Bundan sonraki aşamada MSP 18 e kadar functional test için bir pipeline oluşturacaz. bunun için terraform ile 3 makine yi terraform ile ayağa kaldıracaz ve cluster oluşturacaz. sonra da ansible ile bunları configure edecez.

functional testleri çalıştıran 3 tane oython dosyası. Bu testlerin yapılması için uygulamanın ayağa kalkması lazım. makineleri de bu nedenle ayağa kaldırıyoruz. 1 master 2 worker node olacak şekilde 3 tane. Cluster hazır olduktan sonra uygulamayı playbookla ayağa kaldıracaz. burada helm i de kullanacaz. uygulama ayağa kalktıktan sonra da program functional testi yapması için uygulamaya bağlanıp yapacak.

Gerçek hayatta da direk pipeline yazılmaz, adım adım denenir. Biz de şimdi adım adım deneyerek yapacaz.

1. adımdayız

Burada selenium tool ile çalışacak functional testlerin yazıldığı üç adet python dosyasını repoya ekleyeip push edecez.

git checkout dev

git branch feature/msp-13

git checkout feature/msp-13

ana dizinimizin altında selenium-jobs

isimli bir klasörü oluşturacaz.

mkdir selenium-jobs

bu klasörün altında test\_owners\_all\_headless.py

isimli bir dosya oluşturup,

dosya içeriğini README den yapıştırıyoruz.

Unit test, aracın kapısının ya da diğer parçalarının ayrı ayrı doğru çalışıp çalışmadığı, integration test parçaların birbirine entegre edildiğindeki uyumu, functional test de parçalar birleştikten sonra arabanın doğru çalışıp çalışmadığı gibi bir benzetme yapabiliriz.

functional test otomatik şekilde yapılıyor. python da yazılan kodlar bunun için.

Selenium bu testi yapmayı sağlayan bir tool.

ikinci ve üçüncü dosyalarımızı da hazırlayalım.

test\_owners\_register\_headless.py

ve

test\_veterinarians\_headless.py

şimdi de bunları push ediyoruz.

git add .

git commit -m 'added selenium jobs written in python'

git push --set-upstream origin feature/msp-13

git checkout dev

git merge feature/msp-13

git push origin dev

1. adımdayız

create-ecr-docker-registry-for-dev

isimli bir job oluşturacaz. docker registry için freestyle bir job oluşturuyoruz.

- job name: create-ecr-docker-registry-for-dev

- job type: Freestyle project

- Build:

Add build step: Execute Shell

Command:

```

```bash

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

APP\_REPO\_NAME="clarusway-repo/petclinic-app-dev"

AWS\_REGION="us-east-1"

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=false \

--image-tag-mutability MUTABLE \

--region ${AWS\_REGION}

bu komutarda || sol taraft doğruysa sağ tarafı yapma demekti (Linux ta) bu isimde bir repo olup olmadığını kontrol ediyor aslında. Repo varsa aynı isimde yeniden oluşturamaya alışmasın diye.

yukarıda PATH değişkenine /usr/local/bin pathini ekliyouz ve jenkins e bunu tanıtıyoruz. artık jenkinste buna gerek yok ancak önceden bunu tanıtmak gerekiyordu.

AWS CLI komutları çalışıyor jenkins server ımızd çünkü terraform ile oluşurken role atanmıştı.

job ımızı save edip build edelim.buildimiz çalıştı ve ecr da repomuz oluştu.

bu execute shell komutunu da infrastrcutre altında bir

create-ecr-docker-registry-for-dev.sh

dosyasında saklayacaz.

git add .

git commit -m 'added script for creating ECR registry for dev'

git push --set-upstream origin feature/msp-14

git checkout dev

git merge feature/msp-14

git push origin dev

push layıp merge ettik.

1. adımdayız

**git checkout -b feature/msp-15** ile yeni bir branch oluşturup oraya gidioyruz.

gerçek hayatta her şey böyle ayrı bir branchta olmayabilir şirket kültürü ile alakalı bu durum. ancak biz güzel bir kullanım çrneği olsun diye böyle hareket ediyoruz.

infrastructure altında dev-k8s-terraform

isimli bir klasör oluşturacaz. bu klasör altında da terraform dosyalarımız olacak.

uygulamanın ayağa kalkması için 3 node ve sec grp ları ile cluster oluşturmamız gerekiyor. bunun için de terraform dosyası hazırlayacaz bu adımda. terraform ile kalkan makineler ubuntu olacak. ancak kubernetes hazır halde ayağa kalkamayacak configürasyonları ansible ile yapacaz. ilke kubernetes dersinde manuel yaptığımız kurulumu ansible ile yapmış olacaz. bu best practice tir.

(README den main.tf içeriğini aldık)

master node a helm kullanırken s3 ü repo olarak kullanması için bir role oluşturduk main.tf de.

master, worker 1 ve worker 2 üç tane makine oluşturuyoruz main.tf te.

master sg ve mutual sg var master makineye bağlanacak.

burada key name = clarus b şekilde kalacak. biz onu ileride sed komutu ile değiştirecez.

subnet\_id olarak kendi default vpc mizden us-east-1a nın id sini yazdık. farklı AZ lerde olan subnetlerde maliyet artar bu nedenle aynı subnette olmaları tercih sebebidir.

worker makinelerin de worker ve mutual sg leri var.

birkaç tane tag olmasınının sebbei ansible da kullanacak olmamız.

output olarak da public ip lerini istemişiz.

sec grpların portları için kubernetes docs a gidip Control plane ve worker node lar için portlara bakıyoruz:

Control plane 
protocol Direction 
TCP 
TCP 
TCP 
TCP 
TCP 
Inbound 
Inbound 
Inbound 
Inbound 
Inbound 
port Range 
6443 
2379-2380 
10250 
10259 
10257 
purpose 
Kubernetes API server 
etcd server client API 
Kubelet API 
kube-scheduler 
Used By 
kube-apiserver, etcd 
Self, Control plane 
Self 
kube-controller-manager Self 
Although etcd ports are included in control plane section, you can also host your own etcd cluster externally or on custom 
ports, 
Worker node(s) 
protocol Direction 
TCP 
TCP 
Inbound 
Inbound 
port Range 
10250 
30000-32767 
purpose 
KubeIetAPl 
NodePort Servicest 
Used By 
Self, Control plane 
All 

terrafomr de implicit dependency vardı, kendisi önce oluştıurması gerekn reasource u oluşturur. biz burada sec grp oluştururken bir sg yi diğerine bağlarsak terraform burada confused oluyor ve yapamıyor. bu nedenle arada mutual sg koyduk, karşılıklı portları buna bağladık. sonsuz döngüye girmesine engel olduk.

explicit dependency de depends\_on ile yaptığımız bizim belirlediğimiz bir dependency.

daha önce kubernetes ayağa akaldıran terraform da birbirine bağlanan sg de self parametresi eklenmişti.

burada mutual sg nin ingressleri:

ingress {

protocol = "tcp"

from\_port = 10250

to\_port = 10250

self = true

}

ingress {

protocol = "udp"

from\_port = 8472

to\_port = 8472

self = true

}

ingress {

protocol = "tcp"

from\_port = 2379

to\_port = 2380

self = true

}

8472 kullanacağımız Flannel tool için default port olduğu için ekledik.

