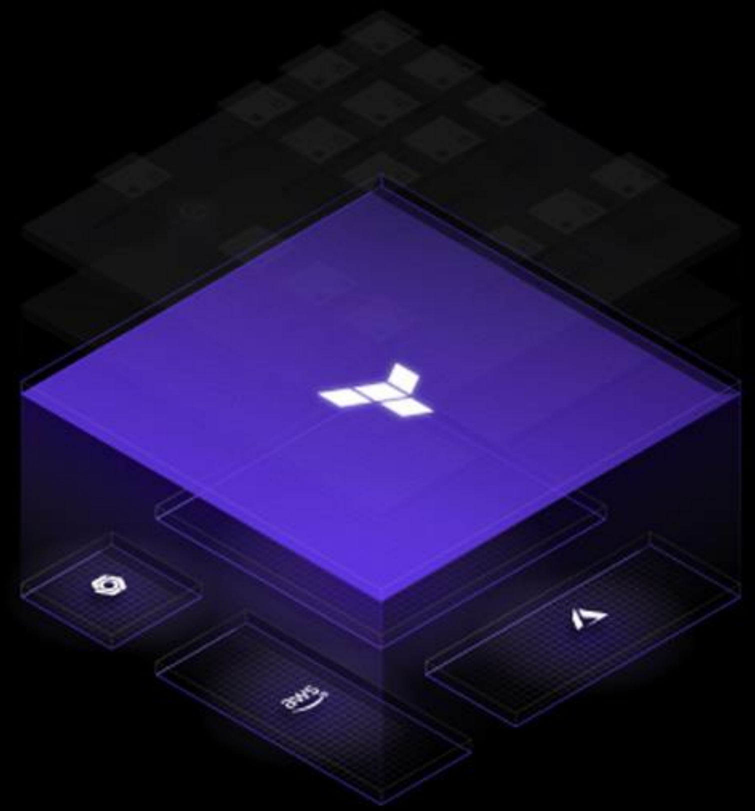
Introduction to **Terraform**

Infrastructure as Code



HashiCorp
Terraform

Vikram
IoT Application Dev & DevOps

/kunchalavikram1427

Agenda

- Need for Infrastructure as Code
- IaC Tools
- Terraform Basics
- Installing Terraform
- Terraform Architecture
- More to be added soon…

/kunchalavikram1427

Infrastructure as Code (IaC)

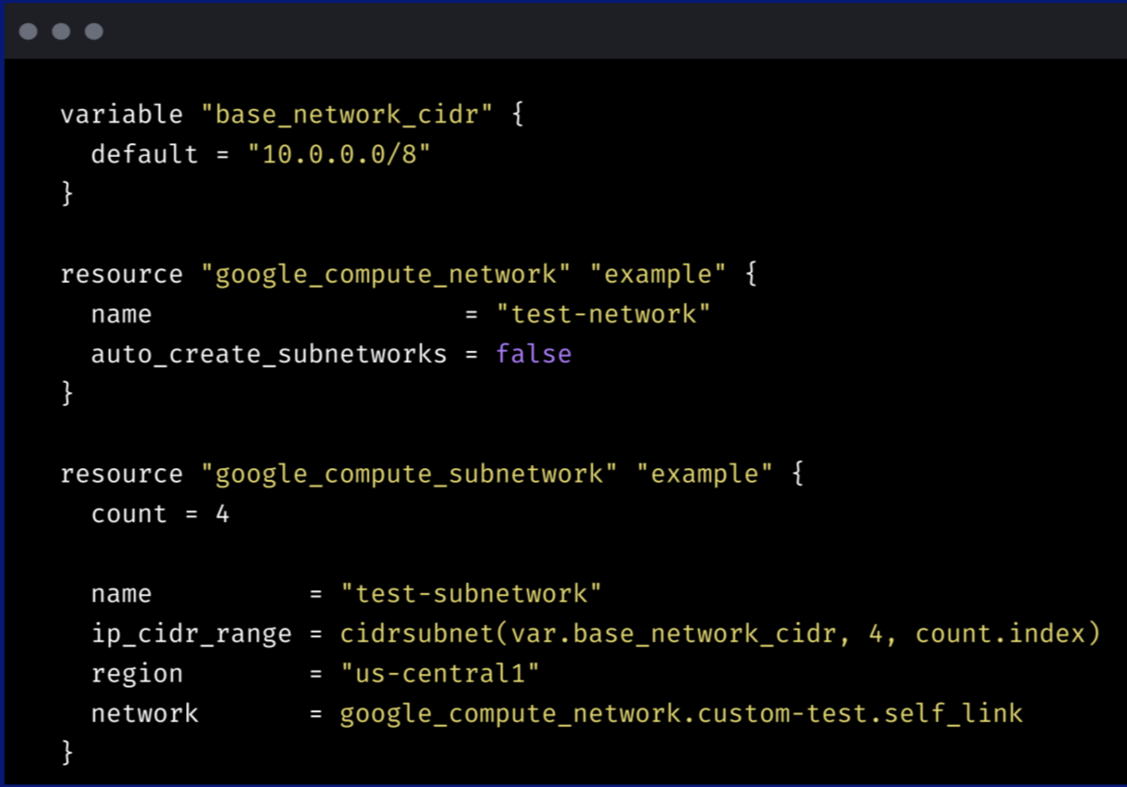
/kunchalavikram1427

Infrastructure as Code

- Infrastructure as Code(IaC) is the process of provisioning, configuring and managing infrastructure like virtual machines, databases, networks, load balancers, security groups etc., through machine-readable configuration files, rather than physical hardware configuration or interactive configuration tools
- It is all about treating your infrastructure configuration and provisioning in the same way you treat your application source code
- The configuration modules are typically stored in version control systems like Git in very well-documented code formats which provides greater accuracy, reduces errors, and increases speed and consistency
- The same configuration files can be used to provision infrastructure for various environments like Dev, Test and Prod
- IaC allows DevOps teams to use different tools and approaches to automatically control and customize the required infrastructure, instead of manually configuring the servers and operating systems
- With the increase in the number of production and delivery cycles, the use of Infrastructure as Code (IaC) tools has changed the way software engineers design, test, and release their applications

Infrastructure as Code

Infrastructure automation becomes possible thanks to template files that are human and machine readable.
These YAML/JSON template files contain instructions for underlying tool on how to manage and provision cloud or server (or both) resources.



```
variable "base_network_cidr" {
  default = "10.0.0.0/8"
}

resource "google_compute_network" "example" {
  name                    = "test-network"
  auto_create_subnetworks = false
}

resource "google_compute_subnetwork" "example" {
  count = 4

  name          = "test-subnetwork"
  ip_cidr_range = cidrsubnet(var.base_network_cidr, 4, count.index)
  region        = "us-central1"
  network       = google_compute_network.custom-test.self_link
}
```

