1. adımdayız.
2. ееео 
   НЕИМ 
   clu%ter 
   ,qaster 

Şimdi MSP-18 de uygulamanın paketlenip, kuberneteste yağa kalkma sürecini jenkinsfile a aktaracğız.

git checkout dev

git branch feature/msp-18

git checkout feature/msp-18

Jenkins klasörü altında package-with-maven-container.sh

dosyası içine mvn clean package komutnu çalıştıracak bir docker komutu giriyoruz.

docker run --rm -v $HOME/.m2:/root/.m2 -v $WORKSPACE:/app -w /app maven:3.6-openjdk-11 mvn clean package

bu paketler bize lazım, dockerfile da .jar dosyasını çalıştıracak image oluşturmuştuk bu dosyalar mvn package komutuyla oluşacak.

bu komutun yapısını daha sonra da kullanmak üzere iyice öğrenmek gerekir.

şimdi yine jenkins klasörü altında prepare-tags-ecr-for-dev-docker-images.sh

isimli bir dosya altında docker image ler için ECR tag lerini oluşturacak komutlarını yazacaz..

MVN\_VERSION=$(. ${WORKSPACE}/spring-petclinic-admin-server/target/maven-archiver/pom.properties && echo $version)

export IMAGE\_TAG\_ADMIN\_SERVER="${ECR\_REGISTRY}/${APP\_REPO\_NAME}:admin-server-v${MVN\_VERSION}-b${BUILD\_NUMBER}"

MVN\_VERSION=$(. ${WORKSPACE}/spring-petclinic-api-gateway/target/maven-archiver/pom.properties && echo $version)

export IMAGE\_TAG\_API\_GATEWAY="${ECR\_REGISTRY}/${APP\_REPO\_NAME}:api-gateway-v${MVN\_VERSION}-b${BUILD\_NUMBER}"

MVN\_VERSION=$(. ${WORKSPACE}/spring-petclinic-config-server/target/maven-archiver/pom.properties && echo $version)

export IMAGE\_TAG\_CONFIG\_SERVER="${ECR\_REGISTRY}/${APP\_REPO\_NAME}:config-server-v${MVN\_VERSION}-b${BUILD\_NUMBER}"

MVN\_VERSION=$(. ${WORKSPACE}/spring-petclinic-customers-service/target/maven-archiver/pom.properties && echo $version)

export IMAGE\_TAG\_CUSTOMERS\_SERVICE="${ECR\_REGISTRY}/${APP\_REPO\_NAME}:customers-service-v${MVN\_VERSION}-b${BUILD\_NUMBER}"

MVN\_VERSION=$(. ${WORKSPACE}/spring-petclinic-discovery-server/target/maven-archiver/pom.properties && echo $version)

export IMAGE\_TAG\_DISCOVERY\_SERVER="${ECR\_REGISTRY}/${APP\_REPO\_NAME}:discovery-server-v${MVN\_VERSION}-b${BUILD\_NUMBER}"

MVN\_VERSION=$(. ${WORKSPACE}/spring-petclinic-hystrix-dashboard/target/maven-archiver/pom.properties && echo $version)

export IMAGE\_TAG\_HYSTRIX\_DASHBOARD="${ECR\_REGISTRY}/${APP\_REPO\_NAME}:hystrix-dashboard-v${MVN\_VERSION}-b${BUILD\_NUMBER}"

MVN\_VERSION=$(. ${WORKSPACE}/spring-petclinic-vets-service/target/maven-archiver/pom.properties && echo $version)

export IMAGE\_TAG\_VETS\_SERVICE="${ECR\_REGISTRY}/${APP\_REPO\_NAME}:vets-service-v${MVN\_VERSION}-b${BUILD\_NUMBER}"

MVN\_VERSION=$(. ${WORKSPACE}/spring-petclinic-visits-service/target/maven-archiver/pom.properties && echo $version)

export IMAGE\_TAG\_VISITS\_SERVICE="${ECR\_REGISTRY}/${APP\_REPO\_NAME}:visits-service-v${MVN\_VERSION}-b${BUILD\_NUMBER}"

export IMAGE\_TAG\_GRAFANA\_SERVICE="${ECR\_REGISTRY}/${APP\_REPO\_NAME}:grafana-service"

export IMAGE\_TAG\_PROMETHEUS\_SERVICE="${ECR\_REGISTRY}/${APP\_REPO\_NAME}:prometheus-service"

biz buradaki variable ların isimlerini daha önce values-template.yaml da ve deploymentlarda image isimlerinde gördük.

işte bu variable ların tanımını burada oluşturuyoruz. buradaki amacımız dinamik bir tagleme yapmak. ileride güncellemelerden sonra hangi image in kullanıldığını bu sayede anlayabiliriz. örneğin iki ay sonra eski bir versiona dönmek istersek rahatça bulabilmemiz lazım bu nedenle tage latest vs demiyoruz da dinamik bir tagleme yapıyoruz.

