WILDBIT – DISTRIBUTED, MULTI-REGION JENKINS CI/CD DEPLOYMENT

Charles Ikenyei

XXX xxx

Wildbit Distributed, Multi-Region, Jenkins CI/CD Deployment

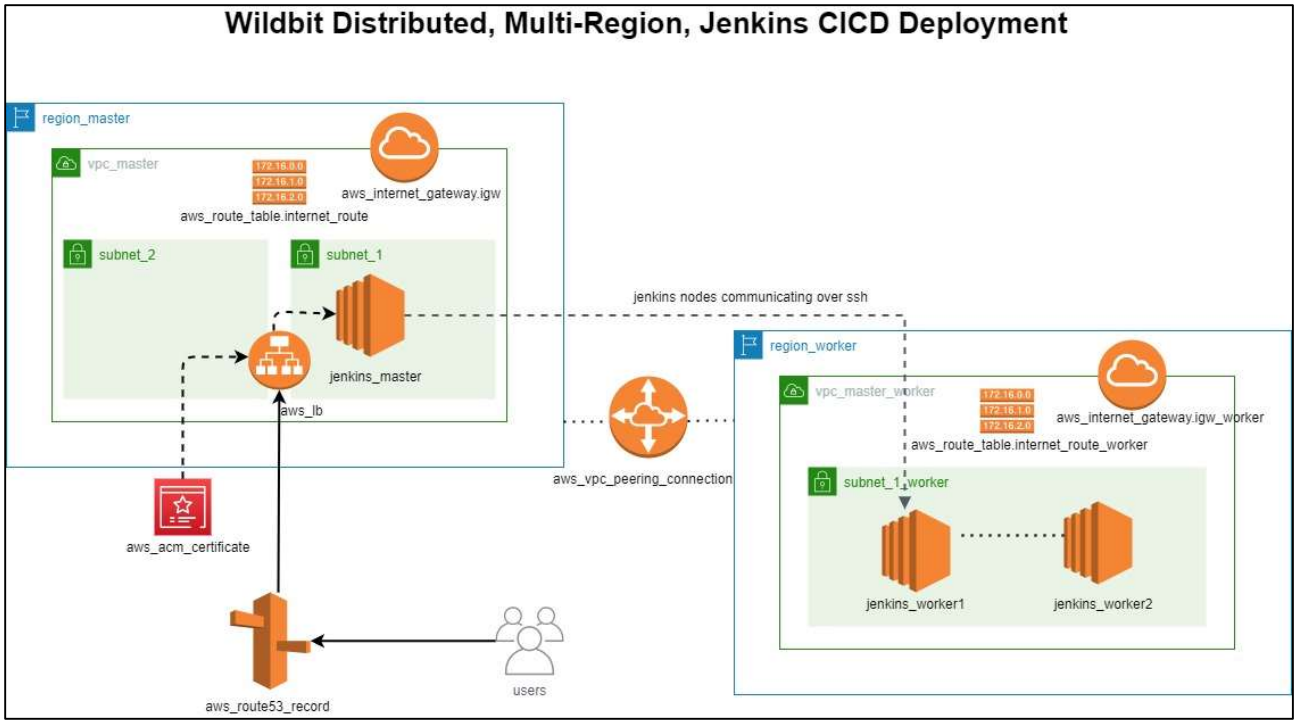

Figure 1: Project Deployment Layout

This project comprises of the following modules:

1) Wildbit_Jenkins_CICD_Documentation
2) FAQ
3) backend.tf
4) provider.tf
5) network.tf
6) variables.tf
7) output.tf
8) security_groups.tf
9) instances.tf
10) dns.tf
11) acm.tf
12) ansible.cfg
13) **Ansible_templates folder**
    - A. Jenkins-master-sample.yml
    - B. Jenkins-worker-sample.yml
    - C. Jenkins_master.yml
    - D. Jenkins_worker.yml
    - E. node.j2
    - F. cred-privkey.j2
    - **G. Inventory_aws folder**
        - I. tf_aws_ec2.yml

1. Environment Setup

Setting up Terraform:

Download the Terraform binary for your Linux OS from Hashicorp website via the link below (https://www.terraform.io/downloads.html).

I. Place the unzipped Terraform binary in the PATH of the OS for ease of access.
II. Test and verity the binary with the "Terraform version" command

Setting up AWS CLI & Ansible:

Depending on your OS Install Python's pip package installer

I. Install AWS CLI & Ansible packages
II. Configure AWS CLI using the "**aws configure**" command

Setting up AWS IAM Permissions for Terraform:

Terraform will need granular permissions to create and delete various AWS resources. Please refer to "**terraform_permissions**"text file containing the granular IAM policy in the project folder. Either of the two options can provide Terraform with the resources that it needs:

I. Create a separate IAM user with required permissions
II. Create an EC2 (IAM role) instance profile with required permissions and attach to the EC2 that you intend to use as the Terraform Controller.

