# Finiata Tasks for DevOps Engineer

**Part1:**

The IPs of the launched instances are as follows:

34.201.72.224

3.86.35.229

3.84.251.40 – created using module

The terraform folder is compressed and attached in the email containing all of the required settings. Port changing for httpbin would also work in the same way as in the example with Apache2.

**Part 2:**

The terraform folder is compressed and attached in the email containing the required setup with remote backend and launching instance with terraform module.

Answers of the questions are as below:

1. **If the third instance mentioned above is already launched, how would you import it in terraform.tfstat.**

Make the desired changes in the terraform working directory and run terraform plan refresh-only. This will show a plan how the resource will be added in the infrastructure, and then run terraform apply -refresh-only. This will not modify the infrastructure but will add the manually added resource in the state file.

1. **How would you rename an existing instance in a terraform state file?**

Terraform move command is used to rename the resources in the state file.

“Terraform state mv old\_resource\_type.old\_resource\_name new\_resource\_type.new\_name”

Resource definition modification is required before changing the state associated with it.

Part 3:

The script has been attached in the email.

The answers of the questions are as below:

 **How would you update the infra changes if you have to update a terraform module?**

If the update is not immediate and can be done on the next terraform apply, we can use Taint command to recreate the resources with new parameters.

We can also directly apply the changes after validating the plan by running terraform apply.

 **How would you tackle Terraform (depends on) cycle dependencies?**

Circular dependencies (depends on) are normally removed by generalizing the main resource and calling it in sub-resource. Lets take an example of AWS Security Group where we need to create a security group 1 with ingress rule of security group 2 and create a security group 2 with ingress rule of security group 1. This would give a Cyclic error. The solution to the problem is to use the “aws\_security\_group” resource to simply create the AWS security groups first without defining any inline ingress or egress rules. You can then use the “aws\_security\_group\_rule” resource to add ingress and egress rules to your respective security groups.

resource "aws\_security\_group" "group\_1" {

name = "Group 1"

vpc\_id = "${var.vpc}"

}

resource "aws\_security\_group" "group\_2" {

name = "Group 2"

vpc\_id = "${var.vpc}"

}

resource "aws\_security\_group\_rule" "allow\_group\_2" {

type = "ingress"

from\_port = 80

to\_port = 80

protocol = "tcp"

security\_group\_id = "${aws\_security\_group.group\_1.id}"

source\_security\_group\_id = "${aws\_security\_group.group\_2.id}"

}

resource "aws\_security\_group\_rule" "allow\_group\_1" {

type = "ingress"

from\_port = 80

to\_port = 80

protocol = "tcp"

security\_group\_id = "${aws\_security\_group.group\_2.id}"

source\_security\_group\_id = "${aws\_security\_group.group\_1.id}"

}

* **On what occasion would you replace Terraform with Ansible**

Terraform is generally an Infrastructure orchestration tool while Ansible is generally a configuration management tool. Both the tools can perform each other’s duties but with limitations like terraform has limitation working on changing the configurations in a complex environment to keep the application and dependencies up to date. If the requirement is to manage the in-depth OS level configuration automation in complex application or infrastructure, ansible should be used instead of terraform.

Also, one of the main reasons when Ansible wins the battle is Ansible is mutable by default. Ansible attempts to keep the configuration changes consistent as per the latest version of the playbook. Any changes introduced do not cause any replacement of the respective infrastructure. It only repairs or modifies configuration on the given component.

Also, if we need to provision bare metal servers we need Ansible as Terraform does not work on bare metal infrastructure deployment.

* **With Terraform and its module, would you maintain your IaC and modules in the same repository or isolate modules from live code and why?**

It depends on the complexity of the infrastructure and the use of the modules in terraform code. But generally the modules should be kept in different repositories. But if we have a family of modules that are all coupled together either functionally or conceptually, and likely to all change together when the requirements change then keeping at least that family of modules together in one repository might make the things easier.

On updating the module over the time the maintenance and management of the module increases, this may cause complexity in module and provider versioning and dependencies. If we use independent repositories for each complex or custom module we can have different advantages over keeping the modules in the same repositories.

1. Module versioning can be applied using release tagging.
2. Independent repositories can allow isolated testing for module functioning and security.
3. If you have a large team that collaborates on a complex infrastructure system, multiple source repositories allow you to localize changes within the repository and lessen the blast radius of failed infrastructure updates across the system. You can scope changes to the teams accountable for the infrastructure.
4. Access control is applied to the entire repo by default. In some circumstances, you might only want a user to access specific subdirectories. In this scenario the modules should be set up with independent repos so that the access can be assigned in the best way.

** In brief words, what can you tell us about DNSSEC?**

DNSSEC is a secure way to authenticate DNS requests. The creation of DNS was at the early ages of the internet and security was not a concern at that time. That means DNS will allow any address given by default without any verification, which can cause cyberattacks and security threats. DNSSEC provides authentication in DNS using digital signatures over the data based on public key. Every DNS zone has a public/private key pair. The zone owner uses the zone's private key to sign DNS data in the zone and generate digital signatures over that data. Private key content is kept secret by the zone owner and the zone's public key is published in the zone itself for anyone to retrieve. Any recursive resolver that looks up data in the zone also retrieves the zone's public key, which it uses to validate the authenticity of the DNS data. The resolver confirms that the digital signature over the DNS data it retrieved is valid. If so, the DNS data is legitimate and is returned to the user. If the signature does not validate, the resolver assumes an attack, discards the data, and returns an error to the user.

** What is a Data Warehouse?**

Datawarehouse is a process/tool for collecting and managing data from different sources to provide meaningful analysis and reporting based on it. Data warehouses are designed to allow the users to run queries and analyses on historical being saved periodically and differently, from different heterogenous resources.

** On AWS, where you would replace ELB with ALB?**

ALB works on application layer instead of the network layer and diverts the traffic based on the variables and the request content instead of the IP and the traffic game like in ELB. If we have an multiple applications sharing the same host the better approach is to use ALB to route the traffic based on the application layer data. Also, when the application behavior is more logical then just routing and the content needs to be served based on the nature of the request, ALB should be used.

** CI/CD best practices for you?**

For the pipeline, the code should be deployed on the target environment in such a way that the build process may not take longer time and the meaningful tests should run on priority to avoid long build time. The series of the steps should be in this way:

1. Checkout code from Git

2. Build and Unit Tests

3. OWASP + Code Analysis (using a tool than can handle both of the tasks like SonarQube)

4. Integration Testing

5. Deployment on target environment.