${ECR\_REGISTRY}/${APP\_REPO\_NAME}

bu kısmı Jenkinsfile başında tanımlayacaz:

envlror—ent { 
App 
app 
AHS 
ECR 
_NAMF— 
_ RE P O_NJW E — " cla r.y - 
t S get-caller-identity 
_ dkr _ _ 
- -query Account - 
-output text' , 
returnStdout : true) . ) 
aNs_Ktypn1R- 
ANSI 
ANSIBLF I-OST CHECKING-"Fa1se" 

ECR\_REGISTRY ile APP\_REPO\_NAME i aws deki ECR repomuzdan alacak.

${BUILD\_NUMBER}

bu kısmı jenkinste tanımlı bir değişken ve hangi buildse o numarayı çekiyor.

${MVN\_VERSION}

developerlarçalıştıkça uyguamada değişiklik yaptıkça mvn versionu da değiştirebilirler bu nedenle bunu da dinamik olarak tage ekledik.

MVN\_VERSION=$(. ${WORKSPACE}/spring-petclinic-visits-service/target/maven-archiver/pom.properties && echo $version)

buradaki ilk nokta linux taki src komutunu ifade ediyor.

e c 2- us ere-jet* ins-server* 
ec2•use 
roe ins - server# 
2.1.2 
ec2- 
us e ins-server* 
test:$echo $version 
test:$_ test-txt 
test:$echo *version 
test.$0 

buradaki örnekte test.txt içindeki versionu echo ile yazdırmak için . test.txt ile bu dosyayı source olarak gösteriyoruz. şöyle de yazılır:

ec2-user@enkins - server* 
2.1.2 
ec2- 
u s e r&jeüins - serve" 
test:$source test . txt 
test: 
echo Sversion 

bu pipeline nightly açlışacak bu nedenle her seferinde verison veremeyiz. MVN\_VERSION u da dinamik şekilde çekmesini yukarıdaki komutla sağlamış olduk. bunu da tage eklemiş olduk.

Jenkinsfile tagleri tanımlayınca envsub komutuyla values-template.yaml ı values.yaml ın yerine geçirecez, override yapacaz. sonra deployment taki image de bu isimle image i çekecek.

jenkins klasörü altında build-dev-docker-images-for-ecr.sh

isimli dosya içine;

docker build --force-rm -t "${IMAGE\_TAG\_ADMIN\_SERVER}" "${WORKSPACE}/spring-petclinic-admin-server"

docker build --force-rm -t "${IMAGE\_TAG\_API\_GATEWAY}" "${WORKSPACE}/spring-petclinic-api-gateway"

docker build --force-rm -t "${IMAGE\_TAG\_CONFIG\_SERVER}" "${WORKSPACE}/spring-petclinic-config-server"

docker build --force-rm -t "${IMAGE\_TAG\_CUSTOMERS\_SERVICE}" "${WORKSPACE}/spring-petclinic-customers-service"

docker build --force-rm -t "${IMAGE\_TAG\_DISCOVERY\_SERVER}" "${WORKSPACE}/spring-petclinic-discovery-server"

docker build --force-rm -t "${IMAGE\_TAG\_HYSTRIX\_DASHBOARD}" "${WORKSPACE}/spring-petclinic-hystrix-dashboard"

docker build --force-rm -t "${IMAGE\_TAG\_VETS\_SERVICE}" "${WORKSPACE}/spring-petclinic-vets-service"

docker build --force-rm -t "${IMAGE\_TAG\_VISITS\_SERVICE}" "${WORKSPACE}/spring-petclinic-visits-service"

docker build --force-rm -t "${IMAGE\_TAG\_GRAFANA\_SERVICE}" "${WORKSPACE}/docker/grafana"

docker build --force-rm -t "${IMAGE\_TAG\_PROMETHEUS\_SERVICE}" "${WORKSPACE}/docker/prometheus"

docker build komutlarını giriyoruz.

bu komutun çözümlemesi:

docker build --force-rm -t "${IMAGE\_TAG\_ADMIN\_SERVER}" "${WORKSPACE}/spring-petclinic-admin-server"

-t ile tagleme yapıyoruz.

--force-rm de intermediate container ları isliyor.

Dockerfile da katmanlı yapı vardı:

petclinic-microsewices-with-db > spring-petclinic-admin-server > 
1 
2 
3 
5 
6 
7 
8 
le 
FROM openjdk:ll-jre 
ARG 
ARG 
ENV 
ADD 
RUN 
RUN 
ADD 
DOCKERIZE VERSION=vØ.6.1 
EXPOSED PORT-=gegø 
SPRING PROFILES ACTIVE docker,mysql 
https : //github.com/jwilder/dockerize/relea± 
tar -xzf dockerize.tar.gz 
chmod +x dockerize 
./target/* . jar /app.jar 
EXPOSE ${EXPOSED PORT} 
ENTRYPOINT ["java , ' 
" ' -Djava . security . egd=file: 

buradaki her aşamadan bir diğerine geçerken image alıyor ve her aşamada container lar ayağa kaldırıyor. biz --force-rm dersek arada takılı kalan container ları silmiş oluyoruz. CI/CD toollarında bu şekilde işlemler yapılır ve gereksiz şeyler silinir.

