6/20/2024

techdome
Solutions

DevOps Intern Assessment

SRINIVASULA REDDY VEERAMREDDY
srinivasulureddy298@gmail.com
https://github.com/Srinu298

Task

Multi-Container Application Deployment with Docker Compose and Kubernetes

1. Open-Source Solution Tools

Recommend open-source tools, software, or practices that are cloud and OEM agnostic:

- **Infrastructure as Code (IaC):** Use **Terraform or Ansible** for provisioning and managing infrastructure across multiple cloud providers.
- **Container Orchestration: Kubernetes** and **Docker** for container management, providing flexibility and scalability without being tied to any specific cloud or OEM.
- **Monitoring and Logging: Prometheus** for monitoring and **Grafana** for visualization, both cloud-agnostic solutions for performance monitoring.
- **Database: PostgreSQL** as an open-source, relational database management system that supports various cloud platforms.
- **Web Server: NGINX** as a high-performance, open-source web server and reverse proxy.

2. Performance Plan

Outline steps to optimize system performance while avoiding proprietary services or hardware:

- **Containerization:** Containerize applications using **Docker** for consistency and portability.
- **Horizontal Scaling:** Utilize Kubernetes for dynamic scaling based on workload demands, ensuring optimal resource allocation.
- **Caching:** Implement **Redis** for caching frequently accessed data to reduce latency and improve response times.
- **Optimization Tools:** Use **Prometheus** and **Grafana** for monitoring and performance tuning based on real-time metrics.

4. Downtime Mitigation Strategy

1. Implement Blue-Green Deployment:

- **Concept:** Blue-green deployment involves running two identical production environments (blue and green). Only one environment is live at any time, while the other undergoes maintenance or updates.
- **Advantages:** This approach allows for zero-downtime updates. Traffic is routed to the active environment, while updates or changes are applied to the inactive one.
- **Implementation:** Use tools like Kubernetes with rolling updates or container orchestration platforms that support blue-green deployment strategies. Tools like Terraform can automate the setup of environments.

2. Use Rolling Updates:

- **Concept:** Rolling updates involve gradually replacing instances or containers in a cluster with updated versions.
- **Advantages:** This approach reduces downtime by updating one instance at a time while the rest of the infrastructure continues to handle traffic.
- **Implementation:** Kubernetes supports rolling updates natively. Define deployment strategies in Kubernetes manifests or Terraform configurations to ensure updates are applied incrementally without service interruption.

3. Automated Testing and Continuous Integration/Continuous Deployment (CI/CD):

- **Concept:** Automate testing and deployment processes to catch errors early and ensure smooth updates.
- **Advantages:** Reduces the risk of downtime caused by manual errors during updates.
- **Implementation:** Use Jenkins or GitLab CI/CD pipelines integrated with Kubernetes or Terraform. Automate unit tests, integration tests, and deployment steps to validate changes before promoting them to production.

4. Implementation Timeline

Create a timeline for implementing the proposed solutions:

- **Phase 1 :** Infrastructure setup using Terraform for cloud-agnostic provisioning.
- **Phase 2 :** Deploy Kubernetes clusters across multiple environments and migrate applications.
- **Phase 3 :** Configure monitoring and logging with Prometheus and Grafana.
- **Phase 4 :** Implement CI/CD pipelines using Jenkins for automated testing and deployment.

Mentioned Below Yml Files are my Daily Practice codes:

Example YML files

Infra Setup YML: Terraform (IAC Tool) opensource

```
main.tf
-------
provider "aws" {
  region = "${var.region}"
  access_key="AKIAUTJE7BDHCZDRTHU5"
  secret_key="UybJ2gstVFfK5RpWU5plQB/Ts8DzSWDszIhdf8qz"
}

amivar_web.tf

variable "ami" { default = "ami-0911e88fb4687e06b" }

vpc.tf
------
resource "aws_vpc" "mainvpc" {
  cidr_block = "${var.vpc-fullcidr}"

  #### this 2 true values are for use the internal vpc dns resolution
  #enable_dns_support   = true
  #enable_dns_hostnames = true

  tags = {
    Name = "MainVPC-Ohio"
  }
}

 variables.tf
-------------
variable "region" {
  default = "us-east-2"
}

variable "vpc-fullcidr"{
  default     = "192.168.0.0/16"
  description = "the vpc cdir"
}


variable "Subnet-Public-AzA-CIDR" {
  default     = "192.168.1.0/24"
  description = "the cidr of the 2a-Public Subnet"
}

variable "Subnet-Private-AzA-CIDR" {
  default     = "192.168.2.0/24"
  description = "the cidr of the 2a-Private Subnet"
}

variable "Subnet-Public-AzB-CIDR" {
  default     = "192.168.3.0/24"
  description = "the cidr of the 2b-Public Subnet"
}

variable "Subnet-Private-AzB-CIDR" {
  default     = "192.168.4.0/24"
  description = "the cidr of the 2b-Private Subnet"
}
#key-pair declaration

variable "key_name" {
  default     = "OHIO-KP-1"
  description = "the ssh key to use in the EC2 machines"
}
```

subnets.tf

```
----------
resource "aws_subnet" "PublicAZA" {
  vpc_id     = "${aws_vpc.mainvpc.id}"
  cidr_block = "${var.Subnet-Public-AzA-CIDR}"

  tags = {
    Name = "2a-PublicSubnet"
  }

  availability_zone = "${data.aws_availability_zones.available.names[0]}"
}

resource "aws_subnet" "PrivateAZA" {
  vpc_id     = "${aws_vpc.mainvpc.id}"
  cidr_block = "${var.Subnet-Private-AzA-CIDR}"

  tags = {
    Name = "2a-PrivateSubnet"
  }

  availability_zone = "${data.aws_availability_zones.available.names[1]}"
}

resource "aws_subnet" "PublicAZB" {
  vpc_id     = "${aws_vpc.mainvpc.id}"
  cidr_block = "${var.Subnet-Public-AzB-CIDR}"

  tags = {
    Name = "2b-PublicSubnet"
  }

  availability_zone = "${data.aws_availability_zones.available.names[0]}"
}
```

```
resource "aws_subnet" "PrivateAZB" {
  vpc_id     = "${aws_vpc.mainvpc.id}"
  cidr_block = "${var.Subnet-Private-AzB-CIDR}"

  tags = {
    Name = "2b-PrivateSubnet"
  }

  availability_zone = "${data.aws_availability_zones.available.names[1]}"
}

resource "aws_route_table_association" "PublicAZA" {
  subnet_id      = "${aws_subnet.PublicAZA.id}"
  route_table_id = "${aws_route_table.public.id}"
}

resource "aws_route_table_association" "PrivateAZA" {
  subnet_id      = "${aws_subnet.PrivateAZA.id}"
  route_table_id = "${aws_route_table.private.id}"
}

resource "aws_route_table_association" "PublicAZB" {
  subnet_id      = "${aws_subnet.PublicAZB.id}"
  route_table_id = "${aws_route_table.public.id}"
}

resource "aws_route_table_association" "PrivateAZB" {
  subnet_id      = "${aws_subnet.PrivateAZB.id}"
  route_table_id = "${aws_route_table.private.id}"
}


securitygroups.tf
-----------------
resource "aws_security_group" "WebServer" {
  name = "WebServer"

  tags = {
    Name = "WebServer-SG"
  }

  description = "ONLY HTTP CONNECTION INBOUD"
  vpc_id      = "${aws_vpc.mainvpc.id}"

  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "TCP"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = "22"
    to_port     = "22"
    protocol    = "TCP"
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

routing-and-network.tf

```
-----------------------

data "aws_availability_zones" "available" {}

resource "aws_internet_gateway" "gw" {
  vpc_id = "${aws_vpc.mainvpc.id}"

  tags = {
    Name = "IGW-MainVPC-Ohio"
  }
}

resource "aws_route_table" "public" {
  vpc_id = "${aws_vpc.mainvpc.id}"

  tags = {
    Name = "Public"
  }

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = "${aws_internet_gateway.gw.id}"
  }
}

resource "aws_route_table" "private" {
  vpc_id = "${aws_vpc.mainvpc.id}"

  tags = {
    Name = "Private"
  }


#natgateway declaration

    route {
      cidr_block     = "0.0.0.0/0"
      nat_gateway_id = "${aws_nat_gateway.PublicAZA.id}"
    }
}
resource "aws_eip" "forNat" {
  vpc = true
}



resource "aws_nat_gateway" "PublicAZA" {
allocation_id = "${aws_eip.forNat.id}"
subnet_id     = "${aws_subnet.PublicAZA.id}"
depends_on    = ["aws_internet_gateway.gw"]
}
```

app-userdata.sh

