DevOps Foundations: Infrastructure as Code Course Handout

Table of Contents

Course Materials

The Github repo for Kubespray is here:
https://github.com/ernestm/k8s-terraform-ansible-sample

The Github repo for the word-cloud-generator app is here:
https://github.com/wickett/word-cloud-generator

The Dockerhub repo for the word-cloud-generator is here:
https://hub.docker.com/r/wickett/word-cloud-generator

Create A Cloud Account

1. In your browser of choice, create an AWS account at https://aws.amazon.com/free/.
2. Log into the AWS console as the root user.
3. Set up MFA on your root user under “Security Credentials” in the pop-down menu from your username in the top right. I use Google Authenticator.

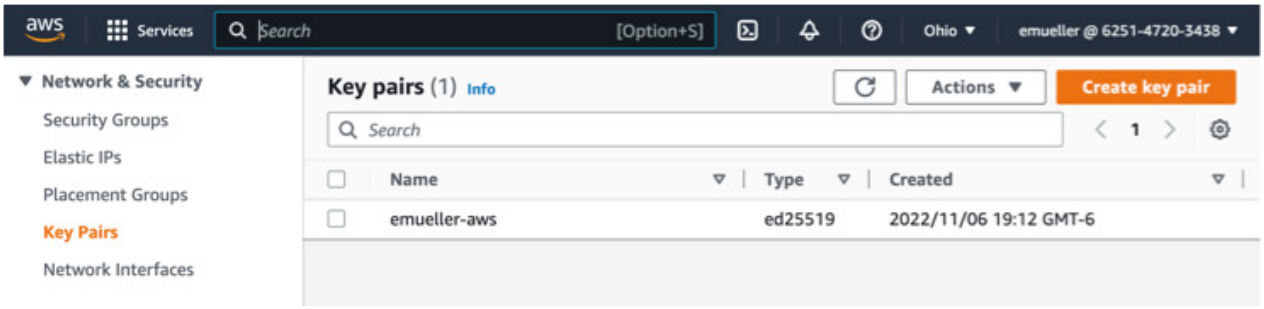
4. Create an IAM user account for yourself in “IAM/Users” in the AWS console with Administrator privilege. Note the password and make an AWS access key and secret key and record them.
5. Log out as the root user and never log in as it again (big security risk). Log into the IAM user and set up MFA.

Create a Cloud Server and Connect to It from Your Local Machine

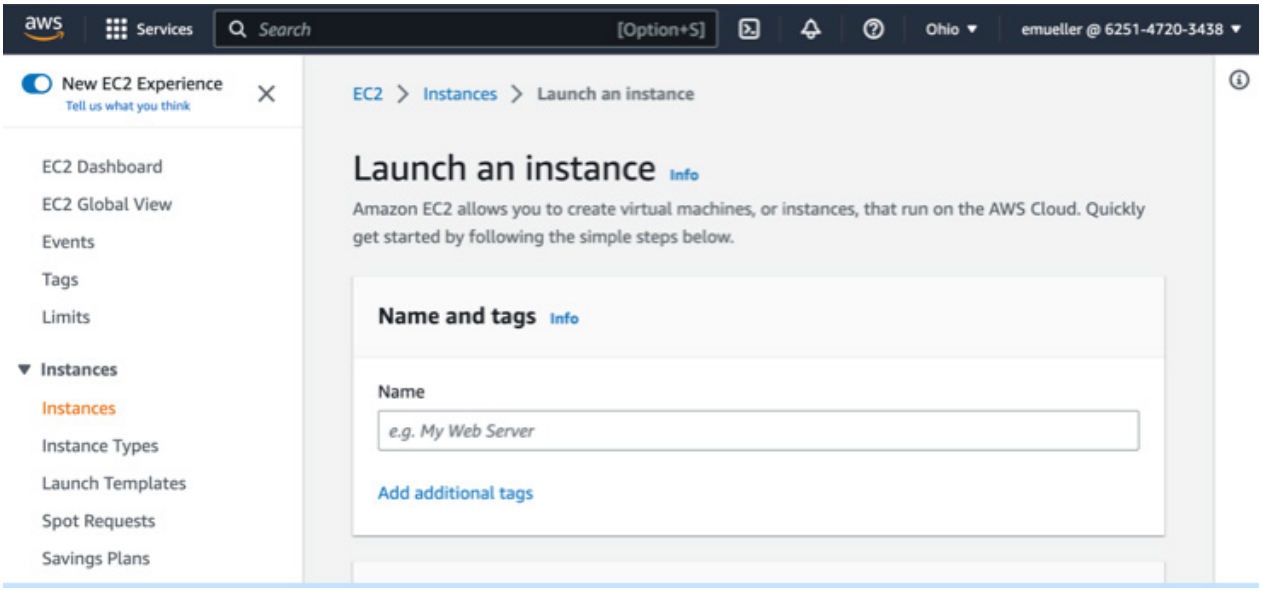
1. On your local machine, create a SSH keypair in bash (in a terminal window on Linux or Mac; on Windows you can install WSL: How to Install WSL2 (Windows Subsystem for Linux 2) on Windows 10 (freecodecamp.org)) I’ll call mine “emueller-aws” but you do you.

