**Percipient**

**AWS Environment Deployment**

**Table of Contents**

1 Introduction ……………………………………………………………………………….. 2-3

1.1 Purpose of this document

1.2 Twinn Framework

2 Installation Plan ……………………………………………………………………………. 3-35

2.1 Scope

2.2 Pre-Installation Requirements

2.3 DB, S/W versions, Hardware requirements etc

2.4 Representative Deployment Diagram

2.5 Pre-installation Checklist

2.6 Components

2.6.1 TWINN Instance Technical Components

2.6.2 Twinn Instance Functional Components

2.7 Installation Procedure

2.7.1 Terraform scripts to create AKS cluster

2.7.2 Connecting to the cluster

2.7.3 Deployment and service yaml files of all the microservices

3.0 Configure Ingress Controller …………………………………………………………… 35-37

4.0 Jenkins Server installation steps………………………………………………………… 38-44

4.1 Jenkins job configuration

4.2 How to build the selected services

5.0 Create the ECR Registry……………………………………………………………………… 44-46

6.0 Docker Files for Backend Services……………………………………………………….. 46-49

## **1.INTRODUCTION**

## **1.1 Purpose of this document**

The purpose of this document is to describe the steps involved in installing the Twinn framework. The primary audience is the Devops/Technical teams who are part of the installation and development activities using the Twinn. The document describes the installation of a new instance of Twinn and the pre-requisites, configuration requirements are mentioned in the below table.

CREATE BASTION SERVER TO ACCESS TO CREATE AND ACCESS ENVIRONMENT IN AWS

**For Citizen-Bastion**

|  |  |
| --- | --- |
|  | **Twin-Transact** |
| **Platform details** | Ubuntu-20.04 |
| **Instance Type** | t2.medium(2 virtual CPUs for the instance, 4GB RAM) |
| **VPC ID** | Vpc-0ae428f0121c564f (default VPC) |
| **Subnet ID** | subnet-0f88c1fb657c18f19 (us-east-1d) |
| **Public IP address(Instance restart IP changes accordingly)** | 184.72.121.114 |
| **Private IP addresses(Instance restart IP changes accordingly)** | 172.31.92.238 |
| **Root Volume Size** | 20GB |
| **Additional Volume Disk** | No |
| **Key pairs** | Percipient-bastion |
| **Security Group** | sg-0293eb37dda66464e (  default VPC security group) |

**For Jenkins-Server**

|  |  |
| --- | --- |
|  | **Prod Env** |
| **Platform details** | Ubuntu-20.04 |
| **Instance Type** | t2.xlarge(2 virtual CPUs for the instance, 8GB RAM) |
| **VPC ID** | vpc-0ae428f0121c5604f (default VPC) |
| **Subnet ID** | subnet-0b09e62f936d3df8e (us-east-1d) |
| **Public IP address(Instance restart IP changes accordingly)** | 54.90.76.21 |
| **Private IP addresses(Instance restart IP changes accordingly)** | 172.31.23.206 |
| **Root Volume Size** | 50GB |
| **Additional Volume Disk** | No |
| **Key pairs** | aws-percipient-jenkins |
| **Security Group** | sg-05ed0f210a63c8d57 (  default VPC security group) |

## **1.2 Twinn Framework**

The TWINN Instance provides for a micro services framework. It is the means by which an FI can power its applications using services (APIs) that serve distinct business purposes.

In contrast to traditional monolithic platforms and SOA (Service-Oriented Architecture) techniques that many FIs still use to run their digital offerings, APIs offer the speed, scalability and reliability required by modern applications.

Furthermore, a micro services framework allows organisations to leverage internal, partner, FinTech and open APIs. These can be re-used across different applications, and are not tied to a specific project.

Percipient’s deliverables for the TWINN Instance comprise technical and functional aspects. The technical components support a comprehensive multi-modal data exposition with polyglot persistence.

# **2. Installation Plan**

## **2.1 Scope**

This is a guide to deploy Twinn Transact Services on AWS cloud.

## **2.2 Pre-Installation Requirements**

* Active Cloud account
* Admin or equivalent access of the account.
* Root or equivalent CLI access to cloud services.

Access to Twinn Transact Azure repository from cloud instance

## **2.3 DB, S/W versions, Hardware requirements**

**For Citizen-Bastion**

|  |  |
| --- | --- |
|  | **Twin-Transact** |
| **Platform details** | Ubuntu-20.04 |
| **Instance Type** | t2.medium(2 virtual CPUs for the instance, 4GB RAM) |
| **VPC ID** | Vpc-0ae428f0121c564f (default VPC) |
| **Subnet ID** | subnet-0f88c1fb657c18f19 (us-east-1d) |
| **Public IP address(Instance restart IP changes accordingly)** | 184.72.121.114 |
| **Private IP addresses(Instance restart IP changes accordingly)** | 172.31.92.238 |
| **Root Volume Size** | 20GB |
| **Additional Volume Disk** | No |
| **Key pairs** | Percipient-bastion |
| **Security Group** | sg-0293eb37dda66464e (  default VPC security group) |

**For Jenkins-Server**

|  |  |
| --- | --- |
|  | **Prod Env** |
| **Platform details** | Ubuntu-20.04 |
| **Instance Type** | t2.xlarge(2 virtual CPUs for the instance, 8GB RAM) |
| **VPC ID** | vpc-0ae428f0121c5604f (default VPC) |
| **Subnet ID** | subnet-0b09e62f936d3df8e (us-east-1d) |
| **Public IP address(Instance restart IP changes accordingly)** | 54.90.76.21 |
| **Private IP addresses(Instance restart IP changes accordingly)** | 172.31.23.206 |
| **Root Volume Size** | 50GB |
| **Additional Volume Disk** | No |
| **Key pairs** | aws-percipient-jenkins |
| **Security Group** | sg-05ed0f210a63c8d57 (  default VPC security group) |

## **2.4 Representative Deployment Diagram**

Cloud Diagram will be included once it is finalized.