sıra image leri push etmeye geldi jenkins folder altında push-dev-docker-images-to-ecr.sh

isimli doysa içine

aws ecr get-login-password --region ${AWS\_REGION} | docker login --username AWS --password-stdin ${ECR\_REGISTRY}

docker push "${IMAGE\_TAG\_ADMIN\_SERVER}"

docker push "${IMAGE\_TAG\_API\_GATEWAY}"

docker push "${IMAGE\_TAG\_CONFIG\_SERVER}"

docker push "${IMAGE\_TAG\_CUSTOMERS\_SERVICE}"

docker push "${IMAGE\_TAG\_DISCOVERY\_SERVER}"

docker push "${IMAGE\_TAG\_HYSTRIX\_DASHBOARD}"

docker push "${IMAGE\_TAG\_VETS\_SERVICE}"

docker push "${IMAGE\_TAG\_VISITS\_SERVICE}"

docker push "${IMAGE\_TAG\_GRAFANA\_SERVICE}"

docker push "${IMAGE\_TAG\_PROMETHEUS\_SERVICE}"

push komutlarını yapıştırıyoruz.

önce ecr için login komutu sonra da push komutları. ecr a yetkisi olduğu için aws ecr get-login-password komutuyla password alabiliyor.

biz bu komutları jenkins pipeline ında .sh dosyaları halinde çalıştıracağız.

git add .

git commit -m 'added scripts for qa automation environment'

git push --set-upstream origin feature/msp-18

push layıp bgünlük burada bırakalım.

$() içindeki ifadeyi linuxte bir komut olarak çıkarıyordu.

Şimdi Jenkins dashboarda gidip scriptleri deneyeceğimiz test-msp-18-scripts

isimli bir job çalıştıracaz.

PATH="$PATH:/usr/local/bin"

APP\_REPO\_NAME="clarusway-repo/petclinic-app-dev" # Write your own repo name

AWS\_REGION="us-east-1" #Update this line if you work on another region

ECR\_REGISTRY="046402772087.dkr.ecr.us-east-1.amazonaws.com" # Replace this line with your ECR name

aws ecr create-repository \

--repository-name ${APP\_REPO\_NAME} \

--image-scanning-configuration scanOnPush=false \

--image-tag-mutability MUTABLE \

--region ${AWS\_REGION}

. ./jenkins/package-with-maven-container.sh

. ./jenkins/prepare-tags-ecr-for-dev-docker-images.sh

. ./jenkins/build-dev-docker-images-for-ecr.sh

. ./jenkins/push-dev-docker-images-to-ecr.sh

Burada PATH değişkenini tanıtıyoruz. ECR\_REGISTRY yi güncellemeyi unutmuyoruz. Repo ismimiz aynı ise onu da değiştiriyoruz çünkü create komutu var.

Apply/Save/Build Now diyoruz.

Burada script dosyalarının başında . Var bu da source komutudur, .sh içindeki komutları çalıştır demektir, shell sürecine alır. biz bu .sh dosyalarını executable yapmadığımız için böyle bir komut ekledik. executable olsaydı ilk . ya gerek olmazdı. sh ./file.sh ya da ./file.sh şeklinde çalışabilirdi executable olsaydı.

3191625ae7fd: 
f7ead5aa2524: 
Pushed 
Pushed 
prometheus-semrice: digest: 
Finished: SUCCESS 
sha256: 33b85e91eaded2337c2seea6fb7bf33e53sse7ø4a831abdøffød18f8525c4693 size: 
2405 

job çalıştı ve

Images (8) 
Q Search artifacts 
Image tag 
v2.1.2-b1 
vets-service- 
v2.1.2-b1 
hystrix- 
dashboard- 
v2.1.2-b1 
discovery- 
server-v2.12- 
bl 
customers- 
service-v2.12- 
v 
Artifact type 
Image 
Image 
Image 
Image 
Image 
c 
Pushed 
2023, 
(UTC+02) 
Haziran 
2023, 
(UTC+02) 
Haziran 
2023, 
(UTC+02) 
Haziran 
2023, 
(UTC+02) 
Haziran 
2023, 
Delete 
Size 
(MB) 
183.65 
185.34 
157.71 
169.20 
183.66 
Details 
Image 
Copy 
Copy 
Copy 
Copy 
Digest 
O sha2 
O sha2 
3 sha2 
O sha2 
O sha2 

ECR repomuza image lerin geldiğini görüyoruz.

oluşan repoyu silebiliriz.

Şimdi ise bir playbook yazacaz. ansible/playbooks altında dev-petclinic-deploy-template

ismiyle.

- hosts: role\_master

mater makinede işlem yapacak.

tasks:

- name: Create .docker folder

file:

path: /home/ubuntu/.docker

state: directory

mode: '0755'

.docker isimli bir directory oluştur ve 755 mode olsun.

- name: copy the docker config file

become: yes

copy:

src: $JENKINS\_HOME/.docker/config.json

dest: /home/ubuntu/.docker/config.json

copy modülü ile src den config.json dosyasını al.