   ```
   ssh-keygen -f /home/ernestm/.ssh/emueller-aws -t ed25519
   ```

2. Pick an AWS region you want to do all this in that is close to you. I used Ohio (us-east-2).
3. In AWS console EC2/Key Pairs, Import Key Pair and choose your SSH public key (the one that ends in .pub). Now it’s available in AWS to put onto servers so you can log in to them.



4. In the AWS console under EC2/Instances, launch Instances and choose Ubuntu and a free tier instance size like t2.micro and select your imported SSH keypair. Allow ssh traffic from your IP (you’ll need to change this if you move around).



5. Once the instance is in Running state, connect to its Public IPv4 address (visible when you click on it in the console) from bash in Linux or WSL on Windows, first to put your SSH keys on it for later use and then to log in.

```
cd .ssh
scp -i emueller-aws emueller-aws ubuntu@13.59.37.13:.ssh/emueller-aws
scp -i emueller-aws emueller-aws.pub
ubuntu@13.59.37.13:.ssh/emueller-aws.pub
cd ..
ssh -i ~/.ssh/emueller-aws ubuntu@13.59.37.13
```

You are now logged into the AWS instance! #Cloud4Lyfe

Update Your New Server and Install Terraform

1. Execute the following commands on your AWS Ubuntu instance to update the OS.

```
sudo apt update
sudo apt --only-upgrade install grub-efi-amd64-signed
sudo apt upgrade
```

2. Reboot the server from the AWS console (Instance state/Reboot instance) and reconnect via SSH as you did above.

3. Now let's install Python and Terraform.

```
sudo apt install python3-virtualenv
sudo apt-get update && sudo apt-get install -y gnupg software-
properties-common
wget -O- https://apt.releases.hashicorp.com/gpg | \
  gpg --dearmor | \
  sudo tee /usr/share/keyrings/hashicorp-archive-keyring.gpg
gpg --no-default-keyring \
  --keyring /usr/share/keyrings/hashicorp-archive-keyring.gpg \
  --fingerprint
echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-
keyring.gpg] \
  https://apt.releases.hashicorp.com $(lsb_release -cs) main" | \
  sudo tee /etc/apt/sources.list.d/hashicorp.list
sudo apt update
sudo apt-get install terraform
```

You should be ready to run Terraform programs now.

```
$ terraform version
Terraform v1.3.8
on linux_amd64
```

Install Kubespray

Kubespray (https://kubespray.io/#/) is a demo system to build a k8s cluster from scratch on a variety of cloud platforms and supporting a variety of system components. The Terraform part is hidden away as a user contributed module, https://github.com/kubernetes-sigs/kubespray/tree/master/contrib/terraform/aws

We've forked this project at https://github.com/ernestm/kubespray so that it doesn't change and invalidate these instructions.

