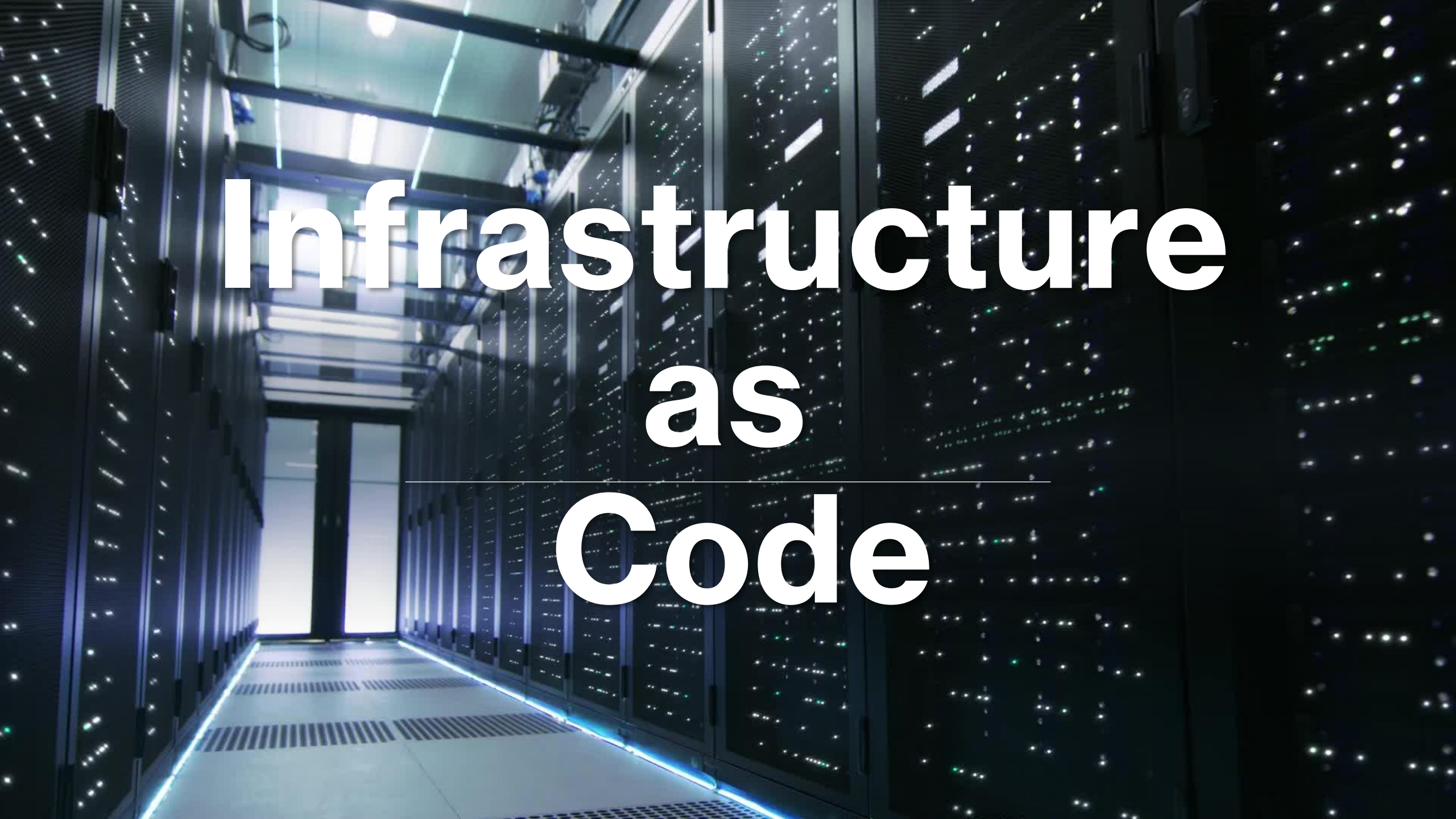

Infrastructure
as
Code

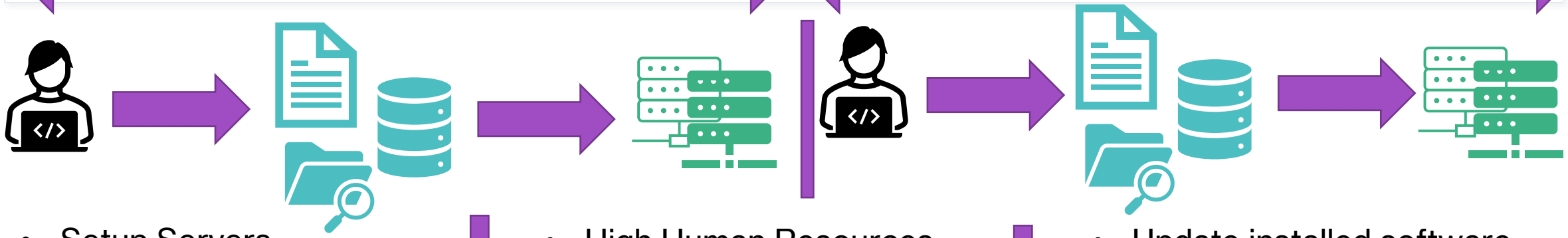
Buzz words or Tools, which we hear in DevOps World

Provisioning	Deployment	Configuration	Orchestration
Infrastructure Setup	Infrastructure Management	Applications Setup and	Applications Management

Manual Provisioning

Setup

Maintenance



- Setup Servers
- Configuring Network
- Create Route Tables
- Install Software
- Configure the Software
- Installing the Database

Prepare Servers

- High Human Resources Cost
- More effort and Time
- More Human Errors Possible

Possible Errors

- Update installed software versions
- Deploy new release
- DB Backup/ Updates
- Recover app
- Add / recover Servers

During Maintenance

For Multiple Environments

Development

Setup

- Setup Servers
- Configuring Network
- Create Route Tables
- Install Software
- Configure the Software
- Installing the Database

Prepare

- High Human Resources Cost
- More effort and Time
- More Human Errors Possible

Possible

Maintenance

- Update installed software versions
- Deploy new release
- DB Backup/ Updates
- Recover app
- Add / recover Servers

During Maintenance

Production

Setup

- Setup Servers
- Configuring Network
- Create Route Tables
- Install Software
- Configure the Software
- Installing the Database

Prepare

- High Human Resources Cost
- More effort and Time
- More Human Errors Possible

Possible

Maintenance

- Update installed software versions
- Deploy new release
- DB Backup/ Updates
- Recover app
- Add / recover Servers

During

Infrastructure as Code

- Automate all the tasks end to end with the concept called IaC
- IaC tools or Programs which carry out all these tasks
- Why so many tools?
- Tools are good at their own areas
- Three major categories of such tasks
  - Infrasture Provisioning (spinning up the servers, network configurations, load balancers creation)
  - Configuring the infrastructure (Installing the software or apps and managing the same = so here we prepare the infrastrure for the deployment).
  - Now Deploying the application at the actual provisioned serever(docker containers, jenkins, k8s orchestratioons)

Different Phases

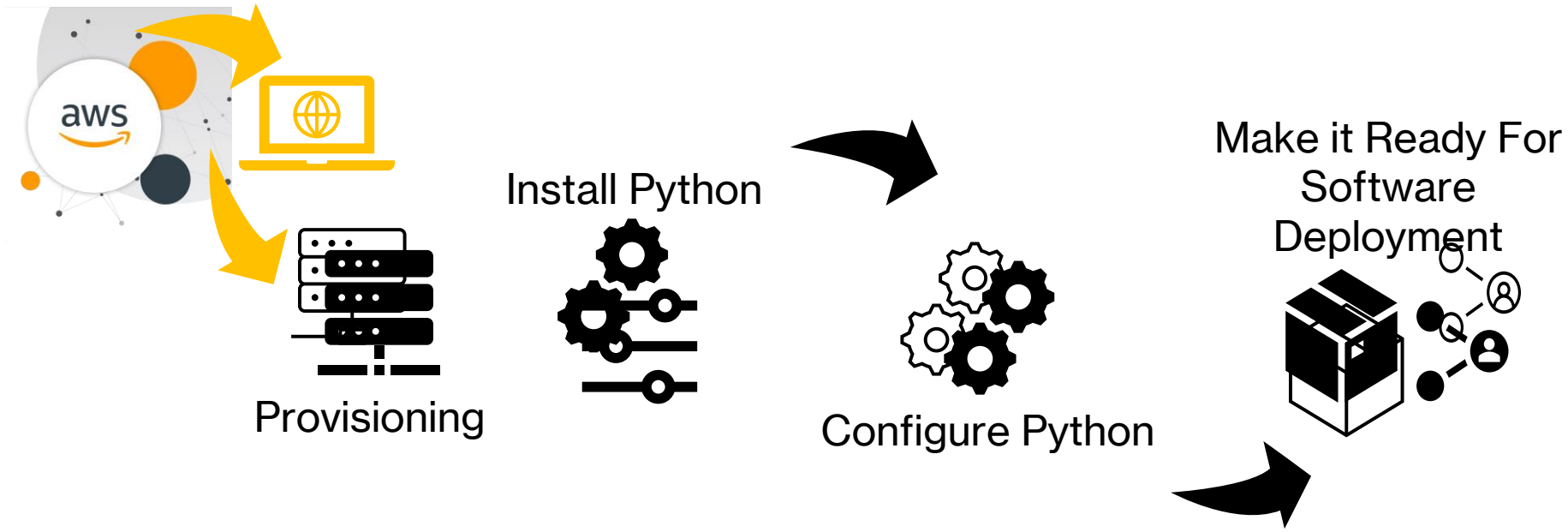
Initial Phase

- Provision and configure
- Initial software installation and configure

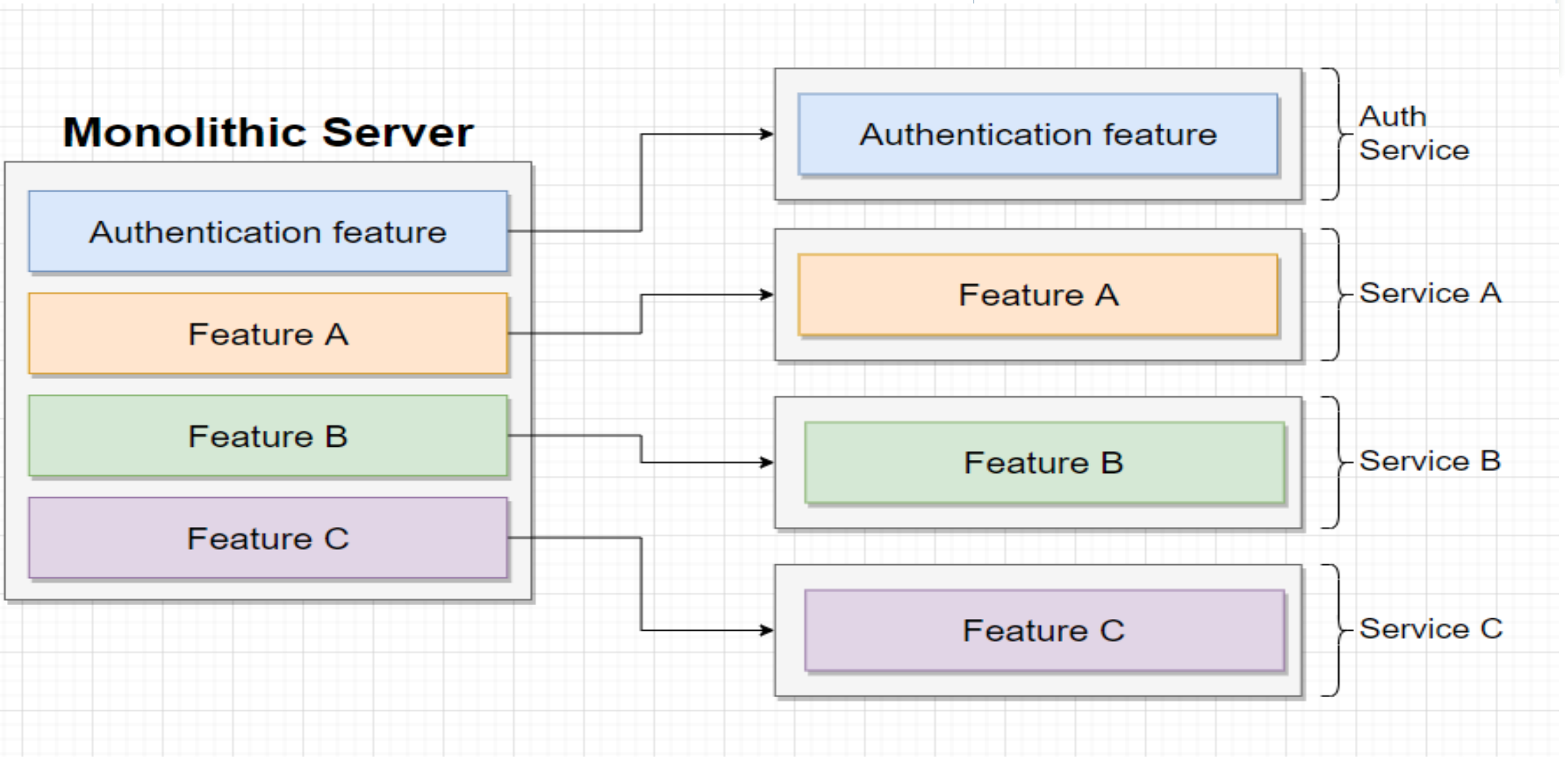
Maintenance Phase

- Adjusting to the software
- Add or remove the servers
- Update and reconfigurations of the softwares

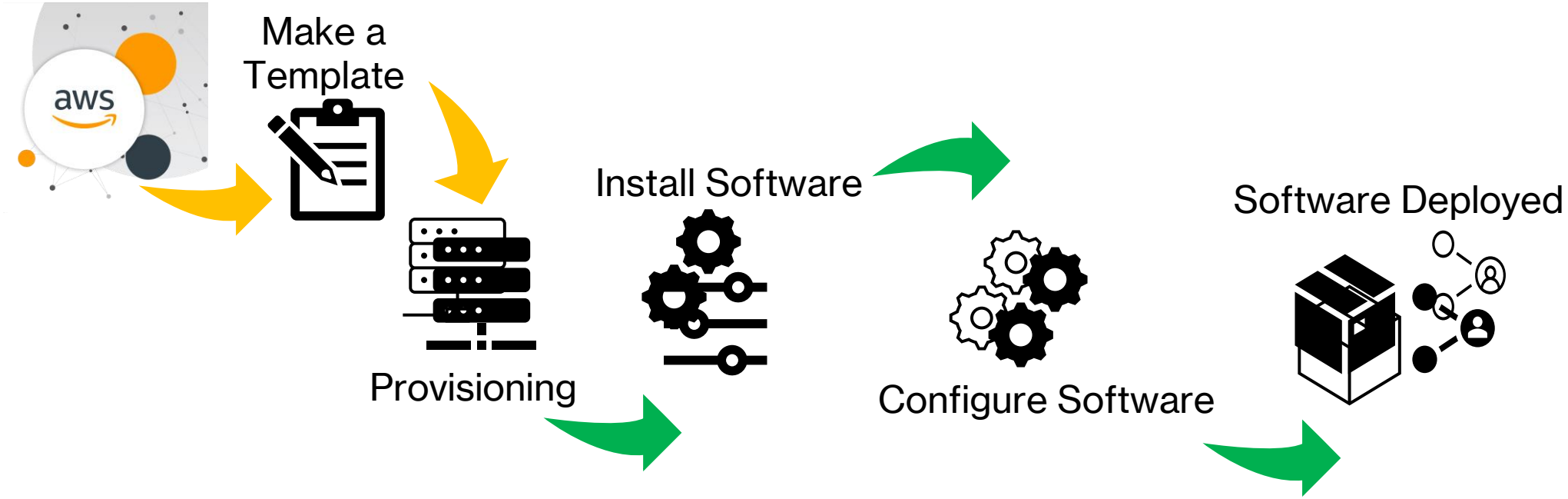
Manual Approach Continued



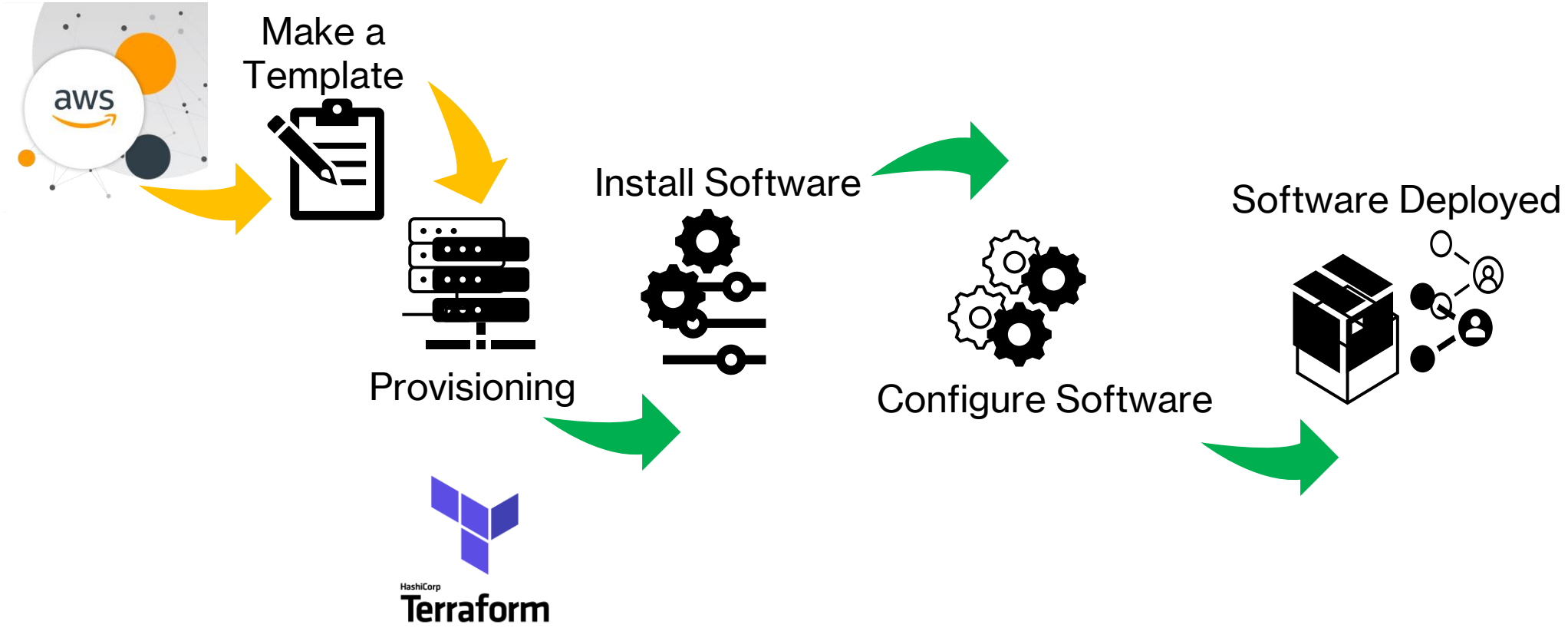
Microservices



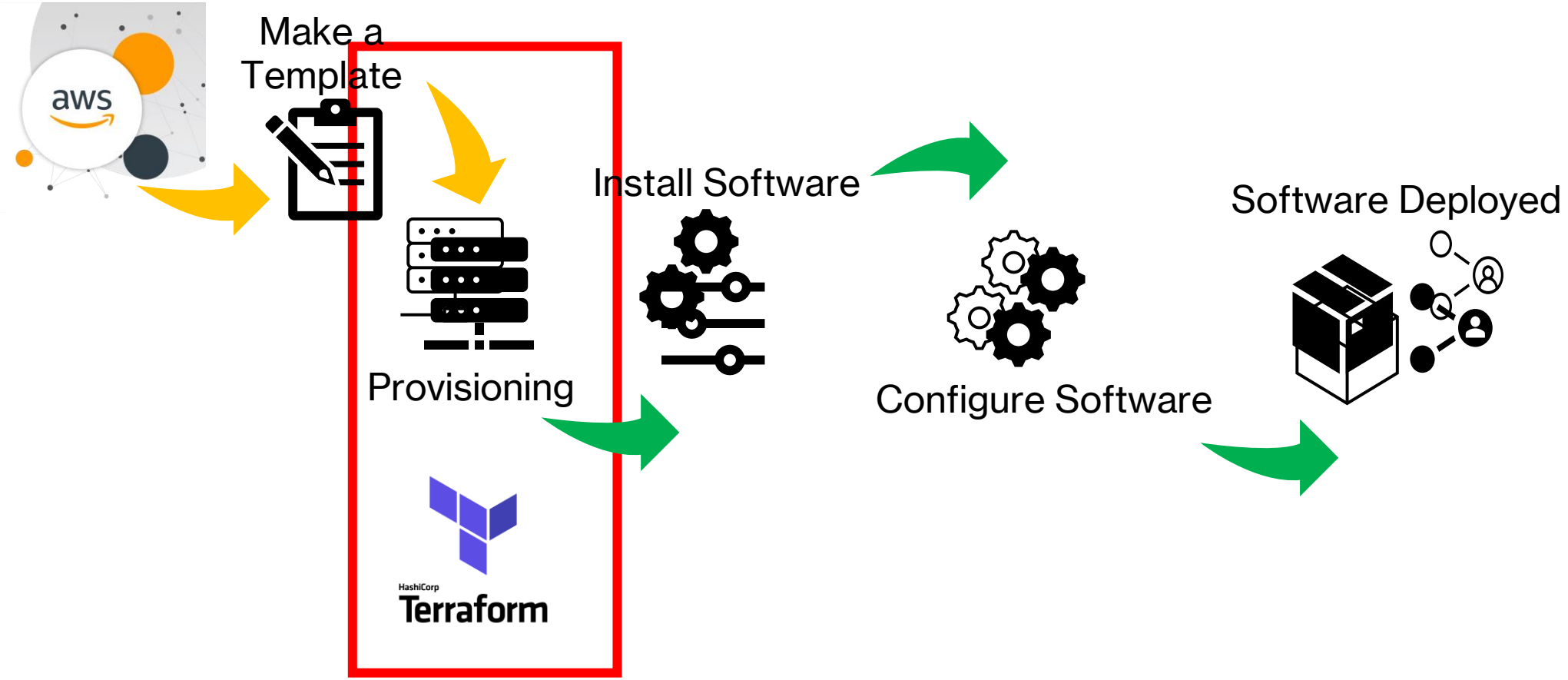
Automated


aws
Make a Template
Provisioning
Install Software
Configure Software
Software Deployed