2. Terraform Infrastructure as Code (IaC)

The commands below are entered in the following sequence:

Terraform init

This command initializes your Terraform project working folder and downloads the required plugins from the repository. It needs to be run before deploying infrastructure.

Terraform fmt

This command keeps the formatting consistent, especially when teams are collaborating and tracking the Terraform code through a version control software.

Terraform validate

This command checks your Terraform code for syntax errors as well as misconfigured resources.

Terraform plan

This command creates the plan of action for Terraform to act on. It checks connectivity to provide the API using the AWS credentials that you provided. It refreshes the state of the resources by calculating the delta between the current and the desired state which is defined in the Terraform code.

Terraform apply

This command applies the changes as suggested by the execution plan.

3. Persisting Terraform State in S3 Backend

Terraform Backends determines how the state is stored. By default, it is stored locally; however, it can be changed by passing the backend configuration to a Terraform block in the Terraform project code. Prior to setting this up, create an S3 bucket where the Terraform State file will be stored and enable versioning. Refer to the Terraform code in the "**backend.tf**" file. Complete the S3 backend configuration by running the "**terraform init**". Whenever the init command is run, terraform will upload the State file to the S3 bucket where it is versioned.

4. Terraform Providers

Terraform providers provide the source code for all Terraform resources. They carry out interactions with vendor APIs such as AWS, Azure or GCP. They also provide the logic for managing, updating, and creating resources in Terraform. This project comprises of a provider in each region and each provider is pegged to an alias where each provider can be invoked to a specific resource via its alias.

A variable for the Terraform user profile and the regions where the resources will be deployed are defined in the "**variables.tf**" file. Next step is to create the "**providers.tf**" file where the provider for each profile and region are defined. "**Terraform init" must be run whenever there are changes to modules or providers.tf file**.

5. Deploying VPCs, Internet Gateways & Subnets

The "**network.tf**" contains the resources that are required to create the VPCs, subnets, route table and internet gateways in each region. Refer to these resources labeled **#1-7** in the comment section of the "**network.tf"** file.

Proceed with the terraform fmt, plan and apply commands. Then inspect the resources via AWS Console. If all the resources are provisioned and tested, then tear down the infrastructure using "**terraform destroy –auto-approve"**

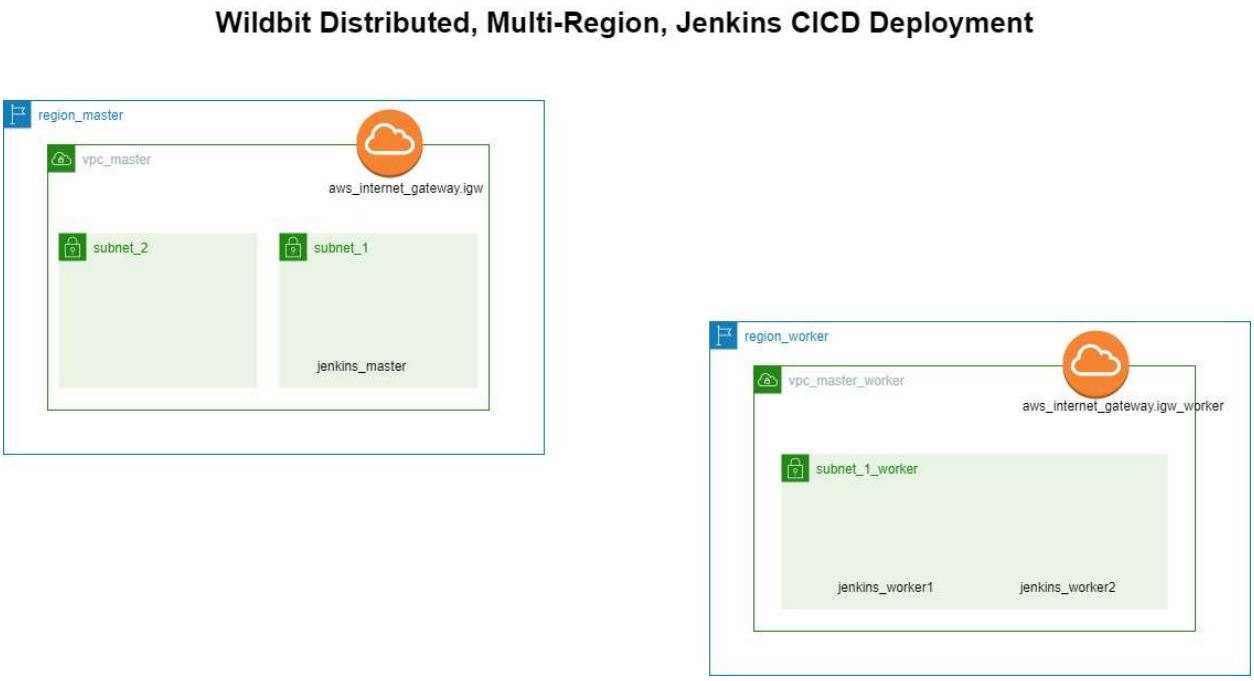

Figure 2: VPC Deployment

6. Deploying Multi-Region VPC Peering

The remaining part of the "**network.tf**" contains the resources that are required to create the VPC Peering, Route Tables, and over-riding the default Route Table with the new Route Table in each region. Refer to these resources labeled **#8-13** in the comment section of the network.tf file.