Common IaC Tools

ANSIBLE

AWS CloudFormation

HashiCorp Terraform

CHEF

Google Cloud Deployment Manager

AZURE ARM

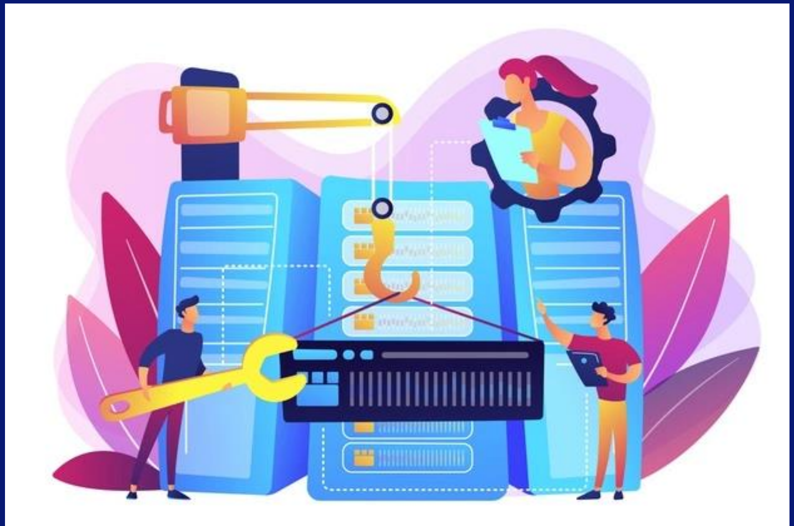
Infrastructure as Code

What might you define as a code?

- The full stack of technologies within the infrastructure of the cloud platform like VM, Databases, Storage, Networking, DNS, Load balancers etc
- Components of the server config files (different files, packages, accounts etc)
- Server instances
- Container Images
- Code delivery pipelines and deployments
- Monitoring and log management
- Compliance and validation tests

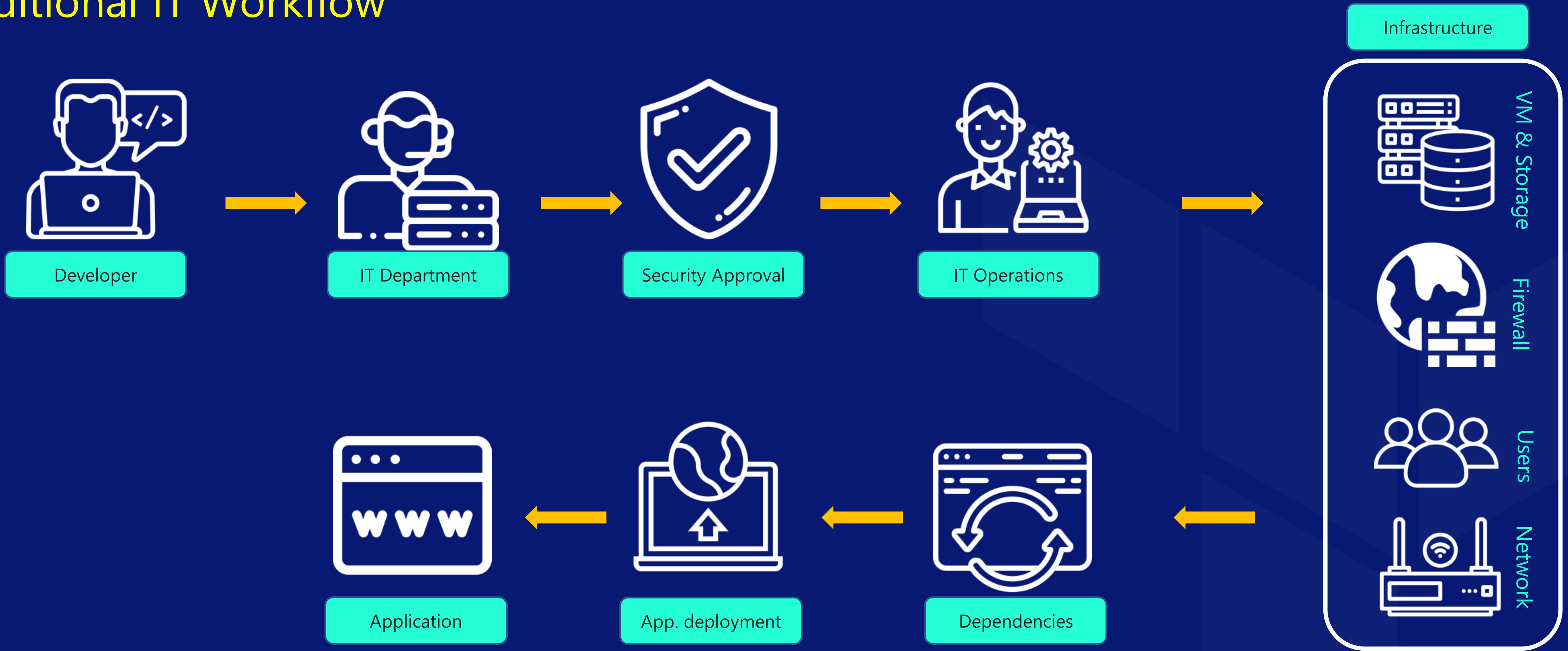
Before IaC

- Manual provisioning of IT infrastructure like setup of servers, configure networking & storage, OS, firewall, install & configure dependencies etc., by expert personnel
- Use of throwaway scripts to automate some tasks
- Long chain of approvals needed. From raising an IT ticket, Security Approvals, Operation team to provision the infrastructure, admin team to add users and firewall rules etc.
- The same goes for decommissioning the infrastructure
- Repeat the process for environments like DEV, TEST and PROD
- Approach is suitable for small on-prem datacenters where infrastructure once created, lives for many years
- Time-consuming, error prone and costly process
- Suffers from Configuration Drift: a minor human error can make environments different as each environment is provisioned manually
- Hard to replicate the same environment without any errors, especially the application working in Dev might end up working differently in Prod
- Hard to track environment changes and prevent inconsistencies



Before IaC

Traditional IT Workflow



Developer raises a IT ticket for infrastructure provisioning. It has to go through security approvals and finally IT operations sets up the required infrastructure. Admin team will install the required dependencies, applications, adds the users and accounts, and finally VMs are added to the patching cycle etc. Finally the developers gets the access to these machines for the applications deployment

