Capstone Project

Monday, May 31, 2021 2:25 PM

INFRA OPTIMIZATION

GitHub Repo: https://github.com/0ktay3min/CapstoneProject.git

DESCRIPTION
Create a DevOps infrastructure for an e-commerce application to run on high-availability mode.
Background of the problem statement:
A popular payment application, **EasyPay** where users add money to their wallet accounts, faces an issue in its payment success rate. The timeout that occurs with the connectivity of the database has been the reason for the issue.
While troubleshooting, it is found that the database server has several downtime instances at irregular intervals. This situation compels the company to create their own infrastructure that runs in high-availability mode.
Given that online shopping experiences continue to evolve as per customer expectations, the developers are driven to make their app more reliable, fast, and secure for improving the performance of the current system.
Implementation requirements:

1. Create the cluster (EC2 instances with load balancer and elastic IP in case of AWS)
2. Automate the provisioning of an EC2 instance using Ansible or Chef Puppet
3. Install Docker and Kubernetes on the cluster
4. Implement the network policies at the database pod to allow ingress traffic from the front-end application pod
5. Create a new user with permissions to create, list, get, update, and delete pods
6. Configure application on the pod
7. Take snapshot of ETCD database
8. Set criteria such that if the memory of CPU goes beyond 50%, environments automatically get scaled up and configured

The following tools must be used:

1. EC2
2. Kubernetes
3. Docker
4. Ansible or Chef or Puppet

The following things to be kept in check:

1. You need to document the steps and write the algorithms in them.
2. The submission of your GitHub repository link is mandatory. In order to track your tasks, you need to share the link of the repository.
3. Document the step-by-step process starting from creating test cases, then executing them, and recording the results.
4. You need to submit the final specification document, which includes:
   - Project and tester details
   - Concepts used in the project
   - Links to the GitHub repository to verify the project completion
   - Your conclusion on enhancing the application and defining the USPs (Unique Selling Points)

1-Creating four Ansible Roles:

a-) Create a directory called *"CapstoneProject"* and inside this directory create another directory called *"roles"*.
b-) Go inside "roles" directory and execute the following commands

Terminal Commands:
$ mkdir CapstonProject
$ cd CapstonProject
$ mkdir roles
$ cd roles
$ ansible-galaxy init ec2
$ ansible-galaxy init k8s_master
$ ansible-galaxy init k8s_slave
$ ansible-galaxy init deployment

```
ares@master:~$ mkdir CapstoneProject
ares@master:~$ cd CapstoneProject
ares@master:~/CapstoneProject$ mkdir roles
ares@master:~/CapstoneProject$ cd roles
ares@master:~/CapstoneProject/roles$ ansible-galaxy init ec2
- Role ec2 was created successfully
ares@master:~/CapstoneProject/roles$ ansible-galaxy init k8s_master
- Role k8s_master was created successfully
ares@master:~/CapstoneProject/roles$ ansible-galaxy init k8s_slave
- Role k8s_slave was created successfully
ares@master:~/CapstoneProject/roles$
```

```
ares@master:~/CapstoneProject/roles$ ansible-galaxy init deployment
- Role deployment was created successfully
ares@master:~/CapstoneProject/roles$
```

2-Setting up Ansible Configuration File :

a-) In Ansible we have two kinds of configuration file — Global & Local. We are going create one local configuration file inside *"CapstoneProject"* folder, and whatever Ansible commands we want to run in future we will run on this folder. Because then only Ansible will be able to read this Local configuration file and can work accordingly.

b-) **Terminal Commands:**
$ sudo vim ansible.cfg => to create a file called ansible.cfg. Once file is created, copy the following parameters inside this file.

```
[defaults]
host_key_checking=False
command_warnings=False
deprecation_warnings=False
ask_pass=False
roles_path= ./roles
force_valid_group_names = ignore
private_key_file= ./ansible.pem
remote_user=ec2-user
[privilege_escalation]
become=True
become_method=sudo
become_user=root
become_ask_pass=False
```

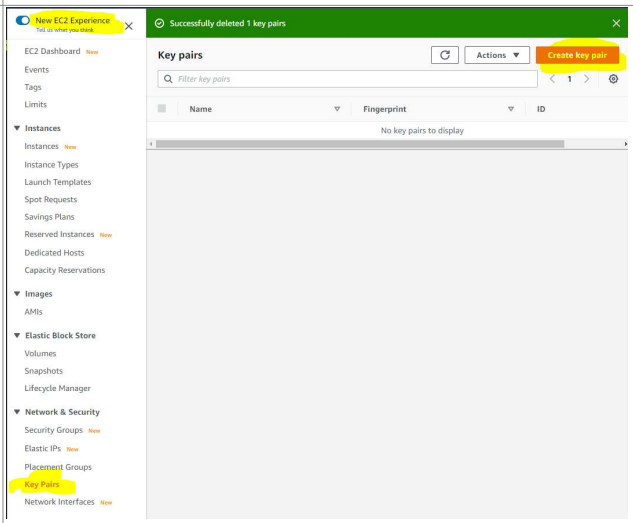
c-) Keywords like *"private_key_file"* which signifies to the aws key pair. When Ansible is going to login to AWS instances to setup K8s via SSH, then it needs the private key file. Also the default remote user of EC2 Instance is "ec2-user".

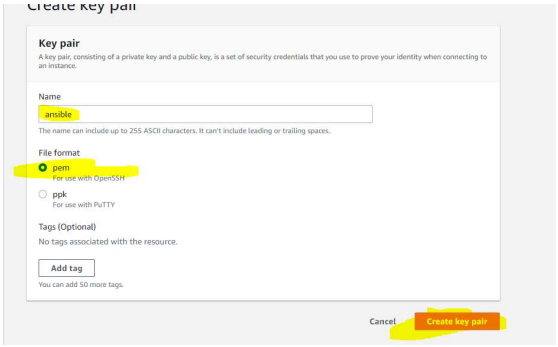
```
ares@master: ~/CapstoneProject
[defaults]
host_key_checking=False
command_warnings=False
deprecation_warnings=False
ask_pass=False
roles_path= ./roles
force_valid_group_names = ignore
private_key_file= ./ansible.pem
remote_user=ec2-user

[privilege_escalation]
become=True
become_method=sudo
become_user=root
become_ask_pass=False
```

3-Creating AWS Key-pair & putting it in the Workspace :

Now we will need to create Key-Pair and download the .pem file where we will be using it to connect to AWS and create EC2 Instances.
a-) Login to you AWS Account and Click EC2.
b-) Click Key pairs on the left panel and click Create key pair.
c-) Name the keypair as *"ansible"*
d-) Click "Create key pair" button on the right bottom corner
e-) Safe ansible.pem file inside main directory called *"CapstoneProject"*.





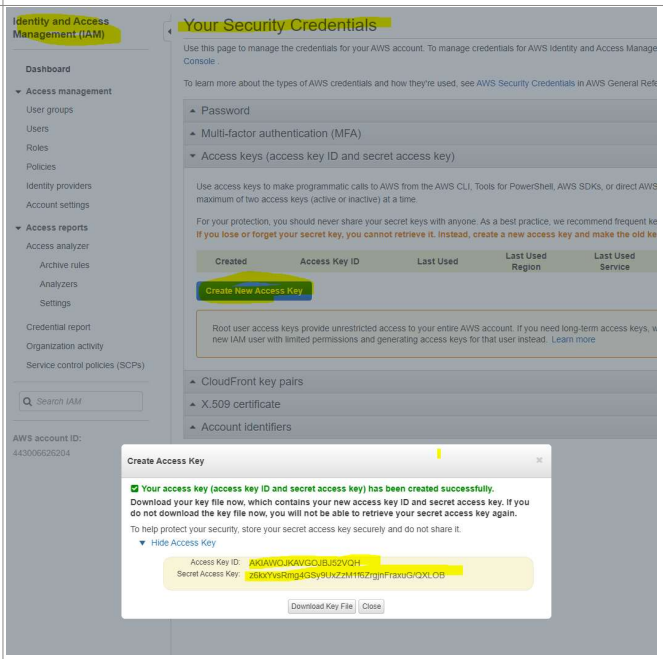
f-) Change the file permission so we can read it.

