

# DevOps Project

## Infrastructure

Terraform was used to create a virtual infrastructure running a cluster of 3 VMs in AWS. The architecture consists of one VPC with an IP range of 10.0.0.0/16, an internal gateway, and a route table to forward the traffic between VMs and Internet, and 3 subnets, each in a different availability zone. The instances were separated in two groups: masters (master1) and workers (worker 1, worker 2). Also, security group rules were added to allow ICMP and SSH from anywhere and allow all traffic from the VMs to the outside. Each EC2 instance has an EBS volume attached to it. I used Elastic IP Addresses instead of DNS records because of the high costs for register a domain name. Terraform files can be found under *tf-config/aws* and the infrastructure can be provisioned with the following commands:

```
terraform init
terraform plan
terraform apply
```

After the provisioning phase, an Ansible playbook was created to update and upgrade the packages, to configure the hostnames of VMs and create a Docker Swarm or a Kubernetes cluster. I created a variable under *vars/main.yml* namely *kubernetes.flag* from where you can choose between Swarm (false) or Kubernetes (true). The playbook can be executed by running the following command:

```
ansible-playbook -i inventory.yml main.yml
```

Password authentication was already deactivated by default, but I included a task to deactivate it just in case. The keypair was created during provisioning step (see Terraform files). After these steps a Docker Swarm or a Kubernetes cluster will be up and running with 1 master and 2 workers.

## Networking

Firewall rules using iptables were set to allow all traffic between VMs, SSH access from local machine and HTTP/HTTPS from everywhere. This step was also automated using another playbook namely *iptables.yml* and can be run using:

```
ansible-playbook -i inventory.yml iptables.yml
```

