SCENARIO:

* We have a Multi Tier Web Application Stack which is already
* Containerized (Web App done in Docker ), and We have also:
* Tested it
* Now it is time to: host for Production

REQUIREMENTS:

* High Availability
* Fault Tolerance
* Easily Scalable
* Platform Independent
* Portable & Flexible
* We should be able to run our containers on:
* Local Machines
* Physical
* cloud
* Virtual Machines
* Dev, QA, Production , Different Environment …… Easily and conveniently

TECHNOLOGIES WE WOULD USE / REQUIRED STEPS:

* Our Java VPROFILE Application Services (i.e. Containerized Apps – vprofile, Check Docker)
* Run it on Kubernetes Cluster for production
* Create EBS volume for DB Pod
* Label Node with zones names
* Kubernetes Definition files for:
* Deployment
* service
* Secret
* Volume

**STEPS / GUIDELINES:**

1. Containerization process

* setup Docker Engine

**Installation Steps:**

1. Setup an EC2 instance using Ubuntu 18

* select T2.micro and use all defaults
* Key = Name and Value = Docker-Engine
* Create a Security group
* Security group name = Docker-SG
* SSH / TCP / 22 / MyIP
* click on next and create a key pair
* Key pair name = docker-key
* download the .csv and click on Launch

1. Follow the Documentation on (https://docs.docker.com/engine/install/ubuntu/),

* Select Ubuntu
* Go to GitBash and SSH into the Docker EC2
* ssh –i Downloads/docker-key.pem ubuntu@public IP of the EC2
* sudo –i
* Updating the apt package index and installing packages to allow apt to use a repository:
* sudo apt-get update
* sudo apt-get install \

apt-transport-https \

ca-certificates \

curl \

gnupg-agent \

software-properties-common -y

* Adding Docker’s official GPG key:

sudo install -m 0755 -d /etc/apt/keyrings

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

sudo chmod a+r /etc/apt/keyrings/docker.gpg

* Setup the Repository :

echo \

"deb [arch="$(dpkg --print-architecture)" signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \

"$(. /etc/os-release && echo "$VERSION\_CODENAME")" stable" | \

sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

* Install Docker Engine

sudo apt-get update

sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin -y

* Verify that the Docker Engine installation is successful by running the hello-world image:

systemctl status docker

sudo docker run hello-world (docker ps –a would show you the dummy container)

* If you want to Add a User to the list of Docker users

sudo usermod –aG docker (username .e.g. Ubuntu)