```
---------------
#!/bin/bash
sudo su
cd
yum update -y
yum install git -y
yum install httpd -y
service httpd start
chkconfig httpd on
cd /var/www/html
```

ec2-machines.tf   # AWS EC2 setup with Terraform IAC

```
----------------
resource "aws_instance" "webapp1"{
  ami                         = "${var.ami}"
  instance_type               = "t2.micro"
  associate_public_ip_address = "true"
  subnet_id                   = "${aws_subnet.PublicAZA.id}"
  vpc_security_group_ids      = ["${aws_security_group.WebServer.id}"]
  key_name                    = "${var.key_name}"

  tags = {
    Name = "WebApp1"
  }

  user_data = "${file("app-userdata.sh")}"
}

resource "aws_instance" "webapp2"{
  ami                         = "${var.ami}"
  instance_type               = "t2.micro"
  associate_public_ip_address = "false"
  subnet_id                   = "${aws_subnet.PrivateAZA.id}"
  vpc_security_group_ids      = ["${aws_security_group.WebServer.id}"]
  key_name                    = "${var.key_name}"

  tags = {
    Name = "WebApp2"
  }

  user_data = "${file("app-userdata.sh")}"
}
```

Ansible Configuration Management : Deployment .yml files (Example)

Ansible Roles:

1)Ansible roles are consists of many playbooks,which is similar to module in puppet and cook books in cheff.
the same in ansible as roles.

2)Roles are a way to group multiple tasks togather into one container to do the automation in very effective manner with clean directory structure.

3)Roles are set of tasks and additional files for a certain role which allows you to break up the configurations.

4)It can easily reuse the codes by anyone if the role is suitable to someone.

5)It can be easily modify and will reduce the syntax errors.

Ansible Galaxy is a repository for Ansible Roles that are available to drop directly into your Playbooks to streamline your automation projects.

1)Launch 2 Ec2 Instances

```
        1)Ansible Master
        2)Ansible Slave

Login to Ansible Master

2) apt-get update -y
3) apt-get install python -y
4) apt-get install ansible -y
5) ssh-keygen
6) cat id_rsa.pub
     -copy text from here and paste in Authorizedkey in Ansible-Slave
7) cd
8) ssh 10.50.1.64
exit
9) cd /etc/ansible/
ls
10) nano hosts
  [webservers]
  slave1 ansible_ssh_host= <slave1-private-ip-address>
11) cd
-------------------------------------
12) ansible -m ping all

13) cd  /etc/ansible/

14) mkdir roles
15) cd roles
16) sudo apt install tree

17) sudo ansible-galaxy init web --offline

18) tree web
19) cd web
20) cd tasks
21) ls
22) nano main.yml
    #tasks file for apache
 - include: install.yml
 - include: configure.yml
 - include: service.yml
23) sudo nano install.yml
---
  -
   apt: "name=apache2 state=latest"
   name: "Install Apache"
24)sudo nano configure.yml
---
   - name: Configure Websiteservic
     copy: src=index.html dest=/var/www/html
25)sudo nano service.yml
---
  - name: Start apache2 service
    service: name=apache2 state=started

26)cd /etc/ansible/roles/web/ cd files
27)sudo nano index.html
<html>
<body>
<center>
<h1> Welcome to Sample web </h1>
</body>
</html>
28)cd /etc/ansible > sudo nano site.yml
---
```

```
-
  hosts: webservers
  roles:
    - web
29)sudo ansible-playbook site.yml --syntax-check
30)sudo ansible-playbook site.yml
31) Open Browser , check with Ansible-slave DNS
=========================================================================
```

Ansible Roles :

-Roles simplifies writing complex playbooks
-Roles allows you to reuse common configuration steps between different types of servers.
-Roles are flexible and easily modified.