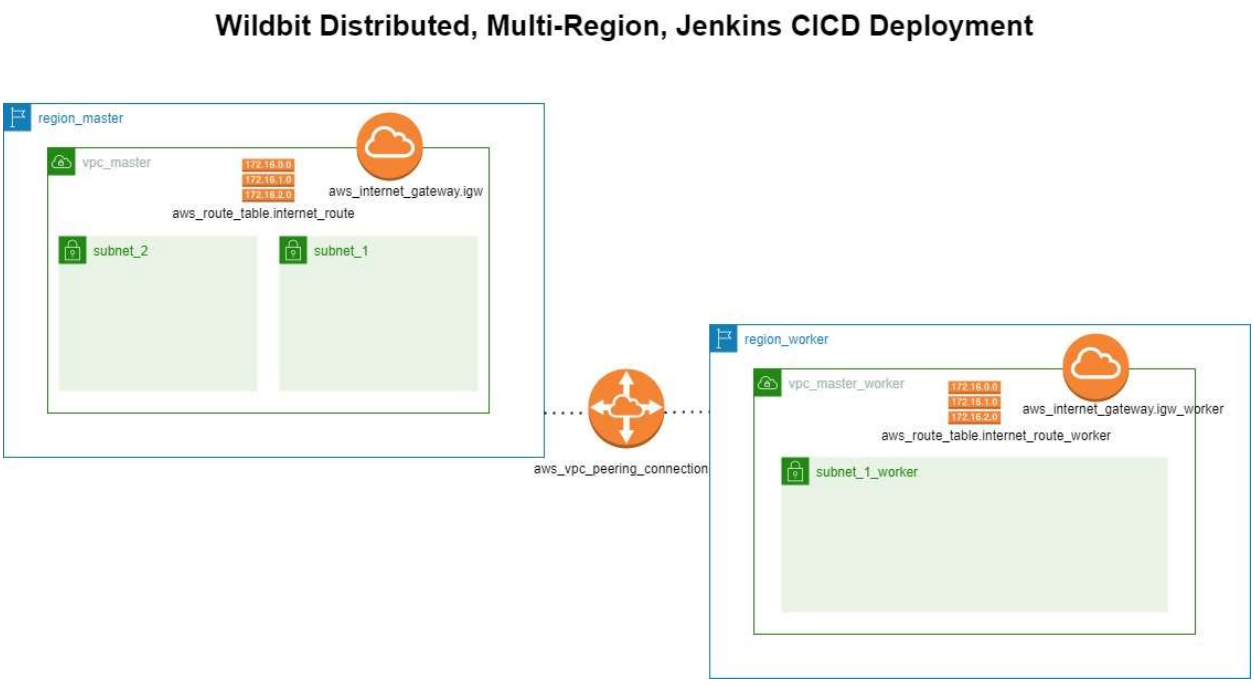

Figure 3: VPC Peering & RT Configuration

Proceed with the terraform fmt, plan and apply commands. Then inspect the resources via AWS Console. If all the resources are provisioned and tested, then tear down the infrastructure using "**terraform destroy –auto-approve"**

7. Deploying Security Groups

Three security groups each for the Application Load Balancer (lb_sg), Jenkins Master Instance and Jenkins Worker(s) instance(s) are created and identified by the resources labeled **#14, 15 & 16** in the "**security_groups.tf**" file.

The lb_sg will allow for inbound **http** and **https** internet traffic and outbound TCP protocol open to all.

The Jenkins Master security group will allow for inbound TCP on port 8080 for the lb_sg and port 22 for the Jenkins Worker.

The Jenkins Worker security group will allow inbound TCP traffic on port 22 from the Jenkins Master and both outbounds for Jenkins Master and Worker(s) security groups will be open to all for internet access.

To increase the of security level by limiting SSH connectivity to just the developers, please replace the "external_ip" variable in the variables.tf file with your local ip_address.