2379 ve 2380 de iki control plane olabilir diye iki port eklendi. 2379 dan 2379 a ve 2380 den 2380 e diye iki ayrı ingress de yazılabilirdi ancak bu şekilde de yazılırsa iki port eklenmiş olur. gerçekte tek control plane olmaz çünkü.

Terraform dosyasının secgrp larını detaylı çalışmamız gerekiyor. Önemli.

git add .

git commit -m 'added dev-k8s-terraform for kubernetes infrastructure'

git push --set-upstream origin feature/msp-15

git checkout dev

git merge feature/msp-15

git push origin dev

komutları ile MSP-15 i bitirelim.

1. adımdayız

bi bu aşamada kubeadm ile kuracaz, daha sonra EKS ile kuracaz. ancak gerçek hayatta biri kubeadm diğer EKS ile kurulmaz. biz recap olsun diye ve çeşitlilik olsun diye böyle yapıyoruz.

git checkout dev

git branch feature/msp-16

git checkout feature/msp-16

git push --set-upstream origin feature/msp-16

burada da test-creating-qa-automation-infrastructure

isimli bir jenkins job çalıştırıp aşağıdaki programlar var mı diye bakacaz:

echo $PATH

whoami

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

python3 --version

pip3 --version

ansible --version

aws --version

terraform --version

bundan sonra da bu job u değiştire değiştire devam edecz bir süre.

yukarıdaki komutları server da çalıştıracak. bu komutlar için github rrepoyu jenkinse tanıtmamıza gerek yok. ancak biz sonraki kodlar için şimdiden tanıtıyoruz.

makinede bu programlar var mı diye neden bakıoruz? gerçek hayatta biz agent ile çalışıyor olacaz ve agent ta bu programlar var mı diye bilmemiz gerekiyor.

jobu built ettik. jenkins user kontrol etti ve programlar yüklü olduğunu gördük:

lø: 35 
lø: 35 
lø: 35 
lø: 35 
lø: 35 
lø: 35 
lø: 35 
lø: 35 
lø: 35 
lø: 35 
lø: 35 
lø: 35 
lø: 35 
lø: 35 
lø: 35 
lø: 35 
: 54 
: 54 
: 54 
: 54 
: 54 
: 54 
: 55 
: 55 
: 55 
+ whoami 
den kin s 
+ python3 
- -version 
Python 3.9.16 
+ pip3 
- -version 
pip 21.3.1 from /usr/Iib/python3.91 site-packages/ pip (python 3 .9) 
lø: 35: 55 
lø: 35: 55 
+ ansible 
- -version 
ensible [core 2.15. a] 
config file = none 
configured module search path 
[ ' /var/lib/jenkins/ . ensible/plugins/modules ' , 
' /usr/share/ansible/plugins/modules ' ] 
lø: 35: 55 
lø: 35: 55 
lø: 35: 55 
lø: 35: 55 
ansible python module location 
/ us r/ local/ lib/ python3.9/site -packages/ ansible 
ansible collection location 
/ var/lib/#nkins/ . ansible/ col lections : / usr/ share/ansible/collections 
executable location 
/usr/local/bin/ansible 
python version 
(/usr/bin/python3) 
lø: 35: 55 
3.9.16 (main, Feb 23 2023, [ecc 11.3.1 20221121 (Red Hat 11.3.1-4)] 
3.1.2 
: 55 
: 55 
: 56 
: 56 
: 56 
: 56 
: 57 
version — 
I ibyaml 
T rue 
+ aws 
- -version 
aws-cIi/2.9.19 Python/3.9.16 Linux/6.I.25-37.47.amzn2Ø23.x86 64 source/x86_64.amzn.2Z23 prompt/off 
+ terraform 
-version 
Terraform VI. 4.6 
on Iinux 
amd64 
Finished . 
• SUCCESS 

şimdi job u configure edip execute shell i değiştiriyoruz:

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

ANS\_KEYPAIR="petclinic-ansible-test-dev.key"

AWS\_REGION="us-east-1"

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

chmod 400 ${ANS\_KEYPAIR}

bu komut bize amazonda key pair oluşturacak. CLI komutu.

bunu neden yapıyoruz, hem key pair oluştur diyoruz hem de şu isimde bir dosyaya kaydet diyoruz. ansible da agent node lara bağlanmak için ssh ile bağlanması gerekiyor ve bu dosya bize orada lazım.

bu execute shelli github ı çekip orada çalıştırdığı için ANS\_KEYPAIR i de o dizine kaydedecek.

Build ediyoru job, key parimizi workspace/test-creating-qa-automation-infrastructure/petclinic-ansbile-test-dev.key olarak kaydetti. chmod 400 de yaptı.

workspace]$ Is 
create-ecr-docker-registr•y-for-dev petclinic-ci-job test-creating-qa-autanation-infrastructure 
spring- - - 
(jenkins@jenkins 
[jenkins@jenkins 
LICENSE 
REAWE.md 
• -dev-docker 
- server 
workspacel$ cd 
test-creating-qa-automation-infrastructure]$ Is 
docker- cunpose-local. yml jenkins 
sprang 
-petclinic-api -gateway 
docker- cmpose. ynl 
- images. Sh docs 
pom. xml 
jobs 
spri ng- pet c I - aonin - server 
spring- petclinic- config- server 
m vnw _ 
package-with -wrapper. Sh 
spring -pet clinic -hystrix-da shboard 
spring -pet clinic-vets-seNice 
spring -pet -Vis 
test - local —deployment. s h 

Amazon console da da key pair oluştuğunu görebiliriz.

