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)