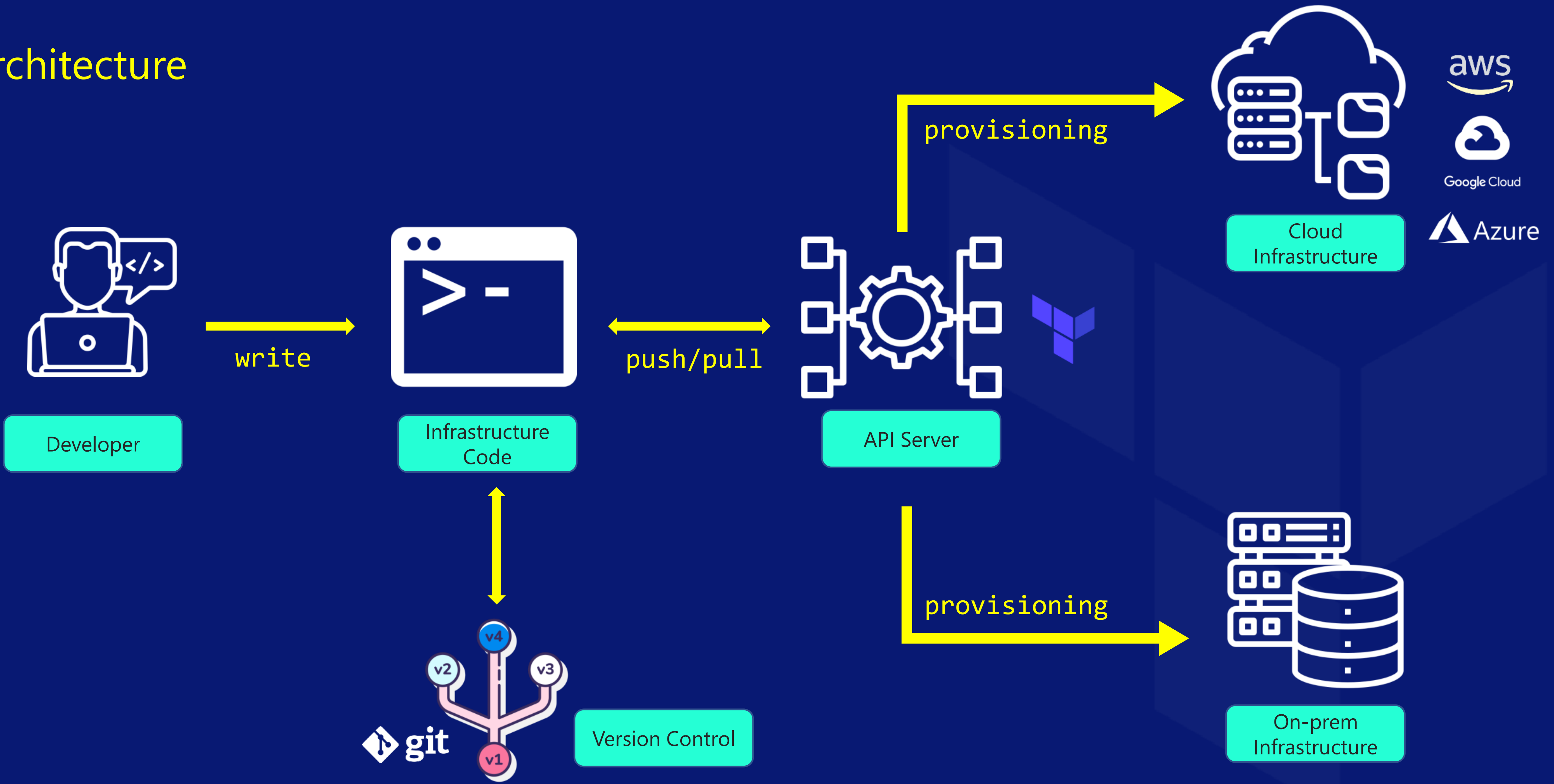
After IaC

- Virtualization and cloud native development eliminated the problem of physical hardware management, enabling developers to provision their own virtual servers or containers on demand
- Without IaC tools, this provisioning would take long time as it has to go through security approvals in a traditional IT environment
- With IaC, provisioning virtualized infrastructure become easy, essentially by using code/configuration files to automate the process of provisioning the infrastructure, which allows for faster deployment/replication of environment
- As the whole infrastructure is now codified, It can be versioned in version control tools like Git
- So, the whole history of how the infrastructure has evolved/changed over time can be easily tracked and well documented

"With IaC, your infrastructure's configuration takes the form of a code file. Since it's just text, it's easy for you to edit, copy, and distribute it. You can—and should—put it under source control, like any other source code file"

Infrastructure as Code

Architecture



/kunchalavikram1427

Mutable vs. Immutable infrastructure

An important decision to make when automating infrastructure with Infrastructure as Code (IaC) and when choosing an IaC solution is whether to establish mutable or immutable infrastructure

Mutable infrastructure

- Infrastructure that can be modified or updated after it is originally provisioned
- Mutable infrastructure gives development teams the flexibility to make ad hoc server customizations to fit development or application requirements or respond to an emergent security issue
- But, it also undermines a key IaC benefit—the ability to maintain consistency between deployments or within versions—and can make infrastructure version tracking much more difficult
- For these reasons, most IaC is implemented as immutable infrastructure

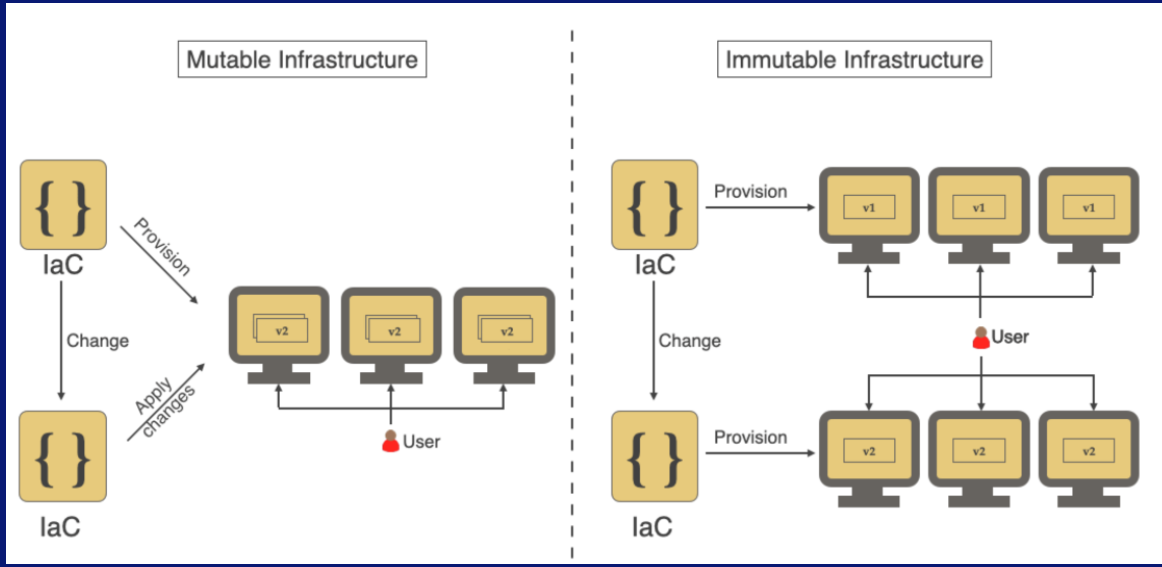
Immutable infrastructure

- Infrastructure that cannot be modified once originally provisioned
- Provision new infrastructure every time if existing infrastructure has to be changed/modified
- It eliminates configuration drift and makes it even easier to maintain consistency between test and deployment environments
- It also makes it easier to maintain and track infrastructure versions and to confidently roll back to any version when necessary
- If immutable infrastructure needs to be changed, it has to be replaced with new infrastructure
- Because new infrastructure can be spun up quickly on the cloud—particularly with IaC—immutable infrastructure is much more feasible and practical than it sounds