şimdi terraform main.tf i çalıştırması için job u configure ediyoruz.

```bash

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

ANS\_KEYPAIR="petclinic-ansible-test-dev.key"

AWS\_REGION="us-east-1"

cd infrastructure/dev-k8s-terraform

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

terraform init

terraform apply -auto-approve -no-color

komutlarını execute shell e aktarıyoruz.

burada main.tf in pathini belirttik.

sed komutuyla da clarus gördüğü yerde key pemi bizim yeni oluşturduğumuz $ANS\_KEYPAIR ile değiştirecek. sınra da terraform init ve terraform apply -auto-approve diyoruz. otoamsyonlar da auto approve nemli yoksa onay bekler kalır.

-no-color da konsol çıktısında karakterlerin daha düzgün gelmesini sağlıyor.

execute shell i değiştirip job ı build ettik.

12:5ø 
12:5ø 
12:5ø 
12:5ø 
12:5ø 
: 25 
: 25 
: 25 
: 25 
: 25 
Outputs : 
kube-master-ip 
"18.207.199.87" 
"3.85.82.159" 
wor - I 
"23.20.111.12" 
wor - 2 
Finished . 
• SUCCESS 

makinelerimiz ayağa kalktı.

makinelere ssh ile bağlanabilip bağlanamadığımızı deneyebiliriz.

burada duruyoruz bugünlük.

jenkins jobdaki komuta apply yerine destroy yazıp çalıştıralım. makineleri destroy etsin

Execute shell 
Command 
See the list of available environment variables 
/usr/local/bin" 
ANS KEYPAIR=" petclinic-ansible-test-dev. key" 
AWS 
cd infrastructure/ dev-k8s-terraform 
sed -i "s/c1arus/$ANS KEYPAIR/g" 
terraform init 
terraform destroy -auto-approve 
Advanced v 
main.tf 
- no- color 

12:55:12 
12: 55:12 Destroy complete! Resources: 
12: 55:12 Finished: SUCCESS 
8 destroyed. 

terminalde de jenkins user ile destroy edebilirdik:

[jenkins@jenkins-server test-creating-qa-automation-infrastructure]$ cd infrastructure/ 
[jenkins@jenkins-serwer infrastructure]$ cd dev-k8s-terraforn/ 
[jenkins@jenkins-serwer Is 
[jenkins@jenkins-server dev-k8s-terraform]$ terraform state list 
data. aws_vpc. name 
petclinic -master-serwer- profile 
_ pet init -master - s -role 
aws 
instance. kube -master 
aws 
instance. worker-I 
aws 
aws_instance. worker-2 
aws_secu rity_group _ petclinic - kube-master•- sg 
pet C I init: - kube-vnrker•- Se 
petclinic -mtual - sg 

key pair ve ECR repomuz kalacak. bir sonraki derste tekrar apply edecez ve devam edecez. (1407. satır)

Bundan sonraki aşamada MSP 18 e kadar functional test için bir pipeline oluşturacaz. bunun için terraform ile 3 makine yi terraform ile ayağa kaldıracaz ve cluster oluşturacaz. sonra da ansible ile bunları configure edecez.

functional testleri çalıştıran 3 tane oython dosyası. Bu testlerin yapılması için uygulamanın ayağa kalkması lazım. makineleri de bu nedenle ayağa kaldırıyoruz. 1 master 2 worker node olacak şekilde 3 tane. Cluster hazır olduktan sonra uygulamayı playbookla ayağa kaldıracaz. burada helm i de kullanacaz. uygulama ayağa kalktıktan sonra da program functional testi yapması için uygulamaya bağlanıp yapacak.

Gerçek hayatta da direk pipeline yazılmaz, adım adım denenir. Biz de şimdi adım adım deneyerek yapacaz.

1. adımdayız

Burada selenium tool ile çalışacak functional testlerin yazıldığı üç adet python dosyasını repoya ekleyeip push edecez.

git checkout dev

git branch feature/msp-13

git checkout feature/msp-13

ana dizinimizin altında selenium-jobs

isimli bir klasörü oluşturacaz.

mkdir selenium-jobs

bu klasörün altında test\_owners\_all\_headless.py

isimli bir dosya oluşturup,

dosya içeriğini README den yapıştırıyoruz.

Unit test, aracın kapısının ya da diğer parçalarının ayrı ayrı doğru çalışıp çalışmadığı, integration test parçaların birbirine entegre edildiğindeki uyumu, functional test de parçalar birleştikten sonra arabanın doğru çalışıp çalışmadığı gibi bir benzetme yapabiliriz.

functional test otomatik şekilde yapılıyor. python da yazılan kodlar bunun için.

Selenium bu testi yapmayı sağlayan bir tool.

ikinci ve üçüncü dosyalarımızı da hazırlayalım.

test\_owners\_register\_headless.py

ve

test\_veterinarians\_headless.py

şimdi de bunları push ediyoruz.

git add .

git commit -m 'added selenium jobs written in python'

git push --set-upstream origin feature/msp-13

git checkout dev

git merge feature/msp-13

git push origin dev

1. adımdayız

create-ecr-docker-registry-for-dev

isimli bir job oluşturacaz. docker registry için freestyle bir job oluşturuyoruz.

- job name: create-ecr-docker-registry-for-dev

- job type: Freestyle project

- Build:

Add build step: Execute Shell

Command:

```

```bash

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

APP\_REPO\_NAME="clarusway-repo/petclinic-app-dev"

AWS\_REGION="us-east-1"

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=false \

--image-tag-mutability MUTABLE \

--region ${AWS\_REGION}

bu komutarda || sol taraft doğruysa sağ tarafı yapma demekti (Linux ta) bu isimde bir repo olup olmadığını kontrol ediyor aslında. Repo varsa aynı isimde yeniden oluşturamaya alışmasın diye.

yukarıda PATH değişkenine /usr/local/bin pathini ekliyouz ve jenkins e bunu tanıtıyoruz. artık jenkinste buna gerek yok ancak önceden bunu tanıtmak gerekiyordu.

AWS CLI komutları çalışıyor jenkins server ımızd çünkü terraform ile oluşurken role atanmıştı.

job ımızı save edip build edelim.buildimiz çalıştı ve ecr da repomuz oluştu.

bu execute shell komutunu da infrastrcutre altında bir

create-ecr-docker-registry-for-dev.sh

dosyasında saklayacaz.

git add .

git commit -m 'added script for creating ECR registry for dev'

git push --set-upstream origin feature/msp-14

git checkout dev

git merge feature/msp-14

git push origin dev

push layıp merge ettik.

1. adımdayız

**git checkout -b feature/msp-15** ile yeni bir branch oluşturup oraya gidioyruz.

gerçek hayatta her şey böyle ayrı bir branchta olmayabilir şirket kültürü ile alakalı bu durum. ancak biz güzel bir kullanım çrneği olsun diye böyle hareket ediyoruz.

infrastructure altında dev-k8s-terraform

isimli bir klasör oluşturacaz. bu klasör altında da terraform dosyalarımız olacak.

uygulamanın ayağa kalkması için 3 node ve sec grp ları ile cluster oluşturmamız gerekiyor. bunun için de terraform dosyası hazırlayacaz bu adımda. terraform ile kalkan makineler ubuntu olacak. ancak kubernetes hazır halde ayağa kalkamayacak configürasyonları ansible ile yapacaz. ilke kubernetes dersinde manuel yaptığımız kurulumu ansible ile yapmış olacaz. bu best practice tir.

(README den main.tf içeriğini aldık)

master node a helm kullanırken s3 ü repo olarak kullanması için bir role oluşturduk main.tf de.

master, worker 1 ve worker 2 üç tane makine oluşturuyoruz main.tf te.

master sg ve mutual sg var master makineye bağlanacak.

burada key name = clarus b şekilde kalacak. biz onu ileride sed komutu ile değiştirecez.

subnet\_id olarak kendi default vpc mizden us-east-1a nın id sini yazdık. farklı AZ lerde olan subnetlerde maliyet artar bu nedenle aynı subnette olmaları tercih sebebidir.

worker makinelerin de worker ve mutual sg leri var.

birkaç tane tag olmasınının sebbei ansible da kullanacak olmamız.

output olarak da public ip lerini istemişiz.

sec grpların portları için kubernetes docs a gidip Control plane ve worker node lar için portlara bakıyoruz:

Control plane 
protocol Direction 
TCP 
TCP 
TCP 
TCP 
TCP 
Inbound 
Inbound 
Inbound 
Inbound 
Inbound 
port Range 
6443 
2379-2380 
10250 
10259 
10257 
purpose 
Kubernetes API server 
etcd server client API 
Kubelet API 
kube-scheduler 
Used By 
kube-apiserver, etcd 
Self, Control plane 
Self 
kube-controller-manager Self 
Although etcd ports are included in control plane section, you can also host your own etcd cluster externally or on custom 
ports, 
Worker node(s) 
protocol Direction 
TCP 
TCP 
Inbound 
Inbound 
port Range 
10250 
30000-32767 
purpose 
KubeIetAPl 
NodePort Servicest 
Used By 
Self, Control plane 
All 

terrafomr de implicit dependency vardı, kendisi önce oluştıurması gerekn reasource u oluşturur. biz burada sec grp oluştururken bir sg yi diğerine bağlarsak terraform burada confused oluyor ve yapamıyor. bu nedenle arada mutual sg koyduk, karşılıklı portları buna bağladık. sonsuz döngüye girmesine engel olduk.

explicit dependency de depends\_on ile yaptığımız bizim belirlediğimiz bir dependency.

daha önce kubernetes ayağa akaldıran terraform da birbirine bağlanan sg de self parametresi eklenmişti.

burada mutual sg nin ingressleri:

ingress {

protocol = "tcp"

from\_port = 10250

to\_port = 10250

self = true

}

ingress {

protocol = "udp"

from\_port = 8472

to\_port = 8472

self = true

}

ingress {

protocol = "tcp"

from\_port = 2379

to\_port = 2380

self = true

}

8472 kullanacağımız Flannel tool için default port olduğu için ekledik.

2379 ve 2380 de iki control plane olabilir diye iki port eklendi. 2379 dan 2379 a ve 2380 den 2380 e diye iki ayrı ingress de yazılabilirdi ancak bu şekilde de yazılırsa iki port eklenmiş olur. gerçekte tek control plane olmaz çünkü.

Terraform dosyasının secgrp larını detaylı çalışmamız gerekiyor. Önemli.

git add .

git commit -m 'added dev-k8s-terraform for kubernetes infrastructure'

git push --set-upstream origin feature/msp-15

git checkout dev

git merge feature/msp-15

git push origin dev

komutları ile MSP-15 i bitirelim.

1. adımdayız

bi bu aşamada kubeadm ile kuracaz, daha sonra EKS ile kuracaz. ancak gerçek hayatta biri kubeadm diğer EKS ile kurulmaz. biz recap olsun diye ve çeşitlilik olsun diye böyle yapıyoruz.

git checkout dev

git branch feature/msp-16

git checkout feature/msp-16

git push --set-upstream origin feature/msp-16

burada da test-creating-qa-automation-infrastructure

isimli bir jenkins job çalıştırıp aşağıdaki programlar var mı diye bakacaz:

echo $PATH

whoami

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

python3 --version

pip3 --version

ansible --version

aws --version

terraform --version

bundan sonra da bu job u değiştire değiştire devam edecz bir süre.

yukarıdaki komutları server da çalıştıracak. bu komutlar için github rrepoyu jenkinse tanıtmamıza gerek yok. ancak biz sonraki kodlar için şimdiden tanıtıyoruz.

makinede bu programlar var mı diye neden bakıoruz? gerçek hayatta biz agent ile çalışıyor olacaz ve agent ta bu programlar var mı diye bilmemiz gerekiyor.

jobu built ettik. jenkins user kontrol etti ve programlar yüklü olduğunu gördük:

lø: 35 
lø: 35 
lø: 35 
lø: 35 
lø: 35 
lø: 35 
lø: 35 
lø: 35 
lø: 35 
lø: 35 
lø: 35 
lø: 35 
lø: 35 
lø: 35 
lø: 35 
lø: 35 
: 54 
: 54 
: 54 
: 54 
: 54 
: 54 
: 55 
: 55 
: 55 
+ whoami 
den kin s 
+ python3 
- -version 
Python 3.9.16 
+ pip3 
- -version 
pip 21.3.1 from /usr/Iib/python3.91 site-packages/ pip (python 3 .9) 
lø: 35: 55 
lø: 35: 55 
+ ansible 
- -version 
ensible [core 2.15. a] 
config file = none 
configured module search path 
[ ' /var/lib/jenkins/ . ensible/plugins/modules ' , 
' /usr/share/ansible/plugins/modules ' ] 
lø: 35: 55 
lø: 35: 55 
lø: 35: 55 
lø: 35: 55 
ansible python module location 
/ us r/ local/ lib/ python3.9/site -packages/ ansible 
ansible collection location 
/ var/lib/#nkins/ . ansible/ col lections : / usr/ share/ansible/collections 
executable location 
/usr/local/bin/ansible 
python version 
(/usr/bin/python3) 
lø: 35: 55 
3.9.16 (main, Feb 23 2023, [ecc 11.3.1 20221121 (Red Hat 11.3.1-4)] 
3.1.2 
: 55 
: 55 
: 56 
: 56 
: 56 
: 56 
: 57 
version — 
I ibyaml 
T rue 
+ aws 
- -version 
aws-cIi/2.9.19 Python/3.9.16 Linux/6.I.25-37.47.amzn2Ø23.x86 64 source/x86_64.amzn.2Z23 prompt/off 
+ terraform 
-version 
Terraform VI. 4.6 
on Iinux 
amd64 
Finished . 
• SUCCESS 

şimdi job u configure edip execute shell i değiştiriyoruz:

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

ANS\_KEYPAIR="petclinic-ansible-test-dev.key"

AWS\_REGION="us-east-1"

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

chmod 400 ${ANS\_KEYPAIR}

bu komut bize amazonda key pair oluşturacak. CLI komutu.

bunu neden yapıyoruz, hem key pair oluştur diyoruz hem de şu isimde bir dosyaya kaydet diyoruz. ansible da agent node lara bağlanmak için ssh ile bağlanması gerekiyor ve bu dosya bize orada lazım.

bu execute shelli github ı çekip orada çalıştırdığı için ANS\_KEYPAIR i de o dizine kaydedecek.

Build ediyoru job, key parimizi workspace/test-creating-qa-automation-infrastructure/petclinic-ansbile-test-dev.key olarak kaydetti. chmod 400 de yaptı.

workspace]$ Is 
create-ecr-docker-registr•y-for-dev petclinic-ci-job test-creating-qa-autanation-infrastructure 
spring- - - 
(jenkins@jenkins 
[jenkins@jenkins 
LICENSE 
REAWE.md 
• -dev-docker 
- server 
workspacel$ cd 
test-creating-qa-automation-infrastructure]$ Is 
docker- cunpose-local. yml jenkins 
sprang 
-petclinic-api -gateway 
docker- cmpose. ynl 
- images. Sh docs 
pom. xml 
jobs 
spri ng- pet c I - aonin - server 
spring- petclinic- config- server 
m vnw _ 
package-with -wrapper. Sh 
spring -pet clinic -hystrix-da shboard 
spring -pet clinic-vets-seNice 
spring -pet -Vis 
test - local —deployment. s h 

Amazon console da da key pair oluştuğunu görebiliriz.

şimdi terraform main.tf i çalıştırması için job u configure ediyoruz.