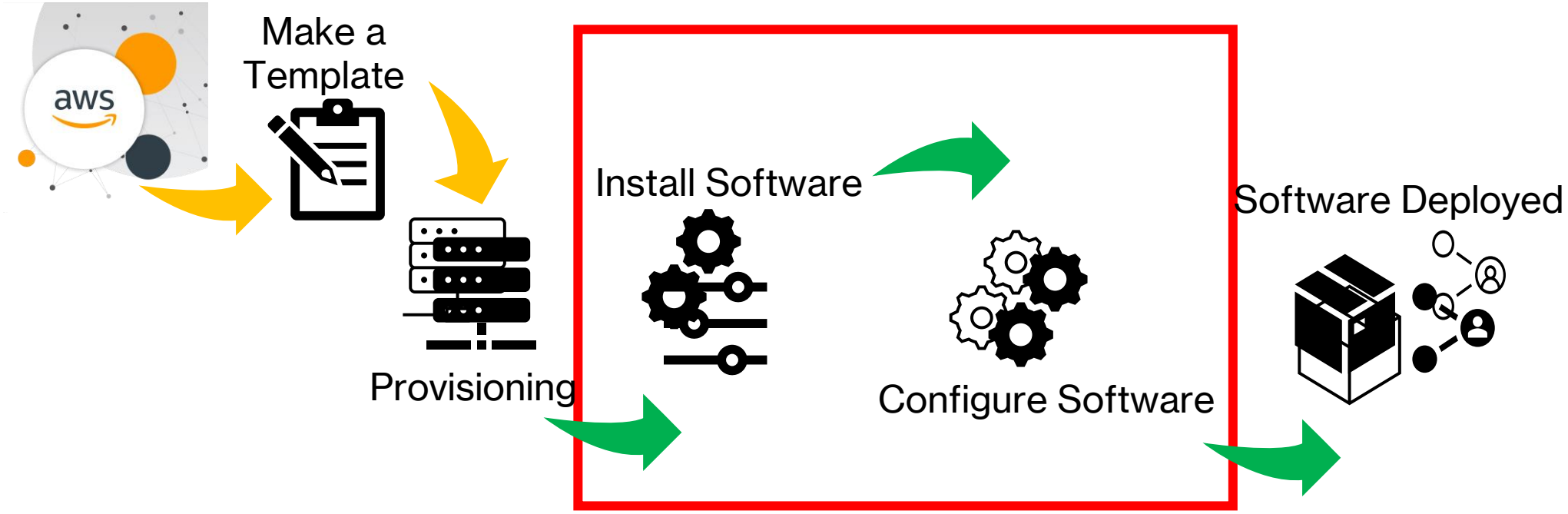
Automated


aws
Make a Template
Provisioning
Install Software
Configure Software
Software Deployed
HashiCorp
Terraform

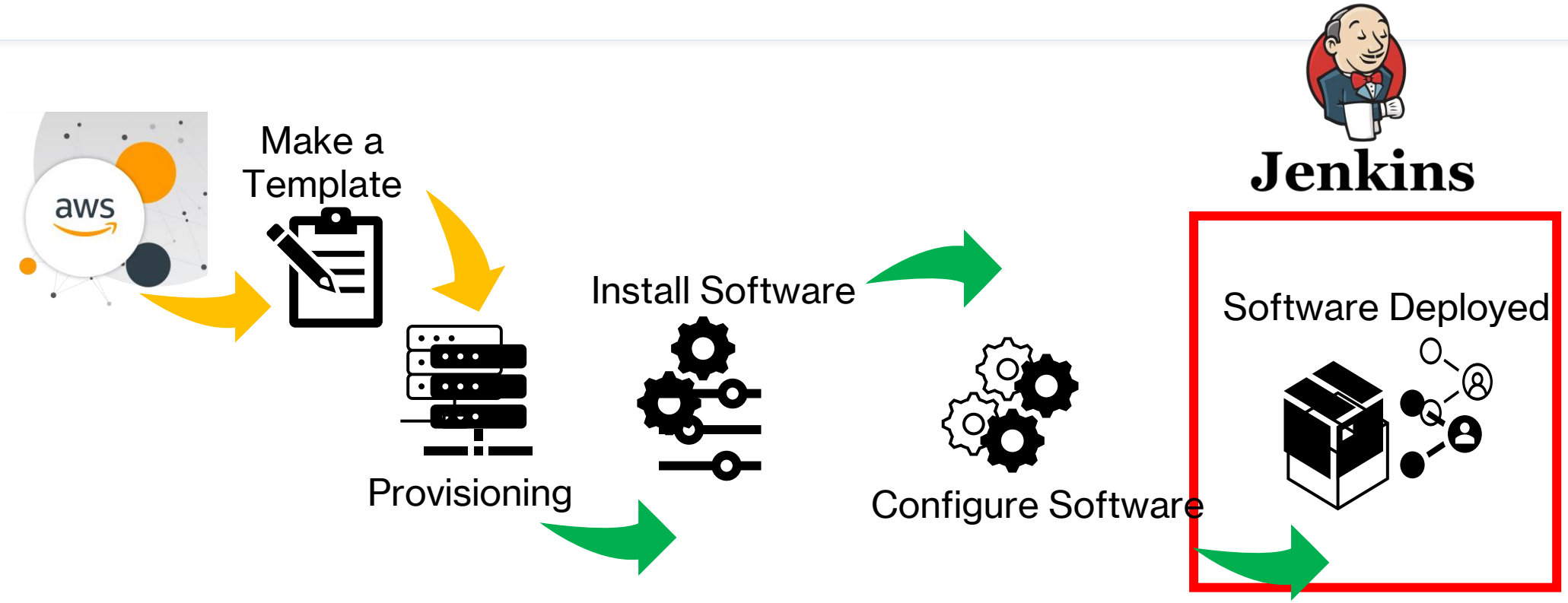
Infrastructure Provisioning



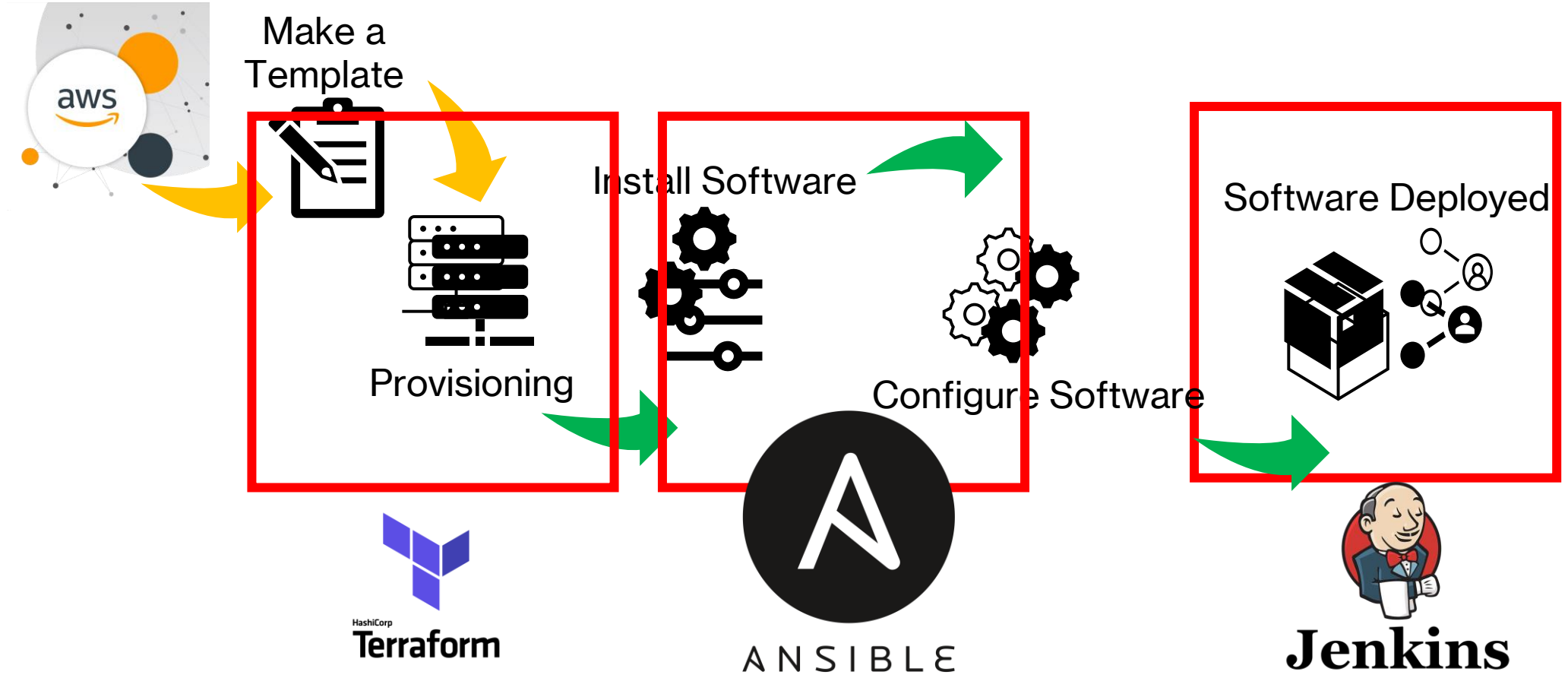
Configuration Management



CI/CD Tools


aws
Make a Template
Provisioning
Install Software
Configure Software
Jenkins
Software Deployed

Why Different Tools



Why Diff Tools

- Automate different tasks in different categories


Get started

Hashicorp

Terraform

Terraform

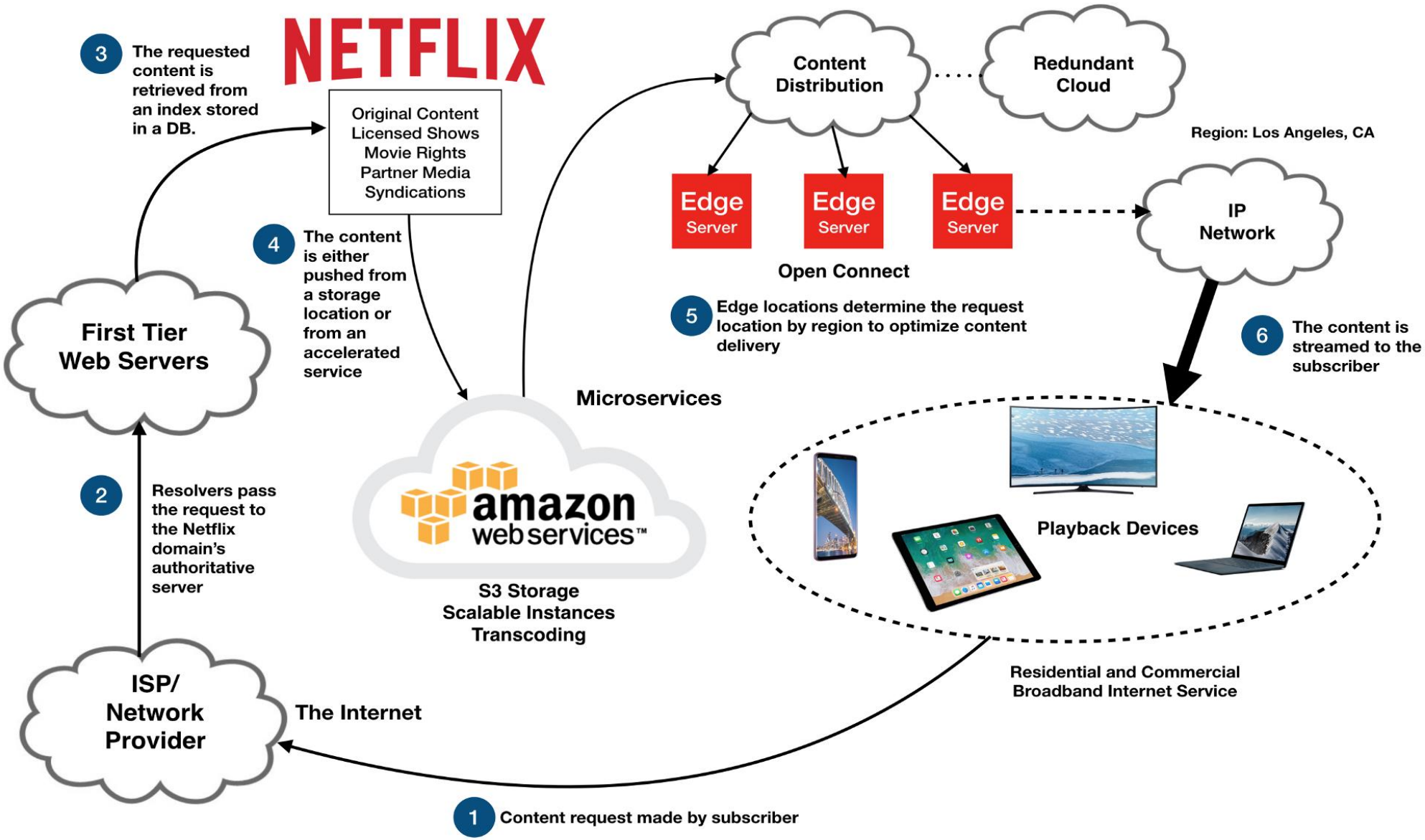
- What is it?
- Why terraform ?
- Difference b/n Terraform and Ansible
- Terraform architecture and commands
- Example

What should you know ?

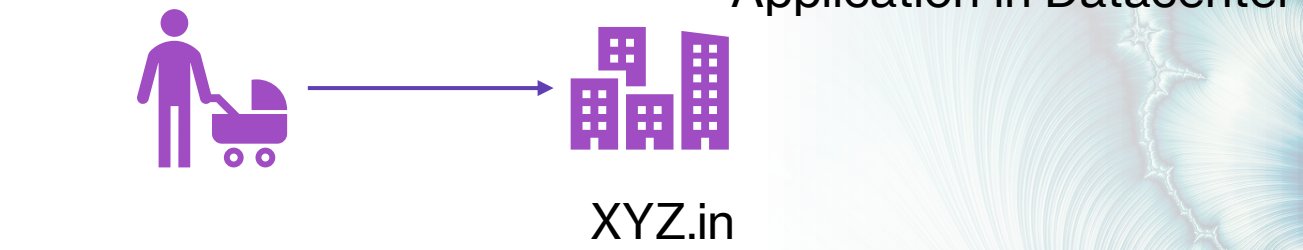
- Terraform Basics : CLI and HCL
- AWS : VPC, Security Groups and EC2 Instances
- GIT/ GITHUB : Basic Commands
- IP Networking : Sub Netting and DNS
- Linux Administration : Basic Linux usage and administration skills
- DevOps Tools : Jenkins and Terraform Cloud
- Any Programming Language

Review

- Configuration : HCL File
- HCL
- Resources
- Deployment
- Provider
- Module
- Variables

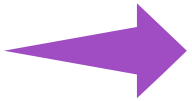

NETFLIX
3 The requested content is retrieved from an index stored in a DB.
Original Content
Licensed Shows
Movie Rights
Partner Media
Syndications
4 The content is either pushed from a storage location or from an accelerated service
First Tier Web Servers
2 Resolvers pass the request to the Netflix domain's authoritative server
ISP/ Network Provider
The Internet
1 Content request made by subscriber
Content Distribution
Redundant Cloud
Edge Server
Edge Server
Edge Server
Open Connect
Region: Los Angeles, CA
IP Network
5 Edge locations determine the request location by region to optimize content delivery
6 The content is streamed to the subscriber
Microservices
amazon web services™
S3 Storage
Scalable Instances
Transcoding
Playback Devices
Residential and Commercial Broadband Internet Service