$JENKINS\_HOME /var/lib/jenkins tir. buradaki .docker docker login komutuyla oluştu. credentials bilgilerini .docker dizini oluşturarak onun içinde tutar.biz de copy modülü ile credential gönderdik. master makine ecr dan image çekerken bunu kullanacak.

- name: deploy petclinic application

shell: |

helm plugin install <https://github.com/hypnoglow/helm-s3.git>

kubectl create ns petclinic-dev

kubectl delete secret regcred -n petclinic-dev || true

kubectl create secret generic regcred -n petclinic-dev \

--from-file=.dockerconfigjson=/home/ubuntu/.docker/config.json \

--type=kubernetes.io/dockerconfigjson

AWS\_REGION=$AWS\_REGION helm repo add stable-petclinic s3://petclinic-helm-charts-<put-your-name>/stable/myapp/

AWS\_REGION=$AWS\_REGION helm repo update

AWS\_REGION=$AWS\_REGION helm upgrade --install \

petclinic-app-release stable-petclinic/petclinic\_chart --version ${BUILD\_NUMBER} \

--namespace petclinic-dev

bu taskte de app deploy edilecek master makinede. bunun için biz aslında jenkins server ı ansible main makine gibi kullanıyoruz.

bu confg dosyasını cluster da bir secret haline getiriyoruz. biz daha önce secret tipi olarak opaque kullanmıştık ancak şimdi Dockr config secret için farklı format kullanacaz.

<https://kubernetes.io/docs/concepts/configuration/secret/#docker-config-secrets>

helm plugin install <https://github.com/hypnoglow/helm-s3.git>

komutuyla helm plugini s3 için kuruyoruz. helm i daha önceki playbook ile kurmuştuk.

kubectl create ns petclinic-dev

bu komut ns oluşturuyor.

kubectl delete secret regcred -n petclinic-dev || true

bu komutla da daha önce regcred isimli bir secret vrsa önce onu iliyoruz sonra create komutu giriyoruz. işlemde hata olmasın diye de true ekliyoruz ki hata alıp orada durmasın. ilk çalıştığında bu secret olmyacak çünkü.

kubectl create secret generic regcred -n petclinic-dev \

--from-file=.dockerconfigjson=/home/ubuntu/.docker/config.json \

--type=kubernetes.io/dockerconfigjson

bu komut secretı oluşturuyor.

AWS\_REGION=$AWS\_REGION helm repo add stable-petclinic s3://petclinic-helm-charts-<put-your-name>/stable/myapp/

burada master makineye s3 reposunu isim vererek tanıtıyoruz. sonraki komut ile repoyu update ediyoruz. s3 repo ismini kendi repmouzun ismiyle değiştiriyoruz.

AWS\_REGION=$AWS\_REGION helm upgrade --install \

petclinic-app-release stable-petclinic/petclinic\_chart --version ${BUILD\_NUMBER} \

--namespace petclinic-dev

helm upgrade --install komutu varsa güncelle yoksa yükle demektir.

önce release e isim verdik, stable-petclinic/ de repya az önceki komutta verdiğimiz isim, petclinic-chart da Config.yaml da yazan charta verdiğimiz isim. --versionu da BUİLD NUMBER olsun diyoruz.

# This is the chart versio 
# to the chart and its tem 
# Versions are expected to 
version: HELM VERSION 

versionu buradan alacak, biz bu HELM\_VERSION u daha sonra BUILD NUMBER olarak değiştirecez daha sonra.

en sonunda da petclinic-dev ns sinde yapsın diyoruz.

bu playbook uygulamayı deploy etmiş olacak. kalan adımda da functional testler yaplıyor.

örnek bir selenium test dosyamız var dummy\_selenium\_test\_headless.py

bunu selenium-jobs folder altında oluşturalım.

bu test google.com a gidip I'm Feeilng Lucky yazısı var mı diye kontrol eden basit bir test.

bunu çalıştıran bir ansible playbook oluşturuyoruz.

pb\_run\_dummy\_selenium\_job.yaml

ismiyle

- hosts: all

tasks:

- name: run dummy selenium job

shell: "docker run --rm -v {{ workspace }}:{{ workspace }} -w {{ workspace }} clarusway/selenium-py-chrome:latest python {{ item }}"

with\_fileglob: "{{ workspace }}/selenium-jobs/dummy\*.py"

register: output

- name: show results

debug: msg="{{ item.stdout }}"

with\_items: "{{ output.results }}"

tüm makinelerde çalışan bir playbook. docker run komutuyla wokrspace i volume olarak bağlıyor.

biden fazla dosya kopyalıyorsak ayrı ayrı copy modülü kullanmaktansa with\_fileglob parmetresi kullanıldı:

Examples 
COOS' that the 

with\_fileglob: "{{ workspace }}/selenium-jobs/dummy\*.py"

buradaki workspace ansible daki bir değişken, ansible da variable ları vars: ile ya da bir file dan alacaz. komut satırında bunu tanıtacaz. Jenkinsfile da.

