM Policy with name of `rke-etcd-worker-policy` and also save it under `templates` for `etcd` or `worker` nodes to enable Rancher to get information from EC2 resources.

```
{
"Version": "2012-10-17",
"Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "ec2:DescribeInstances",
            "ec2:DescribeRegions",
            "ecr:GetAuthorizationToken",
            "ecr:BatchCheckLayerAvailability",
            "ecr:GetDownloadUrlForLayer",
            "ecr:GetRepositoryPolicy",
            "ecr:DescribeRepositories",
            "ecr:ListImages",
            "ecr:BatchGetImage"
        ],
        "Resource": "*"
    }
]
}
```

- Create an IAM Role with name of `car-rental-rke-role` to attach RKE nodes (instances) using `rke-controlplane-policy` and `rke-etcd-worker-policy`.
- Create a security group for External Application Load Balancer of Rancher with name of `rke-alb-sg` and allow HTTP (Port 80) and HTTPS (Port 443) connections from anywhere.
- Create a security group for RKE Kubernetes Cluster with name of `rke-cluster-sg` and define following inbound and outbound rules.
    - Inbound rules;
        - Allow HTTP protocol (TCP on port 80) from Application Load Balancer.
        - Allow HTTPS protocol (TCP on port 443) from any source that needs to use Rancher UI or API.
        - Allow TCP on port 6443 from any source that needs to use Kubernetes API server(ex. Jenkins Server).
        - Allow SSH on port 22 to any node IP that installs Docker (ex. Jenkins Server).
    - Outbound rules;
        - Allow SSH protocol (TCP on port 22) to any node IP from a node created using Node Driver.
        - Allow HTTP protocol (TCP on port 80) to all IP for getting updates.
        - Allow HTTPS protocol (TCP on port 443) to `35.160.43.145/32`, `35.167.242.46/32`, `52.33.59.17/32` for catalogs of `git.rancher.io`.
        - Allow TCP on port 2376 to any node IP from a node created using Node Driver for Docker machine TLS port.
    - Allow all protocol on all port from `rke-cluster-sg` for self communication between Rancher `controlplane`, `etcd`, `worker` nodes.
- Log into Jenkins Server and create `rancher-key.pem` key-pair for Rancher Server using AWS CLI

```
aws ec2 create-key-pair --region us-east-1 --key-name rancher-key.pem --query
KeyMaterial --output text > ~/.ssh/rancher-key.pem
chmod 400 ~/.ssh/rancher-key.pem
```

- Launch an EC2 instance using `Ubuntu Server 20.04 LTS (HVM) (64-bit x86)` with `t3a.medium` type, 16 GB root volume, `rke-cluster-sg` security group, `rke-role` IAM Role, `Name:Rancher-Cluster-Instance` tag and `rancher-key.pem` key-pair. Take note of `subnet id` of EC2.
- Attach a tag to the `nodes (intances)`, `subnets` and `security group` for Rancher with `Key = kubernetes.io/cluster/Rancher` and `Value = owned`.

- Log into `Rancher-Cluster-Instance` from Jenkins Server (Bastion host) and install Docker using the following script.

```
# Set hostname of instance
sudo hostnamectl set-hostname rancher-instance-1
# Update OS
sudo apt-get update -y
sudo apt-get upgrade -y
# Install and start Docker on Ubuntu 19.03
# Update the apt package index and install packages to allow apt to use a
repository over HTTPS
sudo apt-get install \
  apt-transport-https \
  ca-certificates \
  curl \
  gnupg \
  lsb-release
# Add Docker’s official GPG key
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/usr/share/keyrings/docker-archive-keyring.gpg
# Use the following command to set up the stable repository
echo \
  "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]
https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list >
/dev/null
# Update packages
sudo apt-get update
# List the versions available in your repo
apt-cache madison docker-ce

# Since Rancher is not compatible (yet) with latest version of Docker install
version 19.03.15 or earlier version using the version string (exp: 5:19.03.15~3-
0~ubuntu-focal) from the second column
sudo apt-get install docker-ce=<VERSION_STRING> docker-ce-cli=<VERSION_STRING>
containerd.io
sudo systemctl start docker
sudo systemctl enable docker

# Add ubuntu user to docker group
sudo usermod -aG docker ubuntu
newgrp docker
```

- Create a target groups with name of `rancher-http-80-tg` with following setup and add the `rancher instances` to it.

```
Target type         : instance
Protocol            : HTTP
Port                : 80
```

```
<!-- Health Checks Settings -->
Protocol            : HTTP
Path                : /healthz
Port                : traffic port
Healthy threshold   : 3
Unhealthy threshold : 3
Timeout             : 5 seconds
Interval            : 10 seoconds
Success             : 200
```