Proceed with the terraform fmt, plan and apply commands. Then inspect the resources via AWS Console. If all the resources are provisioned and tested, then tear down the infrastructure using "**terraform destroy –auto-approve"**

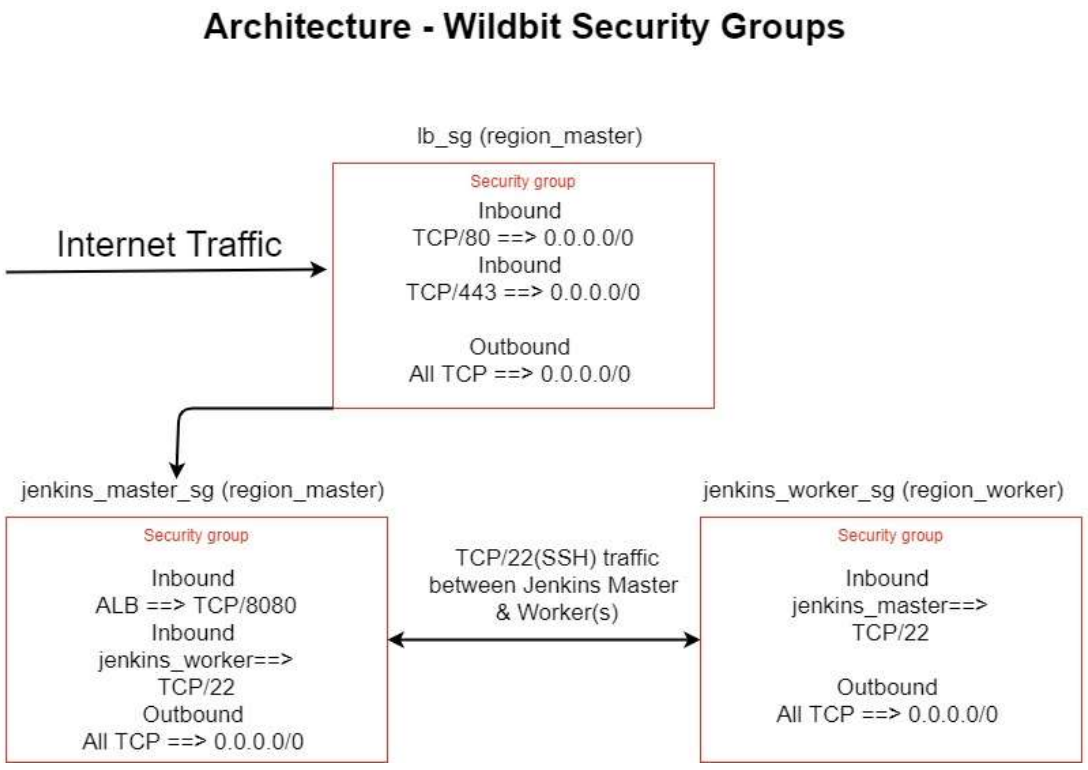

Figure 4: Deploying Security Groups

8. Deploying App EC2 Instances

Step #1: Fetching the AMI ID using the AWS SSM Parameter Store

Using AWS Systems Manager (SSM) Parameter Store to fetch AMI ids for Jenkins Master and Jenkins Worker(s) instances. This AMI ID will be passed to the EC2 Instance during deployment. Refer to the "**instances.tf**" file and the list of all Linux AMIs can be queried via the link in the reference section.

Proceed with the terraform fmt, plan and apply commands. Then inspect the resources via AWS Console. If all the resources are provisioned and tested, then tear down the infrastructure using "**terraform destroy –auto-approve"**

Step #2: Deploy EC2 Key Pairs

Per region, Public and Private key pairs are generated via "**ssh-keygen -t rsa"** command where the public key is baked into each EC2 instance that will be deployed. Refer to Resources **#17 & 18** in the "**instances.tf file**".

Proceed with the terraform fmt, plan and apply commands. Then inspect the resources via AWS Console. If all the resources are provisioned and tested, then tear down the infrastructure using "**terraform destroy –auto-approve"**

Step #3: Deploy Jenkins Master & Jenkins Worker(s) EC2 Instances

The workers_count variable and the Instance_type variable is specified in the "**variables.tf"** file.

The public_ip of both the Jenkins Master and Worker instances that are required for ssh login are displayed using the output block in the "**output.tf**" file.

With respect to the Jenkins worker(s) instances, the count variable increases the number of instances to be generated and a '**for**' loop within the set iterates over every created Jenkins Worker instance and outputs the "**instance public ip**" that is mapped to the "**value**" parameter in the output block. Refer to Resources labeled **#19 & 20** in the instances.tf file for the EC2 deployments.