## **2.5 Pre-installation Checklist**

## **2.6 Components**

### **2.6.1 TWINN Instance Technical Components**

The complete list of software components for instantiating the TWINN Instance is listed below:

MongoDB (No SQL data store)

AXON (Aggregates, Commands & event buses to enable CQRS pattern)

Vault (for API key management and security)

Kafka (message bus & event streaming)

Zookeeper (Configuration & version management)

AWS KMS (Key Management Service)

Jenkins (Automated Test and Deploy)

Terraform (Automated deployment and change infra)

API Gateway (API gateway to provide capability to integration with the FI Development Environment. Includes authentication & routing capabilities)

Cloud Watch/ Grafana (Monitoring)

Cache (distributed in memory caching)

Eureka (Load Balancer)

Zuul (Micro gateway and router)

BPMS (Business Process Management Service)

Spring Config Server (stores & serves distributed configuration servers)

Oauth Server (Security) – Cognito for AWS

S3 - File Storage (File Storage assuming AWS)

AWS Image Registry (ECR – Elastic Docket Container registry for container management

Assumes AWS)

### **2.6.2 Twinn Instance Functional Components**

The TWINN Instance makes available the following functional components:

An industry-standard, technology-agnostic seed data model covering the following domains. These APIs will enable customer management and account creation processes.

Customer with Party and Arrangement

Deposits (CASA, Time Deposits, Savings)

## **2.7 Installation Procedure**

This section will describe the detailed steps for the installation which are broadly described as below.

1. **Cluster Definition**

**Cluster** is a set of nodes that run containerized applications. Containerizing applications packages an app with its dependences and some necessary services. They are more lightweight and flexible than virtual machines.

**Bastion server installation steps:**

Bastion Server is a special purpose server instance that is designed to be the primary access point from the internet and acts as proxy to your other EC2 instances.

Step 1:

For Launching new server , click on **Launch Instance**

Graphical user interface, application

Description automatically generated

Step 2:

Choose **Ubuntu Server 20.04 LTS**

Graphical user interface, text, application, email

Description automatically generated

Step 3:

Select instance type as **t2.medium**, and click on **configure instance details**.

Table

Description automatically generated

Step 4:

Here select **VPC** and **subnet** as **default**

Graphical user interface, text, application, email

Description automatically generated

Step 5:

Default storage is 8GB, increased to **20 GB** as per requirement

Graphical user interface, text, application, email

Description automatically generated

Step 6:

Click on select an existing security group, select default security group and click on **Review and Launch**

Graphical user interface, text, application

Description automatically generated

Step 7:

Check the launched instance details under **Review Instance Launch** and click on **Launch**

Graphical user interface, application

Description automatically generated

Step 8:

Here we can review security group and instance details and click on **Launch**

Table

Description automatically generated

For time being we can open all the ports for connecting cluster to everyone once the project stable, we remove all the ports.

Step 9:

Click on Create new Key pair and name it as percipient-bastion. Click on **Launch instance**

Graphical user interface, text, application, email

Description automatically generated

Once we click on Launch instance , Bastion Server will be available under EC2 Dash Board.

CREATE EKS CLUSTER AND AMAZON ELASTIC CONTAINER REGISTRY

**Install Terraform on Bastion.**

**Step 1- Register HashiCorp GPG keys**

curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -

**Step 2- Add HashiCorp package repository**

sudo apt-add-repository "deb [arch=$(dpkg --print-architecture)] https://apt.releases.hashicorp.com $(lsb\_release -cs) main"

**Step 3- Update "Ubuntu" packages list**

sudo apt update

**Step 4- Install Terraform on Ubuntu**

sudo apt install terraform

**Step 5 - Check Terraform version**

# Check version of Terraform

terraform -v

Terraform Version : 1.1.15

# Check PATH of Terraform

which terraform

**Install AWS CLI on Citizen-Bastion and configure the aws configuration.**

sudo apt install awscli

Reference Document: <https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2-linux.html>

Configure the Access key and Secret access key

aws configure

* **export AWS\_ACCESS\_KEY\_ID=***<Generate the access key for each user>*
* **export AWS\_SECRET\_ACCESS\_KEY=***<Generate the secret access key for each user>*
* **export AWS\_DEFAULT\_REGION=***us-east-1*

**Terraform script to create EKS cluster from Ubuntu.**

GIT Hub Repo: <https://uniconnect-ace.visualstudio.com/uniconnect-ace/_git/Devops-Configs>

Text

Description automatically generated

**eks-cluster.tf**

|  |
| --- |
| #  # EKS Cluster Resources  # \* IAM Role to allow EKS service to manage other AWS services  # \* EC2 Security Group to allow networking traffic with EKS cluster  # \* EKS Cluster  #  resource "aws\_iam\_role" "percipient-citizen-cluster" {  name = "terraform-eks-percipient-citizen-cluster"  assume\_role\_policy = <<POLICY  {  "Version": "2012-10-17",  "Statement": [  {  "Effect": "Allow",  "Principal": {  "Service": "eks.amazonaws.com"  },  "Action": "sts:AssumeRole"  }  ]  }  POLICY  }  resource "aws\_iam\_role\_policy\_attachment" "percipient-citizen-cluster-AmazonEKSClusterPolicy" {  policy\_arn = "arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"  role = aws\_iam\_role.percipient-citizen-cluster.name  }  resource "aws\_iam\_role\_policy\_attachment" "percipient-citizen-cluster-AmazonEKSVPCResourceController" {  policy\_arn = "arn:aws:iam::aws:policy/AmazonEKSVPCResourceController"  role = aws\_iam\_role.percipient-citizen-cluster.name  }  resource "aws\_security\_group" "percipient-citizen-cluster" {  name = "terraform-eks-percipient-citizen-cluster"  description = "Cluster communication with worker nodes"  vpc\_id = aws\_vpc.percipient-citizen.id  egress {  from\_port = 0  to\_port = 0  protocol = "-1"  cidr\_blocks = ["0.0.0.0/0"]  }  tags = {  Name = "terraform-eks-percipient-citizen"  }  }  resource "aws\_security\_group\_rule" "percipient-citizen-cluster-ingress-workstation-https" {  cidr\_blocks = [local.workstation-external-cidr]  description = "Allow workstation to communicate with the cluster API Server"  from\_port = 443  protocol = "tcp"  security\_group\_id = aws\_security\_group.percipient-citizen-cluster.id  to\_port = 443  type = "ingress"  }  resource "aws\_eks\_cluster" "percipient-citizen" {  name = var.cluster-name  role\_arn = aws\_iam\_role.percipient-citizen-cluster.arn  vpc\_config {  security\_group\_ids = [aws\_security\_group.percipient-citizen-cluster.id]  subnet\_ids = aws\_subnet.percipient-citizen[\*].id  }  depends\_on = [  aws\_iam\_role\_policy\_attachment.percipient-citizen-cluster-AmazonEKSClusterPolicy,  aws\_iam\_role\_policy\_attachment.percipient-citizen-cluster-AmazonEKSVPCResourceController,  ]  } |