* To check if the User exist in the group

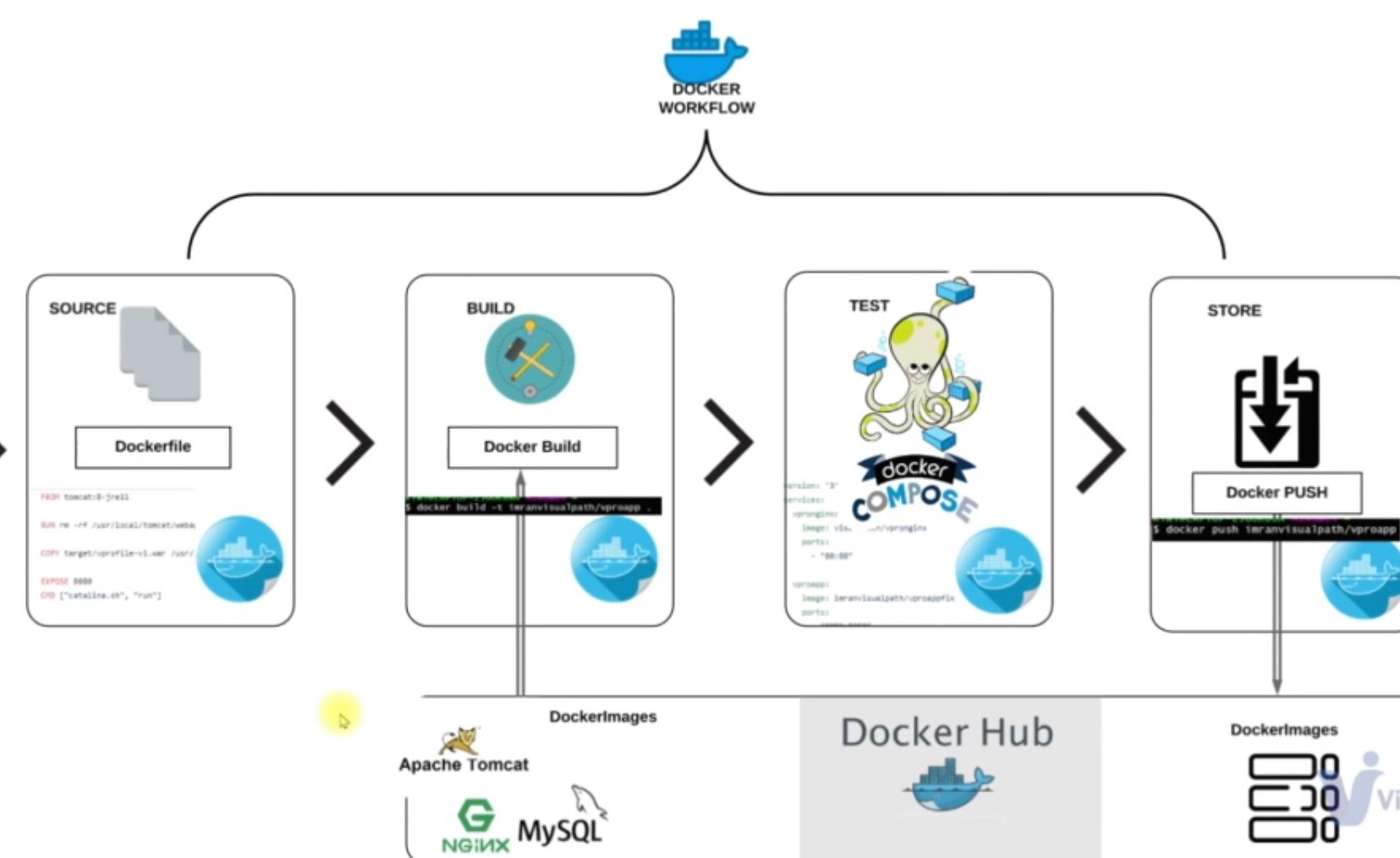
sudo vim /etc/group

* Create container volume

STEPS TO CREATE DOCKER VOLUMES:

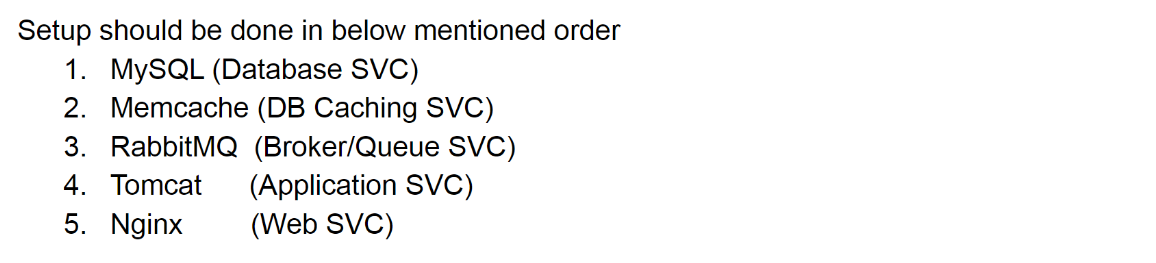
1. Using a Database framework, search for mysql in hub.docker.com
2. sudo -i
3. docker pull mysql:5.7
4. docker images (to findout the size of the directory)
5. docker inspect mysql:5.7 (To findout the port mapping, ID, name and many more in json format)
6. Binding with host directory volume
7. mkdir vprodbdata
8. docker run - -name vprodb –d –e MYSQL\_ROOT\_PASSWORD=secretpass –p 3030:3306 –v /home/ubuntu/vprodbdata:/var/lib/mysql mysql:5.7
9. docker ps (to check running status)
10. ls vprodbdata/
11. docker exec –it (container name ….. vprodb) /bin/bash (to login)
12. cd /var/lib/mysql
13. ls
14. you should find same data
15. I want to stop & delete the container to see if it remains in the system directory
16. docker stop vprodb
17. docker rm vprodb
18. ls
19. ls vprodbdata/ (the data should still be there)
20. docker volume (to show a list of commands for volume)
21. E.g. ….. docker volume create mydbdata
22. docker volume ls
23. docker run - -name vprodb –d –e MYSQL\_ROOT\_PASSWORD=secretpass –p 3030:3306 –v /mydbdata:/var/lib/mysql mysql:5.7
24. ls /var/lib/docker/volumes/mydbdata/\_data/
25. docker inspect (container name ….. vprodb) (To get info for the command below)
26. mysql -h (IP of the container) -u root -p(password)
27. you should be logged in
28. show databases;