Then, to configure a VPN network between instances I used OpenVPN which can be installed also through a playbook `vpn.yml`. Next, I configured master1 to be the server by running `openvpn-install.sh` script inside it. This script generated `.ovpn` config files that can be used further by the clients. To generate two files, the script should be run twice. The traffic between EC2 instances was then tunnelled through `10.8.0.0/24` network using the `tun0` interface attached by the OpenVPN to the instances as illustrated in the next figure.

```

ubuntu@ip-10-0-1-71:~$ ping 10.8.0.3
PING 10.8.0.3 (10.8.0.3) 56(84) bytes of data.
64 bytes from 10.8.0.3: icmp_seq=1 ttl=63 time=1.31 ms
From 10.8.0.1 icmp_seq=2 Redirect Host(New nexthop: 3.0.8.10)
64 bytes from 10.8.0.3: icmp_seq=2 ttl=63 time=1.79 ms
From 10.8.0.1 icmp_seq=3 Redirect Host(New nexthop: 3.0.8.10)
64 bytes from 10.8.0.3: icmp_seq=3 ttl=63 time=1.78 ms
From 10.8.0.1 icmp_seq=4 Redirect Host(New nexthop: 3.0.8.10)
64 bytes from 10.8.0.3: icmp_seq=4 ttl=63 time=2.09 ms
From 10.8.0.1 icmp_seq=5 Redirect Host(New nexthop: 3.0.8.10)
64 bytes from 10.8.0.3: icmp_seq=5 ttl=63 time=1.72 ms
From 10.8.0.1 icmp_seq=6 Redirect Host(New nexthop: 3.0.8.10)
64 bytes from 10.8.0.3: icmp_seq=6 ttl=63 time=1.66 ms
^C
--- 10.8.0.3 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5009ms
rtt min/avg/max/mdev = 1.312/1.725/2.091/0.229 ms
ubuntu@ip-10-0-1-71:~$
1000
link/ether 0a:4c:ff:08:93:1f brd ff:ff:ff:ff:ff:ff
inet 10.0.1.81/24 brd 10.0.1.255 scope global dynamic eth1
    valid lft 2728sec preferred lft 2728sec
inet6 fe80::84c:ffff:fe98:931f/64 scope link
    valid lft forever preferred lft forever
4: tun0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN group default qlen 100
    link/none
    inet 10.8.0.3/24 brd 10.8.0.255 scope global tun0
        valid lft forever preferred lft forever
    inet6 fe80::d281:4f68:2666:706d/64 scope link stable-privacy
        valid lft forever preferred lft forever
ubuntu@ip-10-0-1-71:~$ ping 10.8.0.2
PING 10.8.0.2 (10.8.0.2) 56(84) bytes of data.
64 bytes from 10.8.0.2: icmp_seq=1 ttl=64 time=0.646 ms
64 bytes from 10.8.0.2: icmp_seq=2 ttl=64 time=0.689 ms
^C
--- 10.8.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1029ms
rtt min/avg/max/mdev = 0.646/0.667/0.689/0.821 ms
ubuntu@ip-10-0-1-71:~$ ping 10.8.0.2^C
ubuntu@ip-10-0-1-71:~$

ubuntu@ip-10-0-1-164:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid lft forever preferred lft forever
    inet6 ::1/128 scope host
        valid lft forever preferred lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc fq_codel state UP group default qlen 1000
    link/ether 0a:49:ff:38:a7:83 brd ff:ff:ff:ff:ff:ff
    inet 10.0.1.164/24 brd 10.0.1.255 scope global dynamic eth0
        valid lft 2875sec preferred lft 2875sec
    inet6 fe80::849:ffff:fe38:a783/64 scope link
        valid lft forever preferred lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc fq_codel state UP group default qlen 1000
    link/ether 0a:a1:57:44:07:67 brd ff:ff:ff:ff:ff:ff
    inet 10.0.1.180/24 brd 10.0.1.255 scope global dynamic eth1
        valid lft 2875sec preferred lft 2875sec
    inet6 fe80::8a1:57ff:fe44:767/64 scope link
        valid lft forever preferred lft forever
4: tun0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN group default qlen 100
    link/none
    inet 10.8.0.3/24 brd 10.8.0.255 scope global tun0
        valid lft forever preferred lft forever
    inet6 fe80::f0f8:9c8d:ca63:4888/64 scope link stable-privacy
        valid lft forever preferred lft forever
ubuntu@ip-10-0-1-164:~$ ping 10.8.0.1
PING 10.8.0.1 (10.8.0.1) 56(84) bytes of data.
64 bytes from 10.8.0.1: icmp_seq=1 ttl=64 time=0.668 ms
64 bytes from 10.8.0.1: icmp_seq=2 ttl=64 time=0.753 ms
64 bytes from 10.8.0.1: icmp_seq=3 ttl=64 time=0.785 ms
64 bytes from 10.8.0.1: icmp_seq=4 ttl=64 time=0.765 ms
64 bytes from 10.8.0.1: icmp_seq=5 ttl=64 time=0.720 ms
64 bytes from 10.8.0.1: icmp_seq=6 ttl=64 time=0.734 ms
64 bytes from 10.8.0.1: icmp_seq=7 ttl=64 time=0.654 ms
64 bytes from 10.8.0.1: icmp_seq=8 ttl=64 time=0.694 ms
64 bytes from 10.8.0.1: icmp_seq=9 ttl=64 time=0.628 ms
64 bytes from 10.8.0.1: icmp_seq=10 ttl=64 time=0.702 ms
^C
--- 10.8.0.1 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9175ms
rtt min/avg/max/mdev = 0.628/0.710/0.785/0.048 ms
ubuntu@ip-10-0-1-164:~$
  
```

## Monitoring

In this step a monitoring step consisting of Prometheus, Alertmanager, Blacbox exporter, Pushgateway and Grafana was created. I also included cAdvisor and node-exporter to collect as many metrics as possible. The stack is under monitoring directory where I added installation scripts and configuration files for each service. All the scripts and configuration files should be run and copy to the master1 node. The following metrics were gathered and displayed using Grafana (<http://18.198.73.192:3000/>):

### VM CPU Usage

$$100 - \frac{(\text{avg} \quad \text{by}(\text{instance}) \quad (\text{rate}(\text{node\_cpu\_seconds\_total}\{\text{mode}=\text{"idle"}\}[1\text{m}]))}{1} * 100$$

### VM Memory Usage

$$\text{node\_memory\_Active\_bytes} / \text{node\_memory\_MemTotal\_bytes} * 100$$