**eks-worker-nodes.tf**

|  |
| --- |
| #  # EKS Worker Nodes Resources  # \* IAM role allowing Kubernetes actions to access other AWS services  # \* EKS Node Group to launch worker nodes  #  resource "aws\_iam\_role" "percipient-citizen-node" {  name = "terraform-eks-percipient-citizen-node"  assume\_role\_policy = <<POLICY  {  "Version": "2012-10-17",  "Statement": [  {  "Effect": "Allow",  "Principal": {  "Service": "ec2.amazonaws.com"  },  "Action": "sts:AssumeRole"  }  ]  }  POLICY  }  resource "aws\_iam\_role\_policy\_attachment" "percipient-citizen-node-AmazonEKSWorkerNodePolicy" {  policy\_arn = "arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy"  role = aws\_iam\_role.percipient-citizen-node.name  }  resource "aws\_iam\_role\_policy\_attachment" "percipient-citizen-node-AmazonEKS\_CNI\_Policy" {  policy\_arn = "arn:aws:iam::aws:policy/AmazonEKS\_CNI\_Policy"  role = aws\_iam\_role.percipient-citizen-node.name  }  resource "aws\_iam\_role\_policy\_attachment" "percipient-citizen-node-AmazonEC2ContainerRegistryReadOnly" {  policy\_arn = "arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly"  role = aws\_iam\_role.percipient-citizen-node.name  }  resource "aws\_eks\_node\_group" "percipient-citizen" {  cluster\_name = aws\_eks\_cluster. percipient-citizen.name  node\_group\_name = "percipient-citizen"  node\_role\_arn = aws\_iam\_role.percipient-citizen-node.arn  subnet\_ids = aws\_subnet.percipient-citizen[\*].id  scaling\_config {  desired\_size = 1  max\_size = 2  min\_size = 1  }  depends\_on = [  aws\_iam\_role\_policy\_attachment.percipient-citizen-node-AmazonEKSWorkerNodePolicy,  aws\_iam\_role\_policy\_attachment.percipient-citizen-node-AmazonEKS\_CNI\_Policy,  aws\_iam\_role\_policy\_attachment.percipient-citizen-node-AmazonEC2ContainerRegistryReadOnly,  ]  } |

**providers.tf**

|  |
| --- |
| terraform {  required\_version = ">= 0.12"  }  provider "aws" {  region = var.aws\_region  }  data "aws\_availability\_zones" "available" {}  # Not required: currently used in conjunction with using  # icanhazip.com to determine local workstation external IP  # to open EC2 Security Group access to the Kubernetes cluster.  # See workstation-external-ip.tf for additional information.  provider "http" {} |

**variables.tf**

Here we must give the type as string. Change wherever the existing cluster name with new cluster name.

|  |
| --- |
| variable "aws\_region" {  default = "us-east-1"  }  variable "cluster-name" {  default = "eks-percipient-citizen"  type = string  } |

**vpc.tf**

|  |
| --- |
| #  # VPC Resources  # \* VPC  # \* Subnets  # \* Internet Gateway  # \* Route Table  #  resource "aws\_vpc" "percipient-citizen" {  cidr\_block = "10.0.0.0/16"  tags = tomap({  "Name" = "terraform-eks-percipient-citizen-node",  "kubernetes.io/cluster/${var.cluster-name}" = "shared",  })  }  resource "aws\_subnet" "percipient-citizen" {  count = 2  availability\_zone = data.aws\_availability\_zones.available.names[count.index]  cidr\_block = "10.0.${count.index}.0/24"  map\_public\_ip\_on\_launch = true  vpc\_id = aws\_vpc.percipient-citizen.id  tags = tomap({  "Name" = "terraform-eks-percipient-citizen-node",  "kubernetes.io/cluster/${var.cluster-name}" = "shared",  })  }  resource "aws\_internet\_gateway" "percipient-citizen" {  vpc\_id = aws\_vpc.percipient-citizen.id  tags = {  Name = "terraform-eks-percipient-citizen"  }  }  resource "aws\_route\_table" "percipient-citizen" {  vpc\_id = aws\_vpc.percipient-citizen.id  route {  cidr\_block = "0.0.0.0/0"  gateway\_id = aws\_internet\_gateway.percipient-citizen.id  }  }  resource "aws\_route\_table\_association" "percipient-citizen" {  count = 2  subnet\_id = aws\_subnet.percipient-citizen.\*.id[count.index]  route\_table\_id = aws\_route\_table.percipient-citizen.id  } |

**workstation-external-ip.tf**

|  |
| --- |
| #  # Workstation External IP  #  # This configuration is not required and is  # only provided as an example to easily fetch  # the external IP of your local workstation to  # configure inbound EC2 Security Group access  # to the Kubernetes cluster.  #  data "http" "workstation-external-ip" {  url = "http://ipv4.icanhazip.com"  }  # Override with variable or hardcoded value if necessary  locals {  workstation-external-cidr = "${chomp(data.http.workstation-external-ip.body)}/32"  } |

**outputs.tf**

|  |
| --- |
| #  # Outputs  #  locals {  config\_map\_aws\_auth = <<CONFIGMAPAWSAUTH  apiVersion: v1  kind: ConfigMap  metadata:  name: aws-auth  namespace: kube-system  data:  mapRoles: |  - rolearn: ${aws\_iam\_role.percipient-citizen-node.arn}  username: system:node:{{EC2PrivateDNSName}}  groups:  - system:bootstrappers  - system:nodes  CONFIGMAPAWSAUTH  kubeconfig = <<KUBECONFIG  apiVersion: v1  clusters:  - cluster:  server: ${aws\_eks\_cluster.percipient-citizen.endpoint}  certificate-authority-data: ${aws\_eks\_cluster.percipient-citizen.certificate\_authority[0].data}  name: kubernetes  contexts:  - context:  cluster: kubernetes  user: aws  name: aws  current-context: aws  kind: Config  preferences: {}  users:  - name: aws  user:  exec:  apiVersion: client.authentication.k8s.io/v1alpha1  command: aws-iam-authenticator  args:  - "token"  - "-i"  - "${var.cluster-name}"  KUBECONFIG  }  output "config\_map\_aws\_auth" {  value = local.config\_map\_aws\_auth  }  output "kubeconfig" {  value = local.kubeconfig  } |

**Commands to run the terraform script**

Change wherever the existing cluster name with new cluster name.