Mutable vs. Immutable infrastructure

Immutable infrastructure

- Configuration drift is a huge problem with infrastructure. It occurs when over a period there are changes made to infrastructure that are not recorded, and your various environments drift from each other in ways that are not easily reproducible
- This usually happens if you have a mutable infrastructure that lives for a long time. The system is more brittle in general for long-lived infrastructure since issues like a slow memory leak, disk out of space due to log accumulation, etc. might occur over a period. It also means that you won't be provisioning the infrastructure as frequently as your applications or configuration and as a result won't be confident in your ability to do so
- These issues can be resolved by using immutable infrastructure



Declarative vs. imperative approach

When choosing an IaC solution, it's also important to understand the difference between a declarative or an imperative approach to infrastructure automation

Imperative approach

- Also known as the procedural approach
- In the Imperative approach, you provision your infrastructure one specific step at a time—spinning VMs, create databases, networking, firewalls etc
- While this can be more work to manage as you scale, it can be easier for existing administrative staff to understand and can leverage configuration scripts you already have in place
- Ex: Using shell commands, one command at a time

Declarative approach

- Also known as the functional approach
- In the declarative approach, you specify the desired final state of the infrastructure you want to provision and the IaC software handles the rest—spinning up the virtual machine (VM) or container, installing and configuring the necessary software, resolving system and software interdependencies, and managing versioning
- Ex: Using shell scripts, one scripts runs many sequential shell commands

Benefits of IaC

Improved consistency and less configuration drift

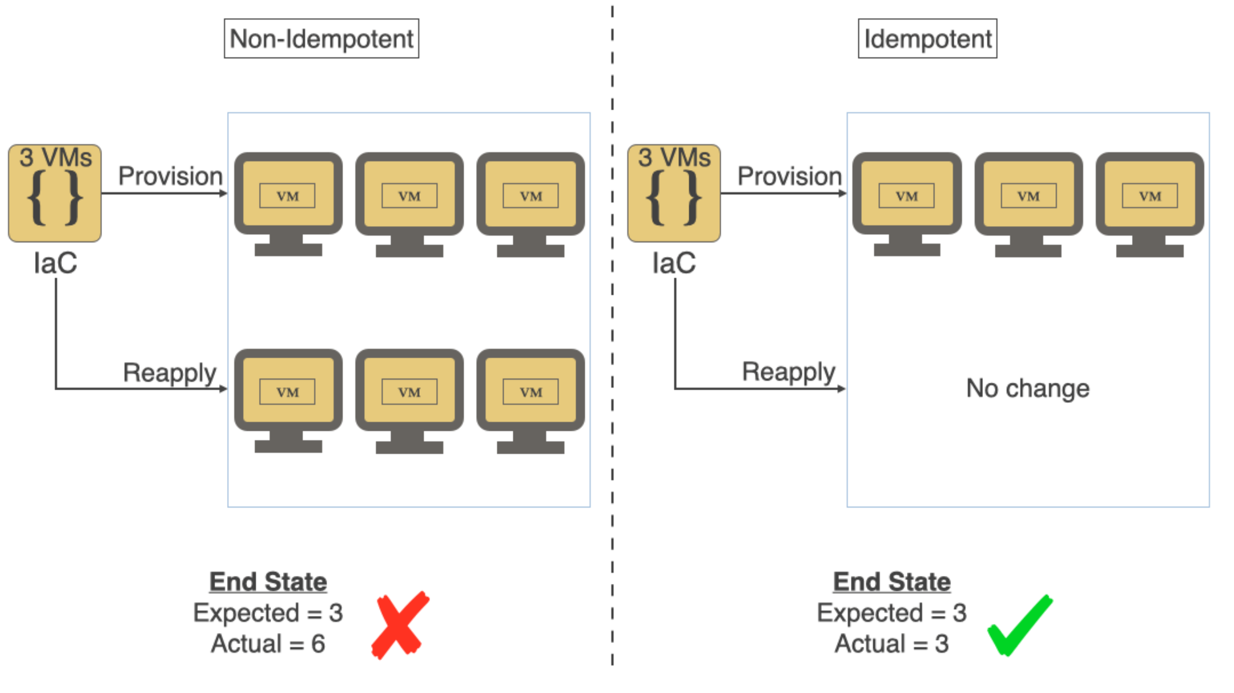
- Without IaC, teams must maintain the settings of individual deployment environments
- As the resources increase over time, it becomes difficult to reproduce the same in all the environment manually
- This can result in issues at deployment, security vulnerabilities, and risks when developing applications and services that need to meet strict regulatory compliance standards
- Configuration drift occurs when ad-hoc configuration changes and updates result in a mismatched development, test, and deployment environments
- IaC prevents drift by provisioning the same environment every time by using same configuration files for each environment

/kunchalavikram1427

Benefits of IaC

Idempotence

- Idempotency means no matter how many times you run your IaC and, what your starting state is, you will end up with the same end state
- This simplifies the provisioning of Infrastructure and reduces the chances of inconsistent results
- If the desired state is achieved, no matter how many times we run the IaC, the system will not change the configuration
- In the diagram you can see that for a Non-Idempotent IaC, if you run it twice it will provision 6 VMs instead of desired 3. In the case of Idempotent IaC, it only provisions the 3 VMs even if you run it multiple times thereby making it more reliable and consistent



Benefits of IaC

Faster time to production/market

IaC automation dramatically speeds the process of provisioning infrastructure for development, testing, and production (and for scaling or taking down production infrastructure as needed). Because it codifies and documents everything, IaC can even automate provisioning of legacy infrastructure, which might otherwise be governed by time-consuming processes (like pulling a ticket)

Faster, more efficient development

By simplifying provisioning and ensuring infrastructure consistency, IaC can confidently accelerate every phase of the software delivery lifecycle. Developers can quickly provision sandboxes and continuous integration/continuous deployment (CI/CD) environments. QA can quickly provision full-fidelity test environments. Operations can quickly provision infrastructure for security and user-acceptance testing. And when the code passes testing, the application and the production infrastructure it runs on can be deployed in one step

Reduced risks and downtimes

Making changes to the infrastructure is not the constraint anymore, because the manual changes are eliminated

Lower costs and improved ROI

In addition to dramatically reducing the time, effort, and specialized skill required to provision and scale infrastructure, IaC lets organizations take maximum advantage of cloud computing’s consumption-based cost structure. It also enables developers to spend less time on plumbing and more time developing innovative, mission-critical software solutions

Benefits of IaC

Any time available resources

Users can get the needed resources they need, when they need

Full control

You have the entire visible control over security, governance, user roles

Faster troubleshooting

Thanks to infrastructure automation, many bottlenecks are eliminated on the step of implementation and testing stage will uncover where the bug popped up

Documented infrastructure description

Documentation is always a good idea, but in case with IaC in DevOps, it serves to describe the desired state of the infrastructure. Thus, any user can read the JSON / YAML file and easily understand how the system must work

Version control

Any change is recorded, so you will know who and when did it and rollback, if the action was undesired. Besides, the team members can make an audit and suggest improvements