çıktısında testi geçip geçmediğiini görmek için register ile output belirtip debug ile yazıdıryoruz. bütün selenium .py testlerini çalıştıracak bu sırasıyla.

burada shell deki item bütün dosyaları sırn diye yazılıyor.asıyla with\_fileglob alsın diye yazılıyor. item ansible da kullanılıyor.

bu playbook u çalıştıracak komutu .sh olarak kaydediyoruz.

ansible/scripts

folder ı oluşturuyoruz.

run\_dummy\_selenium\_job.sh

isimli dosya içinde

PATH="$PATH:/usr/local/bin"

ansible-playbook --connection=local --inventory 127.0.0.1, --extra-vars "workspace=${WORKSPACE}" ./ansible/playbooks/pb\_run\_dummy\_selenium\_job.yaml

playbook u çalıştırma komutunu kaydettik.

yukarıdaki komutu all da değil localde çalıştır demiş olduk.

workspace i ${WORKSPACE} olarak tanımladık. buradaki bu dosyayı çalıştır demiş olduk.

biz selenium testlerini jenkins server da çalıştıramayız başka dependencies ler yüklü olmalı. onlar da image içinde var.

cd petclinic-microservices-with-db/

ansible-playbook --connection=local --inventory 127.0.0.1, --extra-vars "workspace=$(pwd)" ./ansible/playbooks/pb\_run\_dummy\_selenium\_job.yaml

komutları sırasıyla çalıştırıp testi çalıştıralım.

ok: [127.Ø.Ø.1] 
TASK [run dununy selenium job] 

localde çalıştırıyor (playbookda host all diyordu)

playbook taki show result testi sayesinde sounçları görebiliyoruz:

TASK [show results] 
ok: [127 .ø.ø.l] changed': True, 'stdout': 'Test passed' , 
' stderr': "Unable to find image 'clarusway/selenium-py-chrome:late 
st' locally\nlatest: Pulling from clarusway/selenium-py-chrome\neac 

yoksa bu sonuçlaır göstermezdi

PLAY 
127.e.e.1 
failed=ø 
ok=3 
rescued=Ø 
changed=l 
ignored=ø 
unreachab1e=Ø 
skipped=ø 

bu kısmı gösterirdi sadece.

git add .

git commit -m 'added scripts for running dummy selenium job'

git push --set-upstream origin feature/msp-18

ile push ediyoruz.

biz bu testleri playbook ile yazıp jenkinsfile da bu şekilde çalıştırdığımız için testler hata alsa da pipeline durmaz çalışmaya devaam eder. ancak biz bunu komut olarak pipeline a ekleseydik test geçemeseydi pipeline dururdu.

şimdi de bütün selenium job lar için pb\_run\_selenium\_jobs.yaml

isimli bir playbook yazalım. aslında az önceki yazdığmız playbook ile aynı. birkaç şey değiştirdik.

- hosts: all

tasks:

- name: run all selenium jobs

shell: "docker run --rm --env MASTER\_PUBLIC\_IP={{ master\_public\_ip }} -v {{ workspace }}:{{ workspace }} -w {{ workspace }} clarusway/selenium-py-chrome:latest python {{ item }}"

register: output

with\_fileglob: "{{ workspace }}/selenium-jobs/test\*.py"

- name: show results

debug: msg="{{ item.stdout }}"

with\_items: "{{ output.results }}"

MATER\_PUBLIC\_IP testin içinde kullanılacak.

şimdi test dosyalarımızın içinde bir değişiklik yapacaz.

APP\_IP = os.environ['MASTER\_PUBLIC\_IP']

url = "[http://"+APP\_IP.strip()+":30001/](about:blank)"

bu değişikliği yapacaz. çünkü 30001 portundan yayınlanacak uygulama.

şimdi run\_selenium\_jobs.sh

dosyasını ansiblce/scripts altında oluşturup bu playbook u çalıştıracak komutu ekliyoruz.

PATH="$PATH:/usr/local/bin"

ansible-playbook -vvv --connection=local --inventory 127.0.0.1, --extra-vars "workspace=${WORKSPACE} master\_public\_ip=${MASTER\_PUBLIC\_IP}" ./ansible/playbooks/pb\_run\_selenium\_jobs.yaml

sıra jenkinsfile ı yazmaya geldi.

jenkins folder altında jenkinsfile-petclinic-nightly

isimli bir dosya oluşturuyoruz.

jenkinsfile da 109. satırda s3 bucket ismini değiştiriyoruz.

git add .

git commit -m 'added qa automation pipeline for dev'

git push

git checkout dev

git merge feature/msp-18

git push origin dev

dev branchına merge ediyoruz.

şimdi petclinic-nightly

isimli pipeline ı oluşturuyoruz.

her gece 00:00 da çalışan bir pipeline olacak bu.

bu pipeline github tan kodları dev branchından çekecek.

Branch Specifier (blank for •any') 
Yd ev 
Add Branch 
Repository browser . 
(Auto) 
Additional Behaviours 
Add 
Script Path ? 

jenkinsfile ın pathini de düzeltiyoruz.

Apply/Save/Build Now diyoruz.

uzun sürecek bir pipeline bu.