* $ terraform init
* $ terraform validate
* $ terraform plan
* $ terraform apply -auto-approve

**Note: Once Create take the statefile and upload to s3 bucket**

**3 Node Cluster**

Graphical user interface, text, application, email

Description automatically generated

**Worker Nodes for Bastion server**

Graphical user interface

Description automatically generated with medium confidence

### **2.7.2 Connecting to the cluster**

**Install AWS CLI on Bastion and configure the aws configuration.**

sudo apt install awscli

Reference Document: <https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2-linux.html>

Configure the Access key and Secret access key

aws configure

* **export AWS\_ACCESS\_KEY\_ID=***<Generate the access key for each user>*
* **export AWS\_SECRET\_ACCESS\_KEY=***<Generate the secret access key for each user>*
* **export AWS\_DEFAULT\_REGION=***us-east-1*

**Install kubectl**

curl -LO [https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl](https://dl.k8s.io/release/$(curl%20-L%20-s%20https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl)

curl -LO [https://dl.k8s.io/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256](https://dl.k8s.io/$(curl%20-L%20-s%20https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256)

echo "$(<kubectl.sha256) kubectl" | sha256sum --check

kubectl: OK

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

**Install docker**

sudo apt-get update

sudo apt-get install ca-certificates curl gnupg lsb-release

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

sudo apt-get update

sudo apt-get install docker-ce docker-ce-cli containerd.io

Or else you can check docker official installation steps.

**Accessing EKS:**

**Citizen-bastion-Env:**

* aws eks --region us-east-1 update-kubeconfig --name eks-percipient-citizen

1. **Deploy the Services MySQL and MongoDB**

Use this command to run the yaml file: **kubectl create -f servicename.yaml**

MySql

|  |
| --- |
| #deployment  ---  apiVersion: apps/v1  kind: Deployment  metadata:  name: mysql-deployment  namespace: citizen  labels:  app: mysql  spec:  replicas: 1  selector:  matchLabels:  app: mysql  template:  metadata:  labels:  app: mysql  spec:  containers:  - name: mysql  image: mysql:5.7  ports:  - containerPort: 3306  volumeMounts:  - mountPath: "/var/lib/mysql"  subPath: "mysql"  name: mysql-data  env:  - name: MYSQL\_ROOT\_PASSWORD  valueFrom:  secretKeyRef:  name: mysql-secrets  key: password  volumes:  - name: mysql-data  persistentVolumeClaim:  claimName: mysql-data-disk  #Service  ---  #Service.yaml  apiVersion: v1  kind: Service  metadata:  name: mysql-service  namespace: citizen  spec:  type: NodePort  selector:  app: mysql  ports:  - protocol: TCP  port: 3306  nodePort: 30036  targetPort: 3306 |

MongoDB

|  |
| --- |
| #mongodb service.yaml  kind: StorageClass  apiVersion: storage.k8s.io/v1  metadata:  name: citizen  namespace: citizen  annotations:  provisioner: kubernetes.io/aws-ebs  volumeBindingMode: Immediate  parameters:  type: gp2  fsType: ext4  ---  apiVersion: v1  kind: Secret  metadata:  name: secret  namespace: citizen  data:  username: YWRtaW4=  password: RjFyM3dhbGw=  ---  apiVersion: v1  kind: Service  metadata:  name: mongodb-service  namespace: citizen  labels:  app: mongo  spec:  ports:  - name: mongo  port: 27017  targetPort: 27017  clusterIP: None  selector:  app: mongo  ---  apiVersion: v1  kind: Service  metadata:  labels:  name: mongo  name: mongo-lb  namespace: citizen  spec:  ports:  - port: 27017  protocol: TCP  targetPort: 27017  selector:  app: mongo  sessionAffinity: None  type: LoadBalancer  status:  loadBalancer: {} |

### **2.7.3 Deployment and service yaml files of all the microservices**

By using this command, we can deploy yaml file for each service:

**kubectl create -f accountservice.yaml**

|  |
| --- |
| #Account-Service  #Deployment.yaml  apiVersion: apps/v1  kind: Deployment  metadata:  labels:  app: accountservice  name: accountservice  namespace: citizen  spec:  replicas: 1  selector:  matchLabels:  app: accountservice  template:  metadata:  labels:  app: accountservice  spec:  containers:  - image: 093471887101.dkr.ecr.us-east-1.amazonaws.com/account-service:677b5ae7-citizen-71  imagePullPolicy: Always  name: accountservice  resources:  requests:  memory: 512Mi  restartPolicy: Always  schedulerName: default-scheduler  ---  #Service.yaml  apiVersion: v1  kind: Service  metadata:  name: accountservice  namespace: citizen  spec:  ports:  - name: accountservice  port: 8081  protocol: TCP  selector:  app: accountservice  type: ClusterIP  #Beneficiary-Service  #Deployment.yaml  apiVersion: apps/v1  kind: Deployment  metadata:  labels:  app: beneficiariesservice  name: beneficiariessservice  namespace: citizen  spec:  replicas: 1  selector:  matchLabels:  app: beneficiariesservice  template:  metadata:  labels:  app: beneficiariesservice  spec:  imagePullSecrets:  - name: acrsecret  containers:  - image: 093471887101.dkr.ecr.us-east-1.amazonaws.com/beneficiary-service:8c95bd68-citizen-14  imagePullPolicy: Always  name: beneficiariesservice  resources:  requests:  memory: 512Mi  restartPolicy: Always  schedulerName: default-scheduler  ---  #Service.yaml  apiVersion: v1  kind: Service  metadata:  name: beneficiariesservice  namespace: citizen  spec:  ports:  - name: beneficiariesservice  port: 8082  protocol: TCP  selector:  app: beneficiariesservice  type: ClusterIP  #Config-Service  #Deployment.yaml  apiVersion: apps/v1  kind: Deployment  metadata:  labels:  app: configservers  name: configservers  namespace: citizen  spec:  replicas: 1  selector:  matchLabels:  app: configservers  template:  metadata:  labels:  app: configservers  spec:  containers:  - image: 093471887101.dkr.ecr.us-east-1.amazonaws.com/config-servers:2cefdf26-citizen-36  imagePullPolicy: Always  name: configservers  resources:  requests:  memory: 512Mi  restartPolicy: Always  #Service.yaml  ---  apiVersion: v1  kind: Service  metadata:  name: configservers  namespace: citizen  spec:  ports:  - name: configservers  port: 8083  protocol: TCP  selector:  app: configservers  type: ClusterIP  #Connector-Service  #Deployment.yaml  apiVersion: apps/v1  kind: Deployment  metadata:  labels:  app: connectorservice  name: connectorservice  namespace: citizen  spec:  replicas: 1  selector:  matchLabels:  app: connectorservice  template:  metadata:  labels:  app: connectorservice  spec:  containers:  - image: 093471887101.dkr.ecr.us-east-1.amazonaws.com/connector-service:8c95bd68-citizen-16  imagePullPolicy: Always  name: connectorservice  resources:  requests:  memory: 512Mi  restartPolicy: Always  schedulerName: default-scheduler  #service.yaml  ---  apiVersion: v1  kind: Service  metadata:  name: connectorservice  namespace: citizen  spec:  ports:  - name: connectorservice  port: 8083  protocol: TCP  selector:  app: connectorservice  type: ClusterIP  #Customer-Service  #Deployment.yaml  apiVersion: apps/v1  kind: Deployment  metadata:  labels:  app: customerservice  name: customerservice  namespace: citizen  spec:  replicas: 1  selector:  matchLabels:  app: customerservice  template:  metadata:  labels:  app: customerservice  spec:  containers:  - image: 093471887101.dkr.ecr.us-east-1.amazonaws.com/customer-service:8c95bd68-citizen-15  imagePullPolicy: Always  name: customerservice  resources:  requests:  memory: 512Mi  restartPolicy: Always  schedulerName: default-scheduler  ---  #Service.yaml  apiVersion: v1  kind: Service  metadata:  name: customerservice  namespace: citizen  spec:  ports:  - name: customerservice  port: 8084  protocol: TCP  selector:  app: customerservice  type: ClusterIP  #Enquiry-Service  #Deployment.yaml  apiVersion: apps/v1  kind: Deployment  metadata:  labels:  app: enquiryservice  name: enquiryservice  namespace: citizen  spec:  replicas: 1  selector:  matchLabels:  app: enquiryservice  template:  metadata:  labels:  app: enquiryservice  spec:  containers:  - image: 093471887101.dkr.ecr.us-east-1.amazonaws.com/enquiry-service:8c95bd68-citizen-17  imagePullPolicy: Always  name: enquiryservice  resources:  requests:  cpu: 250m  memory: 512Mi  restartPolicy: Always  schedulerName: default-scheduler  ---  #Service.yaml  apiVersion: v1  kind: Service  metadata:  name: enquiryservice  namespace: citizen  spec:  ports:  - name: enquiryservice  port: 8085  protocol: TCP  selector:  app: enquiryservice  type: ClusterIP  #Eureka-Server  #Deployment.yaml  apiVersion: apps/v1  kind: Deployment  metadata:  labels:  app: eurekaserver  name: eurekaserver  namespace: citizen  spec:  replicas: 1  selector:  matchLabels:  app: eurekaserver  template:  metadata:  labels:  app: eurekaserver  spec:  containers:  - image: springcloud/eureka  imagePullPolicy: Always  name: eurekaserver  resources:  requests:  memory: 512Mi  restartPolicy: Always  #Service.yaml  ---  apiVersion: v1  kind: Service  metadata:  name: eurekaserver  namespace: citizen  spec:  ports:  - name: eurekaserver  port: 8761  protocol: TCP  selector:  app: eurekaserver  type: LoadBalancer  #Gs-Service  #Deployment.yaml  apiVersion: apps/v1  kind: Deployment  metadata:  labels:  app: gsservice  name: gsservice  namespace: citizen  spec:  replicas: 1  selector:  matchLabels:  app: gsservice  template:  metadata:  labels:  app: gsservice  spec:  containers:  - image: 093471887101.dkr.ecr.us-east-1.amazonaws.com/gs-service:27162282-citizen-43  imagePullPolicy: Always  name: gsservice  restartPolicy: Always  schedulerName: default-scheduler  #service.yaml  ---  apiVersion: v1  kind: Service  metadata:  name: gsservice  namespace: citizen  spec:  ports:  - name: gsservice  port: 6050  protocol: TCP  selector:  app: gsservice  type: ClusterIP  #Lead-Service  #Deployment.yaml  apiVersion: apps/v1  kind: Deployment  metadata:  labels:  app: leadservice  name: leadservice  namespace: citizen  spec:  replicas: 1  selector:  matchLabels:  app: leadservice  template:  metadata:  labels:  app: leadservice  spec:  containers:  - image: 093471887101.dkr.ecr.us-east-1.amazonaws.com/lead-service:8c95bd68-citizen-25  imagePullPolicy: Always  name: leadservice  restartPolicy: Always  schedulerName: default-scheduler  #Service.yaml  apiVersion: v1  kind: Service  metadata:  name: leadservice  namespace: citizen  spec:  ports:  - name: leadservice  port: 8091  protocol: TCP  selector:  app: leadservice  type: ClusterIP  #Loan-Service  #Deployment.yaml  ---  apiVersion: apps/v1  kind: Deployment  metadata:  labels:  app: loanservice  name: loanservice  namespace: citizen  spec:  replicas: 1  selector:  matchLabels:  app: loanservice  template:  metadata:  labels:  app: loanservice  spec:  containers:  - image: 093471887101.dkr.ecr.us-east-1.amazonaws.com/loan-service:8c95bd68-citizen-21  imagePullPolicy: Always  name: loanservice  resources:  requests:  memory: 512Mi  restartPolicy: Always  #Service.yaml  ---  apiVersion: v1  kind: Service  metadata:  name: loanservice  namespace: citizen  spec:  ports:  - name: laonservice  port: 8086  protocol: TCP  selector:  app: loanservice  type: ClusterIP  #Money-Service  #Deployment.yaml  ---  apiVersion: apps/v1  kind: Deployment  metadata:  labels:  app: moneymgmt  name: moneymgmt  namespace: citizen  spec:  replicas: 1  selector:  matchLabels:  app: moneymgmt  template:  metadata:  labels:  app: moneymgmt  spec:  containers:  - image: 093471887101.dkr.ecr.us-east-1.amazonaws.com/money-service:8c95bd68-citizen-22  imagePullPolicy: Always  name: moneymgmt  resources:  requests:  #cpu: 250m  memory: 512Mi  restartPolicy: Always  #Service.yaml  ---  apiVersion: v1  kind: Service  metadata:  name: moneymgmt  namespace: citizen  spec:  ports:  - name: moneymgmt  port: 8087  protocol: TCP  selector:  app: moneymgmt  type: ClusterIP  #Persist-Service  #Deployment.yaml  apiVersion: apps/v1  kind: Deployment  metadata:  labels:  app: persistanceservice  name: persistanceservice  namespace: citizen  spec:  replicas: 1  selector:  matchLabels:  app: persistanceservice  template:  metadata:  labels:  app: persistanceservice  spec:  containers:  - image: 093471887101.dkr.ecr.us-east-1.amazonaws.com/persist-service:8c95bd68-citizen-23  imagePullPolicy: Always  name: persistanceservice  resources:  requests:  cpu: 250m  memory: 512Mi  restartPolicy: Always  schedulerName: default-scheduler  #Service.yaml  apiVersion: v1  kind: Service  metadata:  name: persistanceservice  namespace: citizen  spec:  ports:  - name: persistanceservice  port: 8088  protocol: TCP  selector:  app: persistanceservice  type: ClusterIP  #Product-Service  #Deployment.yaml  apiVersion: apps/v1  kind: Deployment  metadata:  labels:  app: productservice  name: productservice  namespace: citizen  spec:  replicas: 1  selector:  matchLabels:  app: productservice  template:  metadata:  labels:  app: productservice  spec:  containers:  - image: 093471887101.dkr.ecr.us-east-1.amazonaws.com/product-service:8c95bd68-citizen-24  imagePullPolicy: Always  name: productservice  restartPolicy: Always  schedulerName: default-scheduler  #Service.yaml  apiVersion: v1  kind: Service  metadata:  name: productservice  namespace: citizen  spec:  ports:  - name: productservice  port: 8089  protocol: TCP  selector:  app: productservice  type: ClusterIP  #Template-Service  #Deployment.yaml  apiVersion: apps/v1  kind: Deployment  metadata:  labels:  app: templateservice  name: templateservice  namespace: citizen  spec:  replicas: 1  revisionHistoryLimit: 10  selector:  matchLabels:  app: templateservice  strategy:  rollingUpdate:  maxSurge: 25%  maxUnavailable: 25%  type: RollingUpdate  template:  metadata:  creationTimestamp: null  labels:  app: templateservice  spec:  containers:  - image: 093471887101.dkr.ecr.us-east-1.amazonaws.com/template-service:c75f1a46-citizen-91  imagePullPolicy: Always  name: templateservice  resources:  requests:  memory: 512Mi  terminationMessagePath: /dev/termination-log  terminationMessagePolicy: File  dnsPolicy: ClusterFirst  #service.yaml  ---  apiVersion: v1  kind: Service  metadata:  name: templateservice  namespace: citizen  spec:  ports:  - name: templateservice  port: 8070  protocol: TCP  targetPort: 8070  selector:  app: templateservice  sessionAffinity: None  type: ClusterIP  status:  loadBalancer: {} |