- Create Application Load Balancer with name of `rancher-alb` using `rke-alb-sg` security group with following settings and add `rancher-http-80-tg` target group to it.

```
Scheme              : internet-facing
IP address type     : ipv4

<!-- Listeners-->
Protocol            : HTTPS/HTTP
Port                : 443/80
Availability Zones  : Select AZs of RKE instances
Target group        : `rancher-http-80-tg` target group
```

- Configure ALB Listener of HTTP on `Port 80` to redirect traffic to HTTPS on `Port 443`.
- Create DNS A record for `rancher.littlepricing.com` and attach the `rancher-alb` application load balancer to it.
- Install RKE, the Rancher Kubernetes Engine, Kubernetes distribution and command-line tool) on Jenkins Server.

```
curl -SsL "https://github.com/rancher/rke/releases/download/v1.1.12/rke_linux-amd64" -o "rke_linux-amd64"
sudo mv rke_linux-amd64 /usr/local/bin/rke
chmod +x /usr/local/bin/rke
rke --version
```

- Create `rancher-cluster.yml` with following content to configure RKE Kubernetes Cluster and save it under `infrastructure` folder.

```
nodes:
  - address: 35.173.42.79
    internal_address: 172.31.69.135
    user: ubuntu
    role: [controlplane, worker, etcd]

services:
  etcd:
```

```
    snapshot: true
    creation: 6h
    retention: 24h

ssh_key_path: ~/.ssh/rancher-key.pem

# Required for external TLS termination with
# ingress-nginx v0.22+
ingress:
  provider: nginx
  options:
    use-forwarded-headers: "true"
```

- Run `rke` command to setup RKE Kubernetes cluster on EC2 Rancher instance *`Warning:` You should add rule to cluster sec group for Jenkins Server using its `IP/32` from SSH (22) and TCP(6443) before running `rke` command, because it is giving connection error.*

```
rke up --config ./rancher-cluster.yml
```

- Check if the RKE Kubernetes Cluster created successfully.

```
mkdir -p ~/.kube
mv ./kube_config_rancher-cluster.yml $HOME/.kube/config
chmod 400 ~/.kube/config
kubectl get nodes
kubectl get pods --all-namespaces
```

Task 3 - Install Rancher App on RKE Kubernetes Cluster

- Install Helm version 3+ on Jenkins Server. Introduction to Helm. Helm Installation.

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

- Add helm chart repositories of Rancher.

```
helm repo add rancher-latest https://releases.rancher.com/server-charts/latest
helm repo list
```

- Create a namespace for Rancher.

```
kubectl create namespace cattle-system
```

- Install Rancher on RKE Kubernetes Cluster using Helm.

```
helm install rancher rancher-latest/rancher \
  --namespace cattle-system \
  --set hostname=rancher.littlepricing.com \
  --set tls=external \
  --set replicas=1
```

- Check if the Rancher Server is deployed successfully.

```
kubectl -n cattle-system get deploy rancher
kubectl -n cattle-system get pods
```

Task 4 - Create Production Environment with Rancher

- To provide access of Rancher to the cloud resources, create a `Cloud Credentials` for AWS on Rancher and name it as `AWS-Training-Account`.
- Create a `Node Template` on Rancher with following configuration for to be used while launching the EC2 instances and name it as `AWS-RancherOs-Template`.

```
Region            : us-east-1
Security group    : create new sg (rancher-nodes)
Instance Type     : t2.medium
Root Disk Size    : 16 GB
AMI (RancherOS)   : ami-0e8a3347e4c5959bd
SSH User          : rancher
Label             : os=rancheros
```

Task 5 - Prepare a Production Pipeline

- Create a Kubernetes cluster using Rancher with RKE and new nodes in AWS (on one EC2 instance only) and name it as `car-rental-cluster`.

```
Cluster Type      : Amazon EC2
Name Prefix       :car-rental-k8s-instance
Count             : 3
etcd              : checked
Control Plane     : checked
Worker            : checked
```

- Create `car-rental-prod-ns` namespace on `car-rental-cluster` with Rancher.

- Create a Jenkins Job and name it as `create-ecr-docker-registry-for-car-rental-prod` to create Docker Registry for `Production` manually on AWS ECR.

```
PATH="$PATH:/usr/local/bin"
APP_REPO_NAME="techproeducation-car-rental"
AWS_REGION="us-east-1"

aws ecr create-repository \
  --repository-name ${APP_REPO_NAME} \
  --image-scanning-configuration scanOnPush=false \
  --image-tag-mutability MUTABLE \
  --region ${AWS_REGION}
```

- Install `Rancher CLI` on Jenkins Server.

```
curl -SsL "https://github.com/rancher/cli/releases/download/v2.4.9/rancher-linux-amd64-v2.4.9.tar.gz" -o "rancher-cli.tar.gz"
tar -zxvf rancher-cli.tar.gz
sudo mv ./rancher-v2.4.9/rancher /usr/local/bin/rancher
chmod +x /usr/local/bin/rancher
rancher --version
```