şimdi jenkinsfile ı konuşalım:

pipet i ne 
Pair 
ECR 
3min 2' 
ECR 
32s 
Auto 
3 min 44 
1 min 
21s 
App A 
Smin 32s 
c 
App B 
bin / u ibs 
operating 
Auto m 
Sis 
App c 
bin / I ibs 
36s 

pipeline ımız bu.

ilk kısımda ECR repo oluşuyor

İkinci kısımda mvn package çalışıyor.

Sonra focker image ler için tagler oluşuyor

Docker image ler build ediliyor.

Image ler ECR a push ediliyor.

Ansible için key pair oluşturuluyor.

Terraform ile makineler ayağa kalkıyor.

k8s setup yaml ile Ansible bu makineleri cluster haline getiriyor.

Uygulamayı helm kullanarak deploy ediyor.

Aplikasyonun deploy edilip edilmediği test ediliyor.

Aplikasyonun functional testleri yapılıyor.

pipeline {

agent any

environment {

APP\_NAME="petclinic"

APP\_REPO\_NAME="clarusway-repo/${APP\_NAME}-app-dev"

AWS\_ACCOUNT\_ID=sh(script:'aws sts get-caller-identity --query Account --output text', returnStdout:true).trim()

AWS\_REGION="us-east-1"

ECR\_REGISTRY="${AWS\_ACCOUNT\_ID}.dkr.ecr.${AWS\_REGION}.amazonaws.com"

ANS\_KEYPAIR="petclinic-${APP\_NAME}-dev-${BUILD\_NUMBER}.key"

ANSIBLE\_PRIVATE\_KEY\_FILE="${WORKSPACE}/${ANS\_KEYPAIR}"

ANSIBLE\_HOST\_KEY\_CHECKING="False"

}

burada kullanacağımız env var ları tanıtıyoruz. account id yi otoamtik almanın yolu. bir komutun çıktısını alıyorsak returnStdout:true).trim()

kullanıp trim ile sağını solunu temizliyoruz.

key file ın yerini gösteriyoruz. ansible da onay istemesin diye ANSIBLE\_HOST\_KEY\_CHECKING="False" yapıyoruz.

stages {

stage('Create ECR Repo') {

steps {

echo "Creating ECR Repo for ${APP\_NAME} app"

sh '''

aws ecr describe-repositories --region ${AWS\_REGION} --repository-name ${APP\_REPO\_NAME} || \

aws ecr create-repository \

--repository-name ${APP\_REPO\_NAME} \

--image-scanning-configuration scanOnPush=true \

--image-tag-mutability MUTABLE \

--region ${AWS\_REGION}

'''

}

}

ECR reposunu oluşturan stage.

stage('Package Application') {

steps {

echo 'Packaging the app into jars with maven'

sh ". ./jenkins/package-with-maven-container.sh"

}

}

mvn package komutunu çalıştıracak script.

stage('Prepare Tags for Docker Images') {

steps {

echo 'Preparing Tags for Docker Images'

script {

MVN\_VERSION=sh(script:'. ${WORKSPACE}/spring-petclinic-admin-server/target/maven-archiver/pom.properties && echo $version', returnStdout:true).trim()

env.IMAGE\_TAG\_ADMIN\_SERVER="${ECR\_REGISTRY}/${APP\_REPO\_NAME}:admin-server-v${MVN\_VERSION}-b${BUILD\_NUMBER}"

MVN\_VERSION=sh(script:'. ${WORKSPACE}/spring-petclinic-api-gateway/target/maven-archiver/pom.properties && echo $version', returnStdout:true).trim()

env.IMAGE\_TAG\_API\_GATEWAY="${ECR\_REGISTRY}/${APP\_REPO\_NAME}:api-gateway-v${MVN\_VERSION}-b${BUILD\_NUMBER}"

MVN\_VERSION=sh(script:'. ${WORKSPACE}/spring-petclinic-config-server/target/maven-archiver/pom.properties && echo $version', returnStdout:true).trim()

env.IMAGE\_TAG\_CONFIG\_SERVER="${ECR\_REGISTRY}/${APP\_REPO\_NAME}:config-server-v${MVN\_VERSION}-b${BUILD\_NUMBER}"

MVN\_VERSION=sh(script:'. ${WORKSPACE}/spring-petclinic-customers-service/target/maven-archiver/pom.properties && echo $version', returnStdout:true).trim()

env.IMAGE\_TAG\_CUSTOMERS\_SERVICE="${ECR\_REGISTRY}/${APP\_REPO\_NAME}:customers-service-v${MVN\_VERSION}-b${BUILD\_NUMBER}"

MVN\_VERSION=sh(script:'. ${WORKSPACE}/spring-petclinic-discovery-server/target/maven-archiver/pom.properties && echo $version', returnStdout:true).trim()

env.IMAGE\_TAG\_DISCOVERY\_SERVER="${ECR\_REGISTRY}/${APP\_REPO\_NAME}:discovery-server-v${MVN\_VERSION}-b${BUILD\_NUMBER}"

