Crytek-Case

Before I start, I created 3 aws instances with using Terraform tool. main.tf file is following;

```
//This Terraform Template creates 3 Ansible Machines on EC2 Instances
//Ansible Machines will run on Amazon Linux 2 with custom security group
//allowing SSH (22) and HTTP (80) connections from anywhere.
//User needs to select appropriate key name when launching the instance.

terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.0"
    }
  }
}

provider "aws" {
  region = "us-east-1"
#  secret_key = ""
#  access_key = ""
}

variable "tags" {
  default = ["control_node", "node_1", "node_2"]
}

resource "aws_instance" "amazon-linux-2" {
  ami = "ami-0022f774911c1d690"
  instance_type = "t2.medium"
  count = 3
  key_name = "key"
  security_groups = ["ansible-session-3-sec-gr"]
  tags = {
    Name = element(var.tags, count.index)
  }
}

resource "aws_security_group" "tf-sec-gr" {
  name = "ansible-session-3-sec-gr"
  tags = {
    Name = "ansible-session-sec-gr"
  }

  ingress {
    from_port   = 80 #HTTP
    protocol    = "tcp"
    to_port     = 80
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 9090 #Prometheus
    protocol    = "tcp"
    to_port     = 9090
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 9100 #Node_Exporter
    protocol    = "tcp"
    to_port     = 9100
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 3000 #Grafana
    protocol    = "tcp"
    to_port     = 3000
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 3306 #MySQL
    protocol    = "tcp"
```

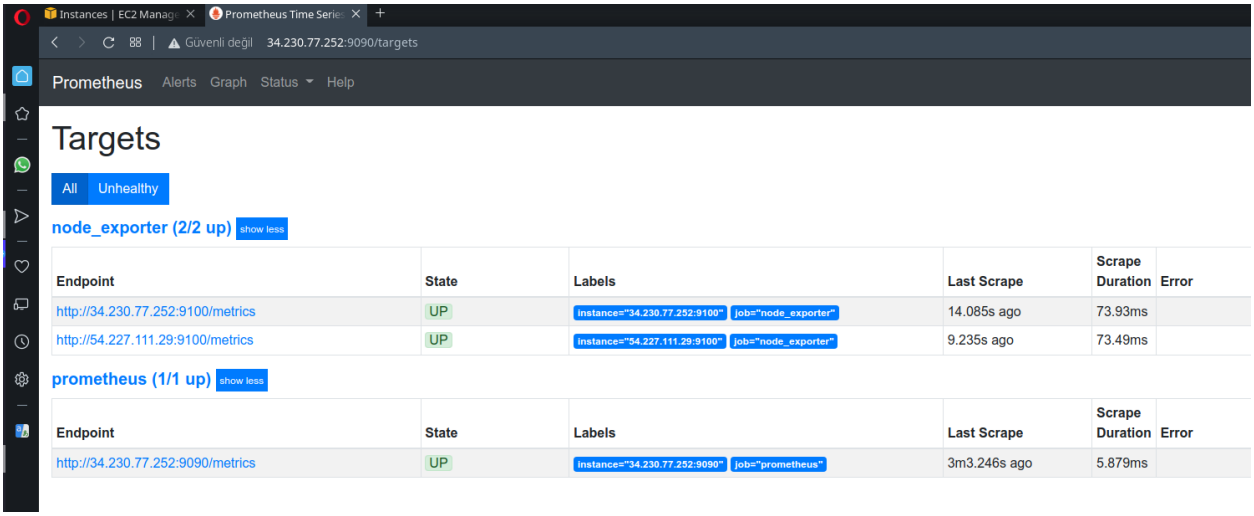
```
    to_port     = 3306
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 22 #SSH Connection
    protocol    = "tcp"
    to_port     = 22
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 82 #Random TCP Port for Phpmyadmin
    protocol    = "tcp"
    to_port     = 82
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    from_port   = 0 #Allows Outbound Traffic
    protocol    = -1
    to_port     = 0
    cidr_blocks = ["0.0.0.0/0"]
  }
}

resource "null_resource" "config" {
  depends_on = [aws_instance.amazon-linux-2[0]]
  connection {
    host = aws_instance.amazon-linux-2[0].public_ip
    type = "ssh"
    user = "ec2-user"
    private_key = file("~/key.pem")
    }

  provisioner "file" {
  source = "./ansible.cfg"
  destination = "/home/ec2-user/ansible.cfg"
}

  provisioner "file" {
    source = "~/key.pem"
    destination = "/home/ec2-user/key.pem"
  }

  provisioner "remote-exec" {
    inline = [
      "sudo hostnamectl set-hostname Control-Node",
      "sudo yum update -y",
      "sudo amazon-linux-extras install ansible2 -y",
      "echo [servers] >> inventory.txt",
      "echo node1  ansible_host=${aws_instance.amazon-linux-2[1].private_ip}  ansible_ssh_private_key_file=~/key.pem ansible_user=ec2-user >>
      "echo node2 ansible_host=${aws_instance.amazon-linux-2[2].private_ip} ansible_ssh_private_key_file=~/key.pem ansible_user=ec2-user >>
      "chmod 400 key.pem"
    ]
  }
}

output "controlnodeip" {
  value = aws_instance.amazon-linux-2[0].public_ip
}
```