3.0 CONFIGURE INGRESS CONTROLLER

Now install NGINX ingress controller in default name space using helm.

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

We have used yaml to launch Ingress controller.

We can up ingress controller by using this command: **kubectl create -f ingress ingress.yaml**

|  |
| --- |
| apiVersion: extensions/v1beta1  kind: Ingress  metadata:  annotations:  kubernetes.io/ingress.class: nginx  name: ingress-rule  namespace: {{.Values.namespace}}  spec:  rules:  - http:  paths:  - backend:  serviceName: accountservice  servicePort: 8081  path: /api/account  - backend:  serviceName: customerservice  servicePort: 8084  path: /api/customer  - backend:  serviceName: enquiryservice  servicePort: 8085  path: /api/customerEnquiry  - backend:  serviceName: enquiryservice  servicePort: 8085  path: /api/accountEnquiry  - backend:  serviceName: enquiryservice  servicePort: 8085  path: /api/loanEnquiry  - backend:  serviceName: enquiryservice  servicePort: 8085  path: /api/beneficiaryEnquiry  - backend:  serviceName: enquiryservice  servicePort: 8085  path: /api/creditCardEnquiry  - backend:  serviceName: enquiryservice  servicePort: 8085  path: /api/accountproductEnquiry  - backend:  serviceName: enquiryservice  servicePort: 8085  path: /api/fundsEnquiry  - backend:  serviceName: moneymgmt  servicePort: 8087  path: /api/money\_management  - backend:  serviceName: loanservice  servicePort: 8086  path: /api/loan-management  - backend:  serviceName: beneficiariesservice  servicePort: 8082  path: /api/beneficiary  - backend:  serviceName: gsservice  servicePort: 6050  path: /api/gs  - backend:  serviceName: leadservice  servicePort: 8091  path: /api/lead  - backend:  serviceName: enquiryservice  servicePort: 8085  path: /api/eventStore  - backend:  serviceName: enquiryservice  servicePort: 8085  path: /api/guarantors  - backend:  serviceName: loanservice  servicePort: 8086  path: /api/loan  - backend:  serviceName: enquiryservice  servicePort: 8085  path: /api/termDepositeEnquiry  - backend:  serviceName: enquiryservice  servicePort: 8085  path: /api/taxEnquiry  - backend:  serviceName: enquiryservice  servicePort: 8085  path: /api/depositAccruedEnquiry  - backend:  serviceName: enquiryservice  servicePort: 8085  path: /api/leadEnquiry  - backend:  serviceName: productservice  servicePort: 8089  path: /api/savingProduct  - backend:  serviceName: productservice  servicePort: 8089  path: /api/taxConfig |

**4.0 Jenkins Server installation steps:**

Step 1:

For launching new **Jenkins Server**, Click on **Launch instance**

Graphical user interface, application

Description automatically generated

Step 2:

Select **Ubuntu Server 20.04 LTS** and click on **select**

Graphical user interface, text

Description automatically generated

Step 3:

Choose instance type as **t2.large** and click on **Configure instance details**

Graphical user interface, application, table

Description automatically generated

Step 4:

Select **Subnet and VPC** as default and click on **Add Storage**

Graphical user interface, text, application, email

Description automatically generated

Step 5:

Default is 8GB and select 50GB as per the requirement

Graphical user interface, text, application, email

Description automatically generated

Step 6:

Click on Select an existing security group and click on **launch-wizard-1**. Click on **Review and Launch.**

Graphical user interface, text, application

Description automatically generated

Step 7:

Check server AMI Details and instance type under **Review Instance Launch**

Graphical user interface, text, application, email

Description automatically generated

Step 8:

Here we can check security group and instance details and click on **Launch**

Table

Description automatically generated

Step 9:

Create new key pair and name it as **aws-percipient-jenkins** and click on **Launch Instance**

Graphical user interface, text, application, email

Description automatically generated

Once we click on Launch instance, Jenkins Server will be available under EC2 DashBoard.

### **4.1 Jenkins job configuration**

1.First Development team needs to merge the code in GitHub repository, and we will build the pipeline

Requirements:

Repo: https://uniconnect-ace.visualstudio.com/uniconnect-ace/\_git/uct-saga-event

Branch: Deployment\_TwinTransact

2.Login into Env twin-transact based on user requirements

3.Logging into http://54.90.76.21:8080/

Username: Percipient\_Citizen

Password: Percipient@123

* Here we are using <http://54.90.76.21:8080/> server (with label name: Jenkins server in aws ec2)
* First, configure Elastic container registry (ECR) credentials and Github repo credentials in jenkins global credentials with kind Username and password

Create a Build jenkins job of type “pipeline”.

|  |
| --- |
| pipeline {  agent any  environment {  AWS\_ACCOUNT\_ID="093471887101"  AWS\_DEFAULT\_REGION="us-east-1"  REPOSITORY\_URI = "${AWS\_ACCOUNT\_ID}.dkr.ecr.${AWS\_DEFAULT\_REGION}.amazonaws.com"  }  parameters {  string(name: 'Branch', defaultValue: 'Deployment\_TwinTransact', description: 'Enter the branch to deploy?')  choice(name: 'Service', choices: ['account-service', 'beneficiary-service', 'config-servers', 'connector-service', 'core-apis', 'customer-service', 'enquiry-service', 'eureka-server', 'gateway-service', 'gs-service', 'id-generator-service', 'lead-service', 'loan-service', 'money-service', 'persist-service', 'product-service', 'smart-apis', 'template-service'], description: 'Select the service you want to deploy')  choice(name: 'environment', choices: ['citizen'], description: 'Select the environment you want to deploy')  }  stages {  stage('Logging into AWS ECR') {  steps {  script {  sh "aws ecr get-login-password --region ${AWS\_DEFAULT\_REGION} | docker login --username AWS --password-stdin ${AWS\_ACCOUNT\_ID}.dkr.ecr.${AWS\_DEFAULT\_REGION}.amazonaws.com"  }  }  }  stage('Cloning our Repo') {  steps {  git branch: '$Branch', credentialsId: 'Github-Key', url: 'https://uniconnect-ace.visualstudio.com/uniconnect-ace/\_git/uct-saga-event'  }  }  stage ('Maven Build') {  steps {  dir("${env.WORKSPACE}/uct-transformer-axon/${Service}") {  sh """  mvn clean install  """  }  }  }    // Building Docker images  stage('Building image') {  steps{  script {  COMMIT\_ID = sh(script: "cd ${WORKSPACE}/uct-transformer-axon/${Service} && git log -n 1 --pretty=format:'%h'", returnStdout: true).trim()  dir("${env.WORKSPACE}/uct-transformer-axon/${Service}") {  sh "cp Dockerfile target && cd target && docker build -t ${REPOSITORY\_URI}/${Service}:${COMMIT\_ID}-$environment-${currentBuild.number} ."  }  }  }  }    // Uploading Docker images into AWS ECR  stage('Pushing to ECR') {  steps{  script {  sh "docker push ${AWS\_ACCOUNT\_ID}.dkr.ecr.${AWS\_DEFAULT\_REGION}.amazonaws.com/${Service}:$COMMIT\_ID-$environment-${currentBuild.number}"  sh "docker rmi ${AWS\_ACCOUNT\_ID}.dkr.ecr.${AWS\_DEFAULT\_REGION}.amazonaws.com/${Service}:$COMMIT\_ID-$environment-${currentBuild.number}"  }  }  }  }  } |

**Parameter name - Default value**

Branch - Enter the branch to build docker image

(Deployment\_TwinTransact)

Service - From choice select the micro-service to build

Environment - Select the respective env to build (citizen)

### **4.2 How to build the selected services**

Graphical user interface, application

Description automatically generated

* Click on citizen-uct-saga-event-job and once it opens
* Click on Build with Parameters on the left and specify the build parameters such as
* Branch: Deployment\_TwinTransact
* Env: citizen

Graphical user interface, application

Description automatically generated

* Click on Build button
* Wait for some time till it builds completes and then check the output whether failed or success.
* Inform the Development team for testing after deployment.
* Finally Deployment is completed Successfully.

**5.0 Create the ECR Registry.**

<https://console.aws.amazon.com/ecr/repositories?region=us-east-1>

Create a Private Repository for all services name “citizen-ecr”

Graphical user interface, text, application, email

Description automatically generated

**Here we can see all images**

Graphical user interface, application

Description automatically generated

**Docker tag and Push to ECR Registry**

* docker tag <service-name>:latest 093471887101.dkr.ecr.us-east-1.amazonaws.com/<service-name>:<GIT\_latest\_tag>
* docker push 093471887101.dkr.ecr.us-east-1.amazonaws.com/<service-name>:<GIT\_latest\_tag>

Image will be available in **ECR**.