Terminal Commands:
$ sudo chmod 400 ansible.pem => File permission 400 will only give *"read"* access to the file.

```
ares@master:~/CapstoneProject$ sudo chmod 400 ansible.pem
ares@master:~/CapstoneProject$ ls -la
total 40
drwxrwxr-x  5 ares ares 4096 Jul 11 11:16 .
drwxr-xr-x 20 ares ares 4096 Jul 11 10:59 ..
-rw-r--r--  1 root root  300 Jul 11 11:15 ansible.cfg
-r--------  1 root root 1675 Jul 11 11:15 ansible.pem
drwxr-xr-x  2 root root 4096 Jul 11 11:16 AWSCredentials
-rwxr-xr-x  1 root root  679 Jul 11 11:15 cred.yml
-rw-r--r--  1 root root  507 Jul 11 11:15 role-binding.yml
drwxrwxr-x  5 ares ares 4096 Jul 11 11:16 roles
-rw-r--r--  1 root root  426 Jul 11 11:15 setup.yml
drwxr-xr-x  2 root root 4096 Jul 11 11:16 ShellScript
ares@master:~/CapstoneProject$
```

4-Creating Ansible Vault to store the AWS Credentials :

Now, we will need to create AIM Access Key in AWS
a-) Go to your AWS account and create Access Key. To create an access key, click your account name and select *"My Security Credentials"*.
b-) In Security Credentials menu, click *"Access Keys"* and then click *"Create New Access Key"* .
Copy Access Key ID and the Secret Access KEY into your secure folder.
c-) Once we have Access Key ID and Secret Access Key, we can create a secure vault using the following command.



Terminal Commands:
$ sudo ansible-vault create cred.yaml => when enter is pressed, it will ask us to provide vault password.
$ sudo chmod 400 cred.yml => File permission 400 will only give *"read"* access to the file.

access_key: AKIAWOJKAVGOJBJ52VQH
secret_key: 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

```
ares@master: ~/CapstoneProject
access_key: AKIAWOJKAVGOJBJ52VQH
secret_key: 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
~
```

5-Writing Code for ec2 Role :

Now we will need to create a role for ec2 instance.

Tasks YML file :

a-) Go inside CapstoneProject/roles/ec2/tasks/
b-) Edit the file called main.yaml with a text editor (nano, vi or vim).
c-) Copy the following syntax inside main.yml

Terminal Commands:
$ sudo nano main.yml

```
---
# tasks file for ec2
- name: Installing boto & boto3 on local system
  pip:
    name: "{{ item }}"
    state: present
  loop: "{{ python_pkgs }}"
- name: Creating Security Group for K8s Cluster
  ec2_group:
    name: "{{ sg_name }}"
    description: Security Group for allowing all port
    region: "{{ region_name }}"
    aws_access_key: "{{ access_key }}"
    aws_secret_key: "{{ secret_key }}"
```

```
ares@master: ~/CapstoneProject/roles/ec2/tasks
---
# tasks file for ec2
- name: Installing boto & boto3 on local system
  pip:
    name: "{{ item }}"
    state: present
  loop: "{{ python_pkgs }}"

- name: Creating Security Group for K8s Cluster
  ec2_group:
    name: "{{ sg_name }}"
    description: Security Group for allowing all port
```

```
    rules:
    - proto: all
      cidr_ip: 0.0.0.0/0
    rules_egress:
    - proto: all
      cidr_ip: 0.0.0.0/0
- name: Launching three EC2 instances on AWS
  ec2:
    key_name: "{{ keypair }}"
    instance_type: "{{ instance_flavour }}"
    image: "{{ ami_id }}"
    wait: true
    group: "{{ sg_name }}"
    count: 1
    vpc_subnet_id: "{{ subnet_name }}"
    assign_public_ip: yes
    region: "{{ region_name }}"
    state: present
    aws_access_key: "{{ access_key }}"
    aws_secret_key: "{{ secret_key }}"
    instance_tags:
      Name: "{{ item }}"
  register: ec2
  loop: "{{ instance_tag }}"
- name: Add 1st instance to host group ec2_master
  add_host:
    hostname: "{{ ec2.results[0].instances[0].public_ip }}"
    groupname: ec2_master
- name: Add 2nd instance to host group ec2_slave
  add_host:
    hostname: "{{ ec2.results[1].instances[0].public_ip }}"
    groupname: ec2_slave
- name: Add 3rd instance to host group ec2_slave
  add_host:
    hostname: "{{ ec2.results[2].instances[0].public_ip }}"
    groupname: ec2_slave
- name: Wait for SSH to come up
  wait_for:
    host: "{{ ec2.results[2].instances[0].public_dns_name }}"
    port: 22
    state: started
```

d-) Let me explain what this code do:

- First we are using "pip" module to install two packages — boto & boto3, because these packages has the capability to contact to AWS to launch the EC2 instances. I used one variable called *"python_pkgs"* & the value of it is stored in the *"CapstoneProject/roles/ec2/vars/main.yml"* file.
- Next I used *"ec2_group"* module to create Security Group on AWS. Although we can create one strong Security Group for our Instances, but to make things simple I allowed ingress & egress in all the ports. But in real scenario we never do this.
- Next I used *"ec2"* module to launch instance on AWS, & here all the parameters are known to us. Only I want to talk about two parameters — first is *"register"* which will store all the Metadata in a variable called *"ec2"* so that in future we can parse the required information from it. Second is *"loop"* which again using one variable which contains one list. Next using *"item"* keyword we are calling the list values one after another. This going to run ec2 module 3 times with different instance tags, which finally will launch 3 instances.
- Next I used *"add_host"* module which has the capability to create one dynamic inventory while running the playbook. Under this module I used *"hostname"* keyword to tell the values to store in the dynamic host group. Here I used that *"ec2"* variable and do the JSON parsing to find the public IP of 1st instance.
- Finally I run *"wait_for"* module to hold the playbook for few seconds till all the node's SSH service started.

Vars YML file :

a-) Go inside CapstoneProject/roles/ec2/vars/
b-) Edit the file called main.yaml with a text editor (nano, vi or vim).
c-) Copy the following syntax inside main.yml

```
# vars file for ec2
instance_tag:
        - master
        - worker1
        - worker2
python_pkgs:
        - boto
        - boto3
sg_name: Allow_All_SG
region_name: us-east-2
subnet_name: subnet-76d35e1d
ami_id: ami-0443305dabd4be2bc
keypair: ansible
instance_flavour: t2.small
volumesize: 30
```

ares@master: ~/CapstoneProject

```
---
# vars file for ec2
instance_tag:
        - master
        - worker1
        - worker2

python_pkgs:
        - boto
        - boto3

sg_name: Allow_All_SG
region_name: us-east-2
subnet_name: subnet-76d35e1d
ami_id: ami-0443305dabd4be2bc
keypair: ansible
instance_flavour: t2.small
volumesize: 30
```

6-Writing Code for k8s_slave Role :

Now we will need to create a role for k8s_slave instance.

Tasks YML file :
a-) Go inside CapstoneProject/roles/k8s_slave/tasks/
b-) Edit the file called main.yaml with a text editor (nano, vi or vim).
c-) Copy the following syntax inside main.yml

Terminal Commands:
$ sudo vim main.yml

```
---
# tasks file for k8s_slave
- name: Add kubeadm repositories on Slave Node
  yum_repository:
    name: kube
    description: Kubernetes repo
    baseurl: https://packages.cloud.google.com/yum/repos/kubernetes-el7-$basearch
    enabled: 1
    gpgcheck: 1
    gpgkey: https://packages.cloud.google.com/yum/doc/yum-key.gpg https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
- name: Installing Docker & kubeadm on Slave Node
  package:
    name:
      - docker
      - kubeadm
      - kubectl
```