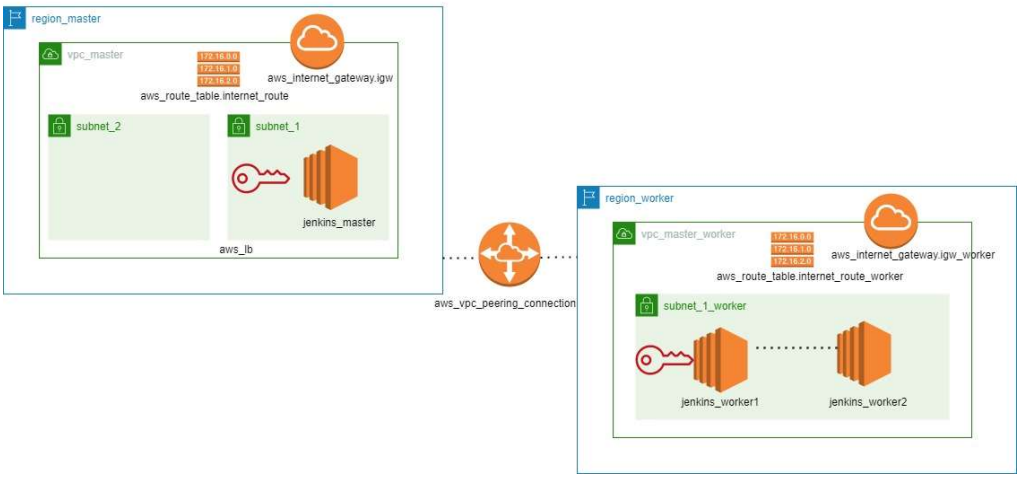

Figure 5: Deploying EC2 Key-Pair

Proceed with the terraform fmt, plan and apply commands. Then inspect the resources via AWS Console. If all the resources are provisioned and tested, then tear down the infrastructure using "**terraform destroy –auto-approve"**

Step #4: Bootstrapping Infrastructure via Terraform Provisioners

The essence of having local-exec Terraform Provisioners is to invoke the ansible-playbook that will run the **Jenkins-master-sample.yml** on the Jenkins Master and **Jenkins-worker-sample.yml** on the Jenkins Worker EC2 instances. Both files are sample yaml files that will test the ansible configuration for Jenkins Master and Jenkins Worker EC2 instances. These sample yaml files will be replaced with the following yaml files during Jenkins installation via Ansible.

A. Jenkins_master.yml
B. Jenkins_worker.yml

The **ansible.cfg** file is modified with three parameters listed below that will enable the dynamic inventory querying by ansible on AWS.

```
[defaults]
inventory       = ./ansible_templates/inventory_aws/tf_aws_ec2.yml
enable_plugins = aws_ec2
interpreter_python = auto_silent
```

The "inventory" parameter above, points ansible configuration to the configuration file for the dynamic host inventory for AWS that is provided by Ansible. This configuration file lets Ansible know which regions to query and how to aggregate the parameters returned by the querying of the AWS instances so that we can parse the hostnames within Ansible playbooks.

The "**tf_aws_ec2.yml**" must be suffixed with "**_aws_ec2.yml**" else AWS will not pick it up as an inventory file. Within the file, the "Keyed_groups" which is where the magic of Ansible dynamic inventory for AWS occurs. Here, we define how the ansible dynamic inventory will create variables for us to parse in as host names to the ansible-playbooks. In essence, the variable will pickup the Jenkins worker instances tag that Terraform creates and then parse them as a hostname within the ansible-playbook whenever it is invoked.

We need to install the boto3 python SDK for AWS which is required by the ansible dynamic inventory plugins.

Finally, the instances are updated with the provisioner before running the terraform commands.

Proceed with the terraform fmt, plan and apply commands. Then inspect the resources via AWS Console. If all the resources are provisioned and tested, then tear down the infrastructure using "**terraform destroy –auto-approve"**

9. Deploying Application Load Balancers & Routing Traffic to EC2 App Instances

The "**alb.tf**" configuration file is created with the resources containing the application load balancer (ALB), the two listeners that will receive http traffic on port 80, the target group and the resource for attaching the Jenkins Master to the target group. The webserver variable that will allow for traffic via **port 80** is defined in the variables.tf file. Refer to the following Resource tags: **#21, 22, 23 & 24**.

An output block containing the dns_name of the deployed load balancer is defined in the output.tf file. This dns_name will allow us to test the web server that comes up after a successful "**terraform apply**".

Finally, the httpd webserver is installed on the node that is directly attached to the application load balancer and this is done by updating the "**Jenkins-master-sample.yml**" file.

Proceed with the terraform fmt, plan and apply commands. Then inspect the resources via AWS Console. If all the resources are provisioned and tested, then tear down the infrastructure using "**terraform destroy –auto-approve"**