MVN\_VERSION=sh(script:'. ${WORKSPACE}/spring-petclinic-hystrix-dashboard/target/maven-archiver/pom.properties && echo $version', returnStdout:true).trim()

env.IMAGE\_TAG\_HYSTRIX\_DASHBOARD="${ECR\_REGISTRY}/${APP\_REPO\_NAME}:hystrix-dashboard-v${MVN\_VERSION}-b${BUILD\_NUMBER}"

MVN\_VERSION=sh(script:'. ${WORKSPACE}/spring-petclinic-vets-service/target/maven-archiver/pom.properties && echo $version', returnStdout:true).trim()

env.IMAGE\_TAG\_VETS\_SERVICE="${ECR\_REGISTRY}/${APP\_REPO\_NAME}:vets-service-v${MVN\_VERSION}-b${BUILD\_NUMBER}"

MVN\_VERSION=sh(script:'. ${WORKSPACE}/spring-petclinic-visits-service/target/maven-archiver/pom.properties && echo $version', returnStdout:true).trim()

env.IMAGE\_TAG\_VISITS\_SERVICE="${ECR\_REGISTRY}/${APP\_REPO\_NAME}:visits-service-v${MVN\_VERSION}-b${BUILD\_NUMBER}"

env.IMAGE\_TAG\_GRAFANA\_SERVICE="${ECR\_REGISTRY}/${APP\_REPO\_NAME}:grafana-service"

env.IMAGE\_TAG\_PROMETHEUS\_SERVICE="${ECR\_REGISTRY}/${APP\_REPO\_NAME}:prometheus-service"

}

}

}

burada variable ları env. ile yazıyoruz, script ile yazmıyoruz çünkü bütün jenkinsfile da kullanılacak bu variable lar. script ile oluşturursak jenkins bunu tanımaz.

stage('Build App Docker Images') {

steps {

echo 'Building App Dev Images'

sh ". ./jenkins/build-dev-docker-images-for-ecr.sh"

sh 'docker image ls'

}

}

image leri build etme stage i. image leri oluşmuş mu diye ls komutu ekliyoruz.

stage('Push Images to ECR Repo') {

steps {

echo "Pushing ${APP\_NAME} App Images to ECR Repo"

sh ". ./jenkins/push-dev-docker-images-to-ecr.sh"

}

}

iamge leri ECR a gönderme adımı.

stage('Create Key Pair for Ansible') {

steps {

echo "Creating Key Pair for ${APP\_NAME} App"

sh "aws ec2 create-key-pair --region ${AWS\_REGION} --key-name ${ANS\_KEYPAIR} --query KeyMaterial --output text > ${ANS\_KEYPAIR}"

sh "chmod 400 ${ANS\_KEYPAIR}"

}

}

terraform ve ansible da kullanacağımız key pairi oluşturma stage i.

stage('Create QA Automation Infrastructure') {

steps {

echo 'Creating QA Automation Infrastructure for Dev Environment'

sh """

cd infrastructure/dev-k8s-terraform

sed -i "s/clarus/$ANS\_KEYPAIR/g" main.tf

terraform init

terraform apply -auto-approve -no-color

"""

script {

echo "Kubernetes Master is not UP and running yet."

env.id = sh(script: 'aws ec2 describe-instances --filters Name=tag-value,Values=master Name=tag-value,Values=tera-kube-ans Name=instance-state-name,Values=running --query Reservations[\*].Instances[\*].[InstanceId] --output text', returnStdout:true).trim()

sh 'aws ec2 wait instance-status-ok --instance-ids $id'

}

}

}

önce terraform ile makineleri yağaa kaldırıyor. burada sed komutuyla maintf teki clarus olan key pairi ANS\_KEYPAIR ile değiştiriyoruz.

env.id ile makineleri filtreleyip bunu $id ye atayıp alttaki cli komutu ile makinelerin 2/2 checked olmasını bekletiyoruz.

wait komutunu kullanabileceğimiz yerler aws docsta var:

3k 
useful? 
OCu"WntationQ 
ke to suggest an 
or ror tneAws 
t our 
'ide 
the AWS 
Guide for help 
Available Commands 11 
bundle-task-complete 
• conversion-task-cancelled 
conversion-task-completed 
• conversion-task-deleted 
• customer-gateway-available 
export-task-cancelled 
export-task-completed 
image-available 
image-exists 
• instance-exists 
instance-running 
• instance-statusN 
• instance-stopped 
• instance-terminated 
• intemet-gateway-exists 
key-pair-exists 
nat-gateway-available 
• nat-gateway-deleted 
network-interface-available 
passVÆrd-data-avaiIabIe 
snapshot-imported 
• spot-instance-request-fulfilled 
subnet-available 
• system-status-ok 
volume ilaOIe 
volume eleted 
volume-in-use 
vpc-available 
vpc-exists 
wc-peering-connection—deleted 
vpc-peering-connection-exists 
V" r I-connection-available 
vpn-connection—deleted 

<https://awscli.amazonaws.com/v2/documentation/api/latest/reference/ec2/wait/index.html>

stage('Create Kubernetes Cluster for QA Automation Build') {

steps {

echo "Setup Kubernetes cluster for ${APP\_NAME} App"

sh "ansible-playbook -i ./ansible/inventory/dev\_stack\_dynamic\_inventory\_aws\_ec2.yaml ./ansible/playbooks/k8s\_setup.yaml"