System consistency

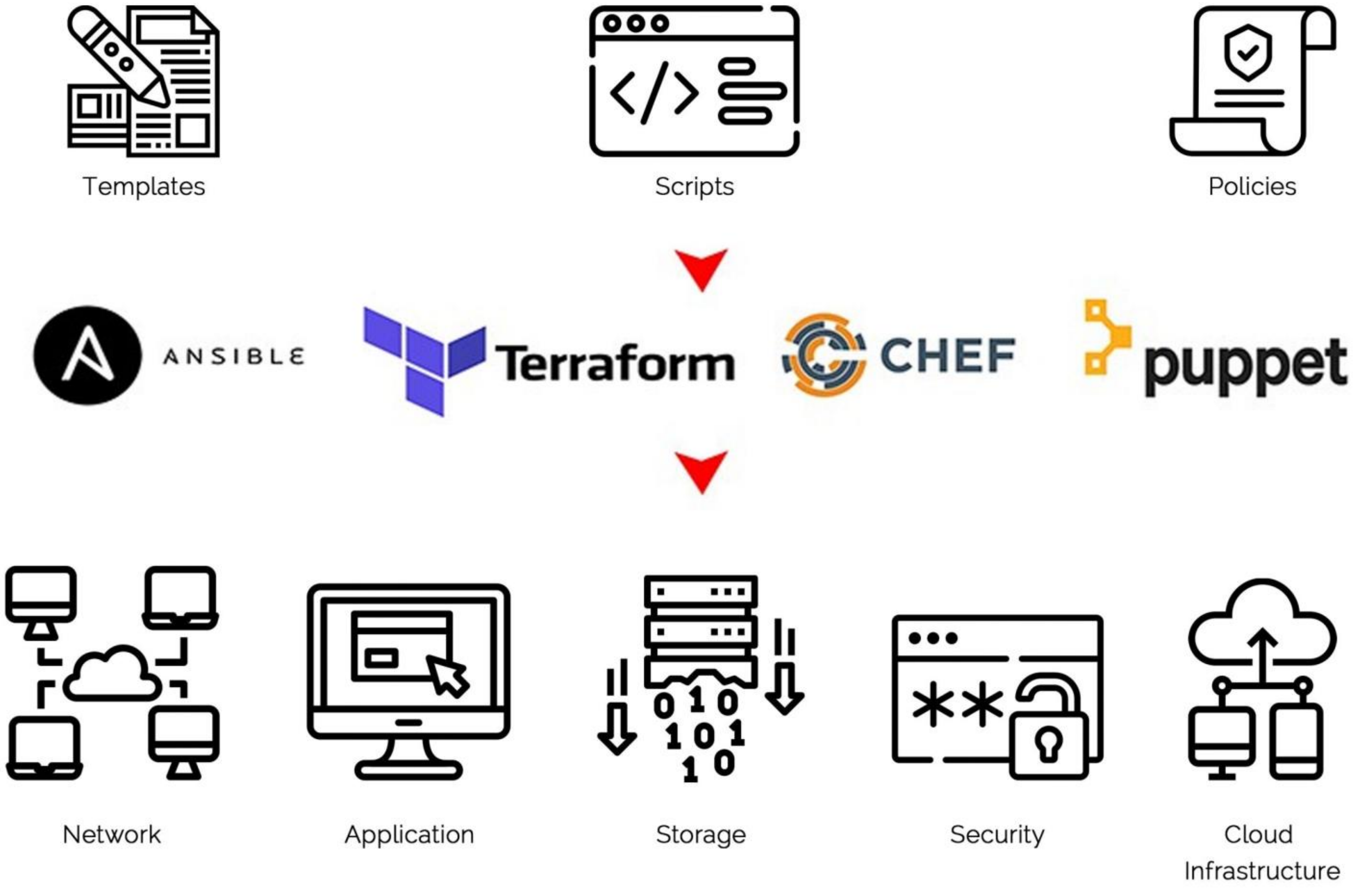
Automation is always about the code built the same way all the time. This enables predictions on how the system will behave and make testing even more efficient

Configuration Management vs Provisioning

- Ansible, Chef, Puppet, and SaltStack are all configuration management tools, which means they are designed to install and manage software on existing servers
- CloudFormation and Terraform are provisioning tools, which means they are designed to provision the servers themselves (as well as the rest of your infrastructure, like load balancers, databases, networking configuration, etc), leaving the job of configuring those servers to other tools like Ansible
- These two categories are not mutually exclusive, as most configuration management tools can do some degree of provisioning and most provisioning tools can do some degree of configuration management
- But the focus on configuration management or provisioning means that some of the tools are going to be a better fit for certain types of tasks
  Ex: Ansible for configuration management and Terraform for infrastructure provisioning

Common IaC Tools

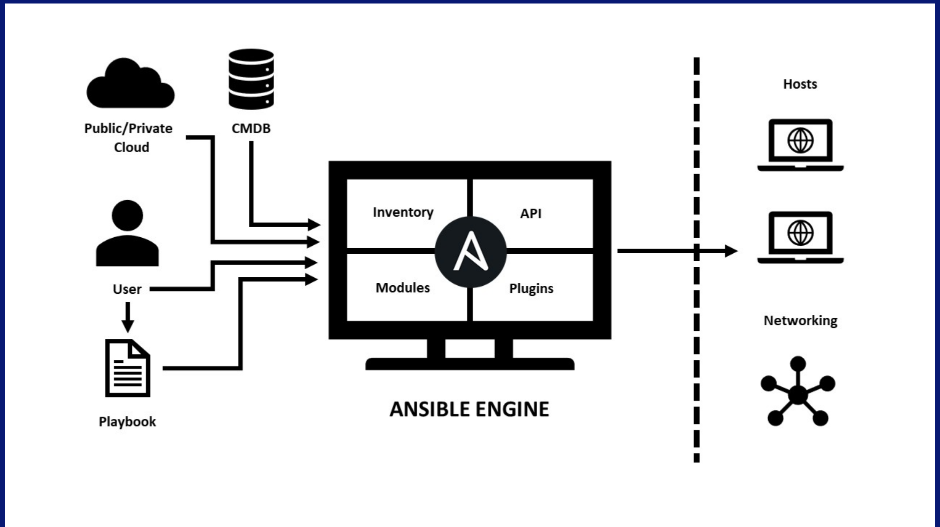
While many open-source IaC tools are available, the most commonly adopted tools are Ansible and Terraform



Common IaC Tools

Ansible

- Ansible is an open source configuration management tool by Red Hat
- It enables cloud provisioning, configuration management, apps multi-tier deployments, service orchestration, security automation, CD pipelines design etc
- Ansible is an agentless system. So, there is no need for any other software utilities or packages
- A declarative automation tool, Ansible lets you create playbooks (written in the YAML configuration language) to specify the desired state of the infrastructure and then does the provisioning
- Users execute playbooks to create and manage the required infrastructure resources
- It does not use agents and can connect to servers and run commands over SSH
- We can even expand the features of Ansible by writing your own Ansible modules and plugins
- Ansible is a popular choice for automating provisioning of Docker containers and Kubernetes deployments