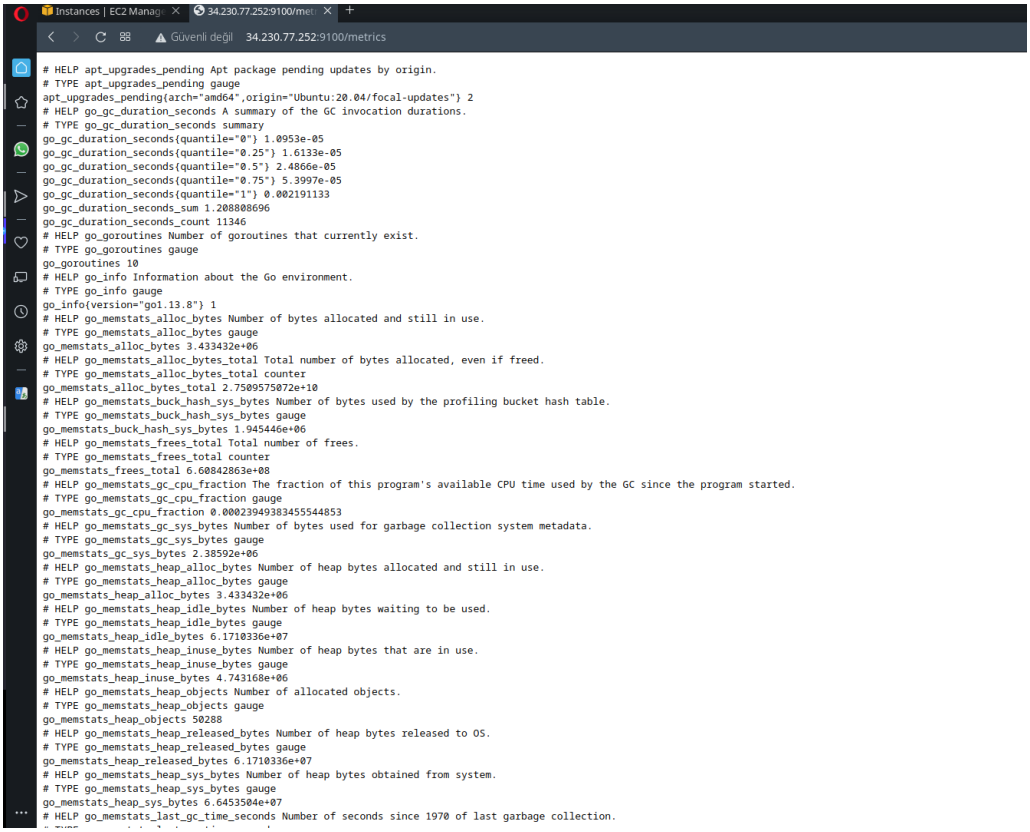
This Terraform file allows me to create 3 aws instances which are control_node,node1 and node2.
The huge advantage of using terraform file is I don't need to connect all of my ansible worker machines to copy my key.pem file, manually create the inventory.txt file and creating ansible.cfg file to home directory. I already created this inventory.txt and ansible.cfg file on my local and the .tf file send these files to my nodes.