On your AWS instance, run ssh-agent with your SSH key so you can pull from git using ssh.

```
eval $(ssh-agent -s)
ssh-add ~/.ssh/emueller-aws
```

Clone our fork of https://github.com/kubernetes-sigs/kubespray

```
git clone git@github.com:ernestm/kubespray.git
```

Set up the Kubespray virtual environment

```
VENVDIR=kubespray-venv
KUBESPRAYDIR=kubespray
ANSIBLE_VERSION=2.12
virtualenv  --python=$(which python3) $VENVDIR
source $VENVDIR/bin/activate
cd $KUBESPRAYDIR
pip install -U -r requirements-$ANSIBLE_VERSION.txt
test -f requirements-$ANSIBLE_VERSION.yml && \
  ansible-galaxy role install -r requirements-$ANSIBLE_VERSION.yml && \
  ansible-galaxy collection -r requirements-$ANSIBLE_VERSION.yml
```

Install AWS Infrastructure with Terraform

Set up some environment variables so Terraform knows where to put your infrastructure. Use the credentials for your account you made above and use the same region you put your initial host in.

```
export TF_VAR_AWS_ACCESS_KEY_ID="aws access key for your IAM user, AKIA..."
export TF_VAR_AWS_SECRET_ACCESS_KEY="aws secret key for your IAM user"
export TF_VAR_AWS_SSH_KEY_NAME="emueller-aws"
export TF_VAR_AWS_DEFAULT_REGION="us-east-2"
```

Run the terraform that does the magic! If you get errors in init or plan, you'll be doing some debugging.

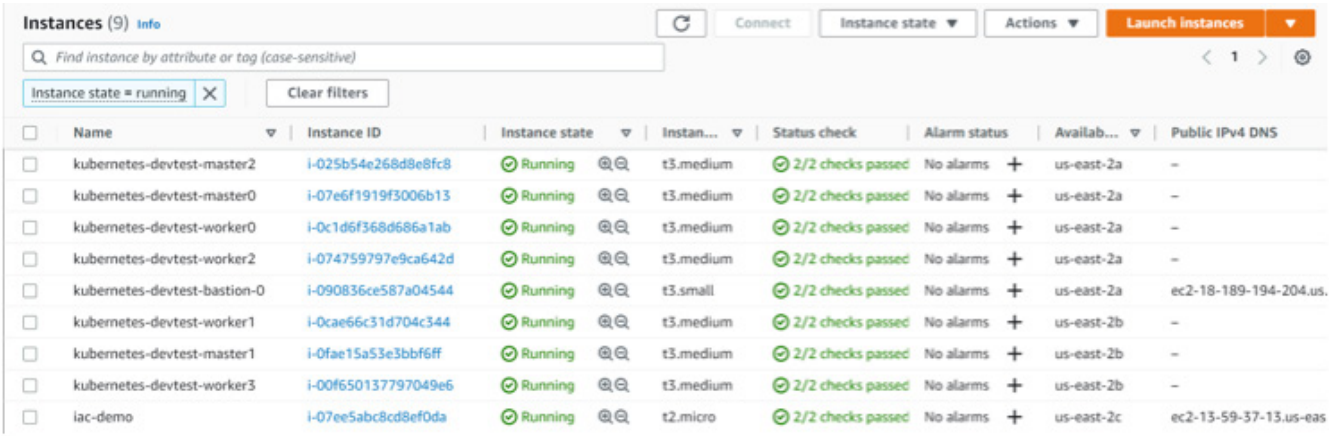
```
cd contrib/terraform/aws
# Initialize terraform (you only have to do this once)
terraform init
# Check for valid tf syntax
terraform validate
# Dry run the terraform and see what it thinks it needs to do
terraform plan
# Really run the terraform to make AWS infrastructure!
terraform apply
```

You should get output looking like:

```
Apply complete! Resources: 42 added, 0 changed, 0 destroyed.

Outputs:
aws_nlb_api_fqdn = "kubernetes-nlb-devtest-de1cceb4b81daedd.elb.us-
east-2.amazonaws.com:6443"
bastion_ip = "3.137.167.28"
default_tags = tomap({})
…
EOT
```

Go look in AWS and you should have a bunch of new Debian servers created!