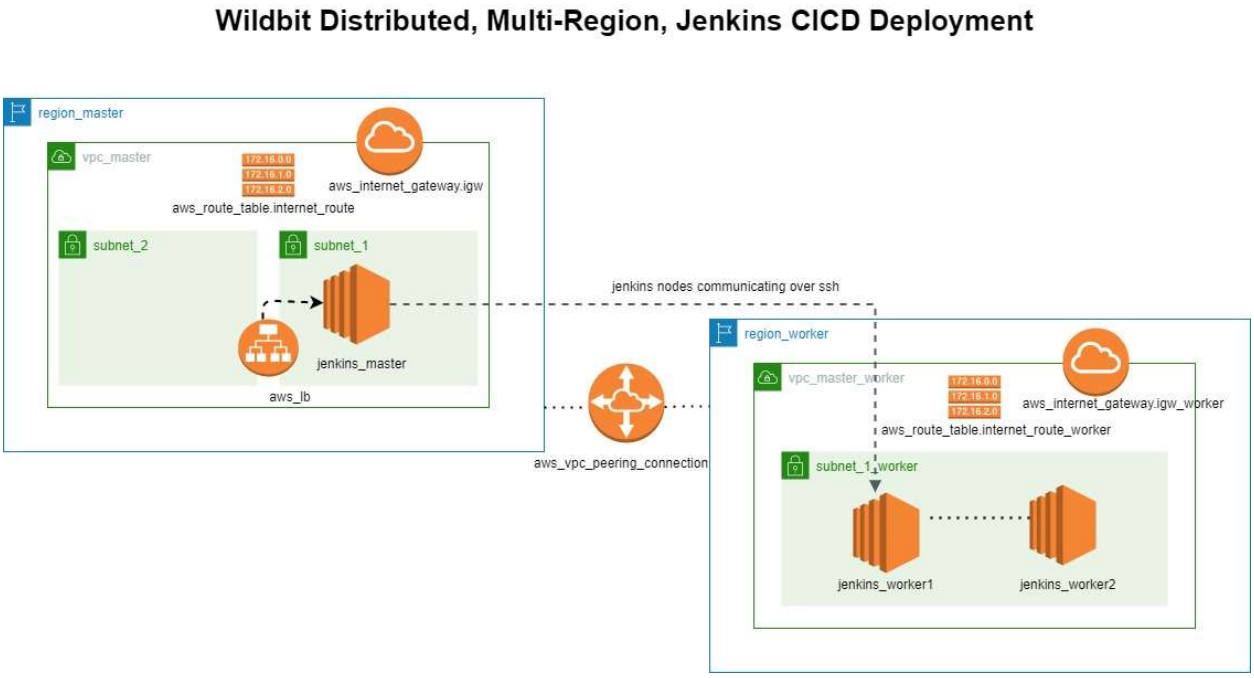

Figure 6: Deploying Application Load Balancer

10. Deploying Route 53 Records (Public Hosted Zone) and an HTTPS endpoint

The ultimate goal is to provide a path for our user traffic to flow through our load balancer to our Jenkins application on the Jenkins Master EC2 instance. You are required to purchase a domain name via

AWS Route53 or a 3rd party vendor. If you decide to go with a 3rd party company then you must register the AWS name servers in your hosted zone to your domain name outside of AWS.

The user traffic will arrive at our public domain that is hosted on AWS Route53. The public domain is required to generate an SSL certificate for the domain to enable a secured connection over HTTPS.

The Route53 will have a Public Hosted Zone that is attached to it. This Public Hosted Zone will have an alias record pointing to the DNS name of the ALB.

The first step is to create a hosted zone on AWS Route 53 and then store the name of the hosted zone in a variable block saved in the variables.tf file.

The Resource tag **#21** is the resource for creating the certificate manager. Before the AWS certificate authority can issue a certificate manager (CM) for our site, AWS CM must verify that we own or control all domain names that we have specified in our request via the **domain_name** parameter in resource #21. By using Route53, ACM automatically adds and uses CNAME record to validate that we own or control the domain name. With a "**dns**" validation specified, ACM provides a CNAME record to insert into your dns database so that it can validate the ownership of the domain name.

Resource tag **#22** in the "**dns.tf**" file creates a record for ACM certificate domain validation.

Resource tag **#23** in the "**acm.tf**" file revalidates the CNAME record that we inserted inside our hosted zone and that completes the generation of the SSL certificate.

The next step is to create a second listener on port 443 in the ALB tagged **Resource #24** and then attach the ACM to this listener so that http request is terminated on the ALB for a secured connection. Also, the listener on port 80 is edited to "**redirect**" traffic to the listener on port 443.