* Containerize your Artifact (On Vagrantfile Vm setup or EC2)



STEPS:

1. Using the resource in the project (C:\gitrepos\cloudcodesandsecurity\DevOps\Projects\Vprofile\_Project\_Setup\Manual\_provisioning)



1. Create a Docker Engine EC2 instance follow steps in the doc: Setup of Docker
2. open GitBash and don’t SSH yet
3. cd c:/
4. mkdir docker-engine
5. cd docker-engine/
6. vagrant init ubuntu/bionic64
7. vim Vagrantfile

Uncomment: config.vm.network “private\_network”, ip: “set and note your IP” (e.g. 192.168.33.12)

:wq

1. vagrant up
2. vagrant ssh
3. don’t do sudo –i yet
4. We need to install Docker Engine in the Vagrant

* Follow the Documentation on (<https://docs.docker.com/engine/install/ubuntu/> ),
* Updating the apt package index and installing packages to allow apt to use a repository:
* sudo apt-get update
* sudo apt-get install \

apt-transport-https \

ca-certificates \

curl \

gnupg-agent \

software-properties-common -y

* Adding Docker’s official GPG key:

sudo install -m 0755 -d /etc/apt/keyrings

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

sudo chmod a+r /etc/apt/keyrings/docker.gpg

* Setup the Repository :

echo \

"deb [arch="$(dpkg --print-architecture)" signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \

"$(. /etc/os-release && echo "$VERSION\_CODENAME")" stable" | \

sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

* Install Docker Engine

sudo apt-get update

sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin -y

* Verify that the Docker Engine installation is successful by running the hello-world image:

systemctl status docker

docker - -version

1. whoami (To get the user )
2. Add the User to the Docker group to have access
3. sudo usermod –a –G Docker (username .e.g. vagrant)
4. exit and login again
5. vagrant ssh
6. id (To check if Vagrant is part of Docker group)
7. docker images (To check the list of images)
8. sudo systemctl status docker
9. We need to build Dockerfile to host the services, in step 1 above
10. Use Intellij IDEA on the system and save it into same folder where the vagrantfile is:
11. Open the Intellij, click on file ….. New ….. Project from Version Control

URL = https://github.com/devopshydclub/vprofile-project.git (The URL of the github project to clone)

Directory = Select the Directory where the Vagrantfile is (Docker-engine) and create a folder named vprofile-project

click on clone

1. Click on Master at the extreme right bottom, to change the branch to origin/docker by clicking on checkout
2. On the left pane, in Docker-files we would write our Dockerfile and save it
3. Go to hub.docker.com and login
4. click on organization tab above and create a free 1 -3 team member
5. Create a Repository inside the Organization
6. click on the first drop down arrow and select the organization name you created
7. Name = vprofileapp (These Repo will be for Tomcat)
8. check public and click on create

* Create another Repository inside the Organization
* click on the first drop down arrow and select the organization name you created
* Name = vprofileweb (These Repo will be for Nginx)
* check public and click on create
* Create another Repository inside the Organization
* click on the first drop down arrow and select the organization name you created
* Name = vprofiledb (These Repo will be for MySQL)
* check public and click on create