## VM Free Space Usage

```
topk(1, node_filesystem_avail_bytes /  
node_filesystem_size_bytes * 100) by (instance)
```

## VM Network Transmit Bytes

```
irate(node_network_transmit_bytes_total{device="eth0"}[1m])
```

## VM Network Receive Bytes

```
irate(node_network_receive_bytes_total{device="eth0"}[1m])
```

## VM Network Transmit Bytes Through Tunnel

```
irate(node_network_transmit_bytes_total{device="tun0"}[1m])
```

## VM Network Receive Bytes Through Tunnel

```
irate(node_network_receive_bytes_total{device="tun0"}[1m])
```

## Container CPU Usage

```
(rate(container_cpu_usage_seconds_total{name!=""}[1m])) * 100
```

## Container Memory Usage

```
container_memory_working_set_bytes/container_spec_memory_limit  
_bytes * 100
```

## VM Free Space Usage

```
(container_fs_usage_bytes{name!=""}/container_fs_limit_bytes{n  
ame!=""}) * 100
```

## VM Network Transmit Bytes

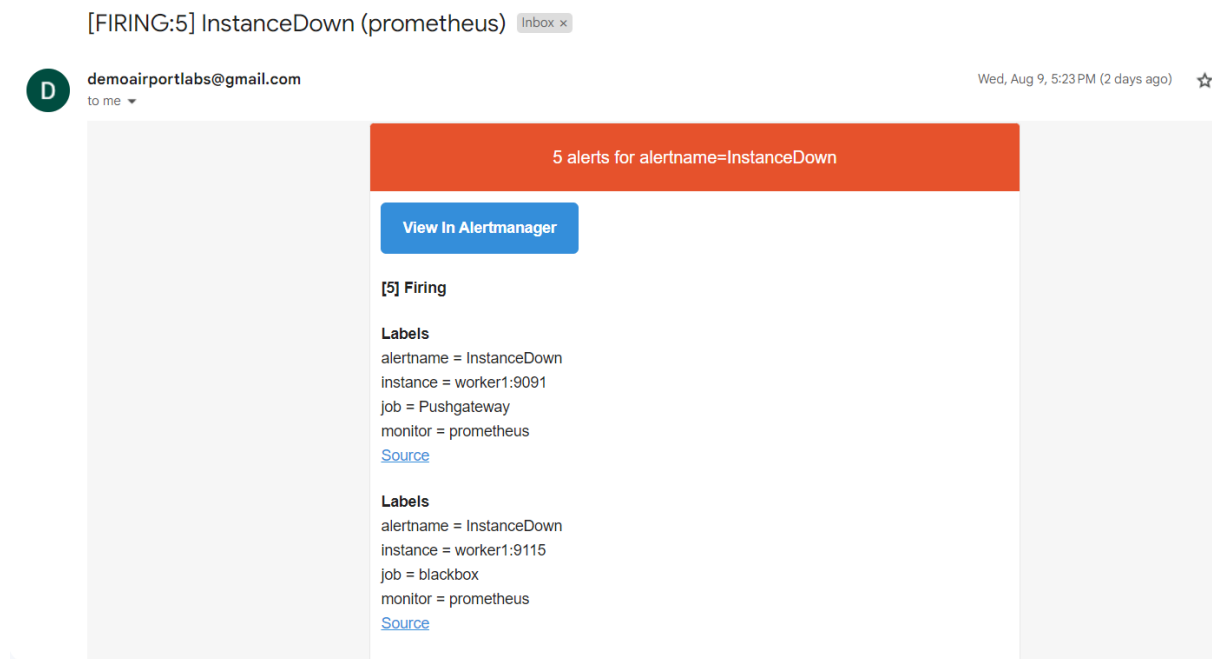
```
irate(container_network_transmit_bytes_total{name!=""}[1m])
```

## VM Network Receive Bytes

```
irate(container_network_receive_bytes_total{name!=""}[1m])
```

Jobs can be analyzed in monitoring/prometheus/Prometheus.yml.

I also created alerts for critical metrics () which send emails in case of node failure as in the following figure.



## CI/CD

In this step I deployed on the previous infrastructure an open source application, where the user first requests the index.html and then it can interact with it (it is only for demo purposes so the application is not responsive).

Two shell scripts were created in demo-app directory: build-pipeline.sh which builds the docker images for each microservice and deploy.pipeline.sh which deploys the application as containers on top of Swarm cluster. Replicas were set to 3 and each replica is hosted on a different node. The application can be accessed at: [http:// 3.121.205.77/](http://3.121.205.77/).

For the CI/CD pipeline I opted for Jenkins which was installed on the master1 node. From Jenkins GUI I created a job which is trigger manually and it builds and deploy the app hosted at: <https://github.com/robertbotez/demo-app>.

## Extra

The application was deployed also using Kubernetes. Moreover, I created a Helm Chart for deploying the app more easily. The infrastructure cost report was attached in the repo.