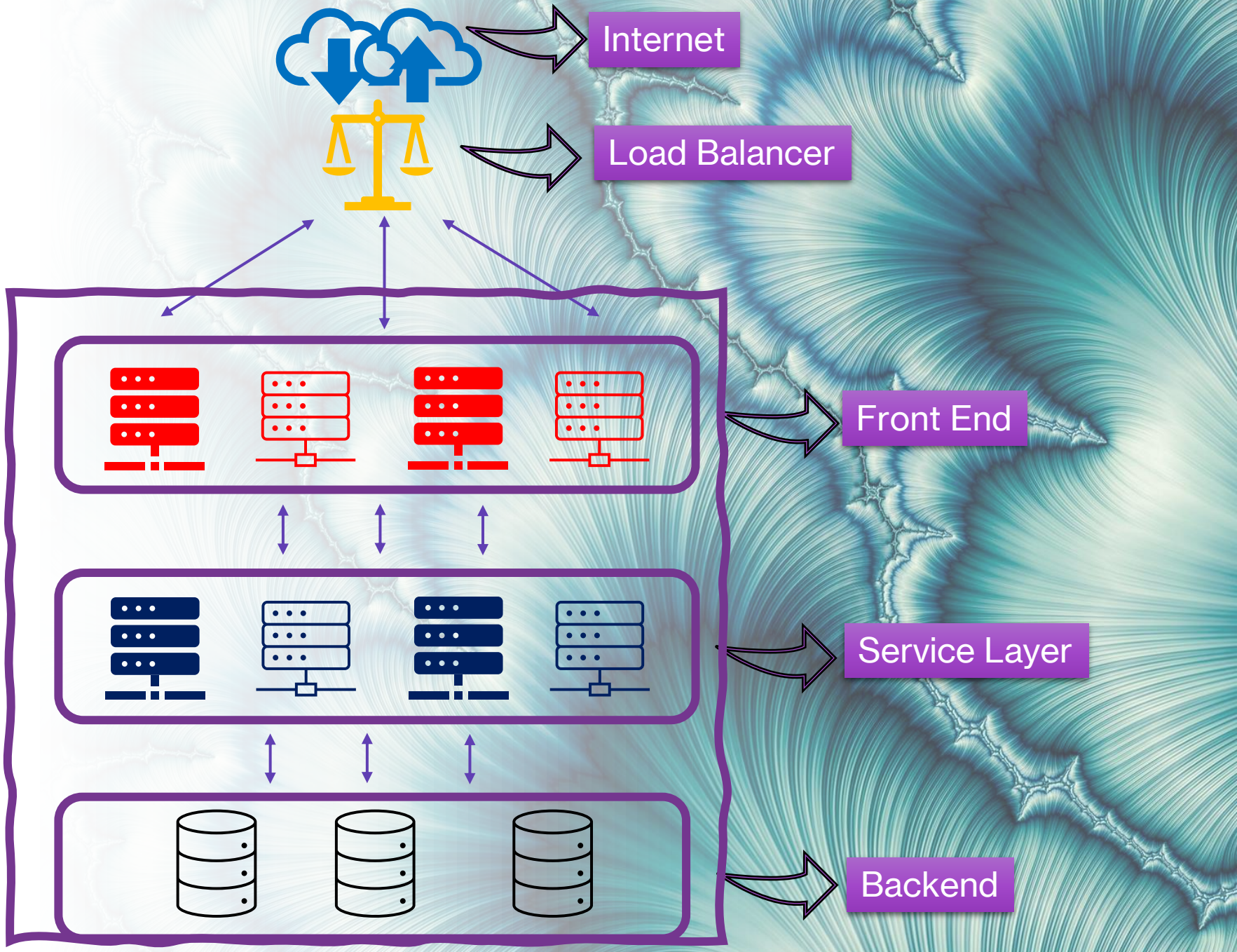
Terraform Usage Scenario



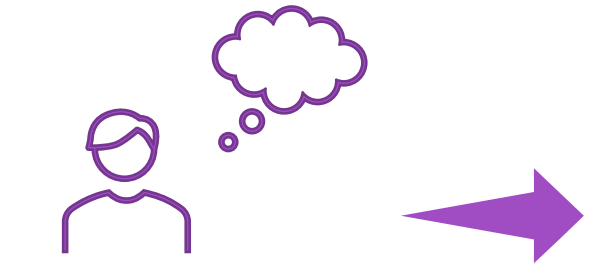
Terraform Usage Scenario

DevOps Engineer

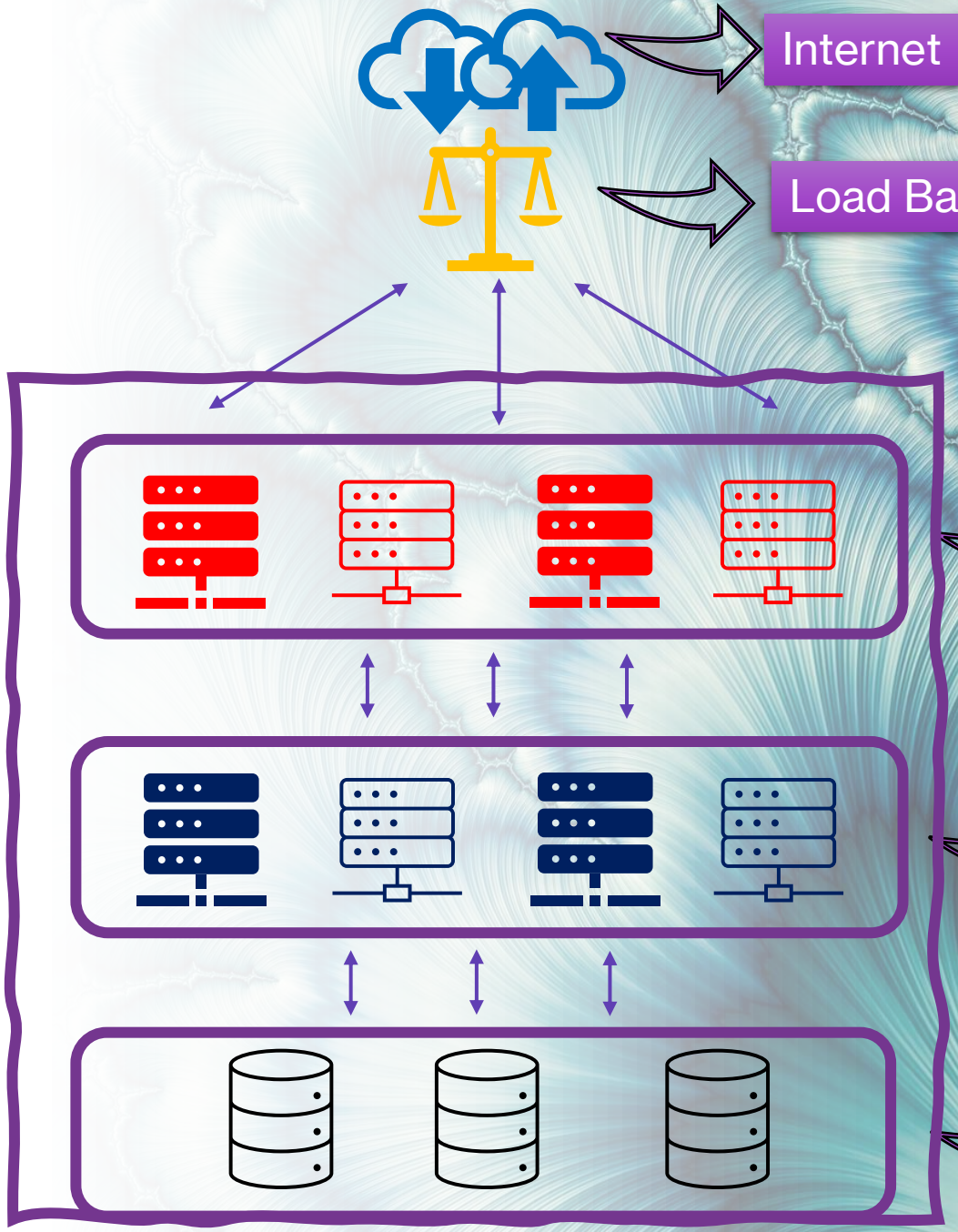




Current State Architecture

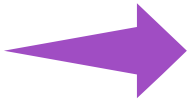


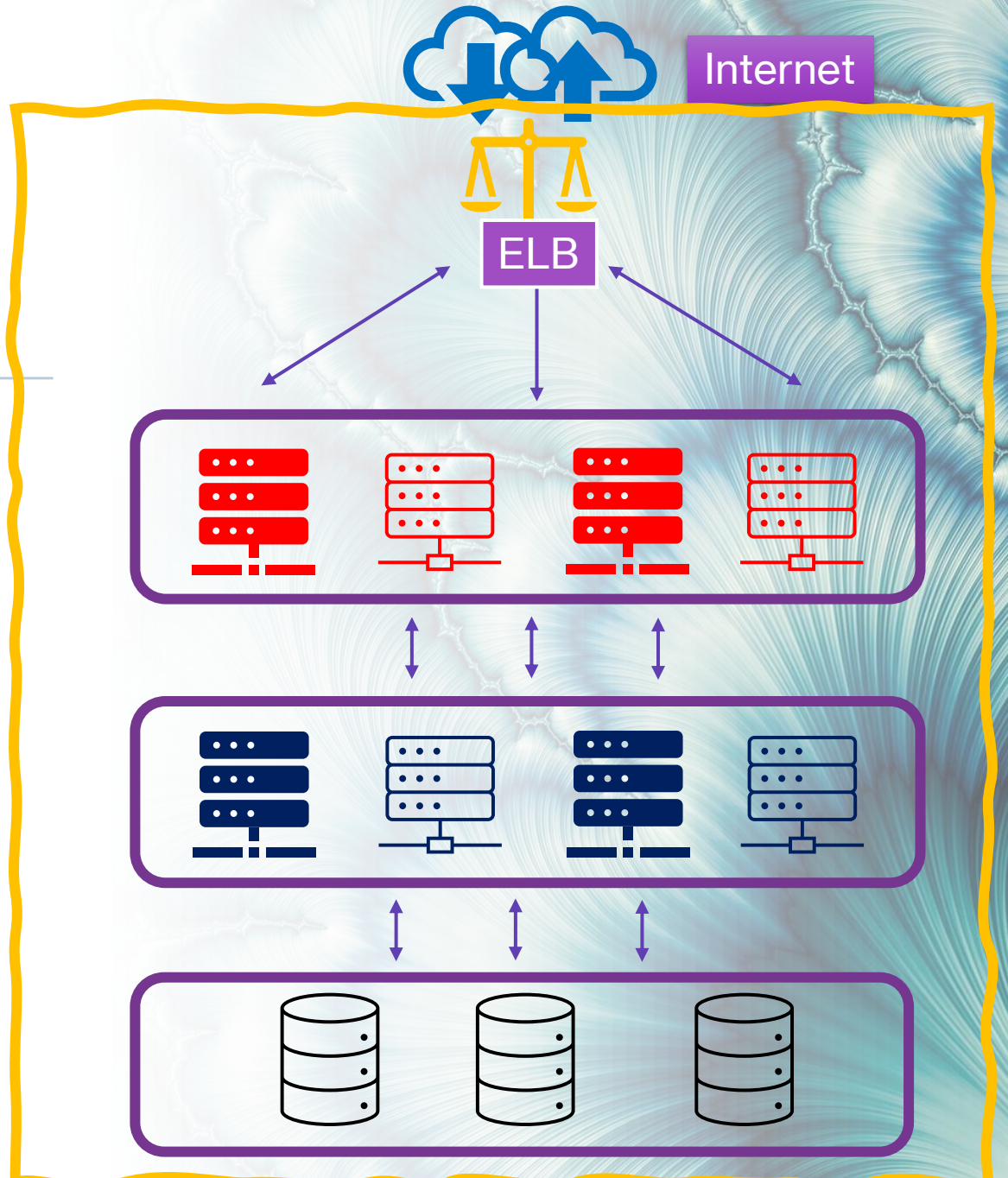
DevOps Engineer



Cloud Based Architecture

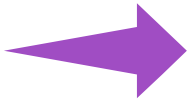
DevOps Engineer

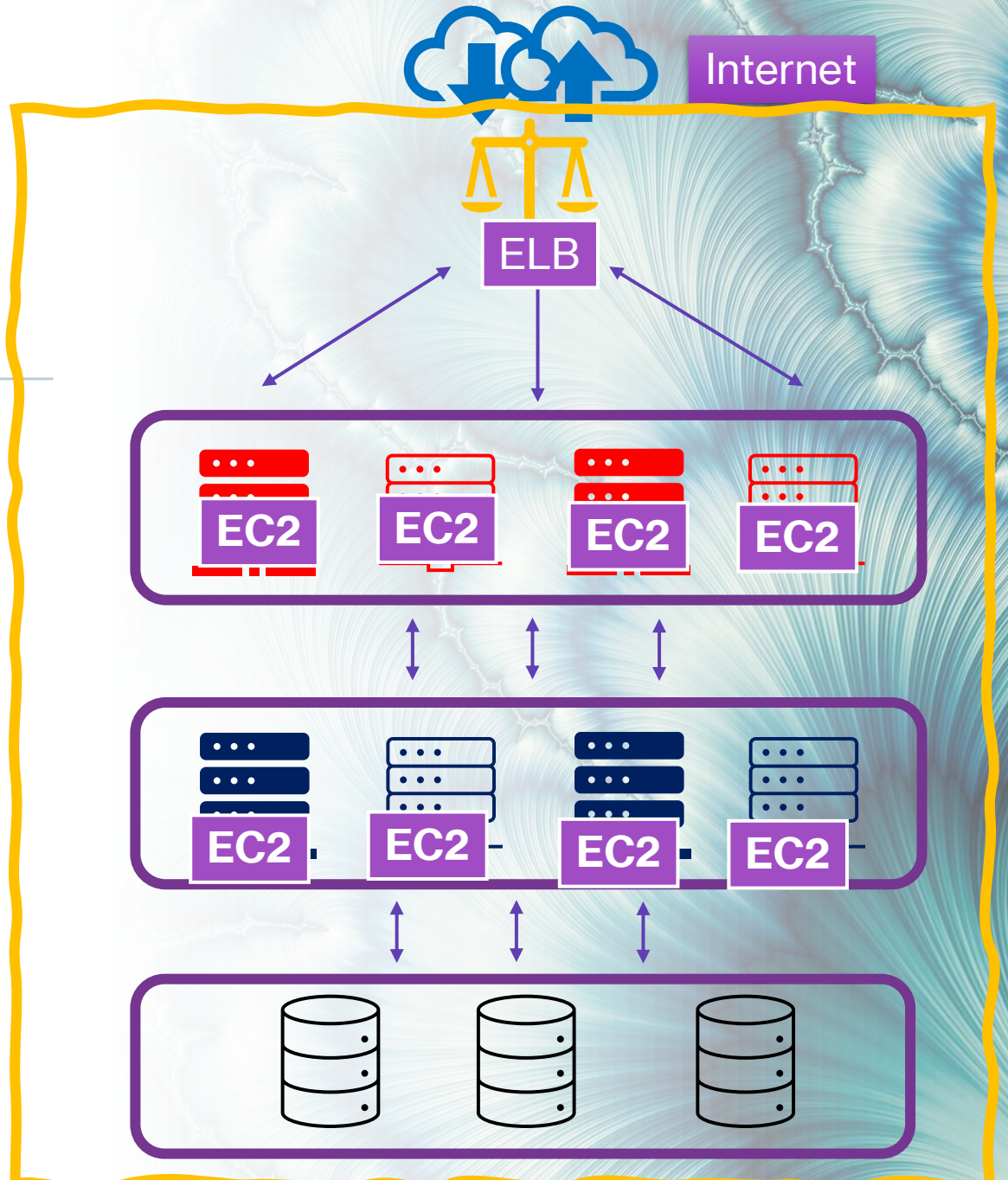




Cloud Based Architecture

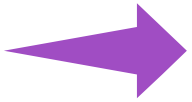
DevOps Engineer

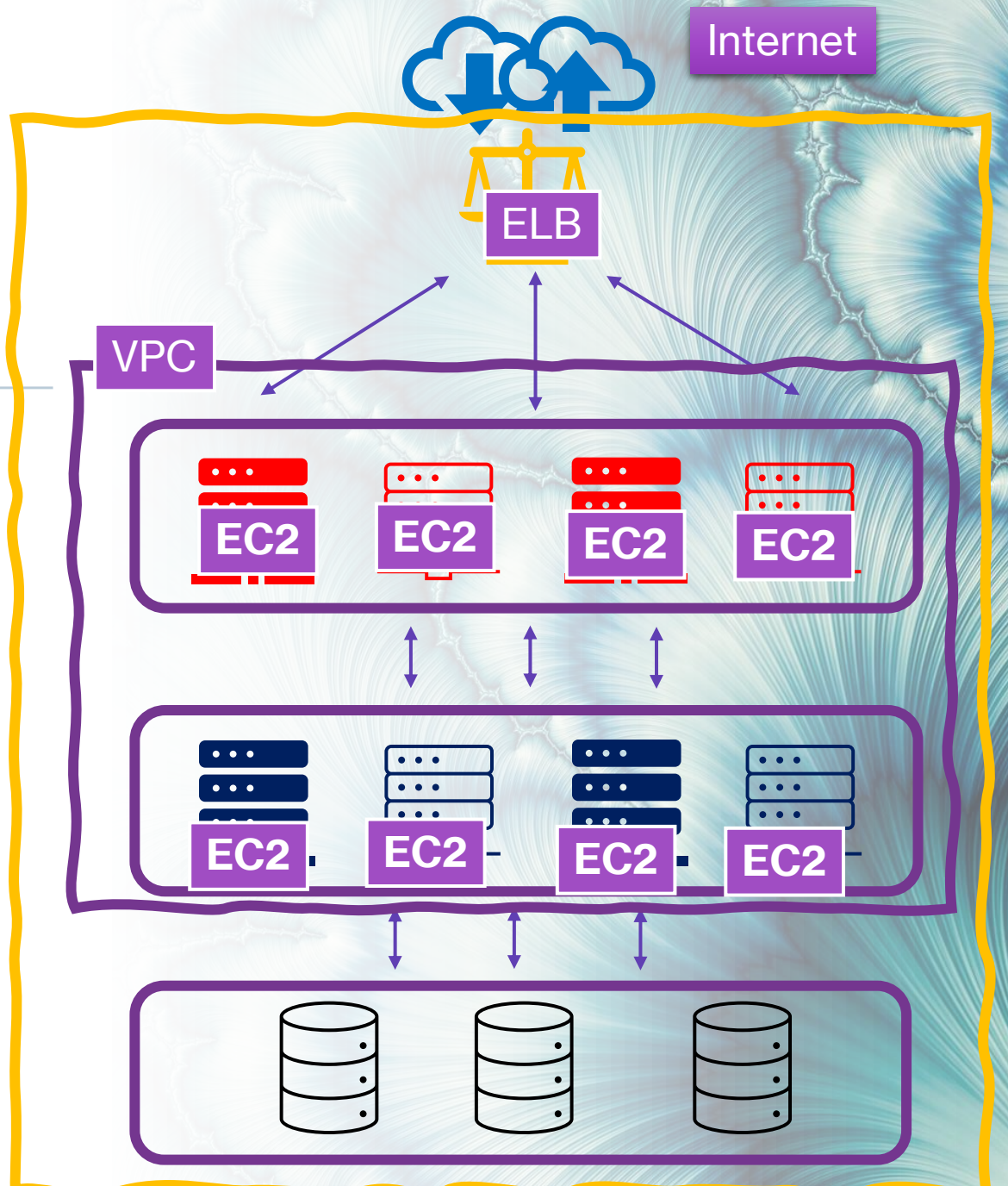




Cloud Based Architecture

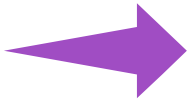
DevOps Engineer

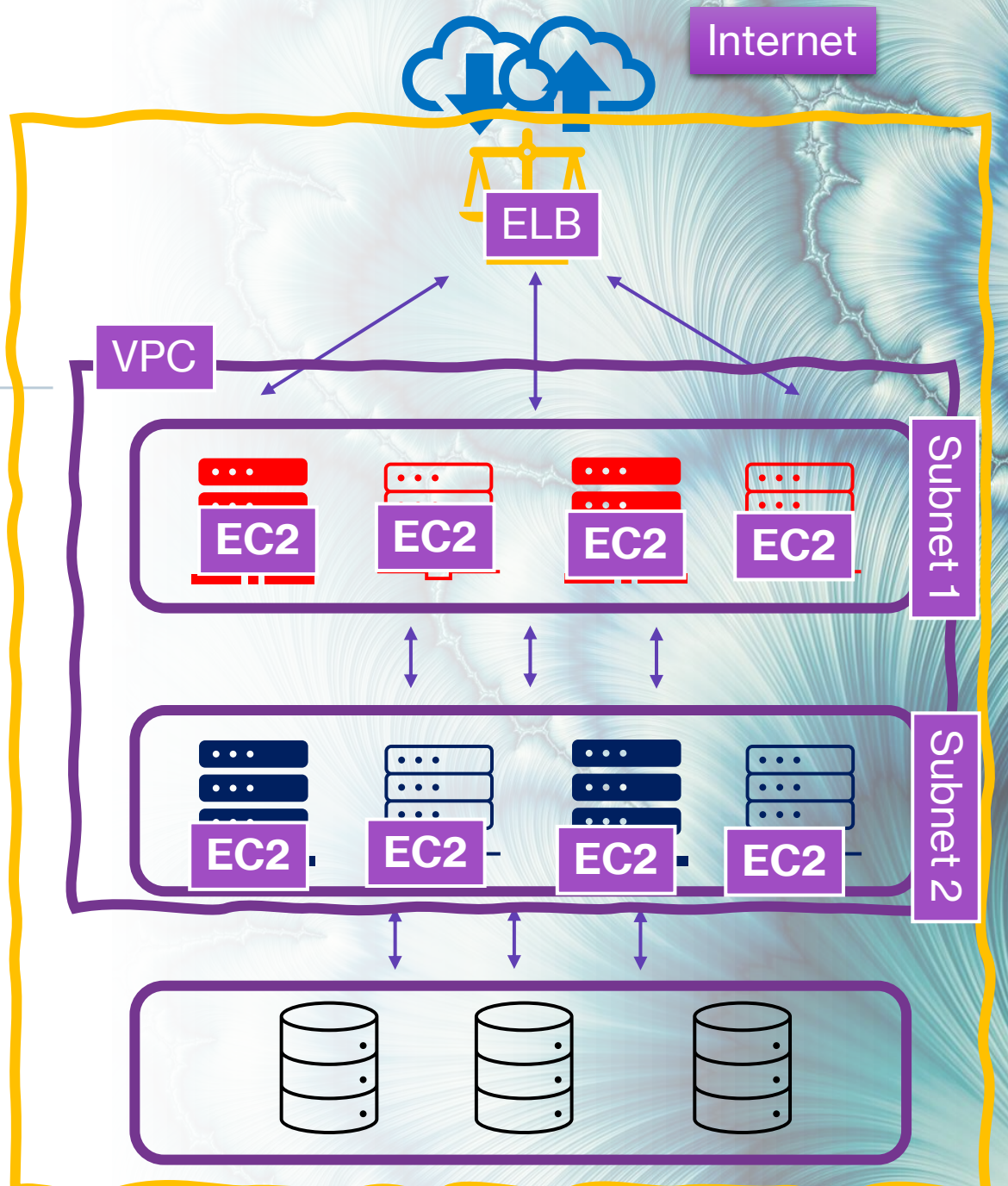




Cloud Based Architecture

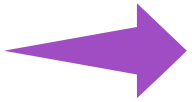
DevOps Engineer

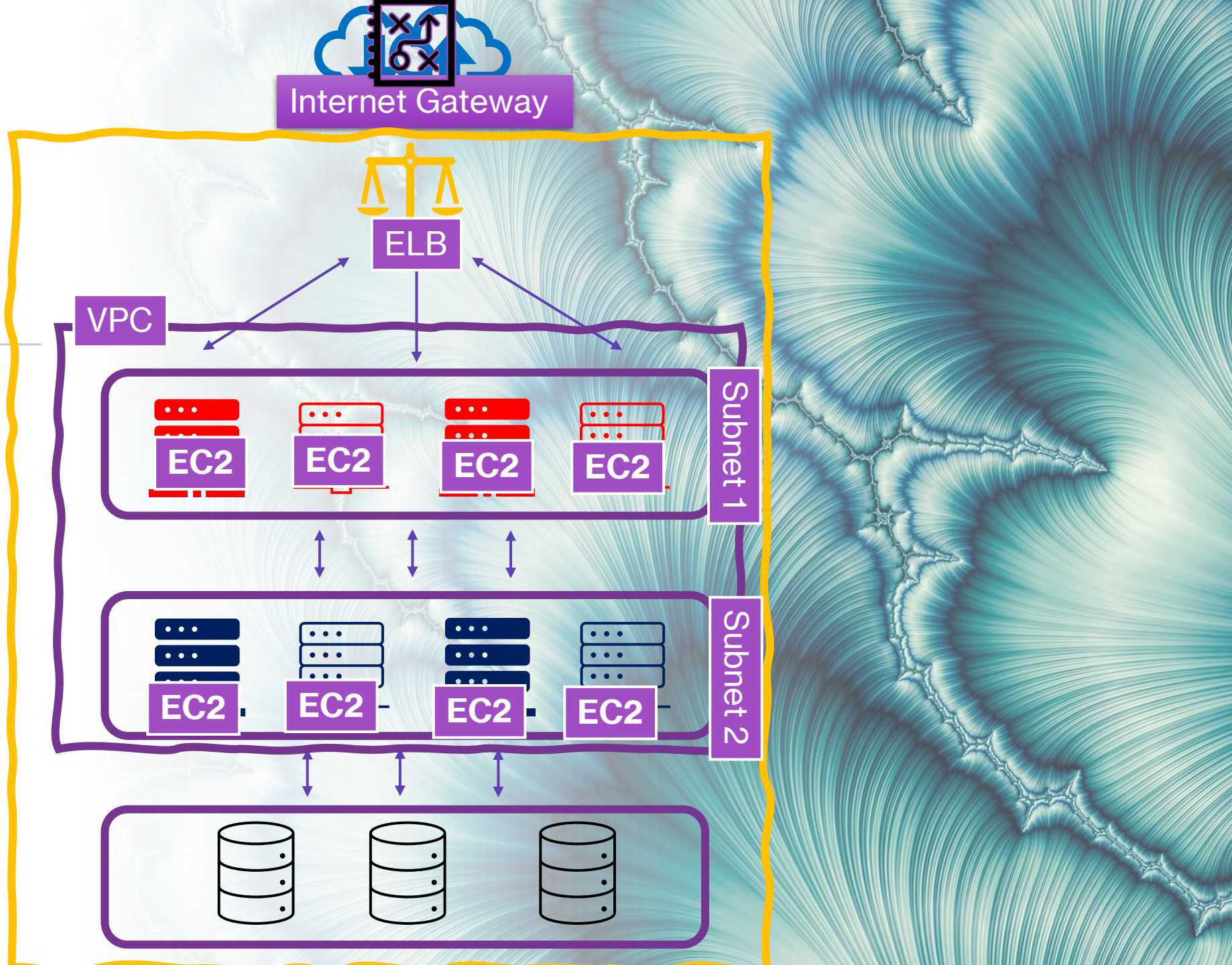




Cloud Based Architecture

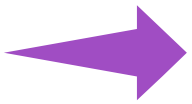
DevOps Engineer

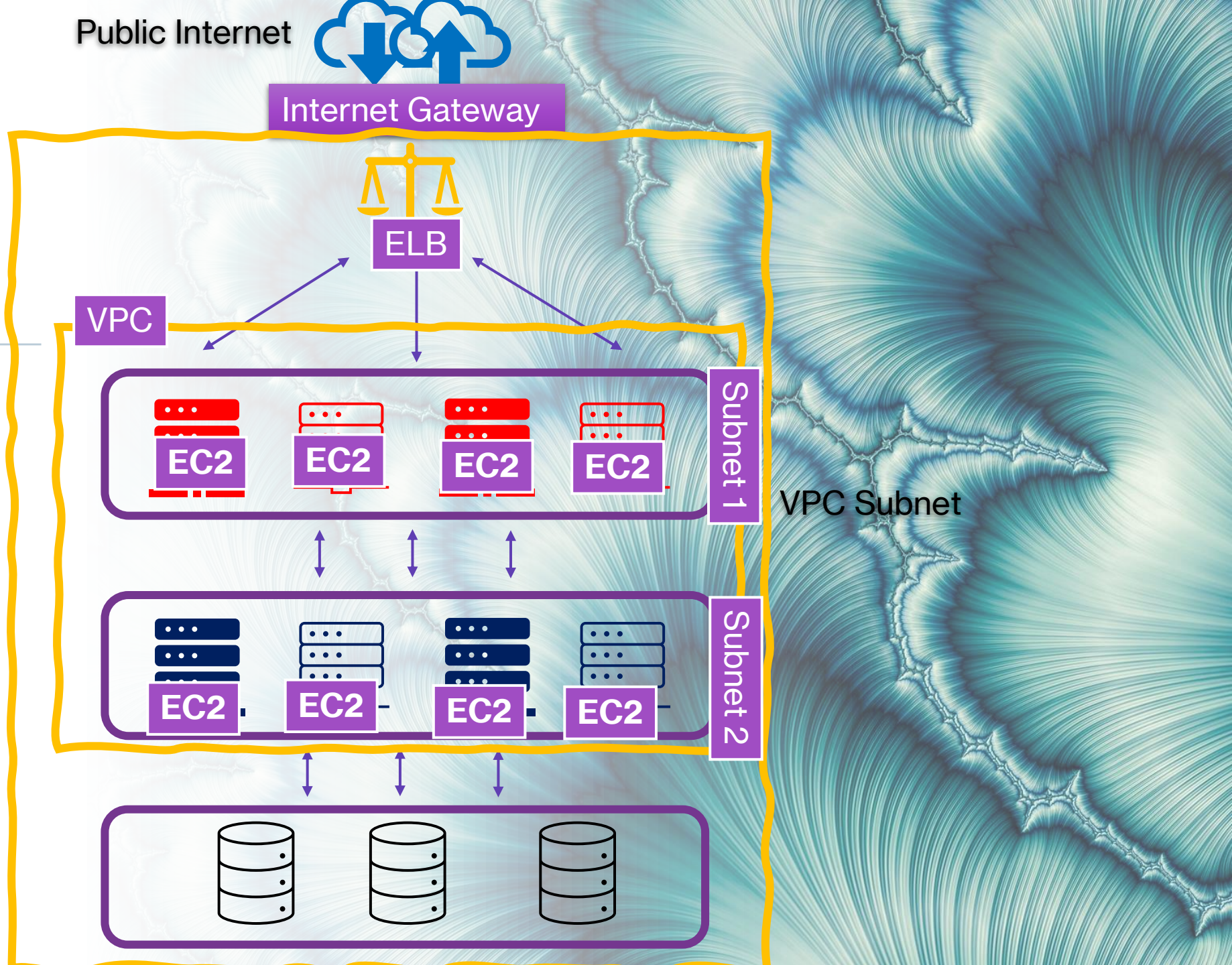




Cloud Based Architecture

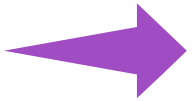
DevOps Engineer

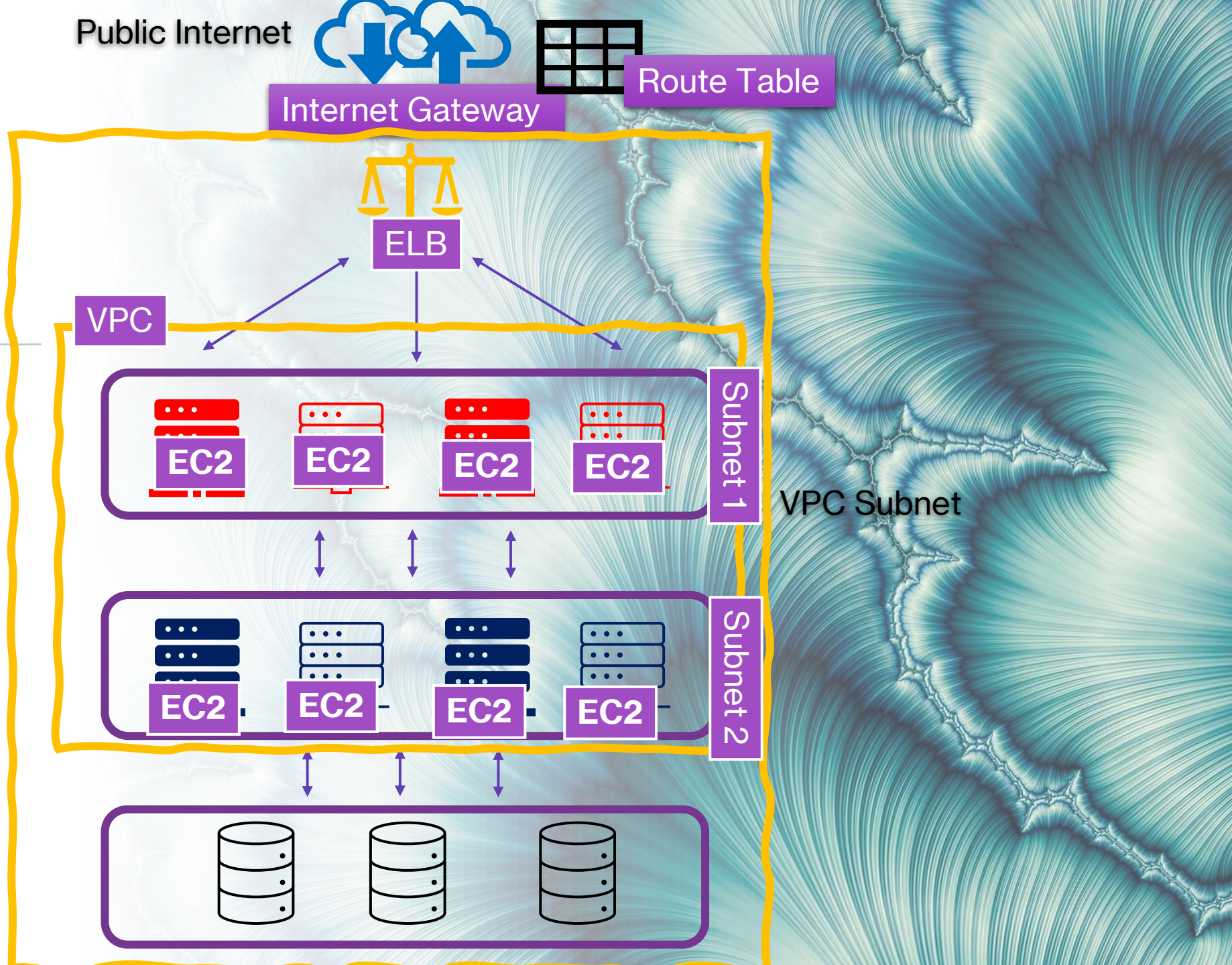




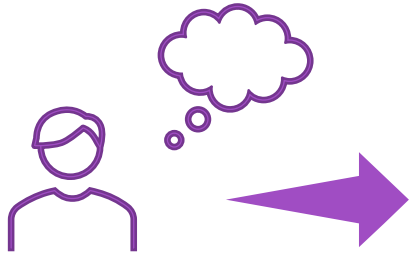
Cloud Based Architecture

DevOps Engineer

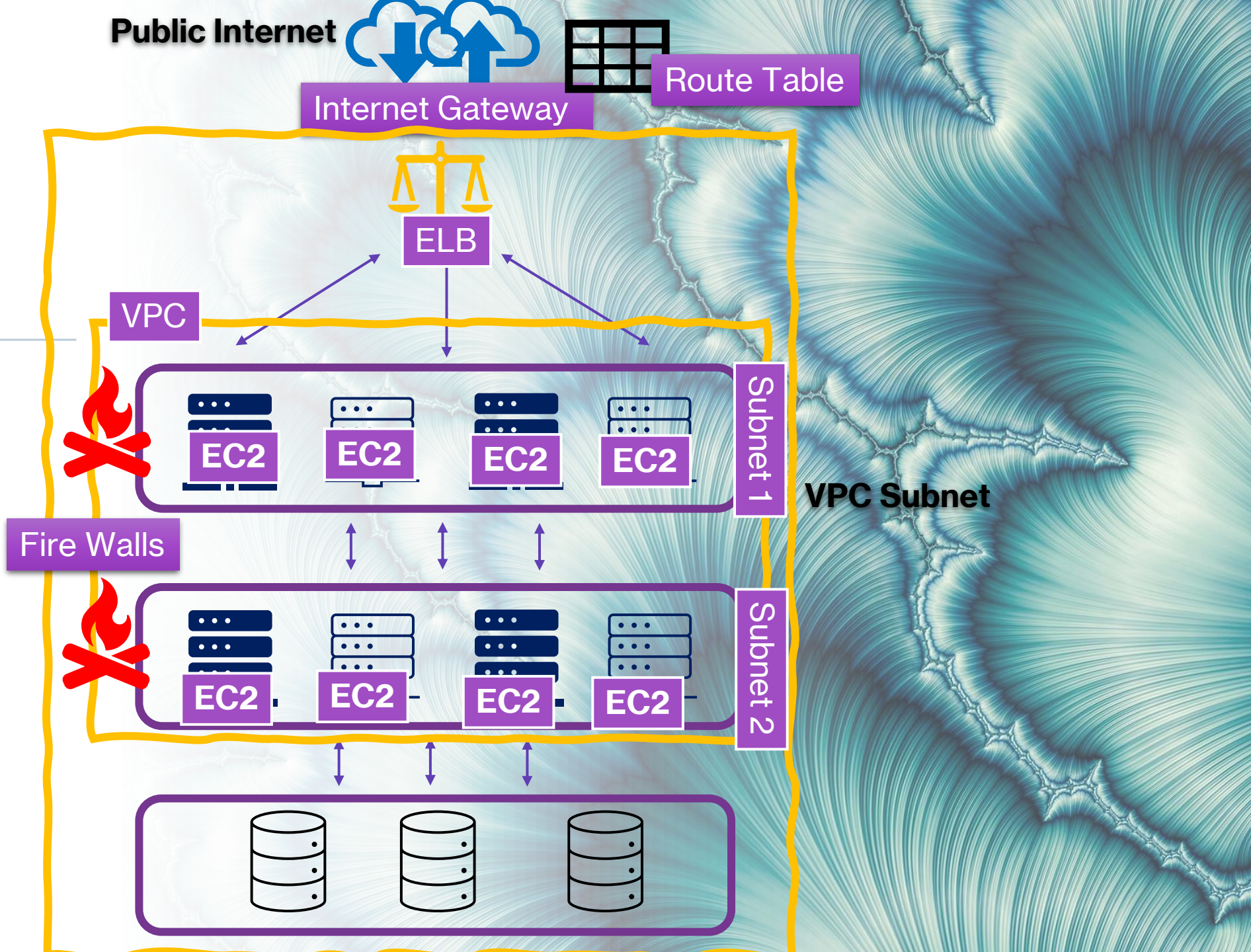




Cloud Based Architecture



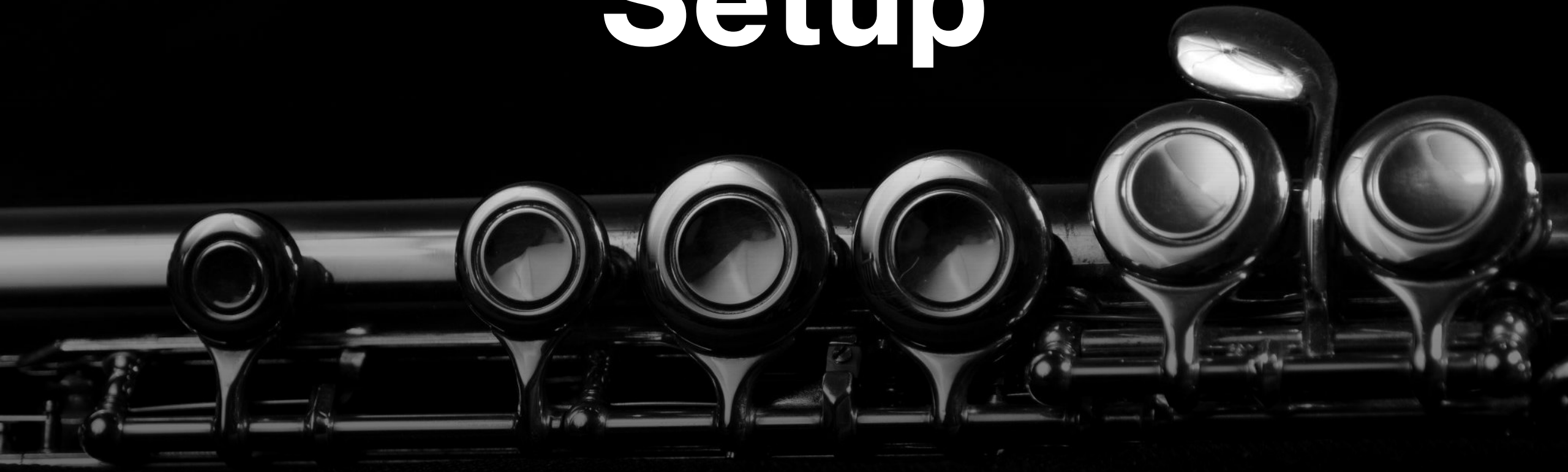
DevOps Engineer



Requirements

- Migrating an Application Infrastructure to AWS Cloud, Using resources like the existing one, this is called as Lift and Shift.
- Improvement in the availability
- Maintain separate Environments for Dev, QA, Stage and Prod

Setup



To get started with TF you need following things

- AWS Setup
- Windows Setup
- Having One IDE/ Text Editor (VS code)
- Basics of Terraform

AWS Setup

Creation of the account (Free tier services will be more than sufficient to get started with learning)

Creating a keypair in the region of your choice

Create a user and provide the programmatic access

Download and Install Terraform

- Using Package manager Chocolatey
  - Chocolatey is windows package manager
- Download exe contained Zip folder for your operating system specification(32bit or 64 bit)
  - Set the environment variable with the value of path of downloaded and extracted terraform
  - Check for the version using **terraform –version**

Setting up VS code

1. Download and install VS code
2. Search for extension terraform, install and enable the same
3. Create a project directory

Quiz

Question 1 of 4

What is the purpose of using a code repository with Terraform?

- ○ to manage code syntax
- ○ to manage cloud integration
- ○ to manage infrastructure builds
- ✓ to manage code versions

  Correct

  Terraform is based on code that might change often. Using a repository to manage code versions is useful.

Quiz

Question 2 of 4

Which specific AWS service is required for configuring communications with the Terraform application?

- S3
- IoT
- IAM

  Correct