ares@master: ~/CapstoneProject

```
---
# tasks file for k8s_slave
- name: Add kubeadm repositories on Slave Node
  yum_repository:
    name: kube
    description: Kubernetes repo
    baseurl: https://packages.cloud.google.com/yum/repos/kubernetes-el7-$basearch
    enabled: 1
    gpgcheck: 1
    gpgkey: https://packages.cloud.google.com/yum/doc/yum-key.gpg https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg

- name: Installing Docker & kubeadm on Slave Node
  package:
    name:
      - docker
      - kubeadm
      - kubectl
      - kubelet
      - iproute-tc
    state: present
```

```
      - kubelet
      - iproute-tc
    state: present
- name: Staring & enabling Docker & kubelet on Slave Node
  service:
    name: "{{ item }}"
    state: started
    enabled: yes
  loop: "{{ service_names }}"
- name: Updating Docker cgroup on Slave Node
  copy:
    dest: /etc/docker/daemon.json
    content: |
      {
      "exec-opts": ["native.cgroupdriver=systemd"]
      }
- name: Restart Docker on Slave Node
  service:
    name: docker
    state: restarted
- name: Updating IP tables on Slave Node
  copy:
    dest: /etc/sysctl.d/k8s.conf
    content: |
      net.bridge.bridge-nf-call-ip6tables = 1
      net.bridge.bridge-nf-call-iptables = 1
- name: Reloading sysctl on Slave Node
  command: sysctl --system
- name: Joining the master node
  command: "{{ hostvars[groups['ec2_master'][0]]['token']['stdout'] }}"
- name: Cleaning Caches on RAM
  shell: echo 3 > /proc/sys/vm/drop_caches

# Installing HTTPD to check hostname in Slave node
- name: Installing HTTPD in Slave Node
  shell:
    cmd: |
      sudo yum update -y
      sudo amazon-linux-extras install -y lamp-mariadb10.2-php7.2 php7.2
      sudo yum install -y httpd mariadb-server
      sudo systemctl start httpd
      sudo systemctl enable httpd
      echo $HOSTNAME > /var/www/html/index.html
```

d-) Let me explain what this syntax do:

- Till docker service restart task, this file is exactly same like the previous file of *"k8s_master"* role. On slave node we don't need to initialize the cluster & also we don't need to setup kubectl. Rest of them we need to do because in slave node also we need install *"kubeadm"* command & Docker as container engine.
- Next I used *"copy"* module to create one configuration file called *"/etc/sysctl.d/k8s.conf"* which will allow the slave node to enabled certain networking rules. Next to enable the rules we need to reload the *"sysctl"* and for that I used *"command"* module.

Now here comes the most interesting part...

- In the beginning of the setup process, we used *"token"* variable on our previous role to store the token command for the slave node to join the cluster. Now each role has their own separate namespaces to store the variables. So we need to go to the namespace of previous hostgroup that we created dynamically.
- For that we use *"hostvars"* keyword and inside it we call the *"ec2_master"* hostgroup. Next in this host group we can have multiple hosts (nodes). To pick the 1st host we use "[0]" option. That means finally *"hostvars[groups['ec2_master'][0]]"* option is calling the namespace of the master node.
- Next using *"['token']['stdout']"* we just parsed the command that we can use in slave to join the master node.
- Finally using *"command"* module I installed HTTPD on the Master Node so I can configure AWS Classic Load Balancer.

Vars YML file :

a-) Go inside CapstoneProject/roles/k8s_slave/vars/
b-) Edit the file called main.yaml with a text editor (nano, vi or vim).
c-) Copy the following syntax inside main.yml

Terminal Commands:
$ sudo vim main.yml

```
---
# vars file for k8s_slave
service_names:
 - "docker"
 - "kubelet"
```

7-Writing Code for k8s_master Role :

Now we will need to create a role for k8s_master instance.

Tasks YML file :

a-) Go inside CapstoneProject/roles/k8s_master/tasks/
b-) Edit the file called main.yaml with a text editor (nano, vi or vim).
c-) Copy the following syntax inside main.yml

```
---
# tasks file for k8s_master
- name: Add kubeadm repositories on Master Node
  yum_repository:
    name: kube
    description: Kubernetes repo
    baseurl: https://packages.cloud.google.com/yum/repos/kubernetes-el7-$basearch
    enabled: 1
    gpgcheck: 1
    gpgkey: https://packages.cloud.google.com/yum/doc/yum-
key.gpg https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
- name: Installing Docker & kubeadm on Master Node
  package:
    name:
      - docker
      - kubeadm
      - kubectl
      - kubelet
      - iproute-tc
      - git
      - httpd-tools
    state: present
- name: Staring & enabling Docker & kubelet on Master Node
  #command: systemctl start docker
  service:
    name: "{{ item }}"
    state: started
    enabled: yes
  loop: "{{ service_names }}"
- name: Pulling the images of k8s master
  command: kubeadm config images pull
- name: Updating Docker cgroup on Master Node
  copy:
    dest: /etc/docker/daemon.json
    content: |
```

```
      {
      "exec-opts": ["native.cgroupdriver=systemd"]
      }
- name: Restart docker on Master Node
  service:
    name: docker
    state: restarted
- name: Initializing k8s cluster
  command: kubeadm init --pod-network-cidr=10.244.0.0/16 --ignore-preflight-errors=NumCPU --ignore-
preflight-errors=Mem
- name: Setting up kubectl on Master Node
  shell:
    cmd: |
      mkdir -p $HOME/.kube
      sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
      sudo chown $(id -u):$(id -g) $HOME/.kube/config
- name: Deploying Flannel on Master Node
  command: kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-
flannel.yml
- name: Creating token for Slave
  command: kubeadm token create  --print-join-command
  register: token
- name: Cleaning Caches on RAM
  shell: echo 3 > /proc/sys/vm/drop_caches
# Installing helm on Master node
#- name: Installing helm on Master Node
#  shell:
#    cmd: |
#     curl https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 > get_helm.sh
#     chmod 700 get_helm.sh
#     ./get_helm.sh
# this will configure RBAC for a new user with permissions to create, list, get, update, and delete pod
s
- name: Configuring RBAC (role based access control) on Master Node
  shell:
    cmd: |
      openssl genrsa -out simplilearn.key 2048
      openssl req -new -key simplilearn.key -out simplilearn.csr -subj "/CN=simplilearn/O=Capstone"
      openssl x509 -req -in simplilearn.csr -CA /etc/kubernetes/pki/ca.crt -
CAkey /etc/kubernetes/pki/ca.key -CAcreateserial -out simplilearn.crt -days 500
      kubectl config set-credentials simplilearn  --client-certificate=simplilearn.crt --client-
key=simplilearn.key
# Installing HTTPD to check hostname in Master node
- name: Installing HTTPD in Master Node
  shell:
    cmd: |
      sudo yum update -y
      sudo amazon-linux-extras install -y lamp-mariadb10.2-php7.2 php7.2
      sudo yum install -y httpd mariadb-server
      sudo systemctl start httpd
      sudo systemctl enable httpd
      echo $HOSTNAME > /var/www/html/index.html
```

```
    dest: /etc/docker/daemon.json
    content: |
      {
      "exec-opts": ["native.cgroupdriver=systemd"]
      }
- name: Restart docker on Master Node
  service:
    name: docker
    state: restarted

- name: Initializing k8s cluster
  command: kubeadm init --pod-network-cidr=10.244.0.0/16 --ignore-preflight-errors=NumCPU --ignore-preflight-errors=Mem

- name: Setting up kubectl on Master Node
  shell:
    cmd: |
      mkdir -p $HOME/.kube
      sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
      sudo chown $(id -u):$(id -g) $HOME/.kube/config

- name: Deploying Flannel on Master Node
  command: kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

- name: Creating token for Slave
  command: kubeadm token create  --print-join-command
  register: token

- name: Cleaning Caches on RAM
  shell: echo 3 > /proc/sys/vm/drop_caches

# Installing helm on Master node
#- name: Installing helm on Master Node
#  shell:
#    cmd: |
#     curl https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 > get_helm.sh
#     chmod 700 get_helm.sh
#     ./get_helm.sh

# this will configure RBAC for a new user with permissions to create, list, get, update, and delete pods
- name: Configuring RBAC (role based access control) on Master Node
  shell:
    cmd: |
      openssl genrsa -out simplilearn.key 2048
      openssl req -new -key simplilearn.key -out simplilearn.csr -subj "/CN=simplilearn/O=Capstone"
      openssl x509 -req -in simplilearn.csr -CA /etc/kubernetes/pki/ca.crt -CAkey /etc/kubernetes/pki/ca.key -CAcreateserial -out simplilearn.crt -days 500
      kubectl config set-credentials simplilearn  --client-certificate=simplilearn.crt --client-key=simplilearn.key

# Installing HTTPD to check hostname in Master node
- name: Installing HTTPD in Master Node
  shell:
    cmd: |
      sudo yum update -y
      sudo amazon-linux-extras install -y lamp-mariadb10.2-php7.2 php7.2
      sudo yum install -y httpd mariadb-server
      sudo systemctl start httpd
      sudo systemctl enable httpd
      echo $HOSTNAME > /var/www/html/index.html
```

d-) Let me explain what this syntax do:

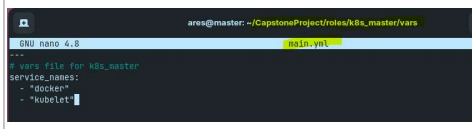
- Here we need to install kubeadm program on our master node to setup K8s cluster. So, for that I'm adding the yum repository provided by K8s community. Here as I'm using AWS Linux 2 for all the instances so we don't need to configure repository for docker cli.
- Next using *"package"* module we are installing *"Docker", "Kubeadm", "Kubectl", "Kubelet","iproute-tc", and "git"* package on our Master Instance.
- Next I used *"service"* module to start the docker and kubelet service. Here again I used the loop on the list called *"service_names"* to run the same module twice.
- Next I used *"command"* module to run one kubeadm command which will pull all the Docker Images required to run Kubernetes Cluster. Here in Ansible we don't have any module to run *"kubeadm"* command, that's why I'm using *"command"* module.
- Next we need to change our Docker default cgroup to *"systemd"*, otherwise kubeadm won't be able to setup K8s cluster. To do that at first using *"copy"* module we are creating one file *"/etc/docker/daemon.json"* and putting some content in it. Next again using *"service"* module we are restarting docker to change the cgroup.
- Next using *"command"* module we are initializing the cluster & then using *"shell"* module we are setting up *"kubectl"* command on our Master Node.
- Next using *"command"* module I deployed Flannel on the Kubernetes Cluster so that it create the overlay network setup and crease user to have access to the cluster.
- RBAC role was created using git repository and *"simplilearn"* user was created to have access to limited cluster resources.
- Also the 2nd *"command"* module is used to get the token for the slave node to join the cluster. Using *"register"* I stored the output of 2nd *"command"* module in a variable called *"token"*. Now this token variable contain the command that we need to run on slave node, so that it joins the master node.
- I used *"shell"* module to clean the buffer cache on my master node, because while doing the setup it is going to create lots of temporary data on RAM.
- Finally using *"command"* module I installed HTTPD on the Master Node so I can configure AWS Classic Load Balancer.

Vars YML file :

a-) Go inside CapstoneProject/roles/k8s_master/vars/
b-) Edit the file called main.yaml with a text editor (nano, vi or vim).
c-) Copy the following syntax inside main.ym

Terminal Commands:
$ sudo vim main.yml

```
# vars file for k8s_master
service_names:
  - "docker"
  - "kubelet"
```



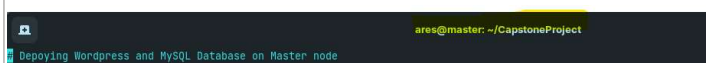
Terminal Commands:
$ kubectl --user=simplilearn -n simplilearn-namespace get pods -o wide => this will let user called simplilearn to list all pods in namespace called simplilearn-namespace

8-Writing Deployment for k8s_master Role :

Now we will need to create a role called ETCD to take a snapshot of the Kubernetes cluster in Master Node.

Tasks YML file :

a-) Go inside CapstoneProject/roles/deployment/tasks/
b-) Edit the file called main.yaml with a text editor (nano, vi or vim).
c-) Copy the following syntax inside main.ym



```
# Depoying Wordpress and MySQL Database on Master node
- name: Depoying Wordpress and MySQL Database on Master node
  shell:
    cmd: |
     git clone https://github.com/0ktay3min/CapstoneProject.git
     cd CapstoneProject/simplilearn
     kubectl apply -k ./


# this will take a snapshot of the Kubernetes cluster
- name: Installing ETCD in Master Node
  shell:
    cmd: |
     sudo wget https://github.com/etcd-io/etcd/releases/download/v3.3.12/etcd-v3.3.12-linux-amd64.tar.gz
     sudo tar xvf etcd-v3.3.12-linux-amd64.tar.gz
     sudo mv etcd-v3.3.12-linux-amd64/etcd* /usr/local/bin/
```

```
- name: Depoying Wordpress and MySQL Database on Master node
  shell:
    cmd: |
     git clone https://github.com/0ktay3min/CapstoneProject.git
     cd CapstoneProject/simplilearn
     kubectl apply -k ./


# this will take a snapshot of the Kubernetes cluster
- name: Installing ETCD in Master Node
  shell:
    cmd: |
     sudo wget https://github.com/etcd-io/etcd/releases/download/v3.3.12/etcd-v3.3.12-linux-amd64.tar.gz
     sudo tar xvf etcd-v3.3.12-linux-amd64.tar.gz
     sudo mv etcd-v3.3.12-linux-amd64/etcd* /usr/local/bin/
```

d-) Let me explain what this code do:
Creating a snapshot of the cluster is crucial, incase if something goes wrong during runtime.
To see the created backup, run the command below.

- I used *"shell"* module to clone my Git Repository called CapstoneProject. Once repository was cloned, ansible ran the following command to deploy WordPress Deployment and MySQL Database.
- Finally using *"shell"* module I installed ETCD on the Master Node so I can take a snapshot of my AWS EC2 Kubernetes Cluster.

9- Creating Setup file:

Now we will need to create the "setup.yml" file which we need to run to create this entire infrastructure on AWS.

a-) Go inside CapstoneProject/
b-) Create a file called setup.yml
c-) Copy the following syntax inside setup.yml

Terminal Commands:
$ sudo vim setup.yml

```
- hosts: localhost
  connection: local
  gather_facts: no
  vars_files:
        - cred.yml
  tasks:
    - name: Running EC2 Role
      include_role:
        name: ec2
- hosts: ec2_master
  gather_facts: no
  tasks:
    - name: Running K8s Master Role
      include_role:
        name: k8s_master
- hosts: ec2_slave
  gather_facts: no
  tasks:
    - name: Running K8s Slave Role
      include_role:
        name: k8s_slave
- hosts: ec2_master
  gather_facts: no
  tasks:
    - name: Running WordPress Deployment, MySQL Database and ETCD on Master Role
      include_role:
        name: deployment
```

d-) Let me explain what this code do.

- Here as you can see, we are running the first "EC2" role on out localhost by contacting AWS API from our localhost.
- By using *"vars_files"*, I declared *cered.yml* file so that *"ec2"* role can access it.
- Our next two steps, I'm running *"k8s_master"*, *"k8s_slave"* and *"deployment"* roles on *"ec2_master"* and *"ec2_slave"* on dynamic hostgroup respectively.

10- Running Ansible Playbook

In this step, we will run ansible playbook called "*setup.yml*" we created.

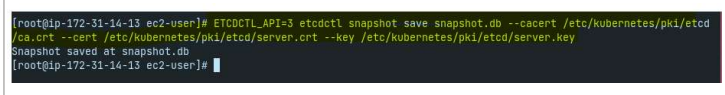
a-) Go inside CapstoneProject/
b-) Run the following command.

Terminal Commands:
$ sudo ansible-playbook setup.yml --ask-vault-pass => By running this command, Ansible will start creating the AWS EC2 Kubernetes cluster with one Master Node and two worker nodes. You will be prompted to enter Vault pass previously created. In my case, I set the password as " *admin* " .

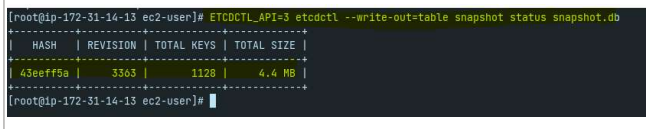
11- Running ETCD on Master Node to Snapshot Kubernetes Cluster

ETCD was already installed during Deployment role. By executing the following command, ETCD backup will be created called "*snapshot.db*".

Terminal command:
$ ETCDCTL_API=3 etcdctl snapshot save snapshot.db --cacert /etc/kubernetes/pki/etcd/ca.crt --cert /etc/kubernetes/pki/etcd/server.crt --key /etc/kubernetes/pki/etcd/server.key => This will create a snapshot of the Kubernetes cluster.



$ ETCDCTL_API=3 etcdctl --write-out=table snapshot status snapshot.db => This command will check created snapshot



12- Stress Testing

In order for us to understand if Horizontal Pod Autoscaler is working correctly, we need to increase the workload.
During "k8s_master" role execution, we installed "httpd-tools" by Apache to help us increase the workload.