The remote-exec feature can be compared to userdata in AWS. After the resource is created, the commands specified under remote-exec "inline" are run. It can be used to preload resources. Here I have changed the name of my control-node host, updated, installed ansible and created an inventory.txt file in each host so that the hosts use their private ip. The purpose of using private ip here is that if the hosts are stopped, public ip addresses will change, while private ip addresses will not change. In this case, inventory.txt will not be a file that needs to be constantly updated.

Using the file provisioner feature of terraform, I copied the files in the locale to all my machines that terraform raised. In this way, I prevented from getting a public key permission denied error.

After Terraform created the hosts, I connected to the control-node using the VSCode Remote SSH Extension. In order to configure the Grafana Dashboard and export the .json file, I first installed prometheus and node exporter on this host. Node exporter will transmit the metrics from prometheus to grafana and the dashboard we have created will give an output according to these metrics. I checked whether it works by typing 9090 port and 9100 that I opened in the terraform file next to the public ip of my host. After pressing the status option on the Prometheus server, I checked whether the node exporters running on both my hosts were up and running from the targets section. It works fine as you can see from the picture. In this case, you can enter grafana and start creating dashboards.

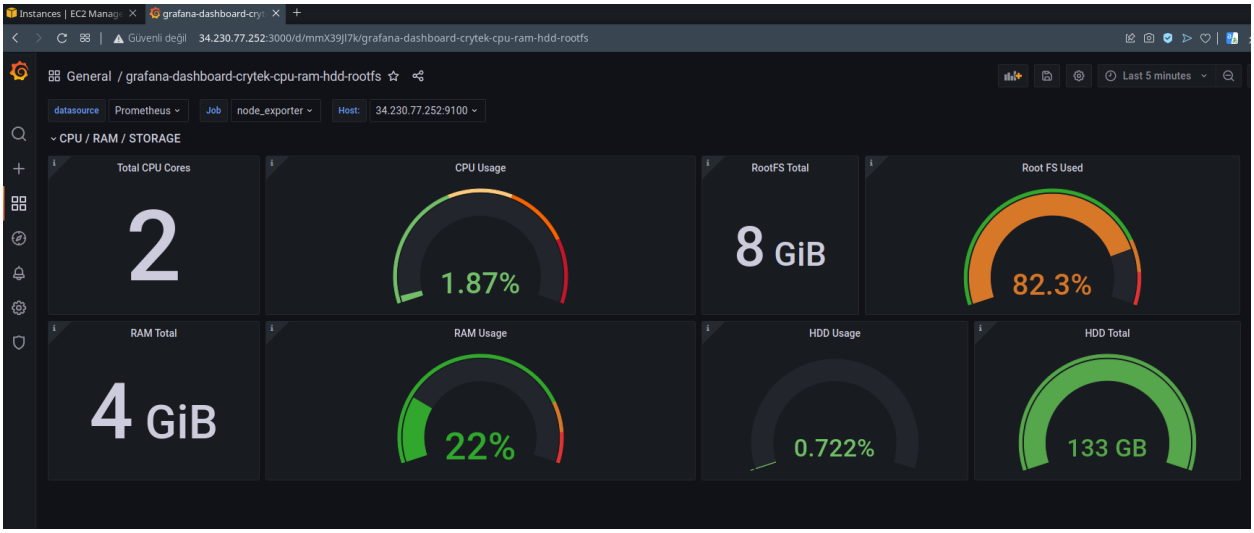




First of all, HDD usages should also appear in the grafana dashboard requested from me. By default on AWS you have no HDD storage when booting an ec2. You must do this manually. For this reason, I manually mounted an HDD to my machines with the

AWS EBS service. I created a mount point by connecting these HDDs to my hosts and mounted them. Then I wrote a query in PromQL so that the grafana dashboard could give the desired data. You can see the queries I wrote below.