```bash

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

ANS\_KEYPAIR="petclinic-ansible-test-dev.key"

AWS\_REGION="us-east-1"

cd infrastructure/dev-k8s-terraform

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

terraform init

terraform apply -auto-approve -no-color

komutlarını execute shell e aktarıyoruz.

burada main.tf in pathini belirttik.

sed komutuyla da clarus gördüğü yerde key pemi bizim yeni oluşturduğumuz $ANS\_KEYPAIR ile değiştirecek. sınra da terraform init ve terraform apply -auto-approve diyoruz. otoamsyonlar da auto approve nemli yoksa onay bekler kalır.

-no-color da konsol çıktısında karakterlerin daha düzgün gelmesini sağlıyor.

execute shell i değiştirip job ı build ettik.

12:5ø 
12:5ø 
12:5ø 
12:5ø 
12:5ø 
: 25 
: 25 
: 25 
: 25 
: 25 
Outputs : 
kube-master-ip 
"18.207.199.87" 
"3.85.82.159" 
wor - I 
"23.20.111.12" 
wor - 2 
Finished . 
• SUCCESS 

makinelerimiz ayağa kalktı.

makinelere ssh ile bağlanabilip bağlanamadığımızı deneyebiliriz.

burada duruyoruz bugünlük.

jenkins jobdaki komuta apply yerine destroy yazıp çalıştıralım. makineleri destroy etsin

Execute shell 
Command 
See the list of available environment variables 
/usr/local/bin" 
ANS KEYPAIR=" petclinic-ansible-test-dev. key" 
AWS 
cd infrastructure/ dev-k8s-terraform 
sed -i "s/c1arus/$ANS KEYPAIR/g" 
terraform init 
terraform destroy -auto-approve 
Advanced v 
main.tf 
- no- color 

12:55:12 
12: 55:12 Destroy complete! Resources: 
12: 55:12 Finished: SUCCESS 
8 destroyed. 

terminalde de jenkins user ile destroy edebilirdik:

[jenkins@jenkins-server test-creating-qa-automation-infrastructure]$ cd infrastructure/ 
[jenkins@jenkins-serwer infrastructure]$ cd dev-k8s-terraforn/ 
[jenkins@jenkins-serwer Is 
[jenkins@jenkins-server dev-k8s-terraform]$ terraform state list 
data. aws_vpc. name 
petclinic -master-serwer- profile 
_ pet init -master - s -role 
aws 
instance. kube -master 
aws 
instance. worker-I 
aws 
aws_instance. worker-2 
aws_secu rity_group _ petclinic - kube-master•- sg 
pet C I init: - kube-vnrker•- Se 
petclinic -mtual - sg 

key pair ve ECR repomuz kalacak. bir sonraki derste tekrar apply edecez ve devam edecez. (1407. satır)

Oluşturduğumuz key pair duruyor mu diye kontrol ettik (silmemiş olmamız lazım)

Jenkins te terraform dosyamızı apply etmemiz gerekiyor. PATH="$PATH:/usr/local/bin"

ANS\_KEYPAIR="petclinic-ansible-test-dev.key"

AWS\_REGION="us-east-1"

cd infrastructure/dev-k8s-terraform

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

terraform init

terraform apply -auto-approve -no-color

Komutunu job ta execute shell e yazıyoruz. Apply/Save/Build Now diyoruz.