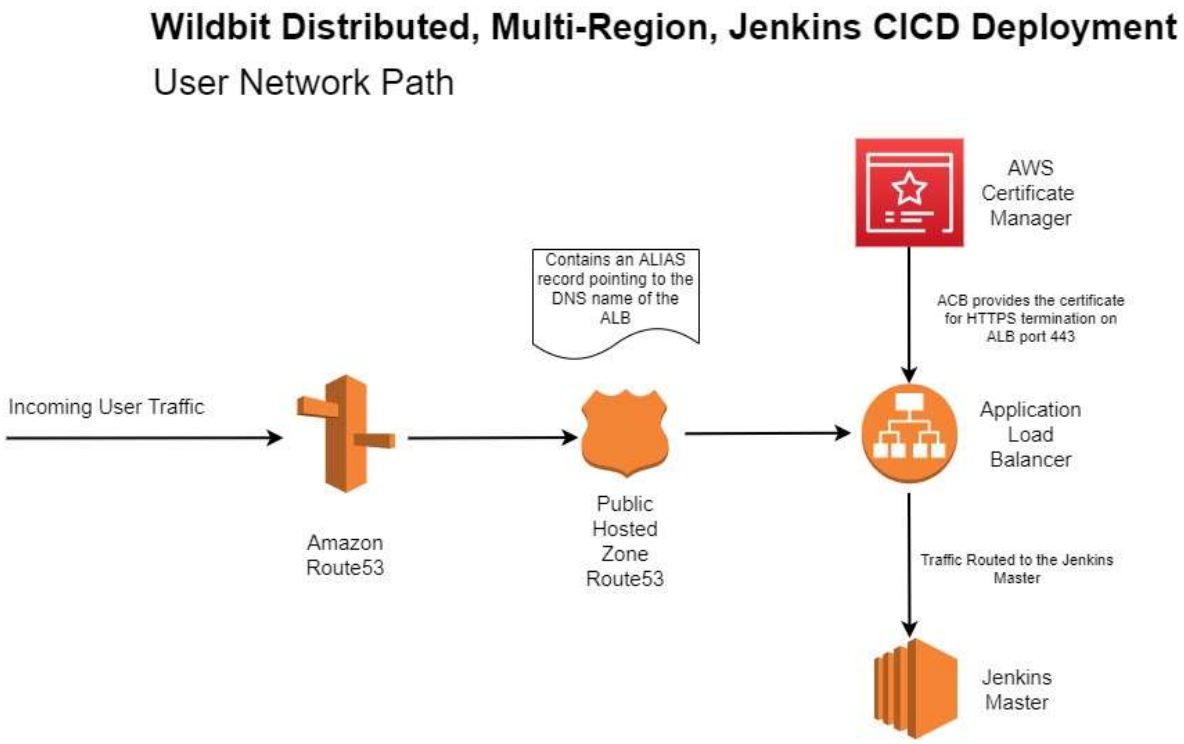

Figure 7: User Network Path Layout

The final resource that will be created (Resource #26) in the "dns.tf" file is a record that routes traffic from Route 53 to the dns name of the ALB.

Proceed with the terraform fmt, plan and apply commands. Then inspect the resources via AWS Console. If all the resources are provisioned and tested, then tear down the infrastructure using "**terraform destroy –auto-approve"** Upon successful completion of Terraform apply, test the project with the url displayed by the "**url**" output block in the output.tf file.

11.Building Playbooks for Jenkins Master & Jenkins Worker Setup

The Chronology of Ansible Playbook for Jenkins Master and Jenkins Worker(s) shown below is converted to configuration files in the "**Jenkins_master.yml**" and "**Jenkins_worker.yml**" files.