I do not attach the .json file that I exported here. A file of about 1000 lines. You can check this file from my github account. I shared the link on the last page of my presentation file.



http://34.230.77.252:3000

http://54.227.111.29:3000

The instances are the same. So output of monitoring is almost exactly the same.

Here are my queries;

CPU Usage Query;

```
(((count(count(node_cpu_seconds_total{instance="$node",job="$job"}) by (cpu))) - avg(sum by (mode)(irate(node_cpu_seconds_total{mode='idle'
```

Total CPU Cores Query;

```
count(count(node_cpu_seconds_total{instance="$node",job="$job"}) by (cpu))
```

RAM Usage Query;

```
((node_memory_MemTotal_bytes{instance="$node",job="$job"} - node_memory_MemFree_bytes{instance="$node",job="$job"}) / (node_memory_MemTotal
```

Total RAM Query;

```
node_memory_MemTotal_bytes{instance="$node",job="$job"}
```

HDD Usage Query;

```
100.0 - 100 * (node_filesystem_avail_bytes{instance=~'$node',device !~'tmpfs',device!~'by-uuid'} / node_filesystem_size_bytes{instance=~'$n
```

Total HDD Query;

```
node_filesystem_avail_bytes{instance=~'$node',device !~'tmpfs',device!~'by-uuid'}
```

And I also create Root File System Total and Usage Queries

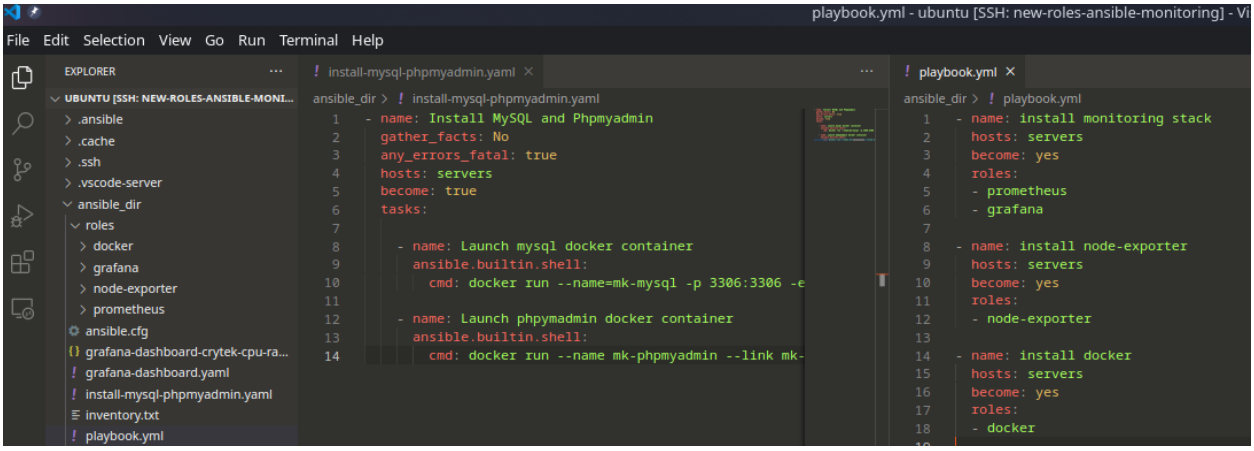
Total RootFS Query;

```
node_filesystem_size_bytes{instance="$node",job="$job",mountpoint="/",fstype!="rootfs"}
```

RootFS Used Query;

```
100 - ((node_filesystem_avail_bytes{instance="$node",job="$job",mountpoint="/",fstype!="rootfs"} * 100) / node_filesystem_size_bytes{instan
```

Dashboard is ready and working smoothly. I exported and started to prepare my playbooks. Programs that must be installed in order to install the desired system within the scope of the task are prometheus, node exporter, grafana and docker. I created these using ansible roles. You can see all my roles under roles on Github. My playbook file that will refer to these roles is as seen in the picture.