- Create Rancher API Key Rancher API Key to enable access to the `Rancher` server. Take note, `Access Key (username)` and `Secret Key (password)`.
- Create a credentials with kind of `Username with password` on Jenkins Server using the `Rancher API Key`.
    - On jenkins server, select Manage Jenkins --> Manage Credentials --> Jenkins --> Global credentials (unrestricted) --> Add Credentials.
    - Paste `Access Key (username)` to Username field and `Secret Key (password)` to Password field.
    - Define an id like `rancher-car-rental-credentials`.
- Create a `Production Pipeline` on Jenkins with name of car-rental-prodwith following script and configure agithub-webhookto trigger the pipeline everycommitonmain branch. car-rental production pipeline should be deployed on permanent prod-environment on car-rental-cluster `Kubernetes cluster under car-rental-prod-ns` namespace.
    - Engine options: Postgrsql
    - Version : 5.7.30
    - Templates: Free tier
    - DB instance identifier:car-rental
    - Master username: root
    - Master password:car-rental
    - Public access: Yes

    - Initial database name:car-rental

- Delete db-deployment.yaml line from k8s/base/kustomization-template.yml file.
- Update k8s/base/db-service.yaml as below.

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    kompose.cmd: kompose convert -f K8S/docker-compose.yml -o K8S/base
    kompose.version: 1.22.0 (955b78124)
  creationTimestamp: null
  labels:
    io.kompose.service: db
  name: db
spec:
  type: ExternalName
  externalName:car-rental.cbanmzptkrzf.us-east-1.rds.amazonaws.com # Change this
line with the endpoint of your RDS.
```

- Prepare a Jenkinsfile for car-rental-prodpipeline and save it asjenkinsfile-car-rental-produnderjenkins` folder.

```
pipeline {
    agent any
    environment {
        PATH=sh(script:"echo $PATH:/usr/local/bin", returnStdout:true).trim()
        APP_NAME="car-rental"
        APP_STACK_NAME="car-rental-App-prod"
        APP_REPO_NAME="techproeducation-car-rental"
        AWS_ACCOUNT_ID=sh(script:'export PATH="$PATH:/usr/local/bin" && aws sts
get-caller-identity --query Account --output text', returnStdout:true).trim()
        AWS_REGION="us-east-1"
        ECR_REGISTRY="${AWS_ACCOUNT_ID}.dkr.ecr.${AWS_REGION}.amazonaws.com"
        RANCHER_URL="https://rancher.littlepricing.com"
        RANCHER_CONTEXT="c-lv94b:p-wthl9"
        RANCHER_CREDS=credentials('rancher-car-api')
    }
    stages {
        stage('Prepare Tags for Docker Images') {
            steps {
                echo 'Preparing Tags for Docker Images'
                script {
                    env.IMAGE_TAG_UI="${ECR_REGISTRY}/${APP_REPO_NAME}:frontend-
prod-ver${BUILD_NUMBER}"
                    env.IMAGE_TAG_API="${ECR_REGISTRY}/${APP_REPO_NAME}:backend-
prod-ver${BUILD_NUMBER}"

env.IMAGE_TAG_GRAFANA_SERVICE="${ECR_REGISTRY}/${APP_REPO_NAME}:grafana-service"
```

```
env.IMAGE_TAG_PROMETHEUS_SERVICE="${ECR_REGISTRY}/${APP_REPO_NAME}:prometheus-
service"
                }
            }
        }
        stage('Build App Docker Images') {
            steps {
                echo 'Building App Dev Images'
                sh """
                  docker build --force-rm -t "${IMAGE_TAG_UI}"
"${WORKSPACE}/bluerentalcars-frontend"
                  docker build --force-rm -t "${IMAGE_TAG_API}"
"${WORKSPACE}/bluerentalcars-backend"
                  docker build --force-rm -t "${IMAGE_TAG_GRAFANA_SERVICE}"
"${WORKSPACE}/docker/grafana"
                  docker build --force-rm -t "${IMAGE_TAG_PROMETHEUS_SERVICE}"
"${WORKSPACE}/docker/prometheus"
                  docker image ls
                """
            }
        }
        stage('Push Images to ECR Repo') {
            steps {
                echo "Pushing ${APP_NAME} App Images to ECR Repo"
                sh """
                  aws ecr get-login-password --region ${AWS_REGION} | docker login
--username AWS --password-stdin ${ECR_REGISTRY}
                  docker push "${IMAGE_TAG_UI}"
                  docker push "${IMAGE_TAG_API}"
                  docker push "${IMAGE_TAG_GRAFANA_SERVICE}"
                  docker push "${IMAGE_TAG_PROMETHEUS_SERVICE}"
                """