1. Return back to the Intellij, right click on the folder (Docker-files)
2. click on New, click on Directory
3. Give the Directory a name (app)
4. Right click on the App directory, click on New and click on file
5. Give it the name = Dockerfile
6. click on Add , to accept the Git request if any
7. Go to hub.docker.com and search for Tomcat official with Tag = 8-jre11

FROM tomcat:8-jre11  
RUN rm -rf /usr/local/tomcat/webapps/\*  
COPY target/vprofile-v2.war /usr/local/tomcat/webapps/ROOT.war  
  
EXPOSE 8080  
CMD ["catalina.sh", "run"]  
WORKDIR /usr/local/tomcat/  
VOLUME /usr/local/tomcat/webapps/

1. Return back to the Intellij, right click on the folder (Docker-files)
2. click on New, click on Directory
3. Give the Directory a name (db)
4. Right click on the db directory, click on New and click on file
5. Give it the name = Dockerfile
6. click on Add , to accept the Git request if any
7. Go to hub.docker.com and search for MySQL official with Tag = 5.7.25

FROM mysql:5.7.25  
LABEL "Project"="Vprofile"  
LABEL "Author"="Ola-Gabriel"  
  
ENV *MYSQL\_ROOT\_PASSWORD*="vprodbpass"  
ENV *MYSQL\_DATABASE*="accounts"  
  
  
ADD db\_backup.sql docker-entrypoint-initdb.d/db\_backup.sql

1. Return back to the Intellij, right click on the folder (Docker-files)
2. click on New, click on Directory
3. Give the Directory a name (web)
4. Right click on the web directory, click on New and click on file
5. Give it the name = Dockerfile
6. click on Add , to accept the Git request if any
7. Go to hub.docker.com and search for Nginx official with Tag = Default is ok, don’t specify

FROM nginx  
LABEL "Project"="Vprofile"  
LABEL "Author"="Ola-Gabriel"  
  
RUN rm -rf /etc/nginx/conf.d/default.conf  
COPY nginvproapp.conf /etc/nginx/conf.d/vproapp.conf

1. Go to the GitBash, and create the Target/vprofile:V2.war (Artifact) for the Tomcat service
2. docker - -version
3. cd /vagrant/
4. ls
5. cd vprofile-project/
6. ls
7. cd Docker-files/
8. ls
9. ls –R
10. cd ../..
11. cd vagrant/
12. cd vprofile-project/
13. ls
14. The pom.xml is the artifact, we need jdk and Maven to build
15. sudo apt install openjdk-8-jdk –y && sudo apt install maven –y
16. We need to fillin the details of the Backend services to connect to the Db
17. check the application.properties file in these (C:\gitrepos\cloudcodesandsecurity\DevOps\ Docker\Containerizing Project\vprofile-project\src\main\resources) for configuration details
18. pwd
19. ls
20. mvn install
21. ls
22. ls target/
23. ls Docker-files/app/
24. cp –r target Docker-files/app/
25. ls
26. cd Docker-files/app/
27. ls
28. docker build –t vprofile/vprofileapp:V1 **.** (These command builds the Docker image and pushes to the repository)
29. docker images
30. cd ../
31. ls
32. cd db/
33. docker build –t vprofile/vprofiledb:V1 **.** (These command builds the Docker image and pushes to the repository)
34. docker images
35. cd ../
36. cd web/
37. docker build –t vprofile/vprofileweb:V1 **.** (These command builds the Docker image and pushes to the repository)
38. docker images
39. We need 2 other Images (Rabbitmq & Memcache), we don’t need to customize them. Just pull them
40. cd web/
41. docker pull memcached
42. docker pull rabbitmq
43. We need to install Docker compose to run all these containers and connect all (the 5 services) together
44. Use the documentation: (docs.docker.com/compose/install/ ) (video timestamp is 51.32mins)
45. Install Docker Desktop and it comes with compose. Use the documentation as a strong guide
46. the path should be: /usr/local/bin/docker-compose
47. sudo chmod +x /usr/local/bin/docker-compose
48. docker-compose - -version
49. Use intellij to write the Docker compose YAML file, save as: docker-compose.yml
50. version: '3'  
    services:  
     vprodb:  
     image: vprofile/vprofiledb:V1  
     ports:  
     - "3306:3306"  
     volumes:  
     - vprodbdata:/var/lib/mysql  
     environment:  
     - MYSQL\_ROOT\_PASSWORD=vprodbpass  
      