  Identity and Access Management (IAM) is required with Terraform so that accesses to AWS resources is possible.
- EC2

Quiz

Question 3 of 4

Terraform is easily installed on Windows by using which approach?

- ○ Compile the application or use an interpreter at run time.
- ○ Download the program installer and run a wizard.
- ✓ Copy a single binary file to a folder in the system's path.

  Correct

  Terraform consists of a single binary file that can be placed in a folder and added to a system's path for ease of use.
- ○ Install the required dependencies along with the binary file.

Quiz

Question 4 of 4

What is Terraform?

- [x] an infrastructure management tool

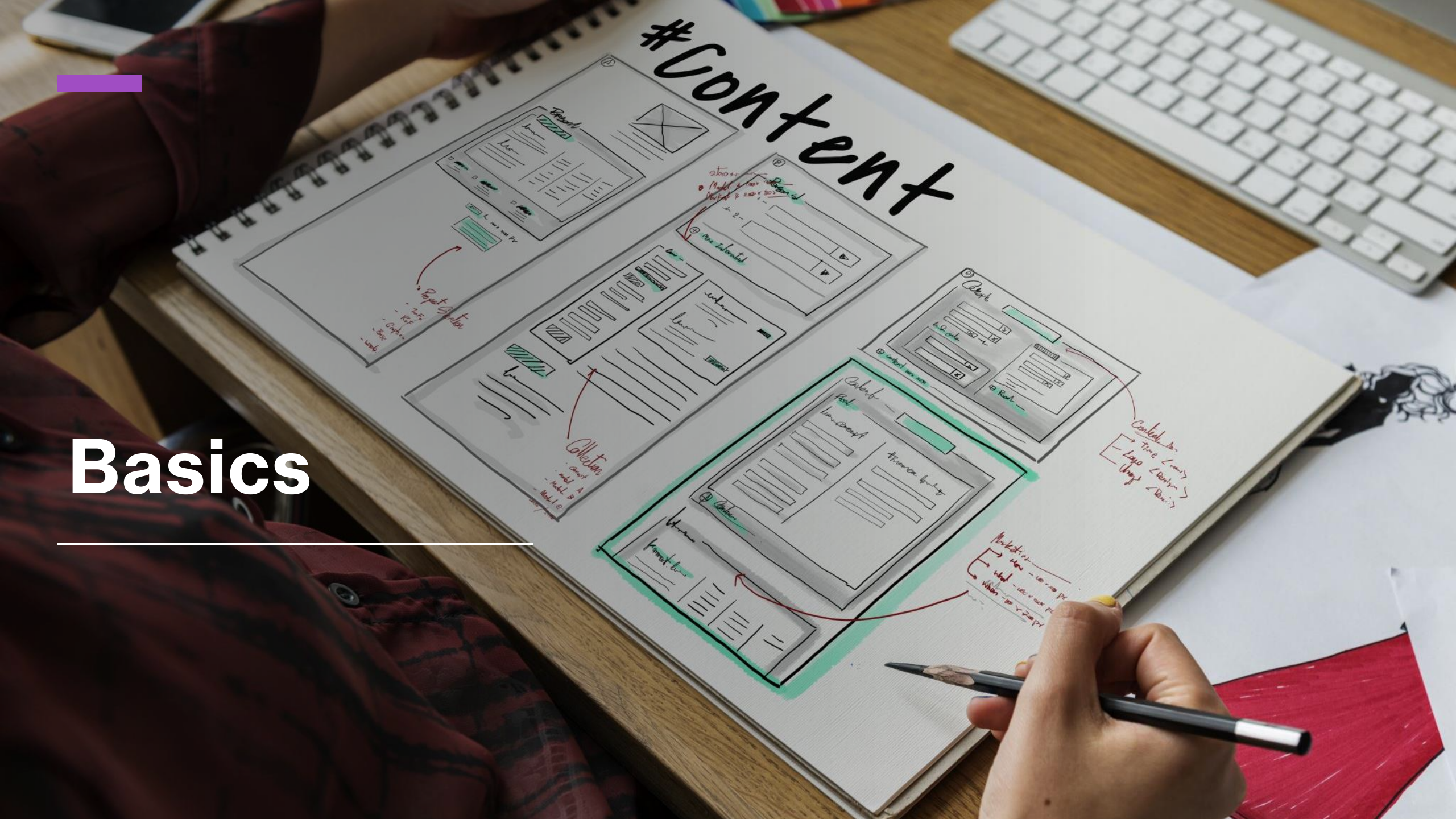
  Correct

  Terraform is an infrastructure management tool that can be used in an infrastructure as code environment.

- [ ] a configuration management tool
- [ ] a server-based management tool
- [ ] a network resource imaging tool

Repository

Clone this repository from my GitHub account



Basics

The Core Terraform Workflow

The core Terraform workflow has three steps:

- **Write** - Author infrastructure as code.
- **Plan** - Preview changes before applying.
- **Apply** - Provision reproducible infrastructure.

Write

- You write Terraform configuration just like you write code: in your editor of choice.

- It's common practice to store your work in a version controlled repository even when you're just operating as an individual.

Plan

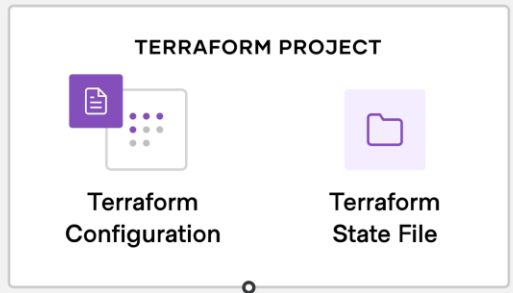
- When the feedback loop of the Write step has yielded a change that looks good, it's time to commit your work and review the final plan.

Apply

- After one last check, you are ready to tell Terraform to provision real infrastructure.

Write

Define infrastructure in configuration files



Plan

Review the changes Terraform will make to your infrastructure

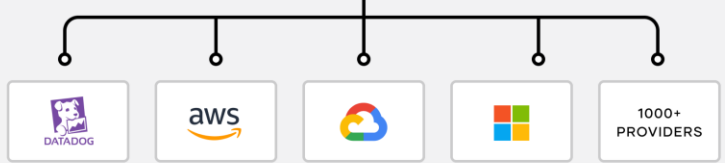
```
$ terraform plan

...

Terraform will perform
the following actions
```

Apply

Terraform provisions your infrastructure and updates the state file.



Initializing a Terraform repository

- Terraform init command
- AWS as a provider
- Try creating a resource S3 Bucket

Terraform Configuration language

- The main purpose of the Terraform language is declaring resources, which represent infrastructure objects.
- All other language features exist only to make the definition of resources more flexible and convenient.
- A *Terraform configuration* is a complete document in the Terraform language that tells Terraform how to manage a given collection of infrastructure.
- A configuration can consist of multiple files and directories.

Terraform Configuration language

Syntax

Configuration syntax

JSON Syntax

Style Conventions

Configuration Syntax

- The *native syntax* of the Terraform language, is a rich language designed to be relatively easy for humans to read and write.
- The Terraform language syntax is built around two key syntax constructs: arguments and blocks.