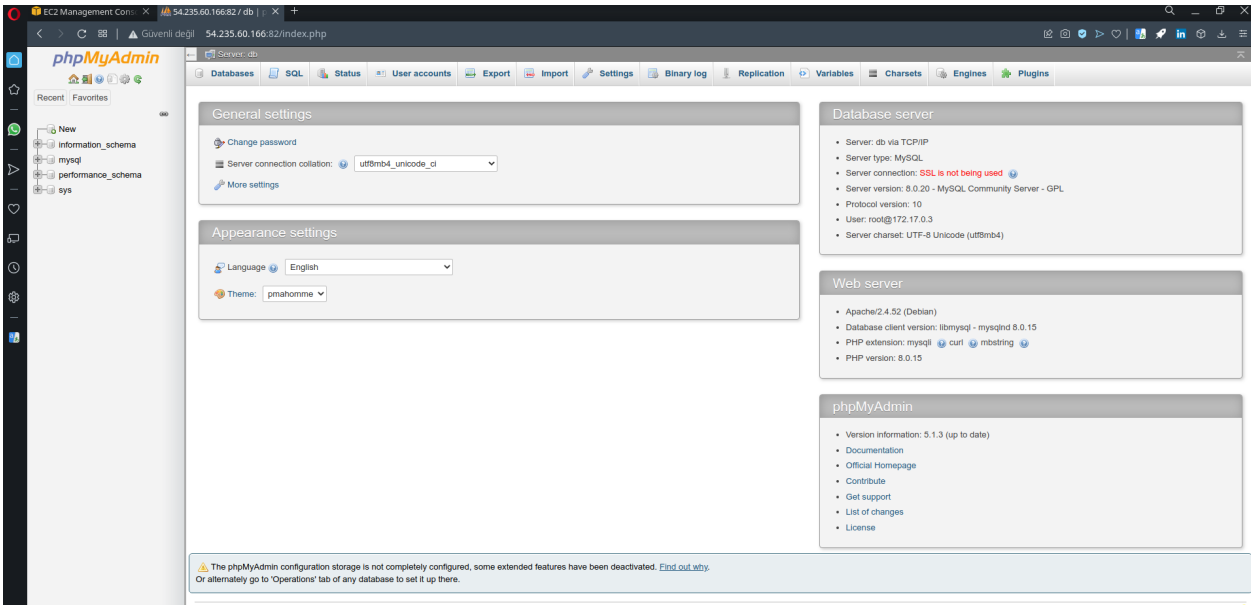


As can be seen from the picture, playbook.yml referring to my roles can be seen. In addition, my yaml file, where I installed mysql and phpmyadmin server, is also visible. I chose to install MySQL and phpmyadmin using a simple docker run command. In this imperative method, I specified the container name, the port to be exported, the password to be used when connecting to the phpmyadmin server and the image to be used in the docker run command.

```
- name: Launch mysql docker container
      ansible.builtin.shell:
        cmd: docker run --name=mk-mysql -p 3306:3306 -e MYSQL_ROOT_PASSWORD=my-secret-pw -d mysql/mysql-server:8.0.20
```

```
- name: Launch phpymadmin docker container
      ansible.builtin.shell:
        cmd: docker run --name mk-phpmyadmin --link mk-mysql:db -p 82:80 -d phpmyadmin/phpmyadmin
```

PHPmyadmin server will broadcast from TCP 82, which I have randomly decided. Besides, when the website is opened using the Phpmyadmin port 82, it asks for a username and password. I have already specified the password in the command where I created the MySQL container. To determine the username, I connected to these containers and created username by running mysql commands in it. Then I logged into the phpmyadmin page as seen below.