```
-new_role
       -default (Folder) : Store data about the role , also store default
variable.
               |_main.yaml
       -files(Folder)    : stores the file that needs to be pushed to the remote
machine.
       -handlers(Folder) : tasks that get triggered from some action.
               |_main.yaml
       -meta (folder): meta data means data about data,Information about
auther,supported platforms and dependecies.
         |_main.yaml
 -tasks (Fol) :Contains main list of tasks to be executed by the role.
               |_main.yaml
 -templates  : contains templates which can be deployed via this role.
 -tests  :
               |_inventory
               |_test.yaml
 -vars   : Stores variables with higher priority than default variablea.Difficult
to override.
               |_main.yaml
```

```
sudo apt-get remove apache2
sudo apt-get purge apache2
```

Docker Compose file: A properly formatted Docker Compose file defining the application architecture and its dependencies.

Docker Compose file: Examples step by step process

Launch 2 VM's

1.docker master

2.docker slave-node (install docker)

On Master Node:

```
apt-get update -y

apt-get install docker.io -y

docker --version

docker swarm init --advertise-addr=192.168.1.219 (private ip)
 -copy docker join token and place in docker slave-node
```

```
docker node ls

docker service create --name httpd_srv --replicas 3 -p 83:80 httpd

docker ps

docker service ls
docker service scale httpd_srv=10

docker ps


docker service create --name digital_srv --replicas 3 -p 85:80 shashikantht/digital

-----------------------------------------------------------------------------------
--------
docker swarm commands :
-----------------------
use this Link : https://docs.docker.com/engine/reference/commandline/swarm/

  ca          Display and rotate the root CA
  init        Initialize a swarm
  join        Join a swarm as a node and/or manager
  join-token  Manage join tokens
                      Eg:docker swarm join-token worker
  leave       Leave the swarm
  unlock      Unlock swarm
  unlock-key  Manage the unlock key
  update      Update the swarm



docker swarm join --token SWMTKN-1-
09h4c69vpcpdikxopyqlg4qrz1ilkc5t0qn70wltmoxcmb8mmw-a5efbgkskapmanxps9pg9qbe4
13.58.184.233:2377


On ubuntu :

sudo curl -L "https://github.com/docker/compose/releases/download/1.25.0/docker-
compose-$(uname -s)-  $(uname -m)" -o /usr/local/bin/docker-compose

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

#Application@1

docker-compose.yml

```
=============
---
version: "3"
services:
  sample1:
    image: httpd
    ports:
      - "80:80"
  sample2:
    image: nginx
    ports:
      - "82:80"


>docker-compose up –d
>docker images
```

```
>docker ps
>Check in the Browser

==============================
```

#Application@2

```
---
services:
  databases:
    environment:
      - MYSQL_ROOT_PASSWORD=password
      - MYSQL_USER=user
      - MYSQL_PASSWORD=password
      - MYSQL_DATABASE=demo_db
    image: mysql
    ports:
      - "3306:3306"
  web:
    image: nginx
    ports:
      - "84:80"
version: "3"
```

#Application@3 Database setup

```
mkdir docker1
cd docker1
nano docker-compose.yml
==================
version: '3.0'
services:
   db:
     image:  hshar/mysql:5.6
     volumes:
       - db_data:/var/lib/mysql
     restart: always
     environment:
       MYSQL_ROOT_PASSWORD: intelli
       MYSQL_DATABASE: docker
       MYSQL_USER: root
       MYSQL_PASSWORD: intelli
   webapp:
     depends_on:
       - db
     image: shashikantht/webapp
     ports:
       - "8000:80"
     restart: always

volumes:
    db_data:

>docker-compose up –d
>docker images
>docker ps
>docker exec -it xxx  bash  (docker1_webapp_1 container-id)

>docker exec -it xxx  bash  ( docker1_db_1 container-id)
       >mysql –u root –p
       >show databases;
       >use docker;
       >show tables;
       >create table emp(name varchar(20),phone varchar(20));
       >select * from emp;
       >exit
>exit
```

check on browser

===

Using Compose is basically a three-step process:

1.Define your app's environment with a Dockerfile so it can be reproduced anywhere.

