Deploying Web Application on AWS Using CI/CD Pipeline, and Monitoring Instances Using Nagios and OSSIM

The project hopes to emulate the environment of an enterprise in the industry, running a web application on AWS servers.

The hardware required for the project is as follows

1. 2 EC2 Small Instances: These will act as our developer machines.

2. 2 EC2 Medium Instances: These will be the production servers.

3. A local machine: This will run the network monitoring software and the IDS.

The software requirements are

1. Nagios Core: The network monitoring software that will monitor all our EC2 instances. It will be running on our local machine.

2. OSSIM: The IDS/IPS for our network. It will be running on our local machine.

3. Docker: For creating images.

4. Kubernetes: Our web application will be deployed using these on our production servers.

5. Jenkins: CI/CD Pipeline.

Working on Project and Implementation

In the industry, the production environment and the developer environment are separated. The production environment is internet-facing while the developer environment isn't. This is achieved by keeping the developer environment in an internal network, so the clients won't be able to access it.

In our project, this shall be done by keeping our developer machines in a separate VPC from the production servers.

The production VPC, depicted as cluster_VPC_1 in the flowchart, will be the internet-facing part of the project.

To connect VPC_1 with VPC_2, we will be using VPC Peering. To do this, the procedure is documented at https://docs.aws.amazon.com/migrationhub-refactor-spaces/latest/userguide/tutorial-using-own-network-vpc-peering.html.

The developer machines will be used to develop the web application. The code will be pushed to GitHub. Doing this will activate the CI/CD Pipeline. The pipeline will create a new Docker Image, and push it on Docker Hub. This image will then be pulled and will be implemented in Kubernetes Clusters. As we have two Production servers, one server will act as the master node and the other will be the worker node.

All of these EC2 instances will be monitored using a local machine. This machine will be running Nagios and OSSIM.

The project will first focus on the deployment of all hardware and networking them. Then, the CI/CD pipeline and microservices will be created. After this, if time permits, the Application itself will be developed.

Day 1

Day 1 will be spent researching VPC Peering. VPC Peering is necessary as it will help connect both the Dev and Prod VPCs. The project will be making use of 4 AWS accounts, each running a single EC2 instance. To bring all instances into a single network, VPC Peering is required. The objective behind using multiple accounts is to save costs. The same reason applies to running the monitoring system and IDS on a local machine.

Day 2

On Day 2, monitoring of an EC2 instance with locally installed Nagios will be tested. The research will make use of a single EC2 instance and a VM, on which Nagios will be installed. Various Nagios configurations will be tested to find the best suits our needs.

Day 3 -

On Day 3, offline tests of the infrastructure were attempted. The infrastructure that is supposed to be deployed on AWS was implemented in VMware. Further research was done on various aspects of the project,
> AWS VPC peering between two VPC
> Install Nagios on the AWS EC2 Instance testing on 1st sprint has not been implemented yet.

AWS VPC Peering - Step By Step Tutorial (Part-12) | #aws #vpc #devops #cloud …

> Kubernetes Monitoring using Prometheus and Grafana
https://github.com/iam-veeramalla/prometheus-Grafana-Zero-to-Hero

Prometheus Docs:- https://prometheus.io/docs/introduction/overview/

Grafana Docs:- https://grafana.com/docs/grafana/latest/

Day 4

On Day 4, created the common Developer github repository to push the code by the developers will collaborate in the Project-group Repository, https://github.com/C2-803100/project-repo

create a new repository on the command line

```
echo "# project-repo" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/C2-803100/project-repo.git
git push -u origin main
```

push an existing repository from the command line

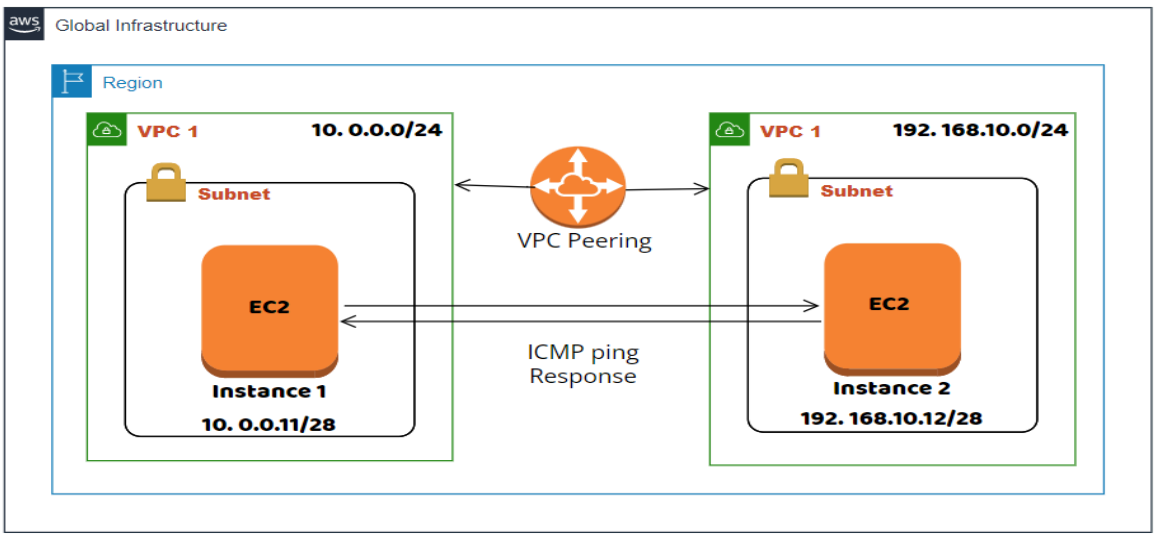
```
git remote add origin https://github.com/C2-803100/project-repo.git
git branch -M main
git push -u origin main
```