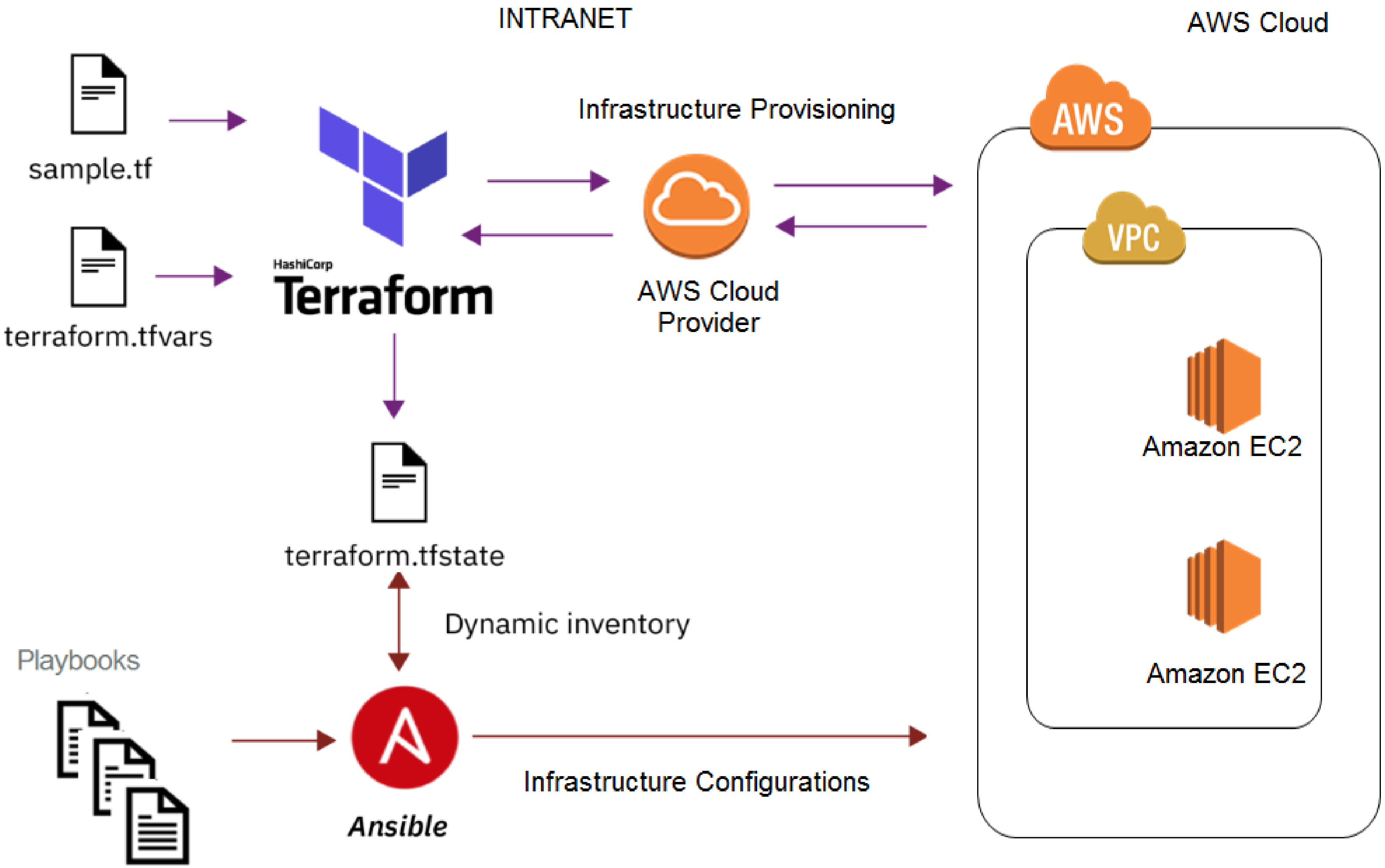
Common IaC Tools

Terraform

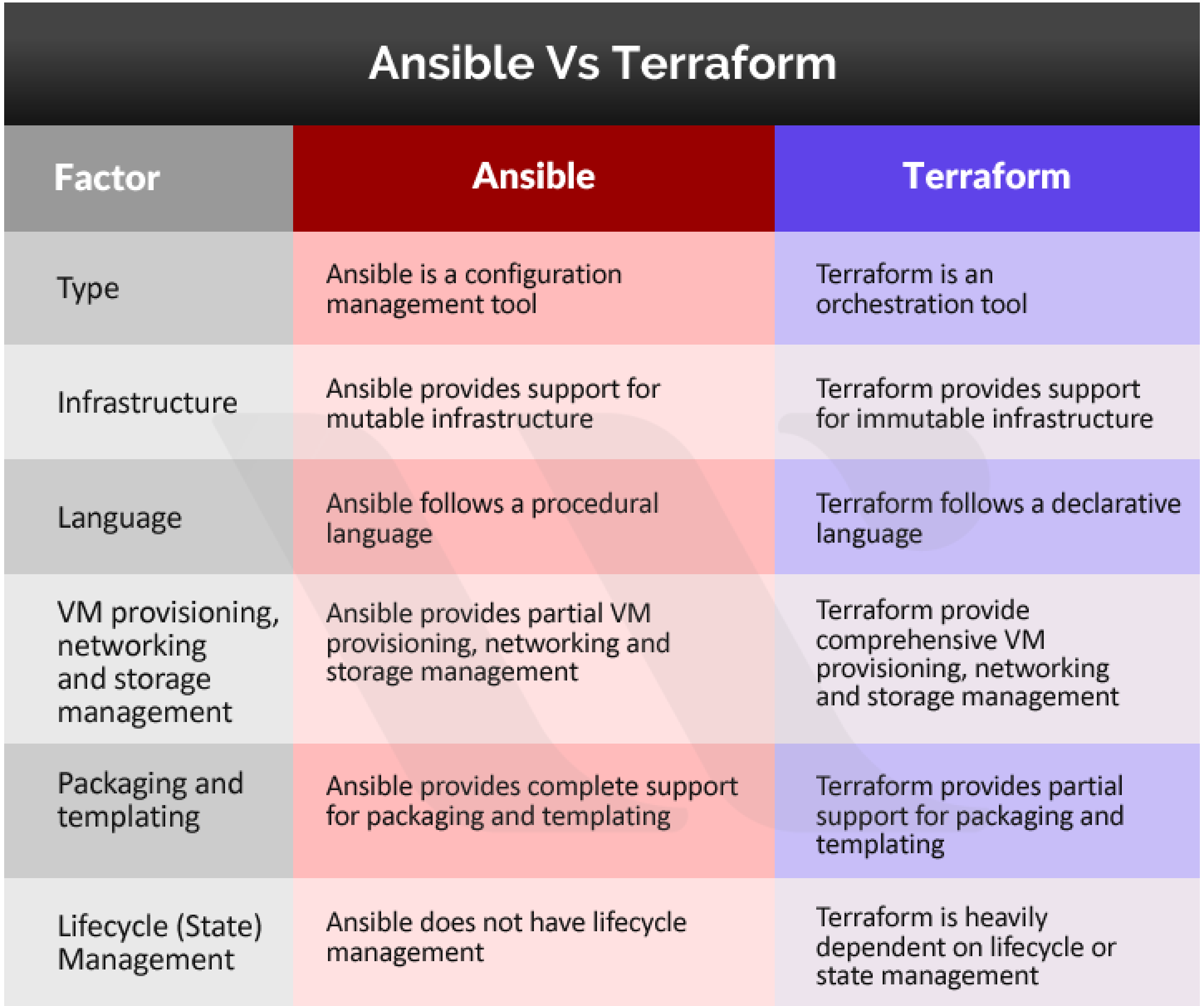
- Terraform is open source IaC tool by HashiCorp
- Terraform is another declarative provisioning and infrastructure orchestration tool that lets engineers automate provisioning of all aspects of their enterprise cloud-based and on-premises infrastructure
- Cloud provider agnostic - i.e. it is compatible with any cloud (AWS, GCP, Azure, DigitalOcean etc) and can be used across multiple clouds
- Usually it is used as a base tool and Ansible can be added to the stack in case we need additional provisioning of the servers after they are being created by terraform
- Most of the IaC tools, like Chef and Ansible, follow the client-server architecture while Terraform uses cloud provider API's to configure the infrastructure and directly communicates with the client
- Uses Configuration files in HCL(HashiCorp Configuration Language) or JSON format to define the desired state of infrastructure
- It has a planning step from which it generates an execution plans are created. Execution plans work literally as dry runs - you always know what will happen when you run the code or apply any changes to it. It helps avoid unexpected or even undesired system behaviour
- It lets you create reproducible infrastructure (prod, staging, test, dev environments)