     vprocache01:  
     image: memcached  
     ports:  
     - "11211:11211"  
      
     vpromq01:  
     image: rabbitmq  
     ports:  
     - "15672:15672"  
     environment:  
     - RABBITMQ\_DEFAULT\_USER=guest  
     - RABBITMQ\_DEFAULT\_PASS=guest  
      
     vproapp:  
     image: vprofile/vprofileapp:V1  
     ports:  
     - "8080:8080"  
     volumes:   
     - vproappdata:/usr/local/tomcat/webapps  
      
     vproweb:  
     image: vprofile/vprofileweb:V1  
     ports:  
     - "80:80"  
    volumes:  
     vprodbdata: {}  
     vproappdata: {}
51. Go to Docker engine in GitBash
52. cd /vagrant/vprofile-project/
53. mkdir compose
54. mv docker-compose.yml compose/
55. cd compose/
56. ls
57. docker-compose - -help (To view a list of commands)
58. docker-compose up –d
59. docker logs (container name) to see logs of the container
60. Go to your browser: (Use the public IP for your Vagrantfile you specified at the beginning):80
61. If everything is fine in the browser, it is time to push the images to Docker Hub
62. docker login
63. username = account name
64. password = set pass
65. docker push vprofile/vprofileapp:V1 (repository name/project name)

* k

1. Create your Kubernetes cluster with Kops,

**STEPS: SETUP WITH KOPS (PREREQUISITES): USING EC2**

1. Login to your AWS account and go to EC2 instances
2. Click on Launch an Instance

* AMI = Ubuntu server 20.04
* instance type = t2.micro
* Number of instances: 1
* Security group = create a security group, name = kops-sg, SSH / TCP / 22 / MyIP
* Keypair = kops-key, download the .csv
* 8GB SSD
* key = name & value = kops
* click on launch

1. Create an S3 Bucket:

* search for S3 Buckets under Services
* click on create a bucket:
* Bucket Name = vprofile-kops-state
* select your EC2 region
* click on create

1. Create an IAM User for AWSCLI & other services (S3):
   * search for IAM under services
   * click on Add User

* user name = kopsadmin
* check programmatic access
* click on Attach existing policies directly
* check on AdministratorAccess
* click on next and click on create
* download the .csv and be very careful with it

1. Create Route53:
   * search for Route 53 under services
   * click on create hosted zone
2. Domain name = kubevpro.groophy.in
3. check Public hosted zone
4. click on Create hosted Zone
5. Scroll down and take note of the Value/Route traffic to info
6. login to your Domain name host provider (truehost)
7. Go to DNS settings / NS Records
8. Type = Nameserver, Host = kubevpro (Domain name provided above), Points to = take the first value in step d above)
9. click on save

* Type = Nameserver, Host = kubevpro (Domain name provided above), Points to = take the second value in step d above)
* click on save
* Repeat step and Do for the third and forth values

1. SSH to your EC2 instance (KOPS) to setup everything:

* Configure the AWSCLI:

1. Go to GitBash
2. cd
3. ssh –i Downloads/kops-key.pem ubuntu@public IP
4. Generate the SSH key to be used by AWSCLI & KOPs
5. ssh-keygen
6. hit enter to accept defaults in the 3 question areas
7. save the Access and Secret keys in word file
8. Install AWSCLI
9. sudo apt update && sudo apt install awscli –y
10. configure the AWSCLI
11. aws configure
12. provide the access and secret keys
13. Region of your EC2
14. Default output = json

* Configure the KOPS: (Use the documentation as a guide: how to install Kubernetes with KOPS)