}

}

clusterı kuaracak ansible playbook unu çalışıtran stage. makineye helm de kuruyor bu playbook.

stage('Deploy App on Kubernetes cluster'){

steps {

echo 'Deploying App on Kubernetes'

sh "envsubst < k8s/petclinic\_chart/values-template.yaml > k8s/petclinic\_chart/values.yaml"

sh "sed -i s/HELM\_VERSION/${BUILD\_NUMBER}/ k8s/petclinic\_chart/Chart.yaml"

sh "helm repo add stable-petclinic s3://petclinic-helm-charts-<put-your-name>/stable/myapp/"

sh "helm package k8s/petclinic\_chart"

sh "helm s3 push --force petclinic\_chart-${BUILD\_NUMBER}.tgz stable-petclinic"

sh "envsubst < ansible/playbooks/dev-petclinic-deploy-template > ansible/playbooks/dev-petclinic-deploy.yaml"

sh "sleep 60"

sh "ansible-playbook -i ./ansible/inventory/dev\_stack\_dynamic\_inventory\_aws\_ec2.yaml ./ansible/playbooks/dev-petclinic-deploy.yaml"

}

}

envsubst komutuyla values-template.yaml ı values.yaml a çevir.

sed komutuyla Cahrt.yaml daki HELM\_VERSION u BUILD\_NUMBER ile değiştir diyoruz bu da jenkins te tanımlı bir değişkendi.

helm repo add ile de s3 reposunu jenkinste helm reposu haline getiriyoruz. helm package ile paketleyip, s3 e push luyoruz. push larken aynı isimte bir paket daha önce gittiyse --force flagiyel override yapıyor, hata almıyoruz.

envsubst komutuyla dev-petclinic-deploy-template i dev-petclinic-deploy.yaml olarak tekrar yaz diyoruz.

daha sonra 60sn bekliyoru ve playbook u çalıştırıyor.

stage('Test the Application Deployment'){

steps {

echo "Check if the ${APP\_NAME} app is ready or not"

script {

env.MASTER\_PUBLIC\_IP = sh(script:"aws ec2 describe-instances --region ${AWS\_REGION} --filters Name=tag-value,Values=master Name=tag-value,Values=tera-kube-ans Name=instance-state-name,Values=running --query Reservations[\*].Instances[\*].[PublicIpAddress] --output text", returnStdout:true).trim()

while(true) {

try{

sh "curl -s ${MASTER\_PUBLIC\_IP}:30001"

echo "${APP\_NAME} app is successfully deployed."

break

}

catch(Exception){

echo "Could not connect to ${APP\_NAME} app"

sleep(5)

}

}

}

}

}

uygulamaı deploy edince uygulama henüz gelmez. 1 dk yı geçebiliyor. hem uygulama geiyor mu kontrol etmek isityoruz hem de selenium testi çaışacak mı bir test yapmak istiyoruz. bunun için de groovy dilinde bir döngü oluşturuyoruz.

MASTER\_PUBLIC\_IP de selenium testleri gerçekleşeceği için burada bunu çekiyoruz. yukarıda id yi çektiğimiz gibi.

try ile komutu çalıştırıyor, curl komutuyla 30001 portundan makineyi görmeye çalışıyor, görürse app is succesfully deployed yazdırıyor, göremiyorsa catch e geçip 5 sn bekleyip döngüye başlıyor. eğer görürse break ile çıkıyor.

stage('Run QA Automation Tests'){

steps {

echo "Run the Selenium Functional Test on QA Environment"

sh 'ansible-playbook -vvv --connection=local --inventory 127.0.0.1, --extra-vars "workspace=${WORKSPACE} master\_public\_ip=${MASTER\_PUBLIC\_IP}" ./ansible/playbooks/pb\_run\_selenium\_jobs.yaml'

}

}

}

bu stage te de pipeline ın amacına geçiyor ve test dosyalarını çalıştıracak playbook u çalıştırıyor. bu çalışınca docker ile 3 tane test dosyasını sırayla çalıştırıyor. bu pipeline her gece çalışacak bir pipeline.

post {

always {

echo 'Deleting all local images'

sh 'docker image prune -af'

echo 'Delete the Image Repository on ECR'

sh """

aws ecr delete-repository \

--repository-name ${APP\_REPO\_NAME} \

--region ${AWS\_REGION}\

--force

"""

echo 'Tear down the Kubernetes Cluster'

sh """

cd infrastructure/dev-k8s-terraform

terraform destroy -auto-approve -no-color

"""

echo "Delete existing key pair using AWS CLI"

sh "aws ec2 delete-key-pair --region ${AWS\_REGION} --key-name ${ANS\_KEYPAIR}"

sh "rm -rf ${ANS\_KEYPAIR}"

}

}

}

en sonda da docker image leri siliyor, ecr repoyu siliyor, terraform destroy ediyor, key pairi siliyor. key pairin dosyasını siliyor.