2.Define the services that make up your app in docker-compose.yml
so they can be run together in an isolated environment.

3.Run docker-compose up and Compose starts and runs your entire app.

Kubernetes deployment manifests: Example cluster Creation

===

1)Launch t2.medium ubuntu 20.04 as k-master

2)Launch t2.micro ubuntu 20.04 as k-WN

....

7) kubeadm init

-this will generate the join token

-paste this token on kubernetes WorkerNode

```
8)  mkdir -p $HOME/.kube
    sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
    sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

```
9)kubectl apply -f
https://github.com/weaveworks/weave/releases/download/v2.8.1/weave-daemonset-k8s.yaml
```

10)kubectl get nodes

11)**nano nginx.yaml**

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

```
          image: nginx
          ports:
            - containerPort: 80


 kubectl create -f nginx.yml

 kubectl get po -o wide

  curl 10.44.0.2

Note : change replicas no ,save and exit nginx.yml

 kubectl apply -f nginx.yml

 kubectl get pods --all-namespaces

 Note:Services

       -clusterip
       -loadbalancer
       -nodeport
       -to delete service : kubectl delete service nginx


kubectl create service clusterip nginx --tcp=80:80
kubectl create service clusterip httpd --tcp=82:80

kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v0.35.0/deploy/static/provider/baremetal/deploy.yaml
```

demo.yml

```
---
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
  name: demo-ingress
spec:
  rules:
    - http:
        paths:
          - backend:
              serviceName: nginx
              servicePort: 80
            path: /nginx
          - backend:
              serviceName: httpd
              servicePort: 82
            path: /httpd

kubectl create -f demo.yml

kubectl get nodes
kubectl get pods
kubectl get svc

kubectl get ing
kubectl  get svc  --all-namespaces
kubectl get svc -n ingress-nginx
```

Dashboard : Kubernetes Dashboard setup

```
 Refer below link for K8S Dashboard.
 https://github.com/kubernetes/dashboard

 1) kubectl apply -f
https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.3/aio/deploy/recommended.yaml
 2) kubectl get svc -n kubernetes-dashboard
 3) kubectl edit svc -n kubernetes-dashboard
    Note: change type from clisterIP to Nodeport
 k8s-app: kubernetes-dashboard
 sessionAffinity: None
 type: NodePort
 status:
 loadBalancer: {}
 kind: List
 metadata: {}

 4)Now Generate token :
   i)create two yml files (sa.yml,crb.yml)
```

#create service account
sa.yml

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin-user
  namespace: kubernetes-dashboard

#Bind ClusterAdmin Role to the service account

 crb.yml
 --------
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: admin-user
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  -
    kind: ServiceAccount
    name: admin-user
    namespace: kubernetes-dashboard

   >kubectl create -f sa.yml
   >kubectl create -f crb.yml
   >kubectl -n kubernetes-dashboard describe secret $(kubectl -n kubernetes-dashboard get secret | grep admin-user | awk '{print $1}')

   kubectl -n kubernetes-dashboard get service kubernetes-dashboard
 on Browser
   k8s-master-public-ip:Nodeport
   copy token from terminal and paste on browser
```

Example 1 : App1 # replace Backend file or frontend file in App1

Install and Set Up kubectl on Windows:
https://kubernetes.io/docs/tasks/tools/install-kubectl-windows/

```
       https://dl.k8s.io/release/v1.23.0/bin/windows/amd64/kubectl.exe
```

download kubectl.exe and place in side folder

configure IAM user in cmd
cd C:\Users\user\Desktop\users

```
       aws eks --region <region-code> update-kubeconfig --name <cluster_name>

       aws eks --region us-east-2 update-kubeconfig --name eks-demo
       aws eks --region ap-south-1 update-kubeconfig --name eks-demo

       kubectl get nodes
       kubectl get svc
       notepad nginx.yml
       kubectl create -f nginx.yml
       kubectl get pods
       kubectl create service loadbalancer nginx --tcp=80:80
       kubectl get svc
       -copy External IP from cmd (a9f1d2b9be55c40beba9981f8d787d1b-1361866503.us-east-1.elb.amazonaws.com)
        Paste on the browser
       -check loadbalancers on top EC2 Dashboard,
```