1. curl –LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl
2. ls (To check the Kubectl binary)
3. chmod +x ./kubectl
4. sudo mv kubectl /usr/local/bin
5. curl –LO https://github.com/kubernetes/kops/releases/download/$(curl –s https://api.github.com/repos/kubernetes/kops/releases/latest | grep tag\_name | cut –d ‘”’ –f 4)/kops-linux-amd64
6. ls
7. sudo chmod +x kops-linux-amd64
8. sudo mv kops-linux-amd64 /usr/local/bin/kops
9. kops - -help
10. nslookup –type=ns kubevpro.groophy.in
11. kops command to create the kubernetes cluster:
12. kops create cluster - -name=kubevpro.groophy.in \ (press Enter)

> - -state=s3://vprofile-kop-states - -zones= us-east-2a, us-east-2b \ (press Enter)

> - -node-count=2 - -node-size=t3.small - -master-size=t3.medium - -dns-zone=kubevpro.groophy.in \ (press Enter)

> - -node-volume-size=8 - -master-volume-size=8 (press Enter to start creating)

1. configure the cluster: The command will show immediately after last command
2. kops update cluster - -name kubevpro.groophy.in - -state=s3://vprofile-kop-states - -yes - -admin (remember to include your s3 bucket)
3. Create a Repo in Github, to host the definition files for: Deployment, service, secrets and volumes
4. Login to Github. and click on create a new repository
5. Name = kube-app
6. Repo should be Public and click on create
7. clone the repo in the intellij Ide
8. Open intellij and click on get from version control
9. URL = The repo url
10. Directory = C:/kube-app
11. click on clone
12. Activate Kubernetes plugin
13. click on file and click on settings
14. On the left pane, click on plugins and search for kubernetes
15. click on install and click on ok when completed
16. check the health of the cluster
17. kops validate cluster - -name vprokube.groophy.in - -state=s3://vprofile-kop-states
18. cat ~/.kube/config (These file is used by kubectl to connect to the clusters)
19. kubectl get nodes (You can confirm by checking the EC2 (3ec2), Autoscaling groups (3groups), VPC and Route 53)
20. If you are not using the Cluster, Please stop to avoid Billing:

kops delete cluster - -name=kubevpro.groophy.in - -state=s3://vprofile-kop-states - -yes

1. sudo poweroff
2. Create EBS volume for DB Pod: (Store MySQL data into EBS volume)

* aws ec2 create-volume - -availability-zone=us-east-2a - -size=3 - -volume-type=gp2
* Take note of the VolumeID and save in word doc
* make sure the DB pod runs in the same zone using Node selector in the definition file which works with labels
* We need to give a Tag so as it can be identified in Definition file
* Go to AWS console service, search for Elastic Block Store and click on volumes
* check the volume you created and scroll down
* click on Tags tab beside Monitoring
* click on add Tags (Key = kubernetes, value = vprokube.groophy.in (cluster name) )
* click on save

1. Label Node with zones names

* kubectl get nodes - -show-labels
* kubectl get nodes
* kubectl describe node (put the name of one of the nodes to view it’s label) | grep us-east-2
* check the zone
* kubectl label nodes (same node you picked) zone=us-east-2a
* Label the next node with us-east-2b
* kubectl describe node (put the name of the second one of the nodes to view it’s label) | grep us-east-2
* check the zone
* kubectl label nodes (same node you picked) zone=us-east-2b

1. Kubernetes Definition files for: (Deployment and Service)
2. Go to your hub.docker.com account
3. click on organization and click on repositories
4. inspect your Images from Docker section
5. Images that will be used:

image vprofiledb and image vprofiledb

1. image vprofileweb (Nginx image) wouldn’t be used. Instead i will spring up an Elastic LoadBalancer instead of that
2. You should have (application.properties) file, check for config details
3. take the passwords and encode them each following steps below
4. go to gitbash,
5. SSH to your instance as Ubuntu user
6. echo –n “vprodbpass” | base64
7. copy the encoded value and paste in word
8. k
9. Definition files to host the secrets such as passwords:
10. Open your Intellij IDE
11. save it as app-secret.yaml under the folder kube-app

apiVersion: v1

kind: Secret

metadata:

name: app-secret

type: Opaque

data:

db-pass: (paste the encoded value)

rmq-pass: (paste the encoded value)