Next install jenkins in AWS EC2 instance.

Day 5

Deadline to completely set-up AWS Infrastructure with Nagios Monitoring, and Microservices prerequisites is set at 9th February 2024. Changes to the project are made. There will be no developer machines. Instead, there will be 1 Master node, and 3 Worker nodes.

8/2/2024
:- working on VPC peering and working on building the infrastructure of our project.



VPC peering connects two VPCs, allowing communication using private IPv4 or IPv6 addresses. It enables instances in both VPCs to communicate as if part of the same network. You can establish connections between your VPCs or with VPCs in other AWS accounts, even across different regions (inter-Region VPC peering).

AWS VPC peering connects Virtual Private Clouds (VPCs) within or across AWS accounts and regions without the need for additional hardware. It facilitates secure data transfer between VPCs, enabling collaboration or resource access. Inter-region VPC peering ensures private IP communication with encryption, reducing the risk of threats like DDoS attacks. This cost-effective solution uses the AWS backbone, avoiding the public internet, making it suitable for sharing resources or replicating data across different geographic regions.

Day 6

Infrastructure has been completely set-up. Each AWS account is running a t2.medium instance with at least 25GB gp3 storage. The reason to run a t2.medium instance is that each machine must be able to meet the minimum requirements for minikube. Each machine is monitored using Nagios NCPA Client.

Sprint 2

Day 7

Further research on minimum requirements revealed that our t2.medium instances won't be adequate. It was decided that t2.large machines would be used. Further changes were made to the project components. Kubeadm would be used in the project instead of Minikube. Simultaneously, research was done on a two-tier web application.

Day 8

It was decided that Terraform would be used to implement the code infrastructure. A Terraform script was written for the same. It was also decided that instead of using 4 EC2 instances, only 2 would be used. This decision was taken to manage the increasing costs of the project. Various Jenkins pipeline scripts were tested on local machines.

Day 9

Work on AlienVault OSSIM was started. It kept crashing in a VirtualBox environment. It also crashed the host machine. The entire day was spent researching on making the IDS function.

Day 10

No work was performed on this day as we had revision classes and a CCPP guidance session.

Day 11

Limited work was performed as OSSIM issues continued. The web application was also found to be inadequate, so it was decided that it would be changed. Jenkins pipelines were tested. At the end of the day, OSSIM was partially working.

Day 12

The cause behind OSSIM not working was found. As the OSSIM server required a static IP to function properly, it was infeasible to use. So, Snort would be used. It was also decided that AWS Elastic Kubernetes Cluster would be used for creating Kubernetes cluster. Research was done on how to implement it. Terraform script was updated to install more dependencies.

Day 13

AWS EKC was tested. We deployed two t2.micro machines using it. However, we faced issues making the service work. Snort was found to be difficult to install on Debian 12, as many of its dependencies were unstable and not included in the apt package. However, we found workarounds for those packages and managed to install it.

Day 14

Final testing for Jenkins pipeline was done. It was decided that t2.large machines would be used. It was also decided that AlienVault OSSIM would be retained as a vulnerability scanner, as it didn’t depend on a static IP. Dockerfile was changed, Nginx image was replaced with Apache.

Day 15

Final day of working on the project. Jenkins pipeline suddenly started behaving erratically, so it had to be rewritten at the last moment. Research was done on why the Kubectl service failed. It was found that t2.micro was inadequate, so we used t2.medium instead. The upgrade made the service work. We scaled out to 3 nodes. Elastic IPs were used. Service Group for the cluster were updated. ACLs were updated to accommodate the working of the cluster. At the end of the day, the project was running as expected.