Json Configuration Syntax

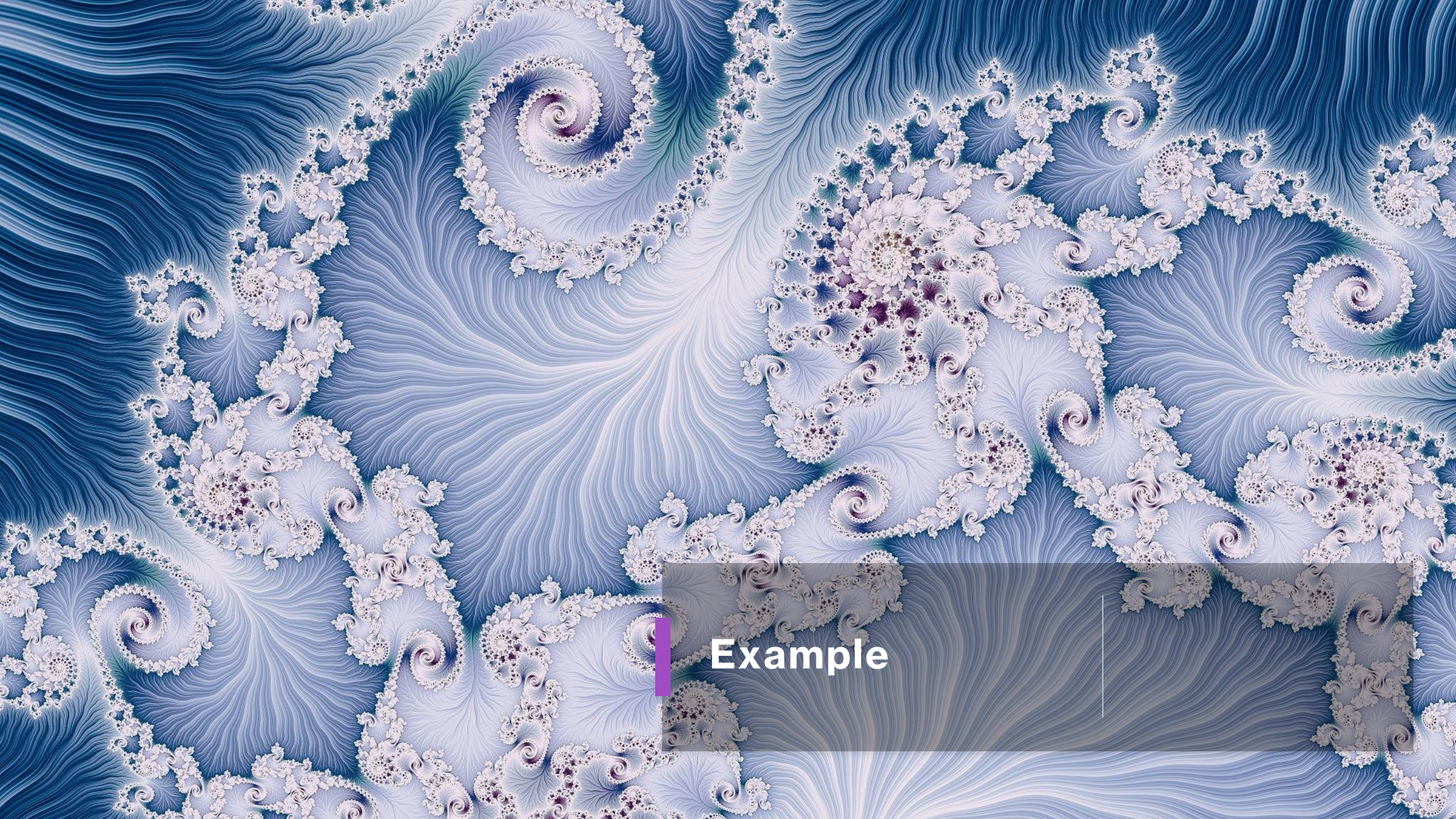
- The JSON syntax is defined in terms of the native syntax.
- Everything that can be expressed in native syntax can also be expressed in JSON syntax, but some constructs are more complex to represent in JSON due to limitations of the JSON grammar.
- Terraform expects native syntax for files named with a `.tf` suffix, and JSON syntax for files named with a `.tf.json` suffix.

Elements of Configuration

- *Blocks* are containers for other content and usually represent the configuration of some kind of object, like a resource.
- *Arguments* assign a value to a name. They appear within blocks.
- *Expressions* represent a value, either literally or by referencing and combining other values.

Files and Directories

- File Extension
  - Code in the Terraform language is stored in plain text files with the `.tf` file extension.
  - There is also a JSON-based variant of the language that is named with the `.tf.json` file extension.
- Directories and Modules
  - A *module* is a collection of `.tf` and/or `.tf.json` files kept together in a directory.
- Terraform always runs in the context of a single *root module.*


Example

Example

The syntax of the Terraform language consists of only a few basic elements:

```
resource "aws_vpc" "main" {
  cidr_block = var.base_cidr_block
}

<BLOCK TYPE> "<BLOCK LABEL>" "<BLOCK LABEL>" {
  # Block body
  <IDENTIFIER> = <EXPRESSION> # Argument
}
```

Copy

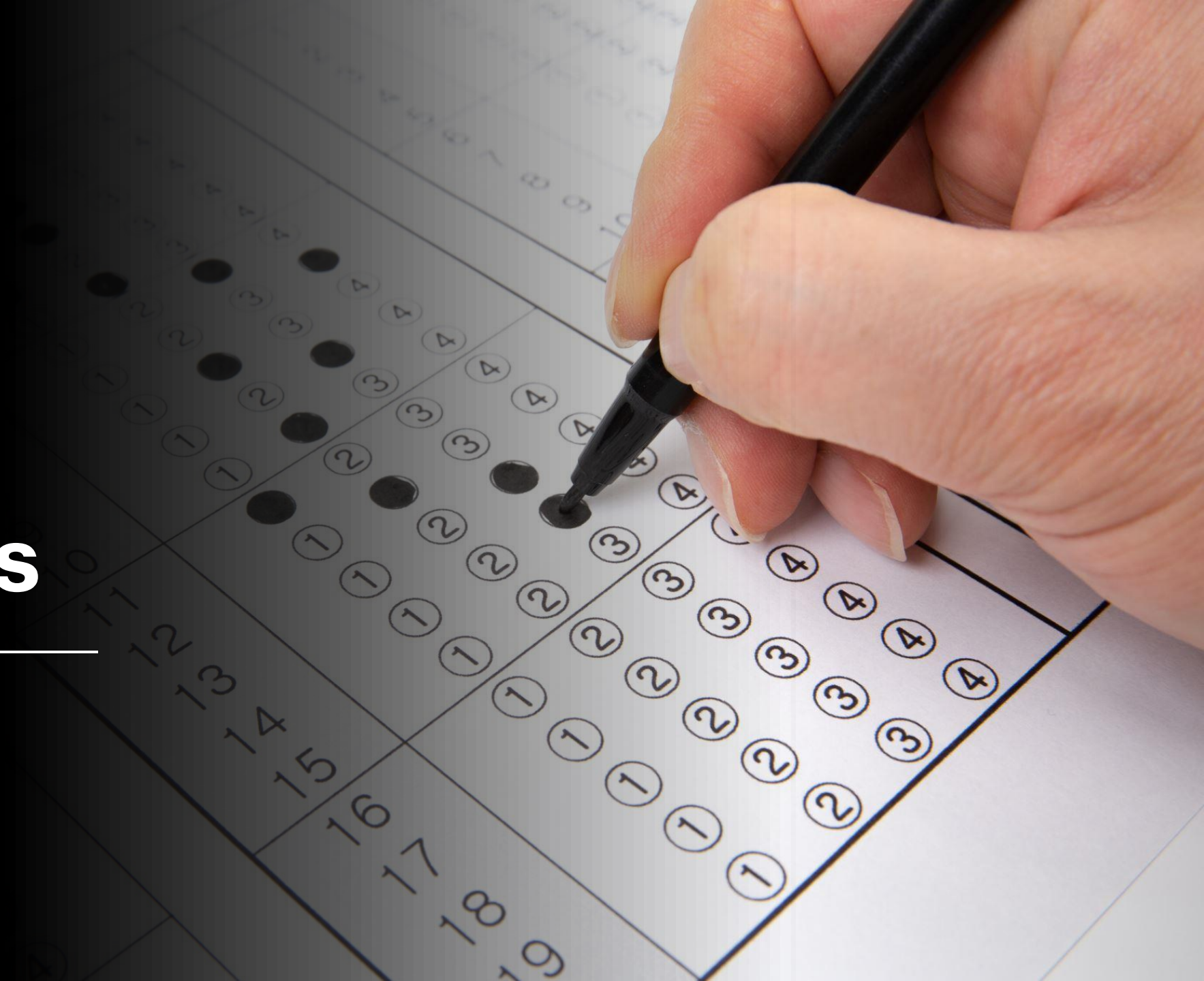
Working Directory Contents

- A Terraform configuration
- A hidden `.terraform` directory
- State data, if the configuration uses the default `local` backend.

What exactly goes into .terraform folder

- A hidden .terraform directory, which Terraform uses to manage cached provider plugins and modules, record which workspace is currently active, and record the last known backend configuration in case it needs to migrate state on the next run.
- This directory is automatically managed by Terraform, and is created during initialization.

Basic Commands



Terraform init

- The terraform init command is used to initialize a working directory containing Terraform configuration files.
- This is the first command that should be run after writing a new Terraform configuration

Usage

Usage: terraform init [options]

Terraform Plan

Usage

Usage: terraform plan [options]

- The `terraform plan` command creates an execution plan, which lets you preview the changes that Terraform plans to make to your infrastructure.
- By default, when Terraform creates a plan it:
  - Reads the current state of any already-existing remote objects to make sure that the Terraform state is up-to-date.
  - Compares the current configuration to the prior state and noting any differences.
  - Proposes a set of change actions that should, if applied, make the remote objects match the configuration.

Terraform apply

- The `terraform apply` command executes the actions proposed in a Terraform plan.
- The most straightforward way to use `terraform apply` is to run it without any arguments at all, in which case it will automatically create a new execution plan

Usage

Usage: terraform apply [options] [plan file]

Terraform Destroy

- The `terraform destroy` command is a convenient way to destroy all remote objects managed by a particular Terraform configuration.

Usage: terraform destroy [options]

Quiz

Question 1 of 6

Which statement describes how Terraform works at a high level?

- [x] It compares code to existing infrastructure and makes appropriate changes.
  Correct
  When deploying an infrastructure via code, Terraform doesn't just go and make changes. First, it checks the known status by comparing local code to the current infrastructure.
- [ ] It defines an infrastructure as code by configuring device settings.
- [ ] It applies infrastructure described in code to any defined systems.
- [ ] It applies code to an existing infrastructure by making changes.

Quiz

Question 2 of 6

How is a Terraform graph created?

- [x] by using the 'graph' command and then a graph generator

  Correct

  After the graph command is used, the graph code can be pasted to create a graph visual with a graph generator.

- [ ] by using the 'graph' command followed by the 'apply' command
- [ ] by using a graph drawing utility plug-in with Terraform
- [ ] by using the built-in 'graph' command and defining the graph size

Quiz

Question 3 of 6

Which step is required before executing the code of a Terraform project for the first time?

- implementation
- ✗ installation
  Incorrect
  Terraform does need to be installed to use it; however, in this case, the focus is on executing code in the project.
- integration
- ✓ initialization
  Correct
  Terraform requires a project to first be initialized by finding any related .tf files. This is done by running the init command.

Quiz

Question 4 of 6

Which Terraform command is used to reach a desired configuration?

- ✓ apply'
  Correct
  The apply command is used in Terraform to reach a desired configuration for deployment. This means Terraform performs the known actions to take.
- ○ state'
- ○ init'
- ✗ plan'
  Incorrect
  The plan command is used in Terraform to create a desired configuration for deployment. This means Terraform knows the required actions to take.

Quiz

Question 5 of 6

In the image displayed, what visual element of Terraform is being utilized?

```
"[root] aws_s3_bucket.tf_course" [label = "aws_s3_bucket.tf_course", shape = "box"]
"[root] provider.aws" [label = "provider.aws", shape = "diamond"]
"[root] aws_s3_bucket.tf_course" -> "[root] provider.aws"
"[root] meta.count-boundary (EachMode fixup)" -> "[root] aws_s3_bucket.tf_course"
"[root] provider.aws (close)" -> "[root] aws_s3_bucket.tf_course"
"[root] root" -> "[root] meta.count-boundary (EachMode fixup)"
"[root] root" -> "[root] provider.aws (close)"
```

- label
- graph

  Correct

  A graph is a visual representation of a Terraform infrastructure project. To create a graph, the graph command can be used. The code output can then be used to build a visual graph.
- shape
- chart

Quiz

Question 6 of 6

The following image shows a sample of everything that Terraform knows about a particular infrastructure. What is this information referred to as?

- ○ deployment
- ✓ state
  Correct
  When an implementation of an infrastructure is planned or executed, Terraform uses a state file to determine that the current status is correct.
- ○ systems
- ○ plan

Terraform Style

- Indent two spaces
- Single meta-arguments first (count = 2)
- Block meta-arguments at last (lifecycles)
- Blank lines for clarity
- Group single arguments
- Achieving readability
- Lining up equal signs

```
resource “aws_instance” “web” {
  count = 2

  ami           = “abc123”
  instance_type = “t2.micro”

  network_interface {
    # ...
  }

  lifecycle {
    create_before_destroy = true
  }
}
```

Getting started with a TF file

Resources

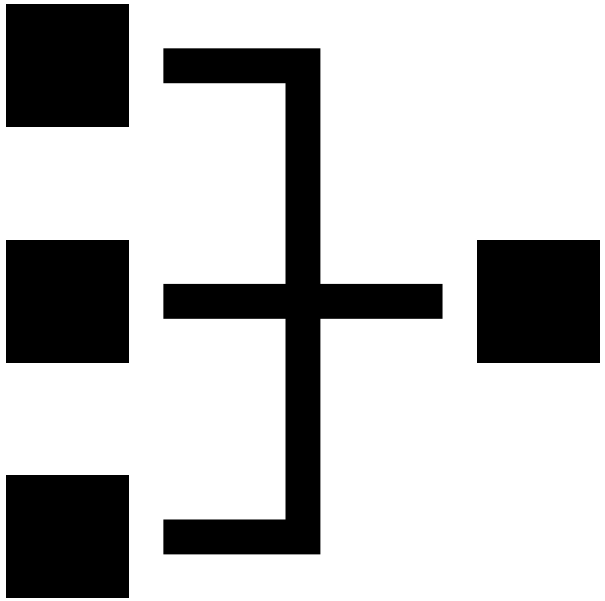
- Building blocks of terraform
- Define “what” of our infrastructure
- Settings varies with different provider

```
provider “aws” {
  profile    = “default”
  region     = “us-west-2”
}

resource “aws_s3_bucket” “tf-course” {
  bucket = “samuelson-terraform-20191107”
  acl    = “private”
}
```

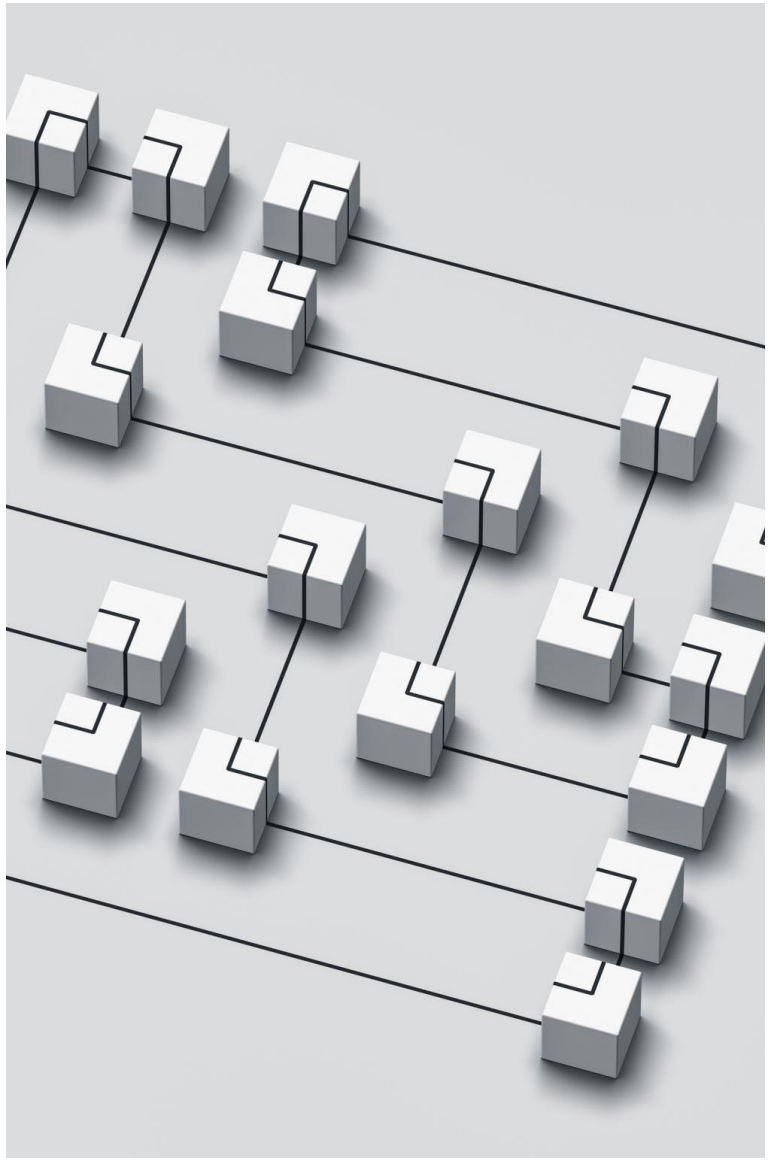
Resources Types

- Buckets
- VPCs
- Security Groups
- Instances
- IAM Users



VPCs and Subnets

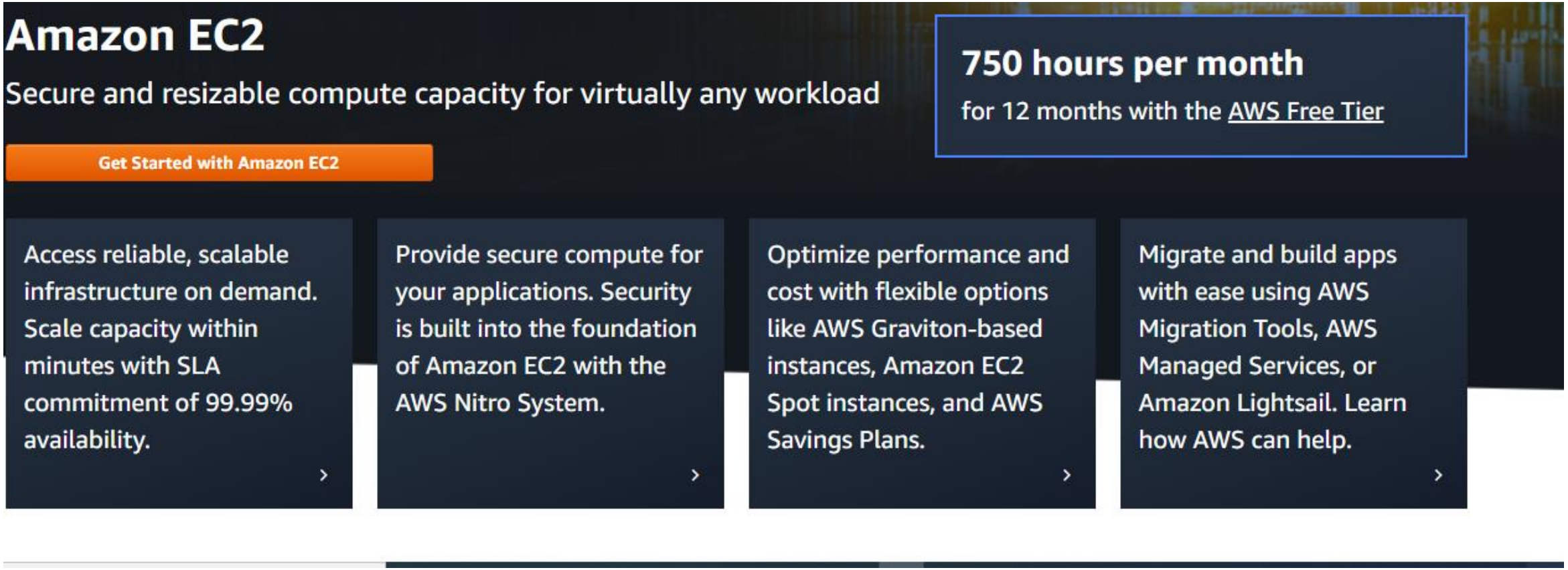
- A *virtual private cloud* (VPC) is a virtual network dedicated to your AWS account.
- You can launch your AWS resources, such as Amazon EC2 instances, into your VPC.
- You can specify an IP address range for the VPC, add subnets, associate security groups, and configure route tables.
- A *subnet* is a range of IP addresses in your VPC.



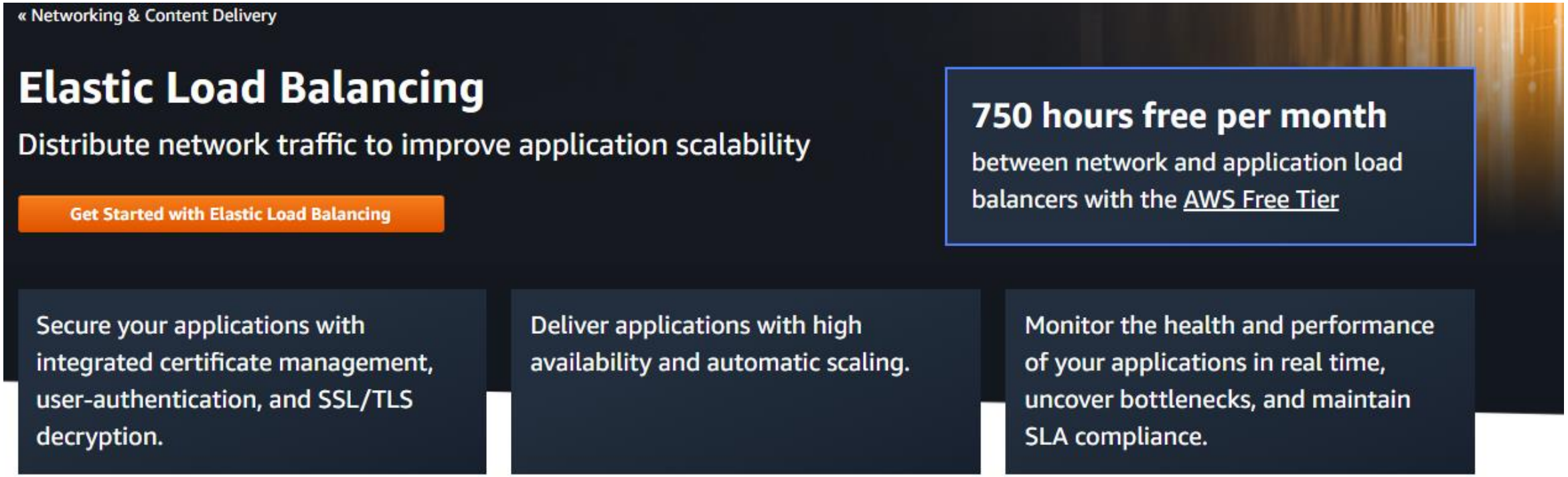
Security Group

- A *security group* acts as a virtual firewall, controlling the traffic that is allowed to reach and leave the resources that it is associated with.
- For example, after you associate a security group with an EC2 instance, it controls the inbound and outbound traffic for the instance.
- When you create a VPC, it comes with a default security group.