1. save through your IDE and commit your changes
2. commit message = secret
3. check (perform code analysis and check TODO)
4. click on commit
5. Quickly run a test
6. click on push commits to kube-app
7. click on push
8. Go to GitBash and Let’s fetch it on the Kops Vm
9. git clone (the repo https link)
10. cd kube-app/
11. ls
12. cat app-secret.yaml
13. Let’s create it
14. kubectl create -f app-secret.yaml
15. kubectl get secret
16. kubectl describe secret
17. kubectl get nodes show labels
19. **DB Deployment definition file: (Open a new file in intellij)**
20. save as (vprodbdep.yaml)

apiVersion: apps/v1

kind: Deployment

metadata:

name: vprodb

labels:

app: vprodb

spec:

selector:

matchLabels:

app: vprodb

replicas: 1

template:

metadata:

labels:

app: vprodb

spec:

containers:

- name: vprodb

image: vprofile/vprofiledb:1 (image name from hub.docker)

args:

- “- -ignore-db-dir=lost+found” (This is to ignore the error created as a result of the format below)

volumeMounts:

- mountPath: /var/lib/mysql

name: vpro-db-data (The name of the volume created below under volumes)

ports:

- name: vprodb-port

containerPort: 3306 (it is specified in the application.properties file)

env:

- name: MYSQL\_ROOT\_PASSWORD

valueFrom:

secretKeyRef:

name: app-secret

key: db-pass

nodeSelector:

zone: us-east-2a

volumes:

- name: vpro-db-data

awsElasticBlockStore:

volumeID: (paste the VolumeID, you saved in word)

fsType: ext4

1. save the code and right click to commit it to github

commit Message = db pod ….. then click on commit

1. Let’s push it

right click on the code and click on push

1. Go to GitBash and SSH then use the Ubuntu user
2. git pull
3. kubectl create -f vprodbdep.yaml
4. kubectl get deploy
5. kubectl get pod
6. kubectl describe pod (name of the db pod created)
7. **Let’s Create the DB Service Definition**
8. Open your Intellij and on the left pane, right click on your dir where other definition files are, then create a new file
9. save it as: db-CIP.yaml
10. open your application.properties to get info for creating the db service

apiVersion: v1

kind: Service

metadata:

name: vprodb

spec:

ports:

- port: 3306

targetPort: vprodb-port

protocol : TCP

selector:

app: vprodb

type: ClusterIP

1. save, commit and push
2. **Let’s create the Memcached Deployment & Service**
3. Open your Intellij and on the left pane, right click on your dir where other definition files are, then create a new file
4. save it as: mcdep.yaml
5. open your application.properties to get info for creating the memcached deployment

apiVersion: apps/v1

kind: Deployment

metadata:

name: vpromc

labels:

app: vpromc

spec:

selector:

matchLabels:

app: vpromc

replicas: 1

template:

metadata:

labels:

app: vpromc

spec:

containers:

- name: vpromc

image: memcached (official Memcached image from docker hub)

ports:

- name: vpromc-port

containerPort: 11211

save, commit and push

1. **Memcache Service Definition file**
2. Open your Intellij and on the left pane, right click on your dir where other definition files are, then create a new file
3. save it as: mc-CIP.yaml
4. open your application.properties to get info for creating the memcached service

apiVersion: v1

kind: Service

Metadata

name: vprocache01

spec:

ports:

- port: 11211

targetPort: vpromc-port

protocol: TCP

selector:

app: vpromc

type: ClusterIP

1. save, commit and push
2. **Let’s create the RabbitMQ Deployment & Service**
3. Open your Intellij and on the left pane, right click on your dir where other definition files are, then create a new file
4. save it as: rmq-dep.yaml
5. open your application.properties to get info for creating the RabbitMQ deployment

apiVersion: apps/v1

kind: Deployment

metadata:

name: vpromq01

labels:

app: vpromq01

spec:

selector:

matchLabels:

app: vpromq01

replicas: 1

template:

metadata:

labels:

app: vpromq01

spec:

containers:

- name: vpromq01

image: rabbitmq (official RabbitMQ image from docker hub)

ports:

- name: vpromq01-port

containerPort: 15672

env:

- name: RABBITMQ\_DEFAULT\_PASS

valueFrom:

secretKeyRef:

name: app-secret

key: rmq-pass

- name: RABBITMQ\_DEFAULT\_USER

value: “guest”

save, commit and push

1. **RabbitMQ Service Definition file**
2. Open your Intellij and on the left pane, right click on your dir where other definition files are, then create a new file
3. save it as: rmq-CIP.yaml
4. open your application.properties to get info for creating the RabbitMQ service

apiVersion: v1

kind: Service

Metadata

name: vpromq01

spec:

ports:

- port: 15672

targetPort: vpromq-port

protocol: TCP

selector:

app: vpromq01

type: ClusterIP

save, commit and push

…………………………….

1. **Let’s create the TOMCAT Deployment, Service & Init containers**
2. Open your Intellij and on the left pane, right click on your dir where other definition files are, then create a new file
3. save it as: vproappdep.yaml
4. open your application.properties to get info for creating the Tomcat deployment

apiVersion: apps/v1

kind: Deployment

metadata:

name: vproapp

labels:

app: vproapp

spec:

selector:

matchLabels:

app: vproapp

replicas: 1

template:

metadata:

labels:

app: vproapp

spec:

containers:

- name: vproapp

image: vprofile/vprofileapp:V1 (Customize image from docker hub containing our Artifact)

ports:

- name: vproapp-port

containerPort: 8080

initContainers: (To check the db container completion status b4 vproapp can start)

- name: init-mydb

image: busybox (From an Image)

command: [‘sh’, ‘-c’, ‘until nslookup vprodb; do echo waiting for mydb; sleep 2; done;’]

- name: init-memcache

image: busybox

command: [‘sh’, ‘-c’, ‘until nslookup vprocache01; do echo waiting for mydb; sleep 2; done;’]

save, commit and push

1. **Tomcat Service Definition file**
2. Open your Intellij and on the left pane, right click on your dir where other definition files are, then create a new file
3. save it as: vproapp-service.yaml
4. open your application.properties to get info for creating the Tomcat service

apiVersion: v1

kind: Service

Metadata:

name: vproapp-service

spec:

ports:

- port: 80

targetPort: vproapp-port

protocol: TCP

selector:

app: vproapp

type: LoadBalancer

save, commit and push

1. **Provision your stack on K8s Cluster**
2. In your intellij, check to make sure the following files have been created & saved successfully. (app-secret.yaml, db-CIP.yaml, mc-CIP.yaml, mcdep.yaml, rmq-CIP.yaml, rmq-dep.yaml, vproapp-service.yaml, vproappdep.yaml, vprodbdep.yaml)
3. Go to GitBash and SSH to your EC2 you created above
4. use Ubuntu user
5. ls
6. cd kube-app/
7. ls
8. git pull (to pull all the files – Definition & Services)
9. kubectl create –f **.**
10. kubectl get deploy
11. kubectl get pod
12. kubectl get svc (complete the LoadBalancer endpoint)
13. Go to AWS console and click on LoadBalancer to check completion status
14. Go to your browser and paste the loadbalancer endpoint
15. **Let’s give our K8s project a URL**
16. You can use Godaddy or Route53
17. Go to AWS console and search for Route53
18. click on Hosted Zones and scroll down to the hosted zone you created, click on it
19. click on create a record and select simple records
20. click on define simple record
21. record name = blog
22. Value / Route traffic (Endpoint) = Alias to Application and Classic Load Balancer
23. Region = select the region of your project
24. loadbalancer = select your loadbalancer endpoint
25. Record type = A-Routes traffic to an IPv4 address and some AWS resources
26. click on Define simple records and click on create records
27. Scroll down to the list of records you have
28. select the url of your record and paste in your browser to test
29. **Clean up process**
30. Go to GitBash
31. In the same Directory where your application is
32. kubectl delete –f **.**
33. kops delete cluster - -name vprokube.groophy.in - -state=s3://vprofile-kops-state --yes
34. cd (return back to Ubuntu user only)