Examine your kubespray/contrib/terraform/aws/terraform.tfstate file. It has the state of the infrastructure that it was created in within it, including details on every single piece. It will look like:

```
{
  “version”: 4,
  “terraform_version”: “1.3.4”,
  “serial”: 63,
  “lineage”: “5c1c68f3-39be-d491-abc8-9fbfce662025”,
  “outputs”: {
    “aws_nlb_api_fqdn”: {
      “value”: “kubernetes-nlb-devtest-de1cceb4b81daedd.elb.us-east-
2.amazonaws.com:6443”,
      “type”: “string”
    },
    “bastion_ip”: {
      “value”: “3.137.167.28”,
      “type”: “string”
    },
    “default_tags”: {
      “value”: {},
      “type”: [
        “map”,
        “string”
      ]
    },
    “etcd”: {
      “value”: “10.250.196.108\n10.250.217.58\n10.250.200.189”,
      “type”: “string”
    },
    “inventory”: {
      “value”: “[all]\nip-10-250-196-108.us-east-2.compute.internal
ansible_host=10.250.196.108\nip-10-250-217-58.us-eas
t-2.compute.internal ansible_host=10.250.217.58\nip-10-250-200-189.us-
east-2.compute.internal ansible_host=10.250.200.18
9\nip-10-250-203-94.us-east-2.compute.internal
ansible_host=10.250.203.94\nip-10-250-221-225.us-east-
2.compute.internal
…
```

It has also put a hosts file for Ansible to use in kubespray/inventory/hosts.

You can change your infrastructure by changing the Terraform code or settings in the tfvars file and then again running:

```
terraform validate
terraform plan
terraform apply
```

Note that it’s safe to stop this infrastructure in the AWS console: select all the servers in this kubernetes cluster and choose Instance State/Stop Instance to shut them all down. When you come back you can do Instance State/Start Instance to all of them and Kubernetes—and any other installed applications—should start back up with no problem! The IP of the kubernetes

bastion will change, but if you run “terraform apply,” it will update the kubespray/inventory/hosts file with the new one for you!

If you want to blow this all away you can use:

```
terraform destroy
```

Install k8s on Your Infrastructure with Ansible

You have the servers, but you don’t have the Kubernetes yet! We’ll install it with Ansible.

```
cd ../../..
cp -rfp inventory/sample inventory/mycluster
```

Uncomment the cloud_provider option in inventory/mycluster/group_vars/all/all.yml and set it to ‘aws’

```
ansible-playbook -i ./inventory/hosts ./cluster.yml -e
ansible_user=admin -b --become-user=root --flush-cache -e
ansible_ssh_private_key_file=~/.ssh/emueller-aws

# Get the controller’s IP address.
CONTROLLER_HOST_NAME=$(cat ./inventory/hosts | grep
“\[kube_control_plane\]” -A 1 | tail -n 1)
CONTROLLER_IP=$(cat ./inventory/hosts | grep $CONTROLLER_HOST_NAME |
grep ansible_host | cut -d‘=’ -f2)

# Get the hostname of the load balancer.
LB_HOST=$(cat inventory/hosts | grep
apiserver_loadbalancer_domain_name | cut -d‘”’ -f2)

# Get the controller’s SSH fingerprint.
ssh-keygen -R $CONTROLLER_IP > /dev/null 2>&1
ssh-keyscan -H $CONTROLLER_IP >> ~/.ssh/known_hosts 2>/dev/null

# Get the kubeconfig from the controller.
mkdir -p ~/.kube
ssh -i ~/.ssh/emueller-aws -F ssh-bastion.conf
admin@$CONTROLLER_IP “sudo chmod 644 /etc/kubernetes/admin.conf”
scp -i ~/.ssh/emueller-aws -F ssh-bastion.conf
admin@$CONTROLLER_IP:/etc/kubernetes/admin.conf ~/.kube/config
sed -i “s^server:.*^server: https://$LB_HOST:6443^” ~/.kube/config
chmod 600  /home/ubuntu/.kube/config

# Install kubectl
curl -LO “https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl”
curl -LO “https://dl.k8s.io/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256”
echo “$(cat kubectl.sha256)  kubectl” | sha256sum --check
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

Now you're ready to poke at your kubernetes cluster! Let's look at all the hosts and all the services currently deployed on them.

```
kubectl get nodes
kubectl get all --all-namespaces
```

Look, a working Kubernetes installation courtesy of Ansible.

Scale Up the K8S Cluster

Look at the cluster and verify you have three worker nodes.