nginx file

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx
          ports:
            - containerPort: 80
```

Above mentioned are Daily practice scripting .yml files just replace frontend and Backend on source deployment files.

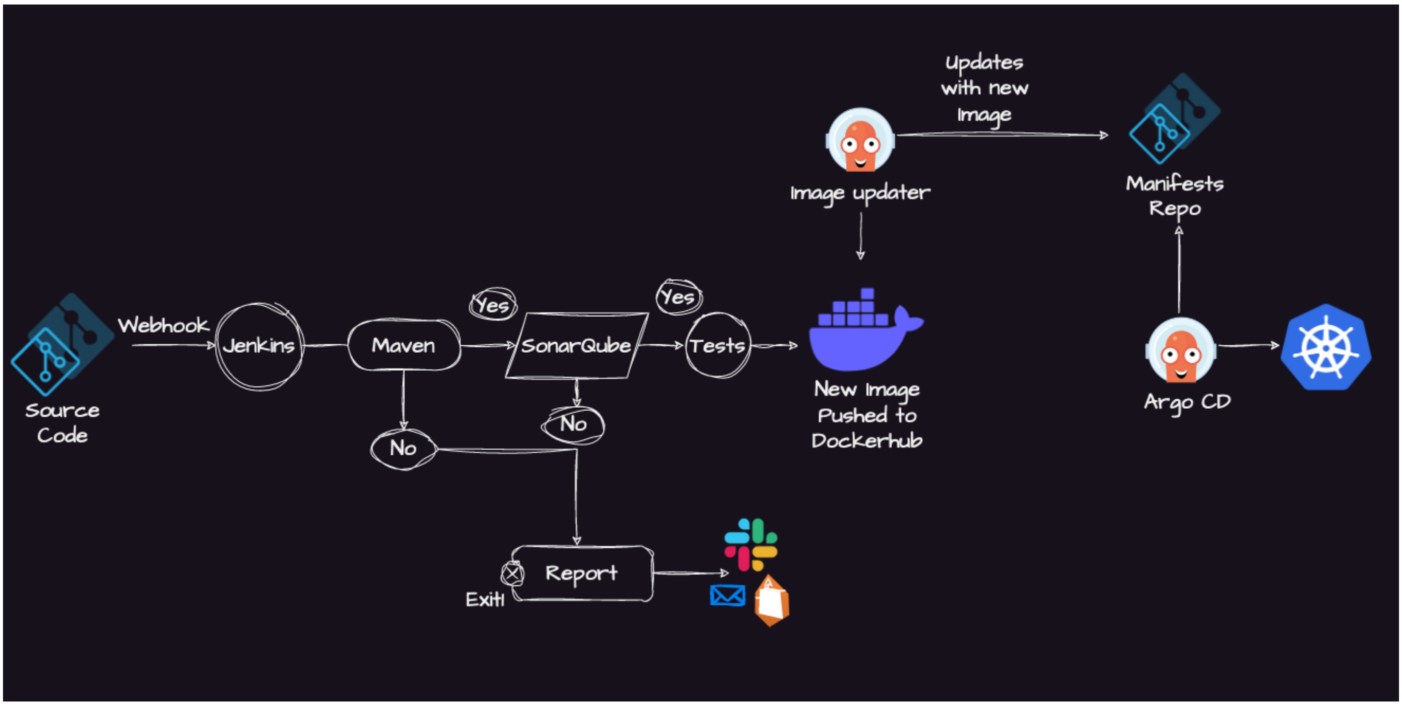
Other Tasks : for Clear Understanding

Project: End to End CI/CD pipeline

1. Setting Up a CI/CD Pipeline (Jenkins CI & Argo CD)

https://github.com/Srinu298/Solulab-task

- We can also Use Ansible in CD with Configure files
- Ansible Tasks and Roles



End to End CI/CD pipeline

Tools

SCM: GitHub

Jenkins: CI

Maven: Build

SonarQube: Code Analysis (QA)

Docker: Docker file Image Created

Argo CD: CD

Kubernetes: Application deployment Container Orchestration