Common IaC Tools

Ansible vs Terraform



Common IaC Tools



Ansible Vs Terraform

Factor	Ansible	Terraform
Type	Ansible is a configuration management tool	Terraform is an orchestration tool
Infrastructure	Ansible provides support for mutable infrastructure	Terraform provides support for immutable infrastructure
Language	Ansible follows a procedural language	Terraform follows a declarative language
VM provisioning, networking and storage management	Ansible provides partial VM provisioning, networking and storage management	Terraform provide comprehensive VM provisioning, networking and storage management
Packaging and templating	Ansible provides complete support for packaging and templating	Terraform provides partial support for packaging and templating
Lifecycle (State) Management	Ansible does not have lifecycle management	Terraform is heavily dependent on lifecycle or state management

Common IaC Tools

AWS CloudFormation

- Launched in 2011, Amazon was the first vendor to provide IaC tool exclusively for using with AWS
- Enables users to define templates that represent software stacks and deploy them automatically to cloud environments
- Templates are created using JSON or YAML files which is easy to understand and a single template can be used across multiple regions and multiple AWS accounts
- With CloudFormation, users can spin up automatically anything from one EC2 machine to a complex application using several AWS services including making changes to the existing services
- We can also recall changes using Rollback Triggers

Azure Resource Manager(ARM) and Google Cloud Deployment Manager(DM)

- Just like AWS, Azure and GCP have their own IaC tools viz. Azure Resource Manager (ARM) and Google Cloud Deployment Manager (DM)
- To configure and implement infrastructure in Azure, you can develop your ARM template using JSON which can be deployed via Powershell script or directly from within the Azure Portal
- Similarly, for DM you can create templates using YAML or Python

https://www.nexastack.com/en/blog/best-iac-tools

IaC Tools

Tool	Tool type	Infrastructure	Architecture	Approach	Language
CHEF	Config management	Mutable	Pull	Declarative & Imperative	Ruby
puppet	Config management	Mutable	Pull	Declarative	DSL & ERB
SALTSTACK	Config management	Mutable	Push & pull	Declarative & Imperative	YAML
AWS CloudFormation	Provisioning	Immutable	Push	Declarative	JSON & YAML
aws Cloud Development Kit	Provisioning	Immutable	Push	Declarative	TS, JS, Python, Java, C#/.Net
ANSIBLE	Config management	Mutable	Push	Declarative & Imperative	YAML
Terraform	Provisioning	Immutable	Push	Declarative	HashiCorp Configuration Language
pulumi Cloud Native Infrastructure as Code	Provisioning	Immutable	Push	Declarative	JS, TS, Python, Go, NET language, incl C#, F#, and VB

Best practices for IaC

Make code your single source of truth. You should explicitly code all the infrastructure specifications in configuration files. Your configuration files should be the single source of truth for all your infrastructure management concerns

Version control all of your configuration files. This probably should go without saying, but put all of your config files under source control. This provides a detailed audit trail for changes

Use little documentation (or none at all) for your infrastructure specifications. Since your config files should be your single source of truth, there should be no need for more documentation. The code will document the state of the machine, and so infrastructure documentation will be updated. External documentation can easily get out of sync with the real configurations, but that won't happen with the config files

Test and Monitor Your Configurations. IaC is code, and like all code, it can be tested. By employing testing and monitoring tools for IaC, you can check for errors and inconsistencies in your servers before you deploy them to production