```
$ kubectl get nodes
NAME                                          STATUS   ROLES           AGE   VERSION
ip-10-250-196-108.us-east-2.compute.internal  Ready    control-plane   92d   v1.25.3
ip-10-250-200-189.us-east-2.compute.internal  Ready    control-plane   92d   v1.25.3
ip-10-250-203-128.us-east-2.compute.internal  Ready    <none>          92d   v1.25.3
ip-10-250-203-94.us-east-2.compute.internal   Ready    <none>          92d   v1.25.3
ip-10-250-217-58.us-east-2.compute.internal   Ready    control-plane   92d   v1.25.3
ip-10-250-218-139.us-east-2.compute.internal  Ready    <none>          92d   v1.25.3
```

You want four worker nodes, you say? Well, step one is to increase the number of workers in Terraform.

Change aws_kube_worker_num in terraform.tfvars to 4

```
terraform validate
terraform plan
terraform apply
```

Now go up to the kubespray root and run the Ansible scaling playbook.

```
ansible-playbook -i ./inventory/hosts ./scale.yml -e
ansible_user=admin -b --become-user=root --flush-cache -e
ansible_ssh_private_key_file=~/.ssh/emueller-aws
```

And now we have five workers reporting in!

```
$ kubectl get nodes

NAME                                          STATUS   ROLES           AGE     VERSION
ip-10-250-196-108.us-east-2.compute.internal  Ready    control-plane   92d     v1.25.3
ip-10-250-198-74.us-east-2.compute.internal   Ready    <none>          3m56s   v1.25.3
ip-10-250-200-189.us-east-2.compute.internal  Ready    control-plane   92d     v1.25.3
ip-10-250-203-128.us-east-2.compute.internal  Ready    <none>          92d     v1.25.3
ip-10-250-203-94.us-east-2.compute.internal   Ready    <none>          92d     v1.25.3
ip-10-250-217-58.us-east-2.compute.internal   Ready    control-plane   92d     v1.25.3
ip-10-250-218-139.us-east-2.compute.internal  Ready    <none>          92d     v1.25.3
```

Deploy an Application Manually

Okay, but you have an empty Kubernetes cluster! Don't you want to run an application?

```
kubectl create deployment nginx --image=nginx
kubectl get pods -l app=nginx
POD_NAME=$(kubectl get pods -l app=nginx -o
jsonpath="{.items[0].metadata.name}")
kubectl port-forward $POD_NAME 8080:80
```

You can run it from another shell on the same box:

```
curl --head http://127.0.0.1:8080
```

You should see the nginx headers.

```
kubectl logs $POD_NAME
kubectl exec -ti $POD_NAME -- nginx -v
kubectl scale --replicas=3 deployment/nginx
kubectl get pods
kubectl expose deployment nginx --port=80 --target-port=80 --name=lb-
service --type=LoadBalancer
kubectl describe service lb-service
```

Get the "LoadBalancer Ingress" from the output and you can curl it:

```
curl --head http://abe7e35901f2b4b30ac667aa1a6cc251-1561050260.us-
east-2.elb.amazonaws.com
```

Go hit the ingress in your browser:

```
http://abe7e35901f2b4b30ac667aa1a6cc251-1561050260.us-east-
2.elb.amazonaws.com/
```

You can remove this deployment with:

```
kubectl delete service lb-service
kubectl delete deployment nginx
```

Install Helm

Learn more about Helm at https://helm.sh.

```
curl https://baltocdn.com/helm/signing.asc | gpg --dearmor | sudo tee
/usr/share/keyrings/helm.gpg > /dev/null
sudo apt-get install apt-transport-https --yes
echo "deb [arch=$(dpkg --print-architecture) signed-
by=/usr/share/keyrings/helm.gpg]
https://baltocdn.com/helm/stable/debian/ all main" | sudo tee
/etc/apt/sources.list.d/helm-stable-debian.list
sudo apt-get update
sudo apt-get install helm
```

Check it out with:

```
helm version
```

Deploy nginx with a Helm Chart

Let's start with the bitnami provided nginx helm chart from
https://artifacthub.io/packages/helm/bitnami/nginx

Add their repo to helm.