Cluster için master ve worker node lar ayağa kalktı.

Ansible ile configurason yapacazğımız için makinelere ssh ile bağlanabiliyor mu denememiz lazım. Bunun için önce bağlanıyor mu diye kontrol edecez. Bunu da jenkins ile yapacaz ve job daki kodu değiştirecez:

ANS\_KEYPAIR="petclinic-ansible-test-dev.key"

ssh -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no -i ${WORKSPACE}/${ANS\_KEYPAIR} ubuntu@172.31.91.243 hostname

Ip adresini kendi node larımızdan birinin private IP ile değiştiriyoruz.

-o UserKnownHostsFile=/dev/null

Komutu ne için, ssh ile bağlanan hostları bir dosyadA TUTUYOR, TUTMASIN DİYE BU KOMUTU GİRİYORUZ.

-o StrictHostKeyChecking=no

Komutunu iste bağlanırken yes diye confirm istemesin diye yazıyoruz.

WORKSPACE jenkins in default variable ı, ANS\_KEYPAIR ise yukarıda tanımlı.

Jenkins in default değişkenlerini görmek için browserda 8080 den sonra /env-vars.html yazıyoruz:

C A Güvenli deéil 3.238.32.125:8080/'env-vars.html/ 
AWS Management... Course: AWS 13-TLI... Google Takvim - 13... O Your Repositories 
Jenkins 
Dashboard 
Q Search (CTRL+K) 
The following variables are available to shell and batch build steps: 
BRANCH NAME 
For e multibranch project, this will be set to the name ofthe branch being built, for example in c 
production from master but not from feature branches; if corresponding to some kind of chang. 
arbitrary (refer to CHAUCE 10 and CHANGE TARGET). 
BRANCH IS PRIMARY 
For e multibranch project, if the SCM source reports that the branch being built is a primary brat 
else unset. Some SCM sources may report more than one branch as a primary branch while othé 
information. 
CHANGE ID 
For e multibranch project corresponding to some kind of change request, this will be set to the 
request number, if supported; else unset. 
CHANGE URL 
For e multibranch project corresponding to some kind of change request, this will be set to the 
unset. 
CHANGE TITLE 
For e multibranch project corresponding to some kind of change request, this will be set to the t 
else unset. 
CHANGE AUTHOR 
For e multibranch project corresponding to some kind of change request, this will be set to the L 
proposed change, If supported; else unset. 
CHANGE AUTHOR DISPLAY NAME 

En sonunda da hostname komutu giriyoruz ki ssh ile bağlanmış mı anlamak için. Bağlandıysa hostun ip sini döndürecek. Bunun yerine herhangi bir linux komutu girilebilir. ssh komutunun özelliği.

Job u build edelim ve çalışıyor mu görelim.

17 : 3ø:ø4 Warning: Permanently added 
19-172-31-36-95 
Finished: SUCCESS 
'172.3: 

Çalıştı.

Şimdi ansible ile configürasyon yapmamız için inventory oluşturacaz.

Ana dizinde ansible/inventory klasörü oluşturuyoruz.

mkdir -p ansible/inventory

-p alt alta klasörleri oluşturuyordu.

hosts.ini isimli bir dosya oluşturup içine

172.31.91.243 ansible\_user=ubuntu

172.31.87.143 ansible\_user=ubuntu

172.31.90.30 ansible\_user=ubuntu

üç makinemizin private ip lerini giriyoruz. Dosya isminin önemi yok daha önce host, inventry, inventory.txt gibi isimler kullandık, yaml dosyası da kullandık. ini de bir dosya formatı. .txt ya da uzantısız da olsa olurdu.

Daha sonra değişiklikleri push luyoruz.

git add .

git commit -m 'added ansible static inventory host.ini for testing'

git push --set-upstream origin feature/msp-16

Şimdi tekrar test-creating-qa-automation-infrastructure

jobumuzun execute shell ini tekrar configure edecez. amacımızı ansible node lara ping atmak.

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

ANS\_KEYPAIR="petclinic-ansible-test-dev.key"

export ANSIBLE\_INVENTORY="${WORKSPACE}/ansible/inventory/hosts.ini"

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

export ANSIBLE\_HOST\_KEY\_CHECKING=False

ansible all -m ping

export komutu ile üç farklı env variable tanımladık. bu env var ların ismini ansivle docs tan alıyoruz:

<https://docs.ansible.com/ansible/latest/reference_appendices/config.html>

Ansible önce env var varsa onu alıyordu:

The configuration file 
Changes can be made and used in a configuration file Which will 
(environment variable if set) 
(in the current directory) 
cfc (in the home directory) 

dosya oluştursaydık, dosaynın olduğu yere gidip komutu orada çalıştıracaktık. Kurulumumuzda default gelen config dosyası yok.

Ansible da iki önemli conf ayarı vardı, bir config ne iki inventory ne?

Execute shelldeki komutu değiştirdikten sonra Apply/Save/Build diyoruz.

üç makineye de ping attı:

19 
19 
19 
19 
19 
19 
19 
19 
19 
19 
19 
19 
19 
19 
19 
19 
19 
19 
19 
19 
19 
19 
172 
172 
172 
.31.36.95 | SUCCESS { 
"ansible facts" : 
" di s covered_i nterpreter_python" • 
"changed": false, 
" ping" : 
" pong " 
.31.44.83 | SUCCESS { 
"ansible facts" : 
" di s covered_i nterpreter_python" • 
"changed": false, 
" ping" : 
" pong " 
.31.35 .247 | SUCCESS { 
"ansible facts" : 
" di s covered_i nterpreter_python" • 
"changed": false, 
" ping" : 
" /usr/bin/python3" 
" /usr/bin/python3" 
" /usr/bin/python3" 
: 45 
: 14 
Finished: 
" pong " 
SUCCESS 

biz dynamic inventory kullanmak istiyoruz. şimdi ansible/inventory

altında dev\_stack\_dynamic\_inventory\_aws\_ec2.yaml

isimli bir dosya oluşturup dynamic inventory yi yazalım. dynamic inventory sounuda aws instanceları kullanıyorsa aws\_ec2 yazması gerekiyordu.

plugin: aws\_ec2

regions:

- "us-east-1"

filters:

tag:Project: tera-kube-ans

tag:environment: dev

instance-state-name: running

keyed\_groups:

- key: tags['Project']

prefix: 'all\_instances'