Instance


Amazon EC2
Secure and resizable compute capacity for virtually any workload
Get Started with Amazon EC2
750 hours per month
for 12 months with the AWS Free Tier
Access reliable, scalable infrastructure on demand. Scale capacity within minutes with SLA commitment of 99.99% availability.
Provide secure compute for your applications. Security is built into the foundation of Amazon EC2 with the AWS Nitro System.
Optimize performance and cost with flexible options like AWS Graviton-based instances, Amazon EC2 Spot instances, and AWS Savings Plans.
Migrate and build apps with ease using AWS Migration Tools, AWS Managed Services, or Amazon Lightsail. Learn how AWS can help.

Load Balancer




Quiz

Question 1 of 9

What is the significance of the "ami" information when selecting and deploying a server instanc

- the instance power
- the server type
- the image used

  Correct

  When deploying an instance, the "ami" information is copied from AWS and placed in code to indicate w image is used to create the instance.
- the cost incurred

Question 2 of 9

An AWS security group can be viewed as _____.

- a firewall instance that protects the defined region within AWS
- a resource group that can be assigned permissions
- ✓ a firewall for instances that can be assigned to resources

  Correct

  In AWS, an instance can have a security group applied. Doing so is like having a firewall with security rule place for specifically assigned resources.
- a resource that uses a firewall to protect any related instances

Question 3 of 9

The proper indentation in a Terraform file is how many spaces?

- ○ two tabbed spaces
- ○ a tabbed space
- ✓ two spaces

  Correct

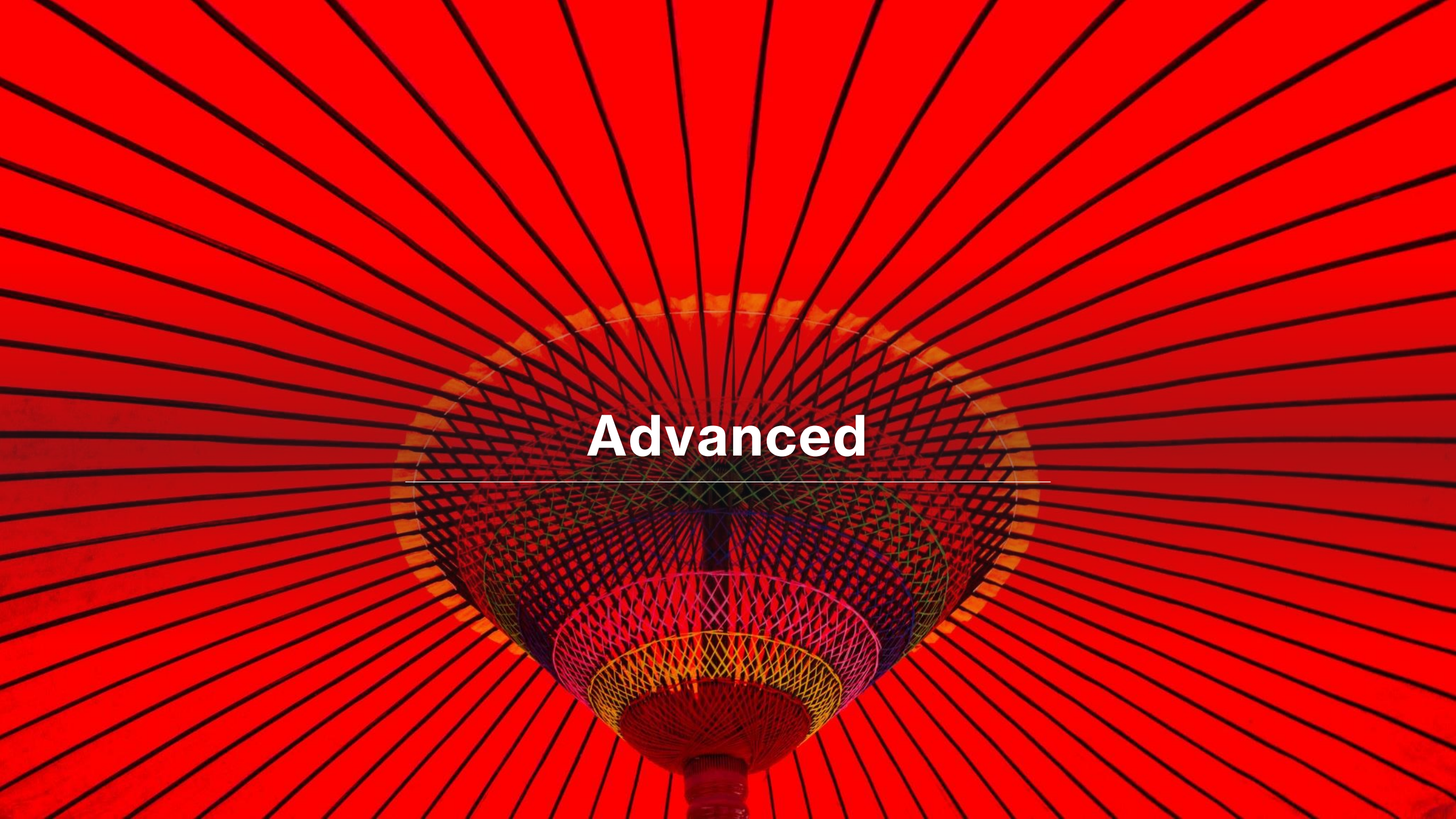
  Coding usually follows style guidelines. In Terraform code, all line indents should be two spaces.
- ○ five spaces

What popular AWS resource type is being used in the image?

```
resource “aws_instance” “web2” {
  ami           = “${data.aws_ami.ubuntu.id}”
  instance_type = “t2.micro”
}
```

- EIP
- S3
- EC2

Correct



Advanced

Advanced Concepts

- Variables
- Split out your data
- Modules

Final Project

3 tier Web Architecture with CICD integrated

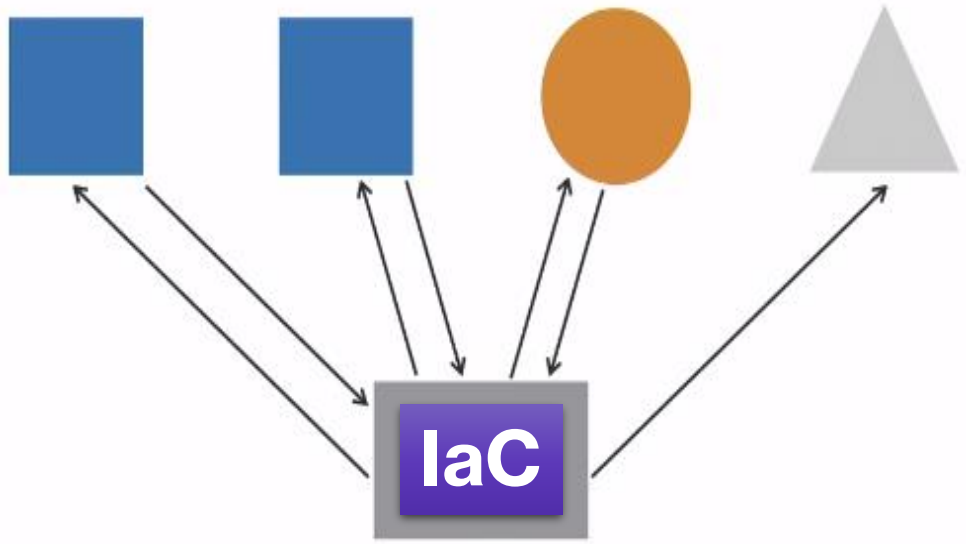
High Level Look

Initial Deployment

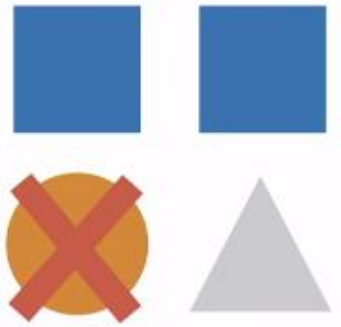
- Terraform init
- Terraform plan/ apply

How Terraform Works

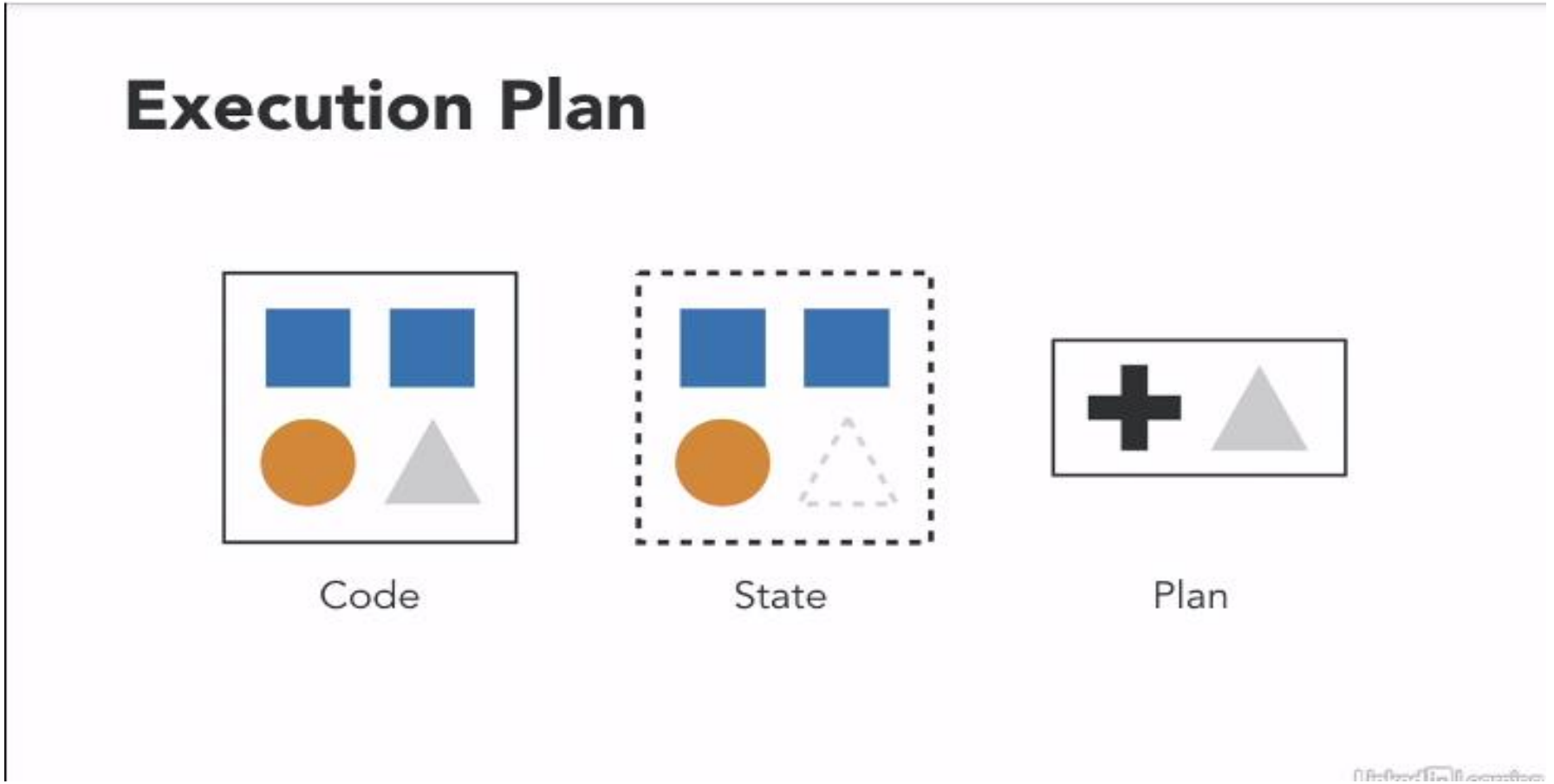
Infrastructure as Code



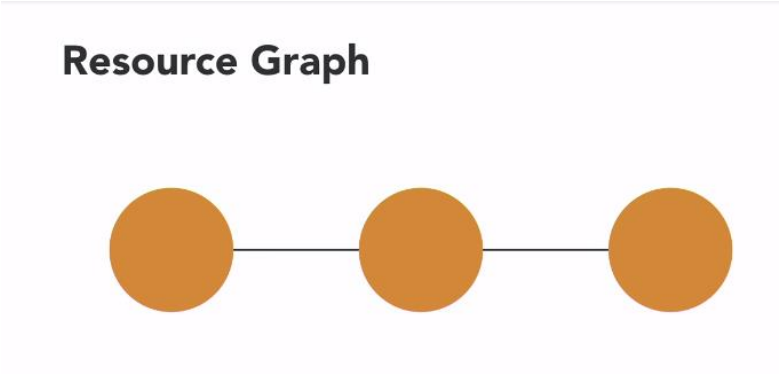
Execution Plan

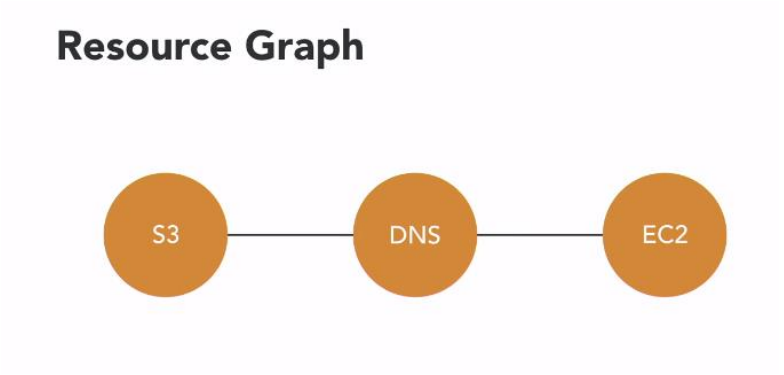


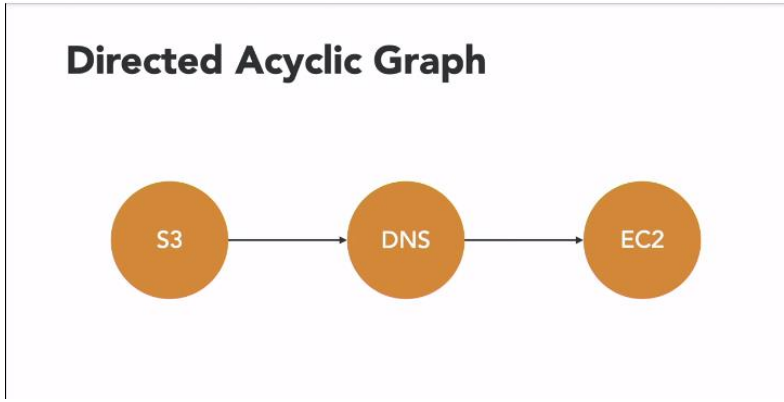
How Terraform Works



How actually it does that

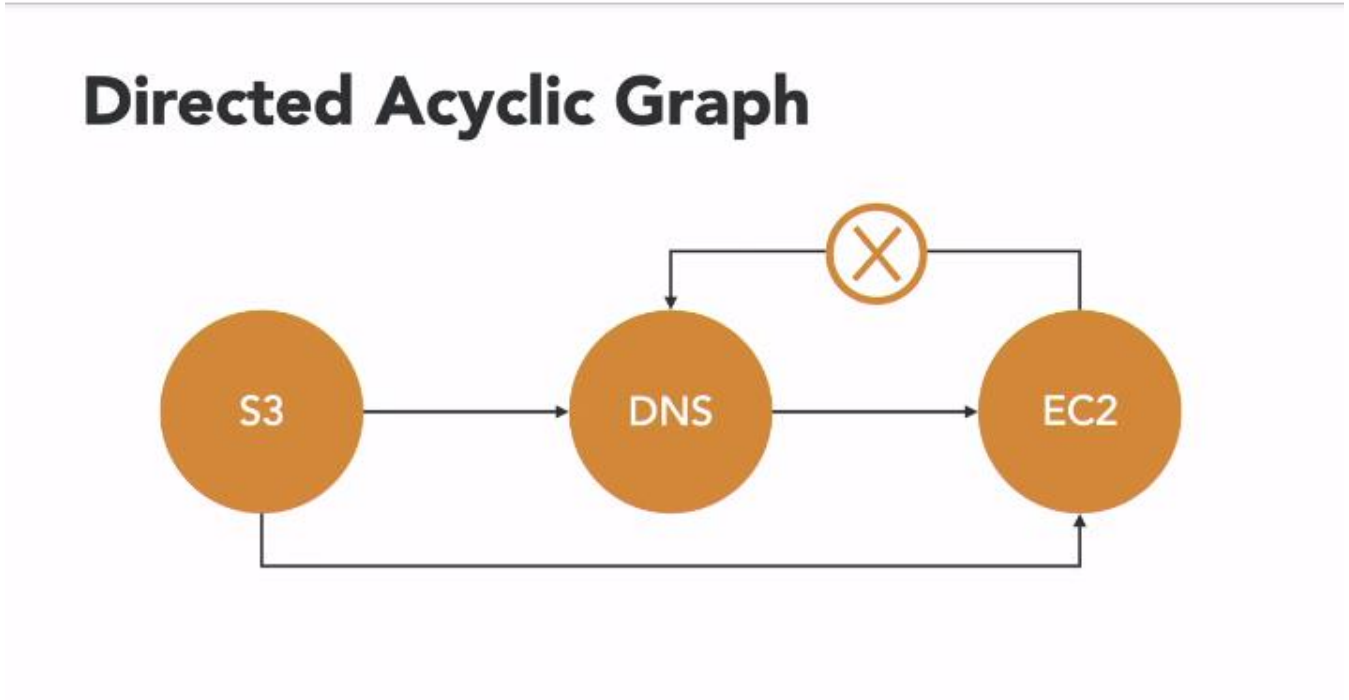


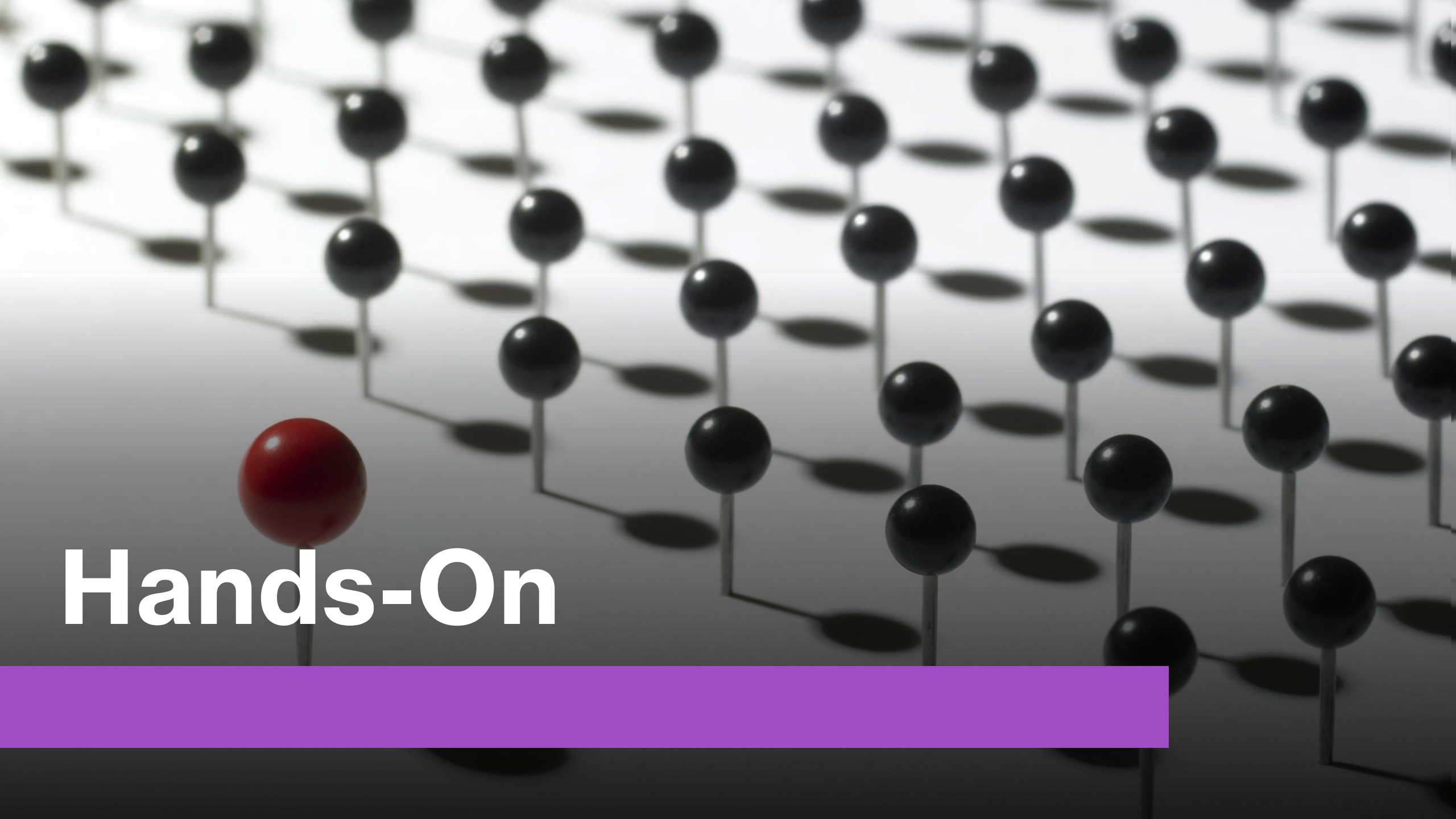




How actually it does that

- Data structure called directed acyclic graph
- A graph is a series of nodes
- Here in terraform each node is a resource
- Graph is directed and also acyclic (no cycles)





Hands-On

Try Some Commands

- Terraform apply/ plan –help
  - Terraform plan –destroy
  - Terraform plan –destroy –out=name.tfplan (binary file)
- Terraform show “plan-name.tfplan”
- Terraform apply

Terraform state

- State of your resources and terraform
- Two different things
  - Reality of your infrastructure, state of your resources on AWS
  - Local terraform configuration with respect to your resources
- Inspecting your state files which is in JSON format (local files)
- Terraform state (for remote)
  - List
  - Show

Terraform Graph

- Terraform graph
- Copy the content
- Open webgraphviz.com
- Paste the copied content to the text box
- Visualize it using generate graph

Terraform apply

- Terraform apply “somepla”
- Terraform plan –destroy “someplan”
- Terraform apply someplan
- Terraform apply –auto-approve

Terraform Console

- The `terraform console` command provides an interactive console for evaluating expressions.