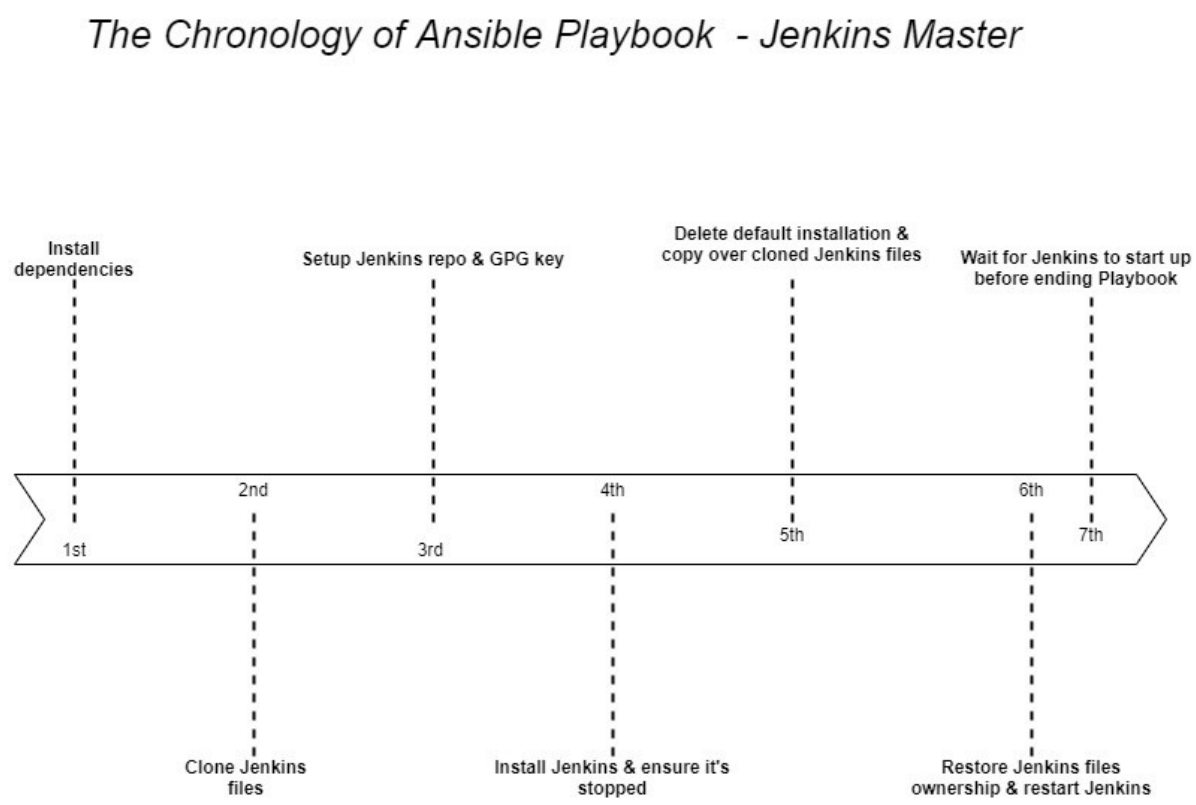

Figure 8: Ansible Playbook - Jenkins Master

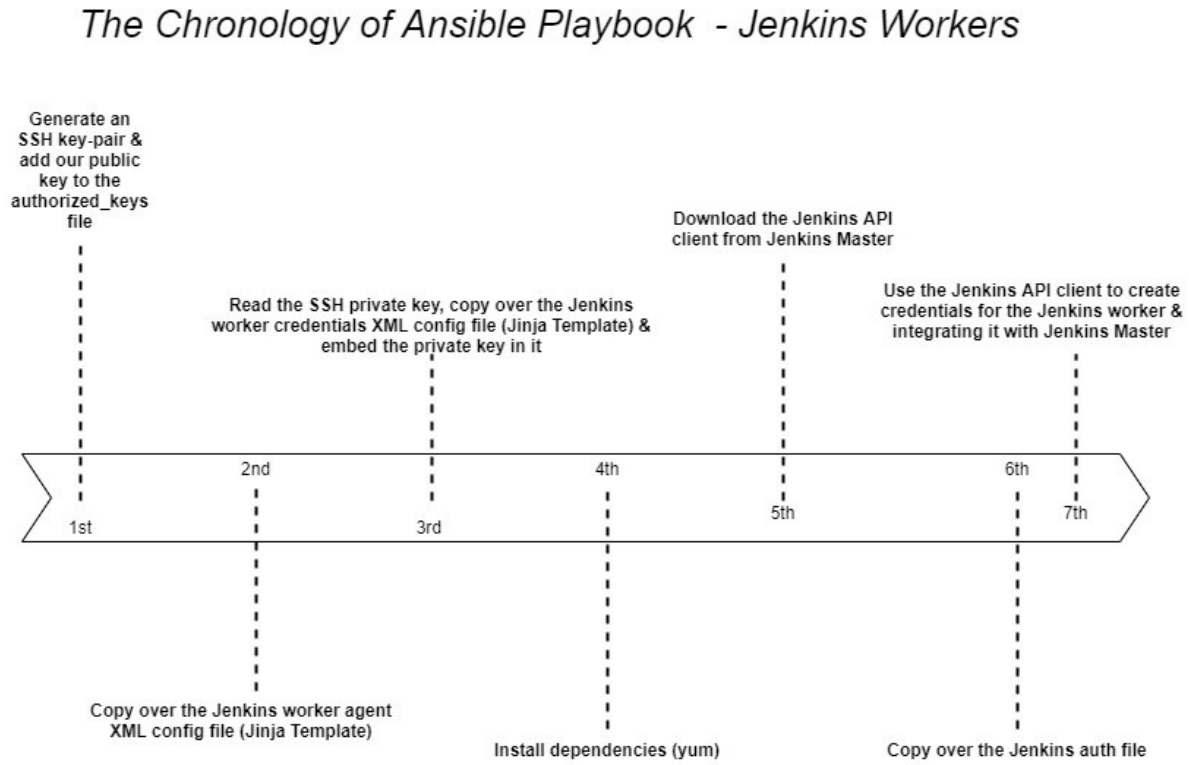

Figure 9: Ansible Playbook - Jenkins Worker(s)

12. Resources

Install unzip Utility

sudo yum -y install unzip

Determine OS Path

echo $PATH

Install Python's Pip Package

Sudo yum -y install python3-pip

Install AWS CLI & Ansible packages

pip3 install awscli --user

pip3 install ansible –user

Verify AWS CLI & Ansible Installation

aws --version

ansible –version

Creating an S3 bucket

aws s3 mb s3://<unique_bucket_name>

List of All AWS Linux AMIs in the Current Region

aws ssm get-parameters-by-path --path /aws/service/ami-amazon-linux-latest --query "Parameters[].Name"

Install boto3

Pip3 install boto3 –user

List hosted zones on AWS

aws route53 list-hosted-zones