We can check services using this command: $ **kubectl get deploy -n citizen**

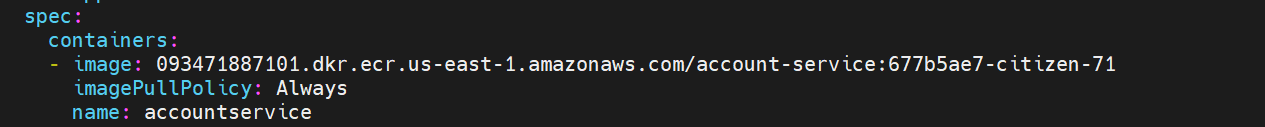
A picture containing graphical user interface

Description automatically generated

To edit the deployment by using this command:

**$ kubectl edit deploy accountservice -n citizen**

Service file will be opened. Replace new image URI with the old URI



Once URI replaced, check the pods with status.

Graphical user interface, text

Description automatically generated

Check the logs by using this command: **$ kubctl logs -f pod-name -n citizen**

Text

Description automatically generated

**6.0 Docker Registry and Docker Files for Backend Services**

To run the micro-services in containers with Docker, first clone the repository from Github and build it (mvn clean install -DskipTests). Now write a Dockerfile to copy the resultant war file into container and run it using java –jar command. Place the Dockerfile under the micro-services repo of your project.

5.1 Dockerfiles For Backend Services

**Account-service**

|  |
| --- |
| FROM java:8-jdk-alpine  COPY  account-service-0.0.1-SNAPSHOT.jar /home/ubuntu/app/  WORKDIR /home/ubuntu/app/  ENTRYPOINT ["java", "-jar", "account-service-0.0.1-SNAPSHOT.jar"] |

**Beneficiary-service**

|  |
| --- |
| FROM java:8-jdk-alpine  COPY  beneficiary-service-0.0.1-SNAPSHOT.jar /home/ubuntu/app/  WORKDIR /home/ubuntu/app/  ENTRYPOINT ["java", "-jar", "beneficiary-service-0.0.1-SNAPSHOT.jar"] |

**Config-service**

|  |
| --- |
| FROM java:8-jdk-alpine  COPY  config-servers-0.0.1-SNAPSHOT.jar /home/ubuntu/app/  WORKDIR /home/ubuntu/app/  ENTRYPOINT ["java", "-jar", "config-servers-0.0.1-SNAPSHOT.jar"] |

**Connector-service**

|  |
| --- |
| FROM java:8-jdk-alpine  COPY  connector-service-0.0.1-SNAPSHOT.jar /home/ubuntu/app/  WORKDIR /home/ubuntu/app/  ENTRYPOINT ["java", "-jar", "connector-service-0.0.1-SNAPSHOT.jar"] |

**Core-apis-service**

|  |
| --- |
| FROM java:8-jdk-alpine  COPY  core-apis-0.0.1-SNAPSHOT.jar /home/ubuntu/app/  WORKDIR /home/ubuntu/app/  ENTRYPOINT ["java", "-jar", "core-apis-0.0.1-SNAPSHOT.jar"] |

**Customer-service**

|  |
| --- |
| FROM java:8-jdk-alpine  COPY  customer-service-0.0.1-SNAPSHOT.jar /home/ubuntu/app/  WORKDIR /home/ubuntu/app/  ENTRYPOINT ["java", "-jar", "customer-service-0.0.1-SNAPSHOT.jar"] |

**Enquiry-service**

|  |
| --- |
| FROM java:8-jdk-alpine  COPY  enquiry-service-0.0.1-SNAPSHOT.jar /home/ubuntu/app/  WORKDIR /home/ubuntu/app/  ENTRYPOINT ["java", "-jar", "enquiry-service-0.0.1-SNAPSHOT.jar"] |

**Eurekha-service**

|  |
| --- |
| FROM java:8-jdk-alpine  COPY  eureka-server-0.0.1-SNAPSHOT.jar /home/ubuntu/app/  WORKDIR /home/ubuntu/app/  ENTRYPOINT ["java", "-jar", "eureka-server-0.0.1-SNAPSHOT.jar"] |

**Gs-service**

|  |
| --- |
| FROM java:8-jdk-alpine  COPY  gs-service-0.0.1-SNAPSHOT.jar /home/ubuntu/app/  WORKDIR /home/ubuntu/app/  ENTRYPOINT ["java", "-jar", "gs-service-0.0.1-SNAPSHOT.jar"] |

**Lead-service**

|  |
| --- |
| FROM java:8-jdk-alpine  COPY  lead-service-0.0.1-SNAPSHOT.jar /home/ubuntu/app/  WORKDIR /home/ubuntu/app/  ENTRYPOINT ["java", "-jar", "lead-service-0.0.1-SNAPSHOT.jar"] |

**Loan-service**

|  |
| --- |
| FROM java:8-jdk-alpine  COPY  loan-service-0.0.1-SNAPSHOT.jar /home/ubuntu/app/  WORKDIR /home/ubuntu/app/  ENTRYPOINT ["java", "-jar", "loan-service-0.0.1-SNAPSHOT.jar"] |

**Money-service**

|  |
| --- |
| FROM java:8-jdk-alpine  COPY  money-service-0.0.1-SNAPSHOT.jar /home/ubuntu/app/  WORKDIR /home/ubuntu/app/  ENTRYPOINT ["java", "-jar", "money-service-0.0.1-SNAPSHOT.jar"] |

**Persist-service**

|  |
| --- |
| FROM java:8-jdk-alpine  COPY  persist-service-0.0.1-SNAPSHOT.jar /home/ubuntu/app/  WORKDIR /home/ubuntu/app/  ENTRYPOINT ["java", "-jar", "persist-service-0.0.1-SNAPSHOT.jar"] |

**Product-service**

|  |
| --- |
| FROM java:8-jdk-alpine  COPY  product-service-0.0.1-SNAPSHOT.jar /home/ubuntu/app/  WORKDIR /home/ubuntu/app/  ENTRYPOINT ["java", "-jar", "product-service-0.0.1-SNAPSHOT.jar"] |

**Smart-apis-service**

|  |
| --- |
| FROM java:8-jdk-alpine  COPY  smart-apis-0.0.1-SNAPSHOT.jar /home/ubuntu/app/  WORKDIR /home/ubuntu/app/  ENTRYPOINT ["java", "-jar", "smart-apis-0.0.1-SNAPSHOT.jar"] |

**Template-apis-service**

|  |
| --- |
| FROM java:8-jdk-alpine  COPY  template-service-0.0.1-SNAPSHOT.jar /home/ubuntu/app/  WORKDIR /home/ubuntu/app/  ENTRYPOINT ["java", "-jar", "template-service-0.0.1-SNAPSHOT.jar"] |