Terminal command:
$ ab -n 10000 -c 100 3.141.201.17:30001/ => By executing this command, we will increase the workload on the server. Once

workload is over %50, Horizontal Pod Autoscaler will create another pod to distribute the workload evenly. Horizontal Pod Autoscaler will create up to 10 pods when workload is increased. As soon as workload goes below %50, pods will be terminated accordingly.

```
Server Software:        Apache/2.4.10
Server Hostname:        3.141.201.17
Server Port:            30001

Document Path:          /
Document Length:        0 bytes

Concurrency Level:      100
Time taken for tests:   3.868 seconds
Complete requests:      124
Failed requests:        0
Non-2xx responses:      125
Total transferred:      43250 bytes
HTML transferred:       0 bytes
Requests per second:    32.06 [#/sec] (mean)
Time per request:       3119.265 [ms] (mean)
Time per request:       31.193 [ms] (mean, across all concurrent requests)
Transfer rate:          10.92 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    2   1.1      2       5
Processing:    20 1227 667.7   1197    2675
Waiting:       20 1137 623.5   1180    2483
Total:         20 1229 668.3   1198    2678

Percentage of the requests served within a certain time (ms)
  50%   1198
  66%   1489
  75%   1589
  80%   1798
  90%   2268
  95%   2282
  98%   2374
  99%   2392
 100%   2678 (longest request)
[root@ip-172-31-14-13 ec2-user]# ab -n 10000 -c 100 3.141.201.17:30001/
```

$ kubectl get hpa => This will check the status of the Horizontal Pod Autoscaler and show us MINPODS, MAXPODS, TARGETS and REPLICAS

```
[root@ip-172-31-14-13 ec2-user]# kubectl get hpa
NAME                  REFERENCE                TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
frontend-wordpress-hpa   Deployment/wordpress   65%/50%   3         10        10         6m50s
[root@ip-172-31-14-13 ec2-user]#
```

$ kubectl get all => To get all PODS, SERVICES, DEPLOYMENT, REPLICASET and HPA.

```
[root@ip-172-31-14-13 ec2-user]# kubectl get all
NAME                                    READY   STATUS    RESTARTS   AGE
pod/wordpress-5c5c8f9747-44q2h          1/1     Running   0          2m3s
pod/wordpress-5c5c8f9747-n8spf          1/1     Running   0          2m3s
pod/wordpress-5c5c8f9747-rt2r4          1/1     Running   0          2m3s
pod/wordpress-mysql-79b47ccbd6-fd84g    1/1     Running   0          2m2s

NAME                      TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
service/kubernetes        ClusterIP      10.96.0.1       <none>        443/TCP        6m4s
service/wordpress         LoadBalancer   10.97.90.236    <pending>     80:30001/TCP   2m3s
service/wordpress-mysql   ClusterIP      10.107.167.81   <none>        3306/TCP       2m3s

NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/wordpress         3/3     3            3           2m3s
deployment.apps/wordpress-mysql   1/1     1            1           2m3s

NAME                                         DESIRED   CURRENT   READY   AGE
replicaset.apps/wordpress-5c5c8f9747         3         3         3       2m3s
replicaset.apps/wordpress-mysql-79b47ccbd6   1         1         1       2m3s

NAME                                                  REFERENCE                TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
horizontalpodautoscaler.autoscaling/frontend-wordpress-hpa   Deployment/wordpress   2%/50%    3         10        3          2m2s
[root@ip-172-31-14-13 ec2-user]#
```

13- Deployment Files

Deployment will be done automatically by the Ansible. During the configuration using Ansible, we created a role called *"deployment"*. Ansible will run this role and application will be deployed in our Kubernetes cluster.

Deployment YAML Files.

a-) kustomization.yml => this file will ran all .yaml files listed inside one by one.

```
secretGenerator:
- name: mysql-pass
  literals:
  - password=YOUR_PASSWORD
resources:
  - ingress.yaml
  - mysql-deployment.yaml
  - wordpress-deployment.yaml
  - rbac.yaml
  - metricserver.yaml
```