Usage

Usage: terraform console [options]


Quiz

You answered 5 of 5 questions correctly. Continue watching Retake quiz

Question 1 of 5

A company is working on two web-based applications on a single cloud network. Which code snippet would be the best candidate to include in a global environment when modularizing code?

```
provider "aws" {
 profile = "default"
 region = "us-west-2"
}
```

```
resource "aws_default_vpc" "default" {}
```

```
resource "aws_default_subnet" "default" "default_az1 {}
```

```
variable "whitelist" {
 type = list(string)
}
```

Correct

Choosing to place the whitelist code into a variable's module is the best way to make it globally available to the different application projects.

Question 2 of 5

What may be a consideration when deciding whether to use code modules in Terraform?

code comments

Question 2 of 5

What may be a consideration when deciding whether to use code modules in Terraform?

- code comments
- new code
- code errors
- ✓ reusable code

Correct

Reusable code is a common practice in many languages. With Terraform, code modules could be designed to be reusable.

Question 3 of 5

Communication between Terraform modules is dependent on _____ .

- arguments
- providers
- ✓ variables

Correct

Terraform code modules allow the separation and grouping of code. Modules communicate by accepting input with variable data and by providing output.

- versioning

Question 4 of 5

Splitting out data in Terraform is a great way to manage code. What is the best way to achieve this?

- Use code state to clean up code.

- [x] variables
  Correct
  Terraform code modules allow the separation and grouping of code. Modules communicate by accepting input with variable data and by providing output.
- [ ] versioning

Question 4 of 5

Splitting out data in Terraform is a great way to manage code. What is the best way to achieve this?

- [ ] Use code state to clean up code.
- [ ] Use code management to remove data.
- [ ] Place all code comments in a text file.
- [x] Use a variables definitions file.
  Correct
  A variables definition file is a Terraform feature that allows for the use of a file that only contains variables that are used in the project.

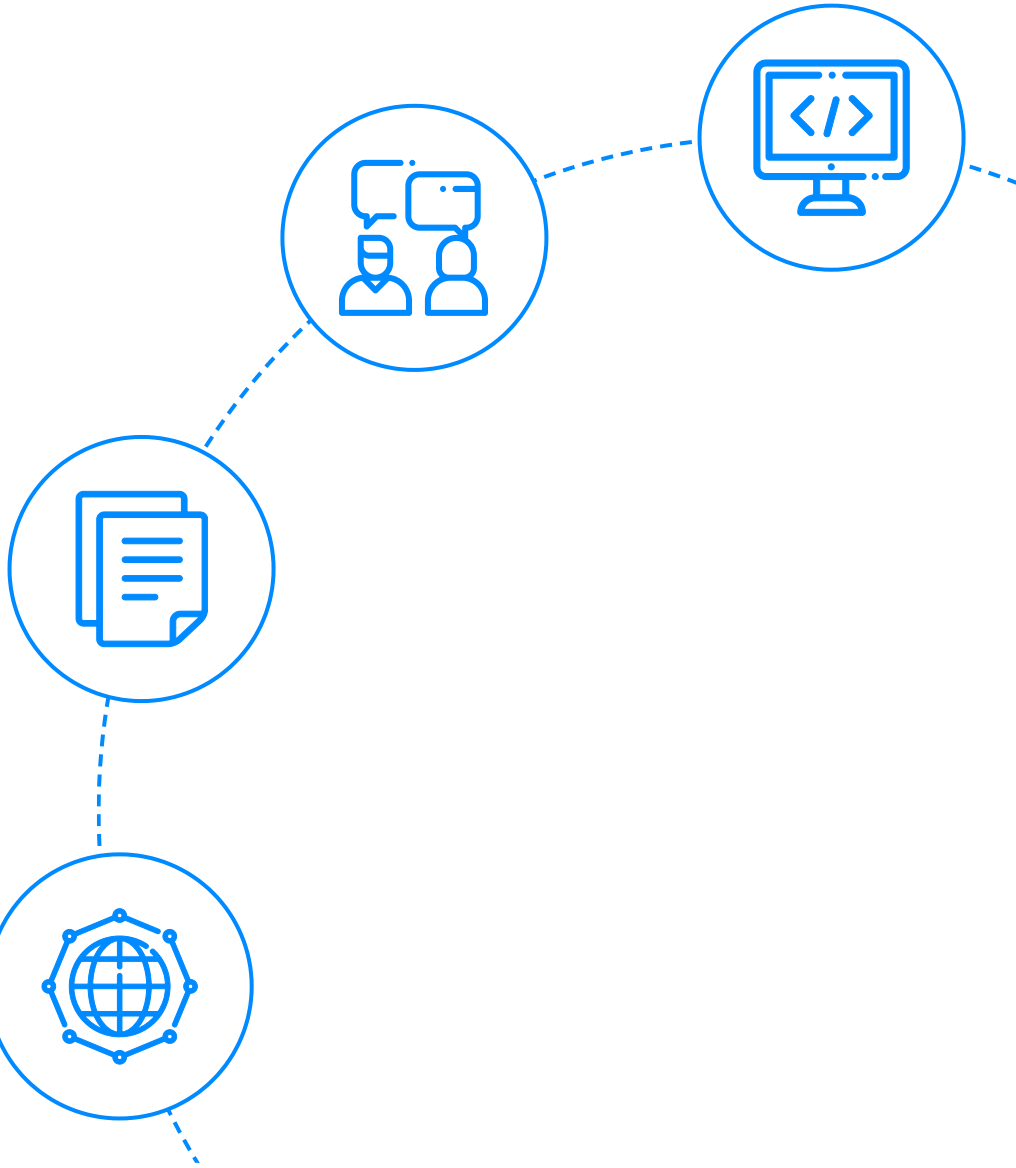
Question 5 of 5

How might a developer best use variables in Terraform code?

- [ ] with static IP values
- [x] with resources that might change
  Correct
  Variables are used in code to represent a value that might change. In Terraform, it is possible that resources may be created and destroyed often. Variables would be useful in this case.
- [ ] with constant resource types
- [ ] with code comments

InterviewBit

Terraform Interview Questions



To view the live version of the page, click here.

Contents

Terraform Interview Questions for Freshers

Terraform Interview Questions for Experienced

Terraform Interview Questions for Experienced

(.....Continued)

19. Explain the architecture of Terraform request flow.
20. Differentiate between Terraform and Cloudformation.
21. Explain the command terraform taint in the context of Terraform.
22. Differentiate between Terraform and Ansible.
23. What do you mean by a Virtual Private Cloud (VPC)? Which command do you use in Terraform to use a VPC service?
24. Explain the command terraform fmt in the context of Terraform.
25. What do you know about Terragrunt? What are its uses?
26. Explain State File Locking in the context of Terraform.
27. What do you know about Terraform core? What are the primary responsibilities of Terraform core?
28. When something goes wrong, how will you control and handle rollbacks in Terraform?
29. What procedures should be taken to make a high-level object from one module available to the other module?
30. What do you understand about remote backend in the context of Terraform?
31. How can you prevent Duplicate Resource Error in Terraform?

Let's get Started

What is Terraform

Terraform is an infrastructure as code (IaC) tool that lets you build, edit, and version infrastructure in a secure and efficient manner. This covers both low-level and high-level components, such as compute instances, memory, and networking, as well as DNS records, SaaS services, and so on. Terraform is capable of managing both third-party services and unique in-house solutions. Terraform uses configuration files to tell it which components are needed to run a single application or a whole datacenter. Amazon Web Services, IBM Cloud, Google Cloud, Platform, DigitalOcean, Linode, Microsoft Azure, Oracle Cloud Infrastructure, OVH, VMware, vSphere, OpenNebula, and OpenStack all use Terraform.

In this article, we will cover the most frequently asked interview questions on Terraform. So, let's get started.

Terraform Interview Questions for Freshers

1. What are the key features of Terraform?

Following are the key features of Terraform:

- **Infrastructure as Code:** Terraform's high-level configuration language is used to describe your infrastructure in declarative configuration files that are human-readable. You may now generate a blueprint that you can edit, share, and reuse.
- **Execution Strategies:** Before making any infrastructure modifications, Terraform develops an execution plan that describes what it will do and asks for your agreement. Before Terraform produces, upgrades, or destroys infrastructure, you can evaluate the changes.
- **Graph of Resources:** Terraform develops or alters non-dependent resources while simultaneously building a resource graph. This allows Terraform to construct resources as quickly as possible while also providing you with more information about your infrastructure.
- **Automation of Change:** Terraform can automate the application of complex changesets to your infrastructure with little to no human intervention. Terraform identifies what happened when you update configuration files and provides incremental execution plans that take dependencies into account.

2. What are the use cases of Terraform?

Following are the use cases of Terraform:

- **Setting Up a Heroku App:**
  - Heroku is a prominent platform as a service (PaaS) for hosting web applications. Developers build an app first, then add add-ons like a database or an email service. The ability to elastically scale the number of dynos or workers is one of the nicest features. Most non-trivial applications, on the other hand, quickly require a large number of add-ons and external services.
  - Terraform may be used to codify the setup required for a Heroku application, ensuring that all essential add-ons are present, but it can also go beyond, such as configuring DNSimple to set a CNAME or configuring Cloudflare as the app's CDN. Best of all, Terraform can achieve all of this without using a web interface in about 30 seconds.

- **Clusters of Self-Service:**
  - A centralised operations team managing a huge and growing infrastructure becomes extremely difficult at a given organisational level. Making "self-serve" infrastructure, which allows product teams to manage their own infrastructure using tooling given by the central operations team, becomes more appealing.
  - Terraform configuration may be used to document knowledge about how to construct and scale a service. You may then publish these configurations across your business, allowing client teams to administer their services using Terraform.
- **Quick Creation of Environments:**
  - It is usual to have both a production and staging or quality assurance environment. These environments are smaller clones of their production counterparts, and they're used to test new apps before they're released to the public. Maintaining an up-to-date staging environment gets increasingly difficult as the production environment grows larger and more complicated.
  - Terraform may be used to codify the production environment, which can then be shared with staging, QA, and development. These settings can be used to quickly create new environments in which to test and then readily discarded. Terraform can help to reduce the challenge of sustaining parallel environments by making it possible to create and destroy them on the fly.
- **Deployment of Multiple Clouds:**
  - To boost fault tolerance, it's common to disperse infrastructure across different clouds. When only one region or cloud provider is used, fault tolerance is restricted by that provider's availability. When a region or an entire provider goes down, a multi-cloud strategy provides for more gentle recovery.
  - Because many existing infrastructure management solutions are cloud-specific, implementing multi-cloud installations can be difficult. Terraform is cloud-agnostic, allowing you to manage numerous providers and even cross-cloud dependencies with a single configuration. This helps operators develop large-scale multi-cloud infrastructures by simplifying management and orchestration.
- **Applications with Multiple Tier Architecture:**
  - The N-tier architecture is a relatively popular structure. A pool of web

3. Is it feasible to use Terraform on Azure with callbacks? Sending a callback to a logging system, a trigger, or other events, for example?

Yes. Azure Event Hubs can be used to accomplish this. This capability is now accessible in the Terraform AzureRM provider. Terraform's Azure supplier provides users with simple functionality. Microsoft Azure Cloud Shell includes a Terraform occurrence that has already been setup.

4. What do you mean by terraform init in the context of Terraform?

The terraform init command creates a working directory in which Terraform configuration files can be found. After creating a new Terraform configuration or cloning an old one from version control, run this command first. It is safe to use this command more than once. Despite the fact that successive runs may result in errors, this command will never overwrite your current settings or state.

Syntax:

```
terraform init [options]
```

The following options can be used in conjunction with the init command :

- **-input=true:** This option is set to true if the user input is mandatory. If no user input is provided, an error will be thrown.
- **-lock=false:** This option is used to disable the locking of state files during state-related actions.
- **-lock-timeout=<duration>:** This option is used to override the time it takes Terraform to get a state lock. If the lock is already held by another process, the default is 0s (zero seconds), which results in an immediate failure.
- **-no-color:** This option disables the color codes in the command output.
- **-upgrade:** This option can be chosen to upgrade modules and plugins throughout the installation process.

5. Why is Terraform preferred as one of the DevOps tools?

Following are the reasons that Terraform is preferred as one of the DevOps tools :

- Terraform allows you to specify infrastructure in config/code, making it simple to rebuild, alter, and track infrastructure changes. Terraform is a high-level infrastructure description.
- While there are a few alternatives, they are all centred on a single cloud provider. Terraform is the only powerful solution that is totally platform-neutral and supports different services.
- Terraform allows you to implement a variety of coding concepts, such as putting your code under version control, writing automated tests, and so on.
- Terraform is the best tool for infrastructure management since many other solutions suffer from an impedance mismatch when attempting to use an API meant for configuring management to govern an infrastructure environment. Instead, Terraform is a perfect match for what you want to do because the API is built around how you think about infrastructure.
- Terraform has a thriving community and is open source, so it's attracting a sizable following. Many people already use it, making it easy to discover individuals who know how to use it, as well as plugins, extensions, and expert assistance. Terraform is also evolving at a much faster rate as a result of this. They have a lot of releases.
- Terraform's speed and efficiency are unrivalled. Terraform's plan command, for example, allows you to see what changes you're about to make before you do them. Terraform and its code reuse feature makes most modifications faster than similar tools like CloudFormation.

6. Mention some of the major competitors of Terraform.

Following are some of the major competitors of Terraform:

- Azure Management Tools.
- Morpheus.
- CloudHealth.
- Turbonomic.
- CloudBolt.
- Apptio Cloudability
- Ansible
- Kubernetes
- Platform9 Managed Kubernetes.

7. What do you understand about Terraform Cloud?

Terraform Cloud is a collaboration tool for teams using Terraform. It offers easy access to shared state and secret data, access controls for approving infrastructure modifications, a private registry for sharing Terraform modules, full policy controls for managing the contents of Terraform configurations, and more. Terraform Cloud is a hosted service that can be found at https://app.terraform.io. Terraform allows small teams to connect to version control, share variables, run Terraform in a reliable remote environment, and securely save remote state for free. Paid tiers provide you with the ability to add more than five people, establish teams with varying levels of access, enforce policies before building infrastructure, and work more efficiently.

Large businesses can utilise the Business tier to scale to multiple concurrent runs, establish infrastructure in private environments, manage user access using SSO, and automate infrastructure end-user self-service provisioning.

8. Explain the destroy command in the context of Terraform.

The terraform destroy command is a simple way to eliminate all remote objects maintained by a Terraform setup. While you should avoid destroying long-lived objects in a production environment, Terraform is occasionally used to manage temporary infrastructure for development, in which case you can use terraform destroy to quickly clean up all of those temporary objects after you're done.

Syntax: `terraform destroy [options]`

You may also execute the following command to build a speculative destroy plan to see what the effect of destroying might be:

```
terraform -destroy plan
```

This will launch Terraform Plan in destroy mode, displaying the proposed destroy changes but not allowing you to execute them.

9. What do you understand about Terraform modules?

A Terraform module is a single directory containing Terraform configuration files. Even a simple arrangement with a single directory having one or more files can be referred to as a module. The files have the extension .tf. This directory is referred to as the root module when Terraform commands are run directly from it. Terraform commands will only use the configuration files in one location: the current working directory. Your configuration, on the other hand, can employ module blocks to call modules from other directories. When Terraform comes across a module block, it loads and processes the configuration files for that module. A module that is called by another configuration is frequently referred to as that configuration's "child module."

10. What are the benefits of using modules in Terraform?

Following are the benefits of using modules in Terraform :

- **Organization of configuration:** By grouping relevant portions of your configuration together, modules make it easier to access, understand, and change your configuration. Hundreds or thousands of lines of configuration can be required to establish even moderately complicated infrastructure. You can organise your configuration into logical components by utilising modules.
- **Encapsulation of configuration:** Another advantage of modules is that they allow you to separate configuration into logical components. Encapsulation can help you avoid unforeseen consequences, such as a change to one element of your configuration causing changes to other infrastructure, and it can also help you avoid basic mistakes like naming two resources with the same name.
- **Maintains consistency and ensures best practices:** Modules can also help you maintain uniformity in your configurations. Consistency not only makes complex configurations easier to grasp, but it also ensures that best practices are followed in all of your settings. Cloud providers, for example, offer a variety of options for establishing object storage services like Amazon S3 or Google Cloud Storage buckets. Many high-profile security problems have occurred as a result of improperly secured object storage, and given the number of sophisticated configuration options involved, it's possible to misconfigure these services by accident.
- **Modules can aid in the reduction of errors:** For example, you might design a module to define how all of your organization's public website buckets would be set, as well as a separate module for private logging buckets. In addition, if a configuration for a particular resource type needs to be altered, using modules allows you to do it in one place and have it applied to all scenarios where that module is used.
- **Aids in reusability:** Setting up the configurations from scratch and writing all of your settings can be time-consuming and error-prone. By reusing configuration generated by yourself, other members of your team, or other Terraform practitioners who have published modules for you to utilise, you can save time and avoid costly errors. You can also share modules you've produced with your colleagues or the broader public, allowing them to profit from your efforts.

11. What are some guidelines that should be followed while using Terraform modules?

Following are some of the guidelines that should be followed while using Terraform modules :

- To publish to the Terraform Cloud or Terraform Enterprise module registries, you must use this convention terraform-<PROVIDER>-<NAME>.
- Start thinking about modules as you write your setup. The benefits of using modules outweigh the time it takes to utilise them properly, even for somewhat complicated Terraform settings maintained by a single person.
- To organise and encapsulate your code, use local modules. Even if you aren't using or publishing remote modules, structuring your configuration in terms of modules from the start will dramatically minimise the time and effort required to maintain and update your setup as your infrastructure becomes more complicated.
- To identify useful modules, go to the Terraform Registry, which is open to the public. By relying on the efforts of others to create common infrastructure scenarios, you may implement your configuration more quickly and confidently.
- Modules can be published and shared with your team. The majority of infrastructure is handled by a group of individuals, and modules are a vital tool for teams to collaborate on infrastructure creation and maintenance.

12. Explain null resource in the context of Terraform.

The null resource is a resource that lets you set up provisioners that aren't directly linked to any current resource. Because a null resource behaves like any other resource, you can configure provisioners, connection details, and other meta-parameters just like any other resource. This gives you more precise control over when provisioners execute in the dependency graph.

13. Explain the command terraform validate in the context of Terraform.

The terraform validate command verifies the configuration files in a directory, focusing solely on the configuration and excluding any outside services such as remote state, provider APIs, and so on. Validate performs checks to see if a configuration is syntactically correct and internally consistent, regardless of any variables or current state. As a result, it's best used for general verification of reusable modules, such as ensuring that attribute names and value types are correct. This command can be executed automatically, for example as a post-save check in a text editor or as a test step for a reusable module in a continuous integration system.

Syntax: `terraform validate [options]`

The following options are available with this command:

- **-json** - Create output in the machine-readable JSON format, appropriate for integration with text editors and other automated systems. Color is always turned off.
- **-no-color** - If supplied, the output will be colourless.

14. Explain the command terraform apply in the context of Terraform.

The terraform apply command is used to carry out the tasks in a Terraform plan. The simplest method to use terraform apply is to run it without any arguments, in which case it will construct a new execution plan (as if you had run terraform plan) and then request you to accept it before doing the activities you specified. Another approach to use terraform apply is to supply it the filename of a saved plan file generated with terraform plan -out=..., in which case Terraform will apply the modifications to the plan without prompting for confirmation. This two-step process is most useful when using Terraform in an automated environment.

Syntax:

```
terraform apply [options] [plan file]
```

15. Explain the command terraform version in the context of Terraform.

The terraform version command shows the current Terraform version as well as any installed plugins.

Syntax:

```
terraform version [options]
```

Unless disabled, the version will display the Terraform version, the platform it's installed on, installed providers, and the results of upgrade and security checks with no extra arguments.

There is one optional flag for this command:

If you specify -json, the version information is formatted as a JSON object, with no upgrade or security information.

16. Mention some of the version control tools supported by Terraform.

Some of the version control tools supported by Terraform are as follows:

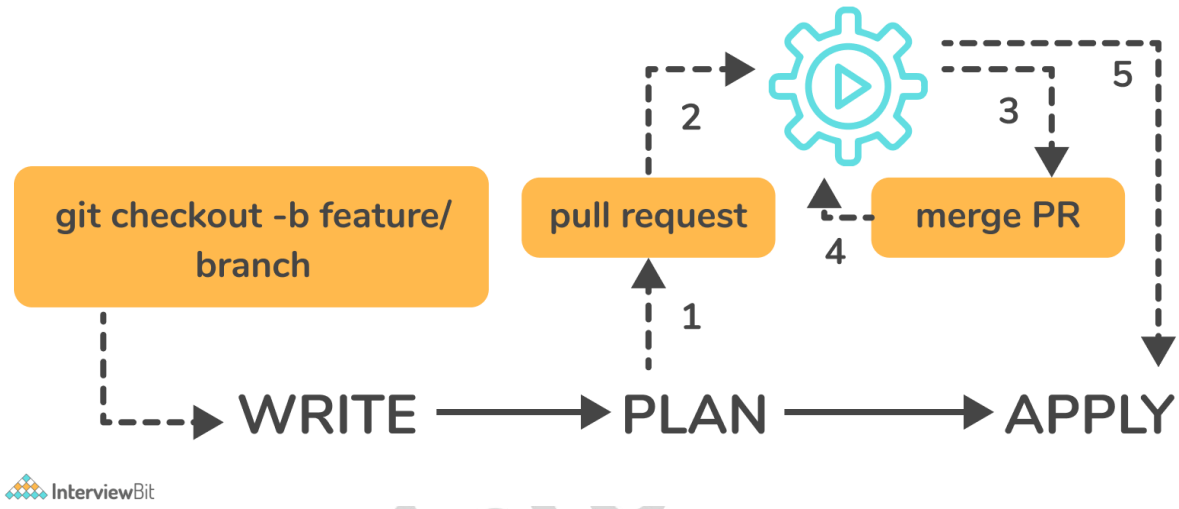
- GitHub
- GitLab CE
- GitLab EE
- Bucket Cloud

17. What do you understand about providers in the context of Terraform?

To interface with cloud providers, SaaS providers, and other APIs, Terraform uses plugins called "providers." Terraform configurations must specify the providers they need in order for Terraform to install and use them. Some providers also require setup (such as endpoint URLs or cloud regions) before they may be used. Terraform may manage a set of resource types and/or data sources that each provider contributes. A provider implements each resource type; Terraform would be unable to manage any infrastructure without them. The majority of service providers set up a specific infrastructure platform (either cloud or self-hosted). Local utilities, such as generating random numbers for unique resource names, can be offered by providers.