- key: tags['Role']

prefix: 'role'

hostnames:

- "ip-address"

compose:

ansible\_user: "'ubuntu'"

filters kısmı hesaptaki onlarca makineden sadece istediklerimizi görmesi için yazdığımız kısım ve ec2 ları taglerine göre ayıklıyor. bu tag ler main.tf te verdiğimiz tagler:

tags — 
Name : "kube-master" 
Project — 
Role = "master" 
environment' "dev? 

instance-state-name:running

kısmı aynı tage sahip olan instance lardan sunning olanları filtrelemesi için eklediğimiz bir parametre.

bu filtrelerin neler olacağını aws cli docs tan alıyoruz:

AWS CLI Command Reference Hame userGuide Forum GitHuo 
O stM 13.839 
• The state ofthe instance 
stopping I 
instance-type The type of instance (for example: j. 
Instance.group•id • The ID of the security group for the instance 
e - The name of the security group for the instance. 
- The public IPV4 address Of the instance 
kernel-id The kernel ID 
- The name of the key pair used when the instance was launched. 
- When launching multiple instances. this is for the instance in the launch 
group (tor example, O, I, 2, and so on) 
launch-time - The time When the instance Was launched. in the ISO 8601 format in the u TC 
zone for example, 2821-eg-29T11 3esz _ You can use a 
wildcard ( • for example, 2e21-E-2gr• , which matches an entire day. 
metadata-options.http-tokens - The metadata request authorization state ( opttcnal I 
required ) 

keyed\_groups da ise filtrelenmiş makineleri grupluyor.

filtrelenmiş makineler içinde sadece worker node lara ya da üçüne birden yapacağımız işlemler var. bu nedenle ayrı ayrı grupladık.

buradaki prefix olmazsa tag in başına \_ geliyor. prefix yazdığımız zaman grupta ismin başına o prefixi getiriyor. \_ kaldırmak için de seperator kullanıorduk. seperator: " " şeklinde.

en alttaki compose ise variable lar atıyordu.

compose • 
ansible user; " 'ubuntu'" 
foc : 
boc • private_dns 

bu değişkenlerin cevabını da gathering yaparak ansible fact te buluyordu.

ubuntu tırnak içinde çünkü ansible fact içinde böyle bir parametre yok, string olarak yazılmış. gerekirse kullanmak için. strign kullanacaksak " ' ' " içinde yazıyoruz.

git add .

git commit -m 'added ansible dynamic inventory files for dev environment'

git push

şimdi de job ın execute shell ini değiştiryoruz

APP\_NAME="Petclinic"

ANS\_KEYPAIR="petclinic-ansible-test-dev.key"

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

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

export ANSIBLE\_HOST\_KEY\_CHECKING=False

ansible-inventory -v -i ./ansible/inventory/dev\_stack\_dynamic\_inventory\_aws\_ec2.yaml --graph

ve dynamic olarak makinelere ulaşabildiğimizi görelim.

dynamic inventory komutu ansible-inventory --graph tı.

-v de açıklamalı getirmesi içindi.

path i de jenkins server da workspace in içinde test-creating-qa-automation-infrastructure

jobunun içindeki ansible/inventory altını gösterdik.

Job build oldu:

No config file found; using defaults 
Using inventory plugin 'ensible collections .amazon.aws .plugins.inventory .aws_ec2' 
to process inventory source 
/ van/ lib/ j enkins/wor kspace/test-creating- -automation - 
infra s t ure/ an s invento ev_s c yamI 
911: 
2ø:øø 
2ø:øø: 
2ø:øø: 
2ø:øø: 
2ø:øø: 
2ø:øø:3ø 
-@ungrouped : 
-@aws_ec2: 
1--34.227.67.192 
1--54.237.25a.2e 
1--54.159.113.195 
kube_ans: 
1--34 
1--54 
1--54 
-@role 
1--34 
1--54 
-@role 
1--54 
. 227.67.192 
. 237.250.20 
. 159.113.195 
worker : 
. 227.67.192 
. 159.113.195 
master : 
. 237.250.20 
Finished: SUCCESS 

şimdi ise dynmaic inventory deki makinelere ping atma işlemini jenkinse yaptıracaz.

job es (execute shell) i değiştirelim:

APP\_NAME="Petclinic"

ANS\_KEYPAIR="petclinic-ansible-test-dev.key"

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

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

export ANSIBLE\_HOST\_KEY\_CHECKING=False

ansible -i ./ansible/inventory/dev\_stack\_dynamic\_inventory\_aws\_ec2.yaml all -m ping

private key gereikyor çünkü ssh ile bağlanaca, export ile tanıtmışız.

yine export ile ssh ile bağlanırken onay sormasın diye ANSIBLE\_HOST\_KEY\_CHECKING=False

tanıttık.

Apply/Save/Build Now yapıyoruz.

20 
20 
: 24: 47 
20 
: 24: 47 
20 
: 24: 47 
20 
: 24: 47 
20 
: 24: 47 
20 
: 24: 47 
20 
20 
: 24: 47 
20 
: 24: 47 
20 
: 24: 47 
20 
: 24: 47 
20 
: 24: 47 
20 
: 24: 47 
20 
20 
: 24: 47 
20 
: 24: 47 
20 
: 24: 47 
20 
: 24: 47 
20 
: 24: 47 
20 
: 24: 47 
20 
: 24 
: 24 
: 24 
: 24 
: 47 
: 47 
: 47 
: 47 
54.159.113.195 SUCCESS { 
"ansible facts" : 
" di s covered_i nterpreter_python" • 
"changed": false, 
" ping" : 
" pong " 
54.237.2se.2e I SUCCESS { 
"ansible facts" : 
" di s covered_i nterpreter_python" • 
"changed": false, 
" ping" : 
" pong " 
" /usr/bin/python3" 
" /usr/bin/python3" 
" /usr/bin/python3" 
34.227.67.192 | SUCCESS { 
"ansible facts" : 
" di s covered_i nterpreter_python" • 
"changed": false, 
" ping" : 
" pong " 
Finished: 
SUCCESS 

ping: pong ları gördük.

şimdi makinelere cluster kuracağız. ansible/playbooks

altında k8s\_setup.yaml

isimli bir yaml dosyası oluşturuyoruz.