b-) ingress.yml=> this will set Ingress Networking

```
apiVersion: v1
kind: Namespace
metadata:
  name: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
---
# Source: ingress-nginx/templates/controller-serviceaccount.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    helm.sh/chart: ingress-nginx-4.0.1
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 1.0.0
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/component: controller
  name: ingress-nginx
  namespace: ingress-nginx
automountServiceAccountToken: true
---
# Source: ingress-nginx/templates/controller-configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  labels:
    helm.sh/chart: ingress-nginx-4.0.1
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 1.0.0
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/component: controller
  name: ingress-nginx-controller
  namespace: ingress-nginx
data:
---
# Source: ingress-nginx/templates/clusterrole.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    helm.sh/chart: ingress-nginx-4.0.1
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
```

```
    app.kubernetes.io/version: 1.0.0
    app.kubernetes.io/managed-by: Helm
  name: ingress-nginx
rules:
  - apiGroups:
      - ''
    resources:
      - configmaps
      - endpoints
      - nodes
      - pods
      - secrets
    verbs:
      - list
      - watch
  - apiGroups:
      - ''
    resources:
      - nodes
    verbs:
      - get
  - apiGroups:
      - ''
    resources:
      - services
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - networking.k8s.io
    resources:
      - ingresses
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - ''
    resources:
      - events
    verbs:
      - create
      - patch
  - apiGroups:
      - networking.k8s.io
    resources:
      - ingresses/status
    verbs:
      - update
  - apiGroups:
      - networking.k8s.io
    resources:
      - ingressclasses
    verbs:
      - get
      - list
      - watch
---
# Source: ingress-nginx/templates/clusterrolebinding.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  labels:
    helm.sh/chart: ingress-nginx-4.0.1
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 1.0.0
    app.kubernetes.io/managed-by: Helm
  name: ingress-nginx
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: ingress-nginx
subjects:
  - kind: ServiceAccount
    name: ingress-nginx
    namespace: ingress-nginx
---
# Source: ingress-nginx/templates/controller-role.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  labels:
    helm.sh/chart: ingress-nginx-4.0.1
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 1.0.0
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/component: controller
  name: ingress-nginx
  namespace: ingress-nginx
rules:
  - apiGroups:
      - ''
    resources:
      - namespaces
    verbs:
      - get
  - apiGroups:
      - ''
    resources:
      - configmaps
      - pods
      - secrets
      - endpoints
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - ''
    resources:
      - services
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - networking.k8s.io
    resources:
      - ingresses
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - networking.k8s.io
    resources:
      - ingresses/status
    verbs:
      - update
  - apiGroups:
      - networking.k8s.io
    resources:
      - ingressclasses
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - ''
    resources:
      - configmaps
    resourceNames:
      - ingress-controller-leader
    verbs:
      - get
      - update
  - apiGroups:
      - ''
    resources:
      - configmaps
```

```
    verbs:
      - create
  - apiGroups:
      - ''
    resources:
      - events
    verbs:
      - create
      - patch
---
# Source: ingress-nginx/templates/controller-rolebinding.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  labels:
    helm.sh/chart: ingress-nginx-4.0.1
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 1.0.0
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/component: controller
  name: ingress-nginx
  namespace: ingress-nginx
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: ingress-nginx
subjects:
  - kind: ServiceAccount
    name: ingress-nginx
    namespace: ingress-nginx
---
# Source: ingress-nginx/templates/controller-service-webhook.yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    helm.sh/chart: ingress-nginx-4.0.1
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 1.0.0
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/component: controller
  name: ingress-nginx-controller-admission
  namespace: ingress-nginx
spec:
  type: ClusterIP
  ports:
    - name: https-webhook
      port: 443
      targetPort: webhook
      appProtocol: https
  selector:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/component: controller
---
# Source: ingress-nginx/templates/controller-service.yaml
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-backend-protocol: tcp
    service.beta.kubernetes.io/aws-load-balancer-cross-zone-load-balancing-enabled: 'true'
    service.beta.kubernetes.io/aws-load-balancer-type: nlb
  labels:
    helm.sh/chart: ingress-nginx-4.0.1
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 1.0.0
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/component: controller
  name: ingress-nginx-controller
  namespace: ingress-nginx
spec:
  type: LoadBalancer
  externalTrafficPolicy: Local
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: http
      appProtocol: http
    - name: https
      port: 443
      protocol: TCP
      targetPort: https
      appProtocol: https
  selector:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/component: controller
---
# Source: ingress-nginx/templates/controller-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    helm.sh/chart: ingress-nginx-4.0.1
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 1.0.0
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/component: controller
  name: ingress-nginx-controller
  namespace: ingress-nginx
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: ingress-nginx
      app.kubernetes.io/instance: ingress-nginx
      app.kubernetes.io/component: controller
  revisionHistoryLimit: 10
  minReadySeconds: 0
  template:
    metadata:
      labels:
        app.kubernetes.io/name: ingress-nginx
        app.kubernetes.io/instance: ingress-nginx
        app.kubernetes.io/component: controller
    spec:
      dnsPolicy: ClusterFirst
      containers:
        - name: controller
          image: k8s.gcr.io/ingress-nginx/controller:v1.0.0
@sha256:0851b34f69f69352bf168e6ccf30e1e20714a264ab1ecd1933e4d8c0fc3215c6
          imagePullPolicy: IfNotPresent
          lifecycle:
            preStop:
              exec:
                command:
                  - /wait-shutdown
          args:
            - /nginx-ingress-controller
            - --publish-service=$(POD_NAMESPACE)/ingress-nginx-controller
            - --election-id=ingress-controller-leader
            - --controller-class=k8s.io/ingress-nginx
            - --configmap=$(POD_NAMESPACE)/ingress-nginx-controller
            - --validating-webhook=:8443
            - --validating-webhook-certificate=/usr/local/certificates/cert
            - --validating-webhook-key=/usr/local/certificates/key
          securityContext:
            capabilities:
              drop:
                - ALL
              add:
                - NET_BIND_SERVICE
            runAsUser: 101
            allowPrivilegeEscalation: true
          env:
            - name: POD_NAME
              valueFrom:
                fieldRef:
```

```
                fieldPath: metadata.name
          - name: POD_NAMESPACE
            valueFrom:
              fieldRef:
                fieldPath: metadata.namespace
          - name: LD_PRELOAD
            value: /usr/local/lib/libmimalloc.so
        livenessProbe:
          failureThreshold: 5
          httpGet:
            path: /healthz
            port: 10254
            scheme: HTTP
          initialDelaySeconds: 10
          periodSeconds: 10
          successThreshold: 1
          timeoutSeconds: 1
        readinessProbe:
          failureThreshold: 3
          httpGet:
            path: /healthz
            port: 10254
            scheme: HTTP
          initialDelaySeconds: 10
          periodSeconds: 10
          successThreshold: 1
          timeoutSeconds: 1
        ports:
          - name: http
            containerPort: 80
            protocol: TCP
          - name: https
            containerPort: 443
            protocol: TCP
          - name: webhook
            containerPort: 8443
            protocol: TCP
        volumeMounts:
          - name: webhook-cert
            mountPath: /usr/local/certificates/
            readOnly: true
        resources:
          requests:
            cpu: 100m
            memory: 90Mi
      nodeSelector:
        kubernetes.io/os: linux
      serviceAccountName: ingress-nginx
      terminationGracePeriodSeconds: 300
      volumes:
        - name: webhook-cert
          secret:
            secretName: ingress-nginx-admission
---
# Source: ingress-nginx/templates/controller-ingressclass.yaml
# We don't support namespaced ingressClass yet
# So a ClusterRole and a ClusterRoleBinding is required
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  labels:
    helm.sh/chart: ingress-nginx-4.0.1
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 1.0.0
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/component: controller
  name: nginx
  namespace: ingress-nginx
spec:
  controller: k8s.io/ingress-nginx
---
# Source: ingress-nginx/templates/admission-webhooks/validating-webhook.yaml
# before changing this value, check the required kubernetes version
# https://kubernetes.io/docs/reference/access-authn-authz/extensible-admission-controllers/#prerequisites
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
metadata:
  labels:
    helm.sh/chart: ingress-nginx-4.0.1
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 1.0.0
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/component: admission-webhook
  name: ingress-nginx-admission
webhooks:
  - name: validate.nginx.ingress.kubernetes.io
    matchPolicy: Equivalent
    rules:
      - apiGroups:
          - networking.k8s.io
        apiVersions:
          - v1
        operations:
          - CREATE
          - UPDATE
        resources:
          - ingresses
    failurePolicy: Fail
    sideEffects: None
    admissionReviewVersions:
      - v1
    clientConfig:
      service:
        namespace: ingress-nginx
        name: ingress-nginx-controller-admission
        path: /networking/v1/ingresses
---
# Source: ingress-nginx/templates/admission-webhooks/job-patch/serviceaccount.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: ingress-nginx-admission
  namespace: ingress-nginx
  annotations:
    helm.sh/hook: pre-install,pre-upgrade,post-install,post-upgrade
    helm.sh/hook-delete-policy: before-hook-creation,hook-succeeded
  labels:
    helm.sh/chart: ingress-nginx-4.0.1
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 1.0.0
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/component: admission-webhook
---
# Source: ingress-nginx/templates/admission-webhooks/job-patch/clusterrole.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: ingress-nginx-admission
  annotations:
    helm.sh/hook: pre-install,pre-upgrade,post-install,post-upgrade
    helm.sh/hook-delete-policy: before-hook-creation,hook-succeeded
  labels:
    helm.sh/chart: ingress-nginx-4.0.1
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 1.0.0
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/component: admission-webhook
rules:
  - apiGroups:
      - admissionregistration.k8s.io
    resources:
      - validatingwebhookconfigurations
    verbs:
      - get
      - update
---
# Source: ingress-nginx/templates/admission-webhooks/job-patch/clusterrolebinding.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
```

```yaml
metadata:
  name: ingress-nginx-admission
  annotations:
    helm.sh/hook: pre-install,pre-upgrade,post-install,post-upgrade
    helm.sh/hook-delete-policy: before-hook-creation,hook-succeeded
  labels:
    helm.sh/chart: ingress-nginx-4.0.1
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 1.0.0
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/component: admission-webhook
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: ingress-nginx-admission
subjects:
  - kind: ServiceAccount
    name: ingress-nginx-admission
    namespace: ingress-nginx
---
# Source: ingress-nginx/templates/admission-webhooks/job-patch/role.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: ingress-nginx-admission
  namespace: ingress-nginx
  annotations:
    helm.sh/hook: pre-install,pre-upgrade,post-install,post-upgrade
    helm.sh/hook-delete-policy: before-hook-creation,hook-succeeded
  labels:
    helm.sh/chart: ingress-nginx-4.0.1
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 1.0.0
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/component: admission-webhook
rules:
  - apiGroups:
      - ''
    resources:
      - secrets
    verbs:
      - get
      - create
---
# Source: ingress-nginx/templates/admission-webhooks/job-patch/rolebinding.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: ingress-nginx-admission
  namespace: ingress-nginx
  annotations:
    helm.sh/hook: pre-install,pre-upgrade,post-install,post-upgrade
    helm.sh/hook-delete-policy: before-hook-creation,hook-succeeded
  labels:
    helm.sh/chart: ingress-nginx-4.0.1
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 1.0.0
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/component: admission-webhook
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: ingress-nginx-admission
subjects:
  - kind: ServiceAccount
    name: ingress-nginx-admission
    namespace: ingress-nginx
---
# Source: ingress-nginx/templates/admission-webhooks/job-patch/job-createSecret.yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: ingress-nginx-admission-create
  namespace: ingress-nginx
  annotations:
    helm.sh/hook: pre-install,pre-upgrade
    helm.sh/hook-delete-policy: before-hook-creation,hook-succeeded
  labels:
    helm.sh/chart: ingress-nginx-4.0.1
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 1.0.0
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/component: admission-webhook
spec:
  template:
    metadata:
      name: ingress-nginx-admission-create
      labels:
        helm.sh/chart: ingress-nginx-4.0.1
        app.kubernetes.io/name: ingress-nginx
        app.kubernetes.io/instance: ingress-nginx
        app.kubernetes.io/version: 1.0.0
        app.kubernetes.io/managed-by: Helm
        app.kubernetes.io/component: admission-webhook
    spec:
      containers:
        - name: create
          image: k8s.gcr.io/ingress-nginx/kube-webhook-certgen:v1.0
@sha256:f3b6b39a6062328c095337b4cadcefd1612348fdd5190b1dcbcb9b9e90bd8068
          imagePullPolicy: IfNotPresent
          args:
            - create
            - --host=ingress-nginx-controller-admission,ingress-nginx-controller-admission.
$(POD_NAMESPACE).svc
            - --namespace=$(POD_NAMESPACE)
            - --secret-name=ingress-nginx-admission
          env:
            - name: POD_NAMESPACE
              valueFrom:
                fieldRef:
                  fieldPath: metadata.namespace
      restartPolicy: OnFailure
      serviceAccountName: ingress-nginx-admission
      nodeSelector:
        kubernetes.io/os: linux
      securityContext:
        runAsNonRoot: true
        runAsUser: 2000
---
# Source: ingress-nginx/templates/admission-webhooks/job-patch/job-patchWebhook.yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: ingress-nginx-admission-patch
  namespace: ingress-nginx
  annotations:
    helm.sh/hook: post-install,post-upgrade
    helm.sh/hook-delete-policy: before-hook-creation,hook-succeeded
  labels:
    helm.sh/chart: ingress-nginx-4.0.1
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 1.0.0
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/component: admission-webhook
spec:
  template:
    metadata:
      name: ingress-nginx-admission-patch
      labels:
        helm.sh/chart: ingress-nginx-4.0.1
        app.kubernetes.io/name: ingress-nginx
        app.kubernetes.io/instance: ingress-nginx
        app.kubernetes.io/version: 1.0.0
        app.kubernetes.io/managed-by: Helm
        app.kubernetes.io/component: admission-webhook
    spec:
      containers:
        - name: patch
```

```
          image: k8s.gcr.io/ingress-nginx/kube-webhook-certgen:v1.0
@sha256:f3b6b39a6062328c095337b4cadcefd1612348fdd5190b1dcbcb9b9e90bd8068
          imagePullPolicy: IfNotPresent
          args:
            - patch
            - --webhook-name=ingress-nginx-admission
            - --namespace=$(POD_NAMESPACE)
            - --patch-mutating=false
            - --secret-name=ingress-nginx-admission
            - --patch-failure-policy=Fail
          env:
            - name: POD_NAMESPACE
              valueFrom:
                fieldRef:
                  fieldPath: metadata.namespace
      restartPolicy: OnFailure
      serviceAccountName: ingress-nginx-admission
      nodeSelector:
        kubernetes.io/os: linux
      securityContext:
        runAsNonRoot: true
        runAsUser: 2000
```