18. Explain the workflow of the core terraform.

Terraform's core workflow consists of three steps:

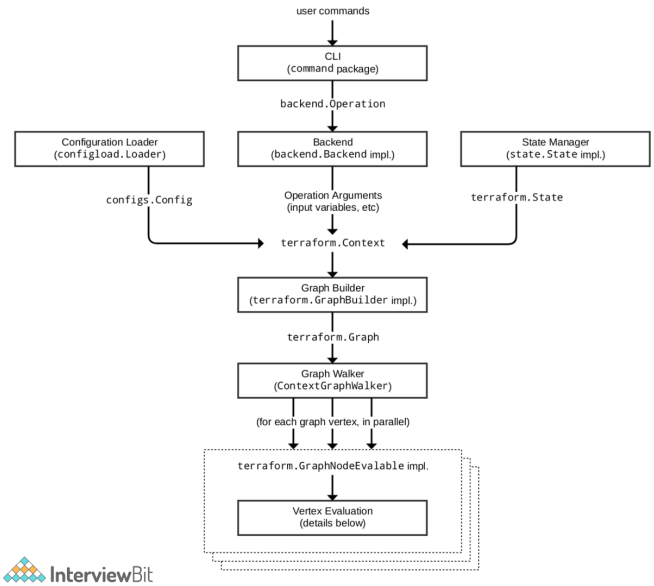


- **Write** - Create infrastructure in the form of code.
- **Plan** - Plan ahead of time to see how the changes will look before they are implemented.
- **Apply** - Create a repeatable infrastructure.

Terraform Interview Questions for Experienced

19. Explain the architecture of Terraform request flow.

A request in Terraform undergoes the following steps as shown in the diagram:



Following are the different components in the above architecture:

Command Line Interface (CLI):

CLI (Common Language Interface) (command package)

Aside from some early bootstrapping in the root package (not shown in the diagram), when a user starts the terraform application, execution jumps right into one of the command package's "command" implementations. The commands.go file in the repository's root directory contains the mapping between user-facing command names and their respective command package types.

The job of the command implementation for these commands is to read and parse any command line arguments, command-line options, and environment variables required for the given command and use them to generate a backend.operation object. The operation is then transferred to the backend that is currently selected.

Backends:

In Terraform, a backend has a variety of responsibilities:

- Carry out operations (e.g. plan, apply)
- To save workspace-defined variables
- To save state

The local backend retrieves the current state for the workspace specified in the operation using a state manager (either statemgr.Filesystem if the local backend is being used directly, or an implementation provided by whatever backend is being wrapped), then uses the config loader to load and do initial processing/validation of the configuration specified in the operation. It then constructs a terraform.context object using these, as well as the other parameters specified in the procedure. The main object actually executes Terraform operations.

Configuration Loader :

Model types in package configs represent the top-level configuration structure. configs.Config represents an entire configuration (the root module and all of its child modules). Although the configs package offers some low-level functionality for creating configuration objects, the major entry point is via configload.Loader, which is found in the sub-package configload. When a configuration is loaded by a backend, a loader takes care of all the complexities of installing child modules (during terraform init) and then locating those modules again. It takes the path to a root module and loads all of the child modules in a recursive manner to create a single configs.

State Manager:

The state manager is in charge of storing and retrieving snapshots of a workspace's Terraform state. Each manager is an implementation of a subset of the statemgr package's interfaces, with most practical managers implementing the entire set of statemgr.Full's operations . The smaller interfaces are mostly for use in other function signatures to be specific about what actions the function might perform on the state manager; there's no reason to design a state manager that doesn't implement all of statemgr. Full.

Graph Builder:

The terraform.Context method invokes a graph builder. A graph builder is used to represent the essential steps for that operation and the dependence relationships between them. Because the graph-building process differs by operation, each has its own graph builder. A "plan" operation, for example, requires a graph created directly from the configuration, whereas an "apply" action creates its graph from the set of modifications stated in the plan being applied.

Graph Walk:

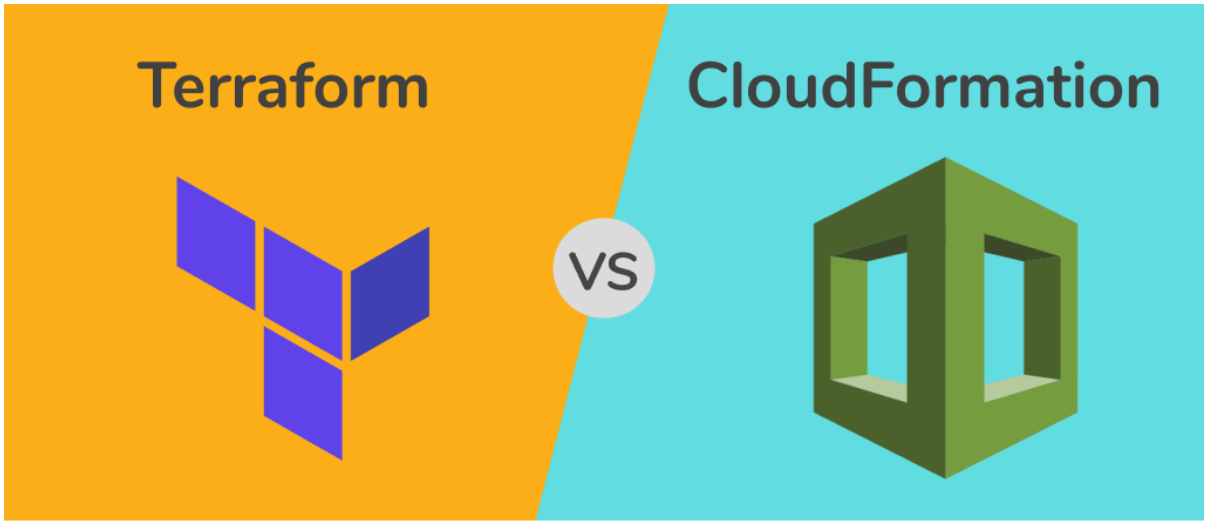
The graph walking method visits each vertex of the graph in a fashion that respects the graph's "happens after" edges. Each vertex in the graph is assessed in such a way that the "happens after" edges are taken into account. The graph walk method will assess many vertices at the same time if possible.

Vertex Evaluation:

During a graph walk, the action executed for each vertex is referred to as execution. Execution performs a series of random operations that make sense for a given vertex type. Before the graph walk begins evaluating further vertices with "happens after" edges, a vertex must be complete correctly. When one or more errors occur during evaluation, the graph walk is interrupted and the errors are returned to the user.

20. Differentiate between Terraform and Cloudformation.

The following points highlight the differences between Terraform and Cloudformation:



- **User-friendliness:**
  - Terraform encompasses numerous Cloud Service Providers such as AWS, Azure, Google Cloud Platform, and many more, while CloudFormation is limited to AWS services. Terraform covers the majority of AWS resources.
- **Based on language:**
  - CloudFormation employs either JSON or YAML as a language. CloudFormation is now simple to read and manage. However, AWS developers are restricted from creating CloudFormation templates that are more than 51MB in size. Developers must establish a layered stack for the templates if the template exceeds this size restriction.
  - Terraform, on the other hand, makes use of Hashicorp's own HCL language (Hashicorp Configuration Language). This language is also JSON compatible.
- **State-management:**
  - Because CloudFormation is an AWS managed service, it examines the infrastructure on a regular basis to see if the provisioned infrastructure is still in good shape. If anything changes, CloudFormation receives a thorough response.
  - Terraform, on the other hand, saves the state of the infrastructure on the provisioning machine, which can be either a virtual machine or a remote computer. The state is saved as a JSON file, which Terraform uses as a map to describe the resources it manages.
  - To summarise, Cloudformation's state is managed out-of-the-box by CloudFormation, which prevents conflicting updates. Terraform stores the state on a local disk, which makes it easier to synchronise the state. Terraform states can also be saved in storage services like S3, which is another recommended practice for state management. This must be defined on the backend, making management easier and safer.
- **Cost:**
  - The nicest aspect about both of these tools is that they are both completely free. Both of these technologies have sizable communities that provide plenty of help and examples. Cloudformation is completely free. The only expense that consumers pay is for the AWS service that CloudFormation provides. Terraform is a completely free and open-source application. Terraform, on the other hand, includes a premium enterprise version with more collaboration and governance features.

21. Explain the command terraform taint in the context of Terraform.

Terraform receives notification from the terraform taint command that a specific item has been degraded or damaged. This is represented by Terraform designating the item as "tainted" in the Terraform state, in which case Terraform will suggest replacing it in the next plan you write. If you want to compel the replacement of a specific object despite the fact that no configuration modifications are required, using the terraform apply -replace option is preferred.

Utilizing the "replace" option while creating a plan is preferable to using terraform taint because it allows you to see the entire impact of the alteration before taking any externally visible action. When you utilise terraform taint to achieve a similar impact, you run the danger of someone else on your team devising a new strategy to counter your tainted object before you've had a chance to consider the implications.

Syntax:

```
terraform taint [options] address
```

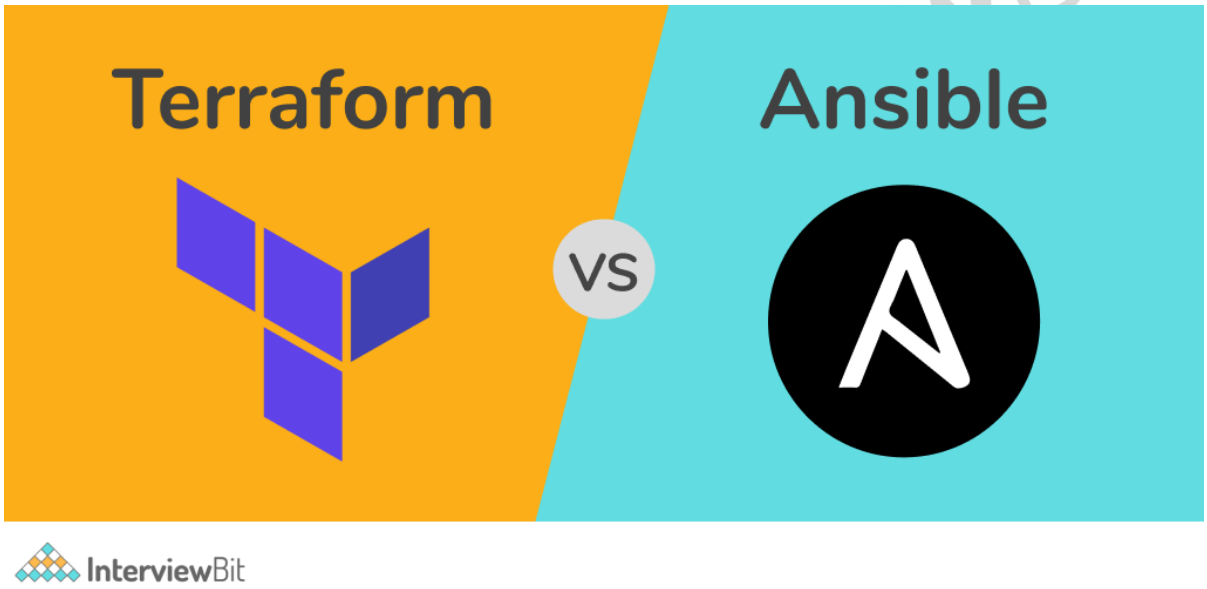
The address option specifies the location of the infected resource. The following options are available with this command:

- **-allow-missing** - Even if the resource is absent, the command will succeed (exit code 0) if it is supplied. Other scenarios, such as a problem reading or writing the state, may cause the command to return an error.
- **-lock=false** - Turns off Terraform's default behaviour of attempting to lock the state for the duration of the operation.
- **-lock-timeout=DURATION** - Instructs Terraform to reattempt procuring a lock for a period of time before issuing an error, unless locking is disabled with -lock=false. A number followed by a time unit letter, such as "3s" for three seconds, is the duration syntax.

22. Differentiate between Terraform and Ansible.

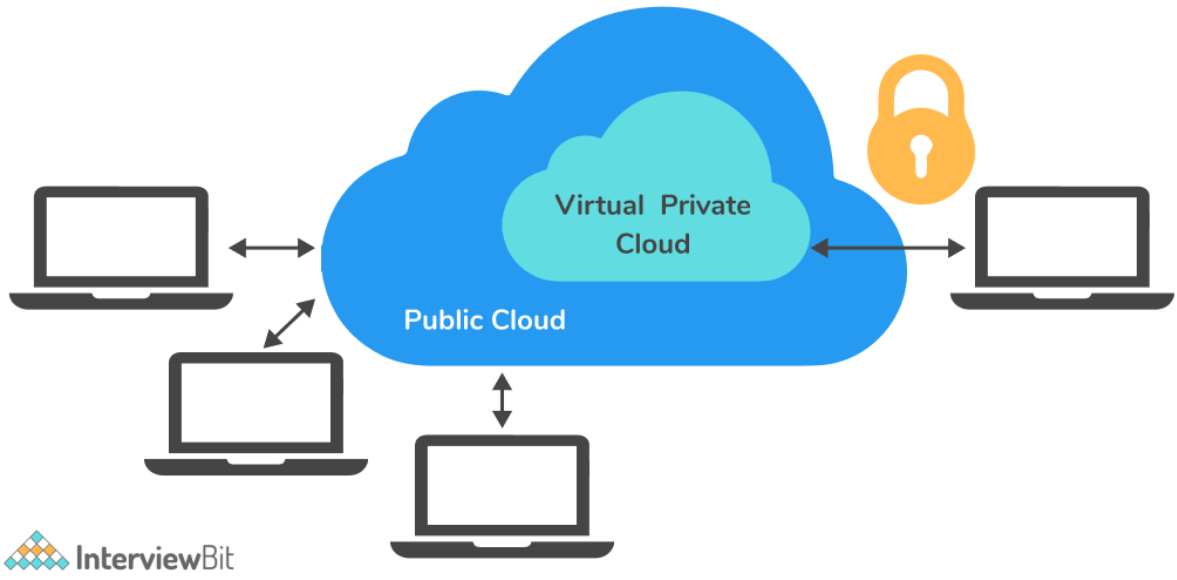
Ansible : Ansible is a remarkably straightforward IT automation technology. Configuration management, application deployment, cloud provisioning, ad-hoc task execution, network automation, and multi-node orchestration are all handled by this software. Complex modifications, such as zero-downtime rolling updates with load balancers, are simple using Ansible.

Following table lists the differences between Ansible and Terraform:



Terraform	Ansible
Terraform is a tool for provisioning.	Ansible is a tool for managing configurations.
It uses a declarative Infrastructure as Code methodology.	It takes a procedural method.
It's ideal for orchestrating cloud services and building cloud infrastructure from the ground up.	It is mostly used to configure servers with the appropriate software and to update resources that have previously been configured.
By default, Terraform does not allow bare metal provisioning.	The provisioning of bare metal servers is supported by Ansible.
In terms of packing and templating, it does not provide better support.	It includes complete packaging and templating support.
It is strongly influenced by lifecycle or state management.	It doesn't have any kind of lifecycle management. It does not store the state.

23. What do you mean by a Virtual Private Cloud (VPC)? Which command do you use in Terraform to use a VPC service?



A Virtual Private Cloud (VPC) is a private virtual network within AWS where you can store all of your AWS services. It will have gateways, route tables, network access control lists (ACL), subnets, and security groups, and will be a logical data centre in AWS. When you create a service on a public cloud, it is effectively open to the rest of the world and can be vulnerable to internet attacks. You lock your instances down and secure them from outside threats by putting them inside a VPC. The VPC limits the types of traffic, IP addresses, and individuals who have access to your instances.

This stops unauthorised users from accessing your resources and protects you from DDOS assaults. Because not all services require internet connection, they can be safely stored within a private network. You can then only allow particular machines to connect to the internet.

We use the command aws_vpc to use a VPC Service in Terraform.

24. Explain the command terraform fmt in the context of Terraform.

Terraform configuration files are rewritten using the terraform fmt command in a consistent structure and style. This command uses a subset of the Terraform language style conventions, as well as some small readability tweaks. Other Terraform commands that produce Terraform configuration will produce files that follow the terraform fmt style, therefore following this style in your own files will assure consistency. Because formatting selections are always subjective, you may disagree with terraform fmt's choices. This command is purposely opinionated and lacks customization options because its primary goal is to promote stylistic consistency throughout Terraform codebases, even though the chosen style will never be everyone's favourite.

Syntax:

```
terraform fmt [options] DIR
```

By default, fmt looks for configuration files in the current directory. If the dir option is provided, it will instead scan the specified directory.

The following are the flags that are available:

- **-list=false** - This option doesn't show files with discrepancies in formatting.
- **-write=false** - This option prevents the input files from being overwritten. (When the input is STDIN or -check, this is implied.)
- **-diff** - Shows the differences in formatting modifications.
- **-check** - Verifies that the input is properly formatted. If all input is properly formatted, the exit status will be 0, else it will be non-zero.
- **-recursive** - Process files from subdirectories as well.

25. What do you know about Terragrunt? What are its uses?

Terragrunt is a lightweight wrapper that adds extra features for maintaining DRY configurations, dealing with many Terraform modules, and managing remote state.

Following are the use cases of Terragrunt:

- **To Keep Our Background Configuration DRY (Don’t Repeat Yourself):** By setting your backend configuration once in a root location and inheriting that information in all child modules, Terragrunt helps you to keep it DRY ("Don't Repeat Yourself").
- **To Keep Our Provider Configuration DRY:** It might be difficult to unify provider configurations across all of your modules, especially if you wish to alter authentication credentials. You may use Terragrunt to refactor common Terraform code and keep your Terraform modules DRY by using it. The provider configurations can be defined once at a root location, just like the backend configuration.
- **To Keep Our Terraform Command Line Interface arguments DRY:** In the Terraform universe, CLI flags are another typical source of copy/paste. It can be difficult and error-prone to have to remember these -var-file options every time. By declaring your CLI parameters as code in your terragrunt.hcl settings, Terragrunt helps you to keep your CLI arguments DRY.
- **To Promote Terraform modules that are immutable and versioned across environments:** Large modules should be considered hazardous, according to one of the most important lessons we've learnt from building hundreds of thousands of lines of infrastructure code. That is, defining all of your environments (dev, stage, prod, and so on) or even a huge amount of infrastructure (servers, databases, load balancers, DNS, and so on) in a single Terraform module is a Bad Idea. Large modules are slow, insecure, difficult to update, code review, test, and are brittle. Terragrunt lets you define your Terraform code once and then promote a versioned, immutable "artifact" of that code from one environment to the next.

26. Explain State File Locking in the context of Terraform.

Terraform's state file locking method prevents conflicts between numerous users doing the same task by blocking activities on a given state file. When one user unlocks the lock, only the other user has access to that state. Terraform will lock your state for any operations that potentially write state if your backend supports it. This prevents outsiders from gaining access to the lock and corrupting your state. All operations that have the potential to write state are automatically locked. There will be no indication that this is happening. Terraform will not continue if state locking fails. The -lock flag can be used to deactivate state locking for most tasks, although it is not advised. Terraform will send a status message if gaining the lock takes longer than planned. If your backend enables state locking, even if Terraform doesn't send a message, it still happens.

27. What do you know about Terraform core? What are the primary responsibilities of Terraform core?

Terraform Core is a binary created in the Go programming language that is statically compiled. The compiled binary is the terraform command line tool (CLI), which is the starting point for anyone who wants to use Terraform. The source code can be found at github.com/hashicorp/terraform.

The primary responsibilities of Terraform core includes:

- Reading and interpolating configuration files and modules using infrastructure as code
- Management of the state of resources
- Resource Graph Construction
- Execution of the plan
- Communication with plugins through RPC

28. When something goes wrong, how will you control and handle rollbacks in Terraform?

In our Version Control System, we need to recommit the previous code version to make it the new and current one. This would start the terraform run command, which would execute the old code. Because Terraform is more declarative, we will make sure that everything in the code reverts to its previous state. If the state file becomes corrupted, we would use Terraform Enterprise's State Rollback feature to restore the previous state.

29. What procedures should be taken to make a high-level object from one module available to the other module?

The steps to make an object from one module available to the other module at a high level are as follows:

- The first step is to define an output variable in a resource configuration. The scope of local and to a module will not be declared until you define resource configuration details.
- Now you must specify the output variable of module A so that it can be utilised in the setup of other modules. You should establish a fresh new and up-to-date key name, with a value that is equal to the output variable of module A.
- You must now create a file named variable.tf for module B. Create an input variable with the exact same name as the key you defined in module B inside this file. This variable permits the resource's dynamic setting in a module. Replicate the process to make this variable available to other modules as well. This is because the scope of the variable established here is limited to module B.

30. What do you understand about remote backend in the context of Terraform?

Terraform's remote backend stores terraform state and can also conduct operations in the terraform cloud. terraform commands such as init, plan, apply, destroy , get, output, providers, state (sub-commands: list, mv, pull, push, rm, show), taint, untaint, validate, and many others can be run from a remote backend. It can be used with a single or several remote terraform cloud workspaces. You can utilise terraform cloud's run environment to conduct remote operations like terraform plan or terraform apply.

31. How can you prevent Duplicate Resource Error in Terraform?

Depending on the situation and the necessity, it can be accomplished in one of three ways.

- By destroying the resource, the Terraform code will no longer manage it.
- By removing resources from APIs
- Importing action will also aid in resource elimination.

Useful Resources

Kubernetes

AWS

Azure

Terraform Tutorial

Terraform Download

Links to More Interview Questions

C Interview Questions	Php Interview Questions	C Sharp Interview Questions
Web Api Interview Questions	Hibernate Interview Questions	Node Js Interview Questions
Cpp Interview Questions	Oops Interview Questions	Devops Interview Questions
Machine Learning Interview Questions	Docker Interview Questions	Mysql Interview Questions
Css Interview Questions	Laravel Interview Questions	Asp Net Interview Questions
Django Interview Questions	Dot Net Interview Questions	Kubernetes Interview Questions
Operating System Interview Questions	React Native Interview Questions	Aws Interview Questions
Git Interview Questions	Java 8 Interview Questions	Mongodb Interview Questions
Dbms Interview Questions	Spring Boot Interview Questions	Power Bi Interview Questions
Pl Sql Interview Questions	Tableau Interview Questions	Linux Interview Questions
Ansible Interview Questions	Java Interview Questions	Jenkins Interview Questions