README. deki içeriği yapıştırıyoruz.

hostvars magic variables dı ansible için.

playbook u inceleyelim:

name: swap off 
shell: I 
free -m 
swapoff -a && sed -i ' / swap / s/AV#/' /etc/fstab 

bu kısım kubectl in çalışması aksamasın diye ram in kullanımını azaltan bir configürasyon. bu alttaki kısım dire docs tan alınıyor.

name : 
shell: I 
cat 
net 
net 
. ipv4. ip_forward 
net 
EOF 
sysctl 
- - system 
Enable 
the nodes to see bridged traffic 
sudo tee /etc/sysct1.d/k8s. conf 
EOF 
-bridge . bridge-nf-ca11-ip6tab1es - 
1 
. bridge . bridge- nf- call-iptables 
1 
1 

bu kısımda paketleri indiriyoruz:

name: Install pakkage that allow apt to be used over HTTPS 
apt: 
{{ package* 
name: " 
state: present 
update_cache: yes 
vars: 
packages 
apt-transport-https 
curl 
ca-certificates 

burada loop ta yapılabilirdi ancak pavkages diye variable tanımlanıp onunla yapılmış.

name: update 
apt-get and install kube packages 
shell: I 
curl https://packages . cloud . google . com/apt/doc/apt-key . 
gpg I 
sudo 
apt- key 
-keyring /usr/share/keyrings/cloud. 
google . gpg 
add - && \ 
echo '"deb [signed-by=/usr/share/keyrings/cloud .google. 
gpg] https://apt . kubernetes . io/ kubernetes-xenial 
sudo tee /etc/apt/sources.list.d/kubernetes. 
ma In 
list && \ 
apt-get update -q && \ 
apt-get install -qy kube1et=1.26.3-00 kubectl=l .26.3-øe 
kubeadm=l . 26.3-00 kubernetes-cni docker 
apt-mark hold kubelet kubeadm kubectl 

bu kısımda kubelet, kubectl, kubeadm kurulumu var burada versionlar önemli.

bu dosyayı kaydettikten sonra

git add .

git commit -m 'added ansible playbooks for dev environment'

git push

şimdi test-creating-qa-automation-infrastructure

jobını configüre ediyoruz.

APP\_NAME="Petclinic"

ANS\_KEYPAIR="petclinic-ansible-test-dev.key"

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

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

export ANSIBLE\_HOST\_KEY\_CHECKING=False

# k8s setup

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

bu komutla cluster kuracağız.

Apply/Save/Build Now diyoruz.

ok: [34.227.67.192] { 
"msg": " [preflight] Running pre-flight Reading 
configuration from the cluster... FYI: You can look at this config file with 
' kubectl -n kube- system get cm kubeadm-config -o yamI'\nCkubeIet-start] Writing kubelet 
configuration to file Writing kubelet 
environment file with flags to file /var/lib/kubelet/kubeadm-flags 
start] Starting the Waiting for the kubelet to perform the TLS 
Bootstrap. node has joined the cluster: \n* Certificate signing request was sent 
to apisemrer and response was received. The Kubelet was informed of the new secure 
connection details . 'kubectl get nodes' on the control-plane to see this node join 
the cluster." 
ok: [54.159.113.195] { 
"msg": " [preflight] Running pre-flight Reading 
configuration from the cluster... FYI: You can look at this config file with 
' kubectl -n kube- system get cm kubeadm-config -o yamI'\nCkubeIet-start] Writing kubelet 
configuration to file Writing kubelet 
environment file with flags to file /var/lib/kubelet/kubeadm-flags 
start] Starting the Waiting for the kubelet to perform the TLS 
Bootstrap. node has joined the cluster: \n* Certificate signing request was sent 
to apisemrer and response was received. The Kubelet was informed of the new secure 
34.227.67.192 
54.159.113.195 
54.237.2se.2e 
connection details . 'kubectl get nodes' on the 
the cluster." 
PLAY RECAP * 
control- plane to 
see this node join 
skipped=e 
skipped=e 
skipped=e 
rescued=e 
rescued=e 
rescued=e 
Finished: SUCCESS 
. ok=14 
ignored=ø 
. ok=14 
ignored=ø 
. ok=2a 
ignored=ø 
changed—Il 
changed—Il 
changed=17 
unreachable=a 
unreachable=a 
unreachable=a 
failed=e 
failed=e 
failed=e 

k8s\_setup.yaml ın çalıştığını gördük.

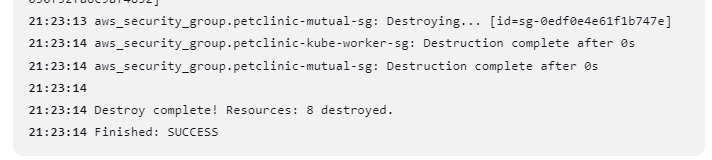
terraform destroy edebiliriz.

cd infrastructure/dev-k8s-terraform

terraform destroy -auto-approve -no-color

komutu ile jobımızı conf edecez.

Apply/Save/Build Now diyoruz.



key pairi silmek için job ı conf ediyoruz:

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

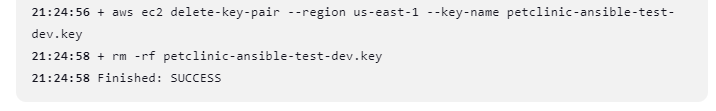
ANS\_KEYPAIR="petclinic-ansible-test-dev.key"

AWS\_REGION="us-east-1"

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

rm -rf ${ANS\_KEYPAIR}

Apply/Save/Build Now diyoruz.



1. adımda deneme yaptığımız komutları kaydetmek için create-qa-automation-environment.sh

isimli bir .sh dosyası oluşturup komutları buraya atalım.

# Environment variables

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

APP\_NAME="Petclinic"

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

AWS\_REGION="us-east-1"

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

export ANSIBLE\_HOST\_KEY\_CHECKING=False

# Create key pair for Ansible

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

chmod 400 ${ANS\_KEYPAIR}

# Create infrastructure for kubernetes

cd infrastructure/dev-k8s-terraform

terraform init

terraform apply -auto-approve -no-color

# Install k8s cluster on the infrastructure

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

# Build, Deploy, Test the application

# Tear down the k8s infrastructure

cd infrastructure/dev-k8s-terraform

terraform destroy -auto-approve -no-color

# Delete key pair

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

rm -rf ${ANS\_KEYPAIR}

daha sonra push ve merge ediyoruz:

git add .

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

git push

git checkout dev

git merge feature/msp-16

git push origin dev