c-) mysql-deployment.yaml => this deploy a pod for SQL Database.

```
apiVersion: v1
kind: Service
metadata:
  name: wordpress-mysql
  labels:
    app: wordpress
spec:
  type: ClusterIP
  ports:
    - port: 3306
  selector:
    app: wordpress
    tier: mysql
---
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: manual1
provisioner: kubernetes.io/gce-pd
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: zk1-pv1
spec:
  storageClassName: manual1
  capacity:
    storage: 1Gi
  accessModes:
  - ReadWriteOnce
  hostPath:
    path: /mr/zk1
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pv-claim
  labels:
    app: wordpress
spec:
  storageClassName: manual1
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress-mysql
  labels:
    app: wordpress
spec:
  selector:
    matchLabels:
      app: wordpress
      tier: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: wordpress
        tier: mysql
    spec:
      containers:
      - image: mysql:5.6
        name: mysql
        env:
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-pass
              key: password
        ports:
        - containerPort: 3306
          name: mysql
        volumeMounts:
        - name: mysql-persistent-storage
          mountPath: /var/lib/mysql
      volumes:
      - name: mysql-persistent-storage
        persistentVolumeClaim:
          claimName: mysql-pv-claim
```

c-) rbac.yaml => Role Based Access Control will grant user specific access to the cluster. In our case, we created a user called *"simplilearn"* where this user can list, update, delete and create pods.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: sa-reader
  namespace: default
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: reader-cr
rules:
- verbs: ["create", "list", "delete", "get", "update"]
  resources: ["pods"]
  apiGroups: [""]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-userA-pods-rb
  namespace: default
subjects:
- kind: ServiceAccount
  name: sa-reader
  namespace: default
roleRef:
  kind: ClusterRole
  name: reader-cr
  apiGroup: rbac.authorization.k8s.io
```

d-) wordpress-deployment.yaml => This will create three pods and set Horizontal Pod Autoscaler accordingly.

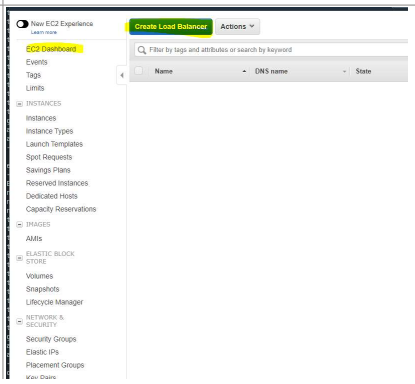
```
apiVersion: networking.k8s.io/v1
```

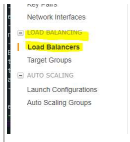
```
kind: Ingress
metadata:
  name: ingress-wordpress
  annotations:
    kubernetes.io/ingress.class: "nginx"
spec:
  rules:
  - http:
      paths:
        - path: /
          pathType: Prefix
          backend:
            service:
              name: wordpress
              port:
                number: 30001
---
apiVersion: v1
kind: Service
metadata:
  name: wordpress
  labels:
    app: wordpress
spec:
  type: LoadBalancer
  ports:
    - port: 80 #default port ofr wordpress
      nodePort: 30001
      name: http
  selector:
    app: wordpress
    tier: frontend
---
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: manual
provisioner: kubernetes.io/gce-pd
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: zk1-pv
spec:
  storageClassName: manual
  capacity:
    storage: 1Gi
  accessModes:
  - ReadWriteOnce
  hostPath:
    path: /mr/zk
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: wp-pv-claim
  labels:
    app: wordpress
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress
  labels:
    app: wordpress
spec:
  replicas: 3
  selector:
    matchLabels:
      app: wordpress
      tier: frontend
  template:
    metadata:
      labels:
        app: wordpress
        tier: frontend
    spec:
      containers:
      - image: wordpress:4.8-apache
        name: wordpress
        env:
        - name: WORDPRESS_DB_HOST
          value: wordpress-mysql
        - name: WORDPRESS_DB_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-pass
              key: password
        ports:
        - containerPort: 80
          name: wordpress
        volumeMounts:
        - name: wordpress-persistent-storage
          mountPath: /var/www/html
        resources:
            limits:
              cpu: 50m
      volumes:
      - name: wordpress-persistent-storage
        persistentVolumeClaim:
          claimName: wp-pv-claim
---
kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v1
metadata:
  name: frontend-wordpress-hpa
spec:
  targetCPUUtilizationPercentage: 50
  minReplicas: 3
  maxReplicas: 10
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: wordpress
```

14- Creating EC2 Load Balancer

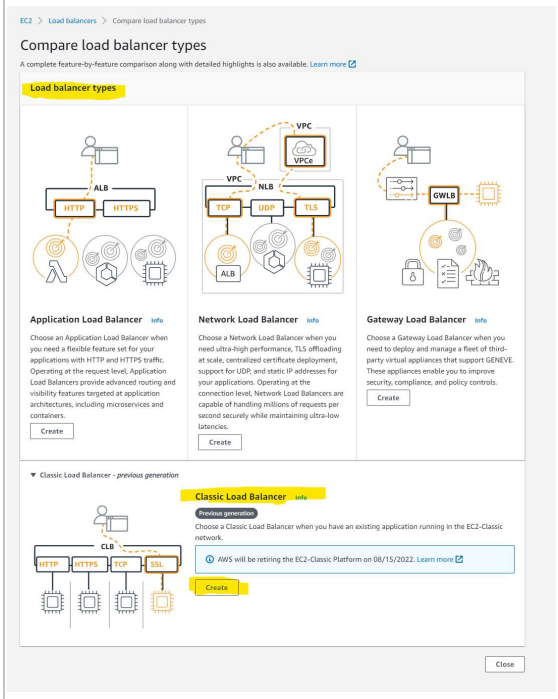
To be able to balance incoming traffic accordingly, I created a Load Balancer inside AWS. This way, any incoming traffic will be balances between nodes inside Kubernetes Cluster.

a-) Click EC2 Dashboard and select *"Load Balancers"*, then click *"Create Load Balancer"*.

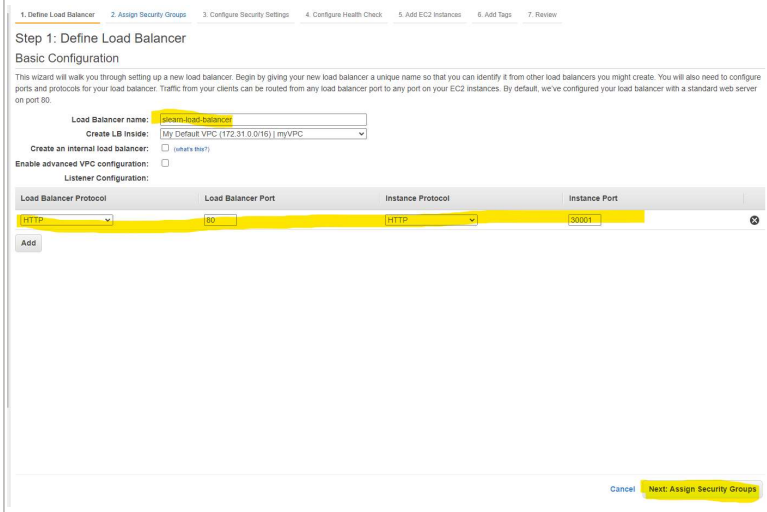




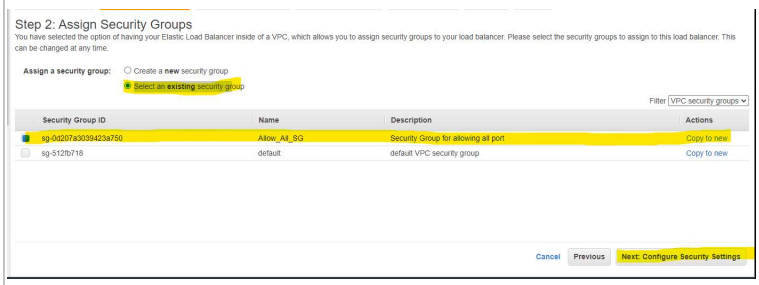
b-) On Load balancer types, select *"Classic Load Balancer"*, then click *"Create"*



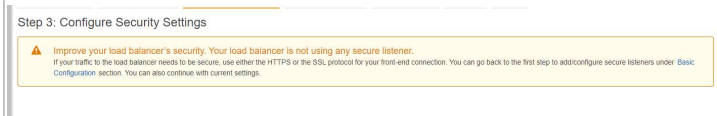
c-) Define a name for your Load Balancer and select the Default VPC. On the listening port side, select port 80 on the Load Balancer side and port 30001 on the Instance Port. Port 30001 was manually defined when Deployment for Wordpress was created. Once port numbers are set correctly, click *"Next: Assign Security Group"*.



d-) On Security Group window, select the existing security group created during EC2 instance creation. In my case, I have created Security Group called "Allow_All_SG". Once security group is selected, press *"Next: Configure Security Settings"*

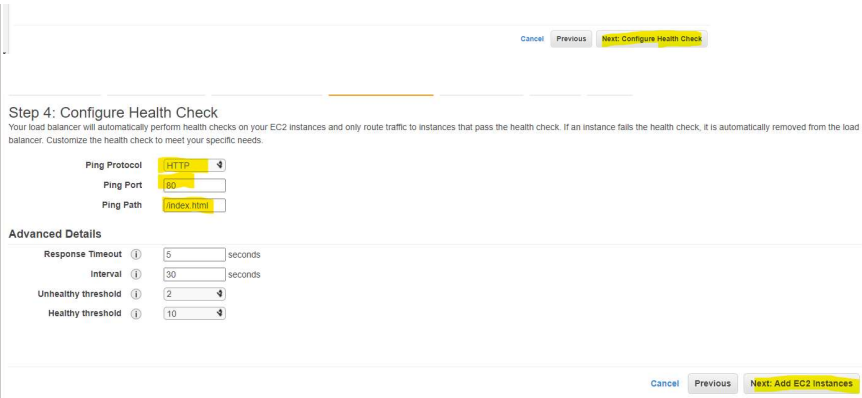


e-) Since we are not setting up HTTPS, we do not need to configure SSL protocols. Click *"Next: Configure Health Check"*.

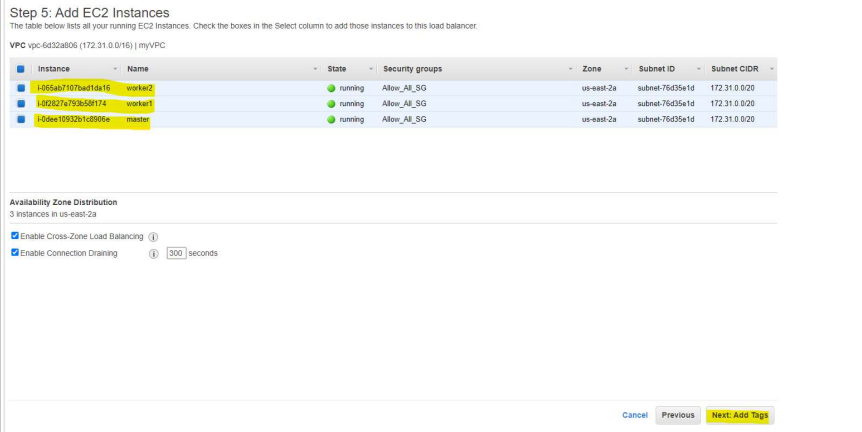


f-) Configuring Health check very crucial for Load Balancer to direct traffic correctly. During k8s_master and k8s_slave role, we installed HTTPD web server on Master and Worker nodes. HTTPD created a file called *"index.html"* inside directory *"/var/www/html/index.html"*. During health check, Load Balancer will ping that /index.html file and will know if instance is up and running. Once Ping Protocol and Ping Port is set correctly, press *"Next: Add EC2 Instances"*.

g-) Select all three instances created, then press *"Next: Add Tags"*.

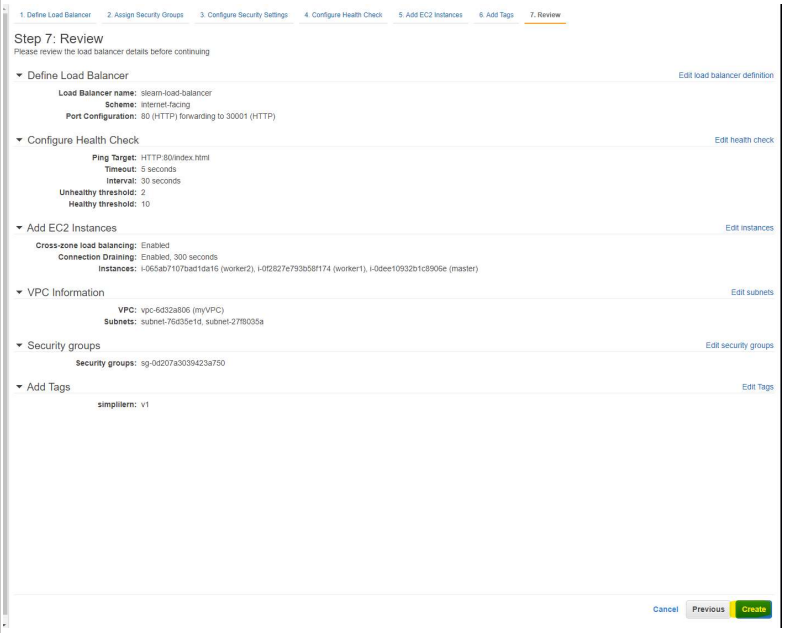


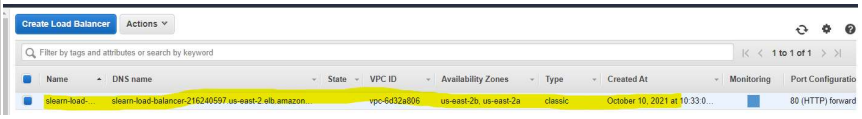
h-) Define a *"Key"* and *"Value"* then press *"Review and Create"*.



i-) Review the Load Balancer Settings, then click *"Create"*.

j-) Load balancer creating may take up to a minute. Once Load Balancer is created, make sure all three instance Status shows *"In service"*.





k-) Click "*Description*" tab on Load Balancer page and copy the DNS name. With this DNS name, you can access your Worpress deployment using any browser.