```
helm repo add my-repo https://charts.bitnami.com/bitnami
helm pull bitnami/nginx
```

There should now be a file called nginx-<some version>.tgz in your directory. Unpack it:

```
tar zxvf nginx-<some-version>.tgz
```

In the README.md you can see all the options.

```
helm install nginx nginx/ --values nginx/values.yaml
```

Get the connection information:

```
export SERVICE_PORT=$(kubectl get --namespace default -o
jsonpath="{.spec.ports[0].port}" services nginx)
export SERVICE_IP=$(kubectl get svc --namespace default nginx -o
jsonpath='{.status.loadBalancer.ingress[0].hostname}')
echo "http://${SERVICE_IP}:${SERVICE_PORT}"
```

It'll show something like

```
http://a7b2151fa5e8646fc935fa36bd77bb40-275720885.us-east-2.elb.amazonaws.com:80
```

You should be able to hit it in your browser!

You can remove it when you want with:

```
helm delete nginx
```

Deploy a Custom Application with a Helm Chart

Make a chart from "scratch".

```
helm create wordcloud
cd wordcloud
vi values.yaml
```

Change the replicaCount to three.
Change the image stanza to read:

```
image:
  repository: wickett/word-cloud-generator
  pullPolicy: IfNotPresent
  # Overrides the image tag whose default is the chart appVersion.
  tag: "latest"
```

And change the service definition to read:

```
service:
  type: LoadBalancer
  port: 8888
```

Check it out:

```
helm lint wordcloud
helm template --validate --debug wordcloud
```

Install it on our cluster:

```
helm install wordcloud wordcloud/ --values wordcloud/values.yaml
```

It'll say:

```
NAME: wordcloud
LAST DEPLOYED: Wed Feb  8 02:08:02 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
     NOTE: It may take a few minutes for the LoadBalancer IP to be available.
           You can watch the status of by running 'kubectl get --
namespace default svc -w wordcloud'
  export SERVICE_IP=$(kubectl get svc --namespace default wordcloud --
template "{{ range (index .status.loadBalancer.ingress 0) }}{{.}}{{
end }}")
  echo http://$SERVICE_IP:8888

$ kubectl get all
NAME                                READY   STATUS    RESTARTS   AGE
pod/wordcloud-786f647486-5lstv      1/1     Running   0          4m43s
pod/wordcloud-786f647486-hq52m      1/1     Running   0          4m43s
pod/wordcloud-786f647486-zqkhm      1/1     Running   0          4m43s

NAME                 TYPE          CLUSTER-IP     EXTERNAL-IP       PORT(S)          AGE
service/kubernetes   ClusterIP     10.233.0.1     <none>            443/TCP          92d
service/wordcloud    LoadBalancer  10.233.27.147   a11649bae78424c6db92e3373933cc54-194045323.
us-east-2.elb.amazonaws.com                                        8888:31487/TCP   4m43s

NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/wordcloud      3/3     3            3           4m43s

NAME                                      DESIRED   CURRENT   READY   AGE
replicaset.apps/wordcloud-786f647486      3         3         3       4m43s
```

Get that load balancer DNS name and port from the output and hit it in your browser, in this case, **http://a11649bae78424c6db92e3373933cc54-194045323.us-east-2.elb.amazonaws.com:8888.** You should see the word cloud generator UI! Use away.

Once done, you are free to:

```
helm delete wordcloud
```

Docker with Word Cloud Generator

Install Docker Desktop:
https://www.docker.com/products/docker-desktop/

Get repo for word-cloud-generator:
https://github.com/wickett/word-cloud-generator

Use Make commands to build and run docker containers:
https://github.com/wickett/word-cloud-generator/blob/master/Makefile

```
make docker-build
make docker-run
```

```
# depending on your OS and processor, you might need to change the
Makefile to run a different container. The instructor used an Apple M2
chip as mentioned in the video.
```

CI for IaC with GitHub Actions

The code for this video is located at:
https://github.com/wickett/word-cloud-generator/blob/master/.github/workflows/build-and-publish.yml

Serverless Framework

Create an AWS account and then set up a free Serverless framework account at:

Serverless Framework https://www.serverless.com/