grafana-dashboard.yaml

```yaml
- hosts: servers
  tasks:
    - name: Import Grafana Dashboard
      community.grafana.grafana_dashboard:
        grafana_url: http://54.227.111.29:3000
        grafana_api_key: "eyJrIjoiV1Z0NGFWM3BCcU5aWXBmaWNpa2E4Mm05SVZVMV1SeUYiLCJuIjoiYWRtaW4iLCJpZCI6MX0="
        state: present
        commit_message: Updated by ansible
        overwrite: yes
        path: "/home/ubuntu/grafana-dashboard-crytek-cpu-ram-hdd-rootfs-1651962337887.json"

    - name: Import Grafana Dashboard on another instance
      community.grafana.grafana_dashboard:
        grafana_url: http://34.230.77.252:3000/
        grafana_api_key: "eyJrIjoidTlKZVpDWW53MmF4N3BzclJsQzdIRFJGWjVuN3NSdkkiLCJuIjoiYWRtaW4iLCJpZCI6MX0="
        state: present
        commit_message: Updated by ansible
        overwrite: yes
        path: "/home/ubuntu/grafana-dashboard-crytek-cpu-ram-hdd-rootfs-new-34.json"
```

Dashboard was created and exported. By putting this dashboard in a single playbook, I prepared the following yaml file so that it can run every time I run the playbook and import it to the grafana dashboard. The Grafana API Key is under the configuration option on the grafana website.

All the operations done so far are the answers to the first 4 items in the task sent to me.

5) Deploying a web application using docker image.

I deployed https://gist.github.com/79194d8f44ae6a2e0c96c006b58118a7.git from this Github address a index.html. After I cloned, I just copied it to my host and I created a Dockerfile to create an image of it.

```
FROM ubuntu

RUN apt-get update

RUN apt-get install nginx -y

COPY index.html /var/www/html/

EXPOSE 80

CMD [“nginx”,”-g”,”daemon off;”]
```

I cloned a stand-alone index.html file from the github address below. I used this index html file to change an nginx page to be loaded by default. I also did this using dockerfile. First of all, if you look at the Dockerfile, I used the ubuntu base image. If your website is written with flask, then the alpine base image, which takes up very little space and has python dependencies, can be used. I chose ubuntu base image. Then I did the update with the RUN command as I was running ubuntu. Then I ran the command to install nginx in the container that will be created with the RUN command. After these commands run, by default nginx will use its own index.html file located in the /var/www/html directory. But when my index.html file is copied to that directory using COPY, the default nginx index.html file will be overwritten and the website image will be changed. Then expose 80 is a documentation instruction letting the container know we are exposing the standard port 80. CMD provides defaults for an executing container.

After I created the Dockerfile;

```
docker build -t nginxdocker .
```

After this command, I checked all of my docker images inside of my instance with “docker images” command.

```
[ec2-user@ip-172-31-15-56 deneme-test]$ docker images
REPOSITORY                                  TAG          IMAGE ID       CREATED         SIZE
nginxdocker                                 latest       db8d985c1290   8 seconds ago   167MB
```

So the image is created. We can create our container from this image.

```
docker container run -d -p 5000:80 nginxdocker:latest
```

After all these processes, I created a static website with a simple .html file pulled from Github. But since this is a simple disposable website, I did not push this image to any repository such as Docker Hub or AWS ECR. As I mentioned above, if this was a three-tier web application, it should use a file called requirements.txt alongside the Dockerfile and install all the necessary dependencies. Also, a database engine to be used for the database layer can be specified using the docker image in a docker-compose.yaml file. After all these processes, if this image will be used many times, it can be pushed to a repo.

```
https://github.com/ozgenhalit/task-cr.git
```

https://github.com/ozgenhalit/task-cr

This github repository contains my ansible project.

```
https://github.com/ozgenhalit/website.git
```

https://github.com/ozgenhalit/website

And this repository has my sample web application that created from docker image used Dockerfile.

Halit ÖZGEN