            }
        }
        stage('Deploy App on Docker Swarm'){
            steps {
                echo 'Deploying App on K8s Cluster'
                sh "rancher login $RANCHER_URL --context $RANCHER_CONTEXT --token
$RANCHER_CREDS_USR:$RANCHER_CREDS_PSW"
                sh "envsubst < K8S/base/kustomization-template.yml >
K8S/base/kustomization.yml"
                sh "rancher kubectl delete secret regcred -n car-rental-prod-ns ||
true"
                sh """
                rancher kubectl create secret generic regcred -n car-rental-prod-
ns --from-file=.dockerconfigjson=$JENKINS_HOME/.docker/config.json --
type=kubernetes.io/dockerconfigjson
                """
                sh "rancher kubectl apply -k K8S/prod/"
                }
        }
    }
```

```
    post {
        always {
            echo 'Deleting all local images'
            sh 'docker image prune -af'
        }
    }
}
```

- Commit the change, then push the script to the remote repo.

Task 6 - Setting Domain Name and TLS for Production Pipeline with Route 53

- Create an `A` record of carrental.littlepricing.com in your hosted zone (in our case `littlepricing.com`) using AWS Route 53 domain registrar and bind it to your car-rental cluster.
- Configure TLS(SSL) certificate for carrental.littlepricing.com using `cert-manager` on car-rental K8s cluster with the following steps.
- Log into Jenkins Server and configure the `kubectl` to connect to car-rental cluster by getting the `Kubeconfig` file from Rancher and save it as `$HOME/.kube/config` or set `KUBECONFIG` environment variable.

```
#createcar-rental-config file under home folder(/home/ec2-user).
nano car-rental-cluster.yaml
# paste the content of kubeconfig file and save it.
chmod 400 car-rental-cluster.yaml
export KUBECONFIG=/home/ec2-user/.kube/car-rental-cluster.yaml
# test the kubectl withcar-rental namespaces
kubectl get ns
```

- Install the `cert-manager` on car-rental cluster. See Cert-Manager info.
    - Create the namespace for cert-manager

```
kubectl create namespace cert-manager
```

    - Add the Jetstack Helm repository.

```
helm repo add jetstack https://charts.jetstack.io
```

    - Update your local Helm chart repository.

```
helm repo update
```

- Install the `Custom Resource Definition` resources separately

```
kubectl apply -f https://github.com/jetstack/cert-
manager/releases/download/v1.5.0/cert-manager.crds.yaml
```

- Install the cert-manager Helm chart

```
helm install \
cert-manager jetstack/cert-manager \
--namespace cert-manager \
--version v1.5.0
```

- Verify that the cert-manager is deployed correctly.

```
kubectl get pods --namespace cert-manager -o wide
```

- Create `ClusterIssuer` with name of `tls-cluster-issuer-prod.yml` for the production certificate through `Let's Encrypt ACME` (Automated Certificate Management Environment) with following content by importing YAML file on Ranhcer and save it under `K8S` folder. *Note that certificate will only be created after annotating and updating the `Ingress` resource.*

```
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-prod
  namespace: cert-manager
spec:
  acme:
    # The ACME server URL
    server: https://acme-v02.api.letsencrypt.org/directory
    # Email address used for ACME registration
    email: devops@techpro.com
    # Name of a secret used to store the ACME account private key
    privateKeySecretRef:
      name: letsencrypt-prod
    # Enable the HTTP-01 challenge provider
    solvers:
    - http01:
        ingress:
          class: nginx
```

- Check if `ClusterIssuer` resource is created.

```
export KUBECONFIG="/home/ec2-user/.kube/car-rental-cluster.yaml"
kubectl apply -f K8S/tls-cluster-issuer-prod.yml
kubectl get clusterissuers letsencrypt-prod -n cert-manager -o wide
```

- Issue production Let's Encrypt Certificate by annotating and adding the `ui` ingress resource with following through Rancher.

```
metadata:
  name: ui
  annotations:
    cert-manager.io/cluster-issuer: "letsencrypt-prod"
spec:
  tls:
  - hosts:
    - carrental.littlepricing.com
    secretName: car-rental-tls
```

- Check and verify that the TLS(SSL) certificate created and successfully issued to carrental.littlepricing.com`by checking URL of`https:/carrental.littlepricing.com`

- Run the `Production Pipeline` car-rental-prod` on Jenkins manually to examine the car-rental application.

Task 7 - Monitoring with Prometheus and Grafana

- Change the port of Prometheus Service to `9090`, so that Grafana can scrape the data.
- Create a Kubernetes `NodePort` Service for Prometheus Server on Rancher to expose it.
- Create a Kubernetes `NodePort` Service for Grafana Server on Rancher to expose it.