HashiCorp

Terraform

Deliver Infrastructure as Code

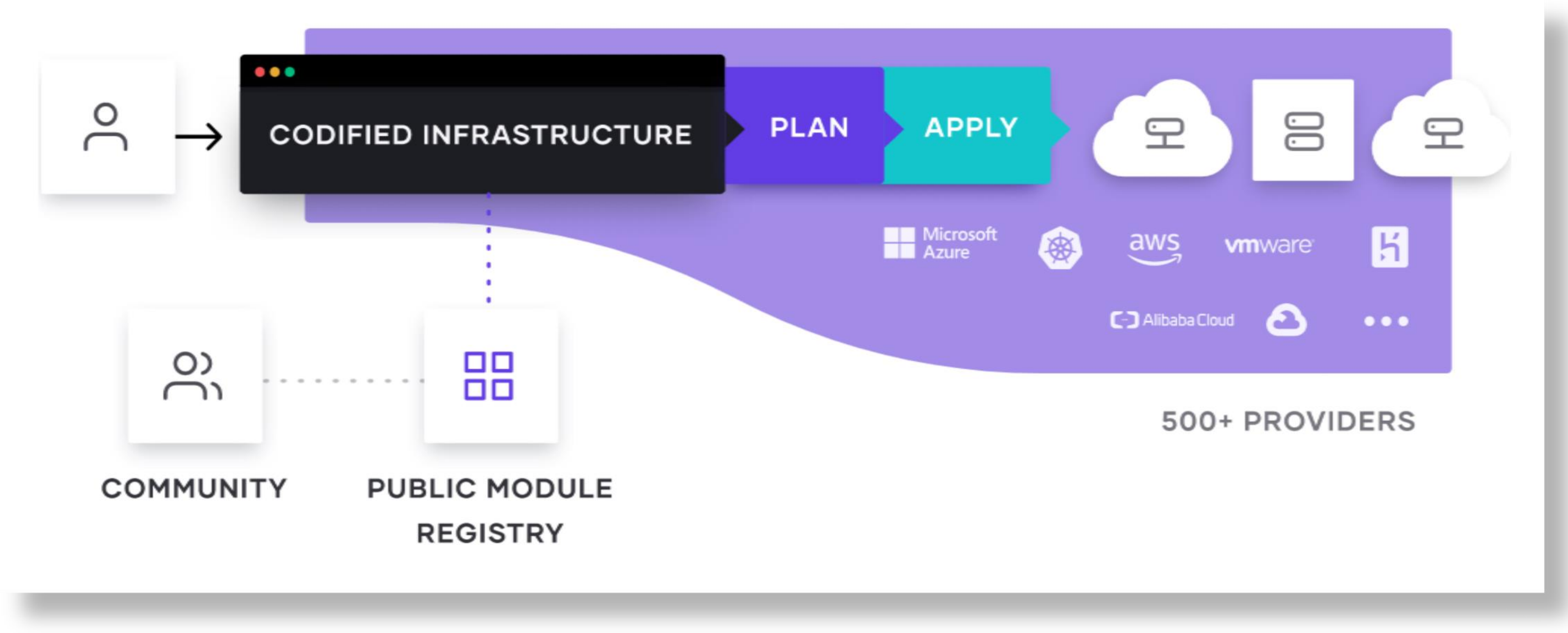
/kunchalavikram1427

Terraform

- Terraform is open source IaC tool by HashiCorp
- Terraform is another declarative provisioning and infrastructure orchestration tool that lets engineers automate provisioning of all aspects of their enterprise cloud-based and on-premises infrastructure
- Cloud provider agnostic - i.e. it is compatible with any cloud (AWS, GCP, Azure, DigitalOcean etc) and can be used across multiple clouds
- Usually it is used as a base tool and Ansible can be added to the stack in case we need additional provisioning of the servers after they are being created by terraform
- Most of the IaC tools, like Chef and Ansible, follow the client-server architecture while Terraform uses cloud provider API's to configure the infrastructure and directly communicates with the client
- Uses Configuration files in HCL(HashiCorp Configuration Language) or JSON format to define the desired state of infrastructure
- It has a planning step from which generates an execution plans are created. Execution plans work literally as dry runs - you always know what will happen when you run the code or apply any changes to it. It helps avoid unexpected or even undesired system behaviour
- It lets you create reproducible infrastructure (prod, staging, test, dev environments)
- Terraform focuses more on server provisioning. When the complete cloud infrastructure is considered as code, and all the parameters were combined in a configuration file, all the members of the team can easily collaborate on them as they would do any other code

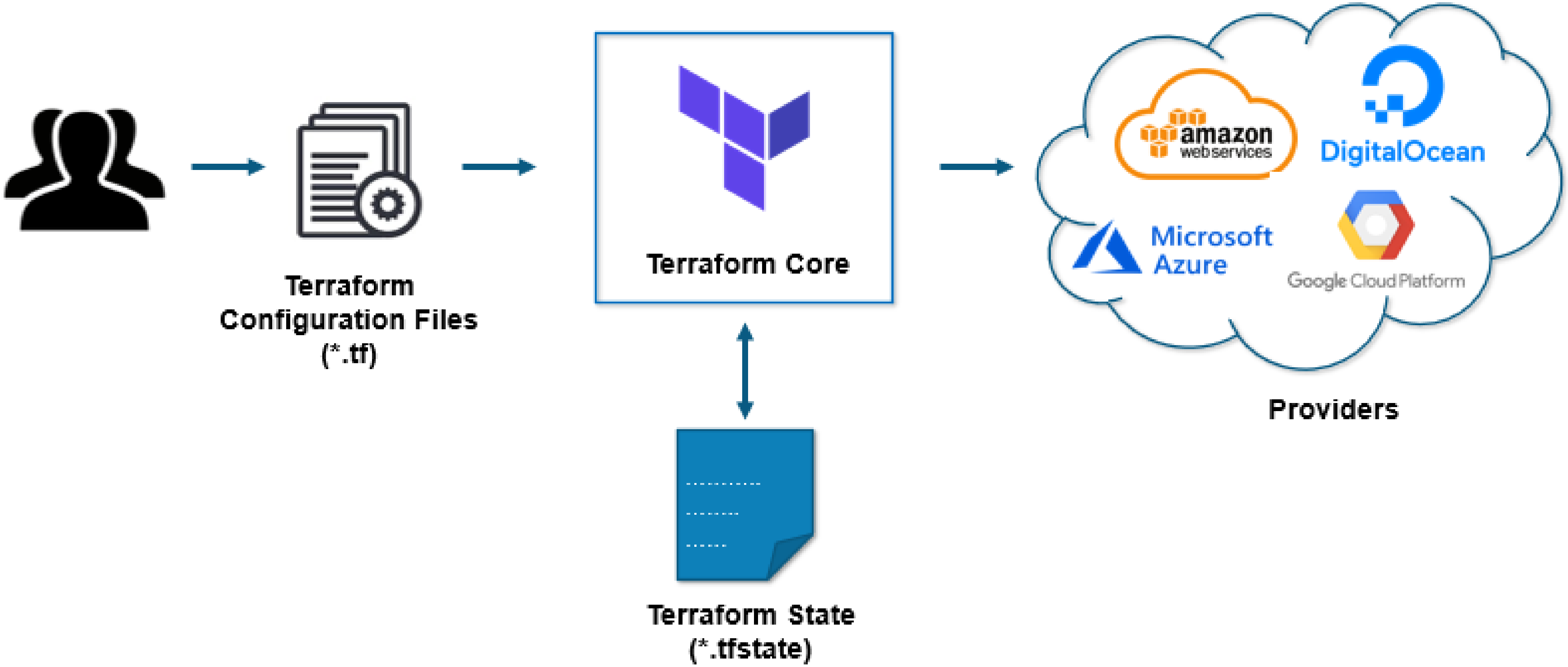
Terraform

Working



Terraform

Architecture



/kunchalavikram1427

Terraform

Advantages

- Multi-Provider – The most versatile feature of Terraform is that it supports multi-cloud orchestration like AWS, Azure, OpenStack, etc. and also premises deployments
- Immutable Infrastructure – When using Chef, Puppet, Salt runs any software updates on servers; this often leads to configuration drift when differences in the configuration lead to bugs that lead to security breaches. Terraform solves this issue by using an immutable infrastructure approach in which every change in configuration leads to a separate configuration snapshot which means de-provisioning the old one and deployment of a new one
- Syntax – HCL (HashiCorp Configuration Language) is a custom language that is used by Terraform
- Dry Runs – Terraform uses a command known as terraform plan, which creates an execution plan; by using this execution plan, we can check whether the set of changes meets the expectation without performing any changes to real resources or state. For example, by running the terraform plan before committing a change to version control, check whether it will work as expected or not
- Client-only architecture – Terraform eliminates the need for additional checks for provisioning the infrastructure, which leverages the cloud provider's API. In Ansible, this is done by connecting through SSH but with limitations. Terraform works on APIs and have a wide variety of options, which helps to make it more secure, reliable, and easy to use
- Super Portability – There are only a single tool and single language for describing the infrastructure which is used for multiple cloud providers. The problem of migrating to vendors is not a problem

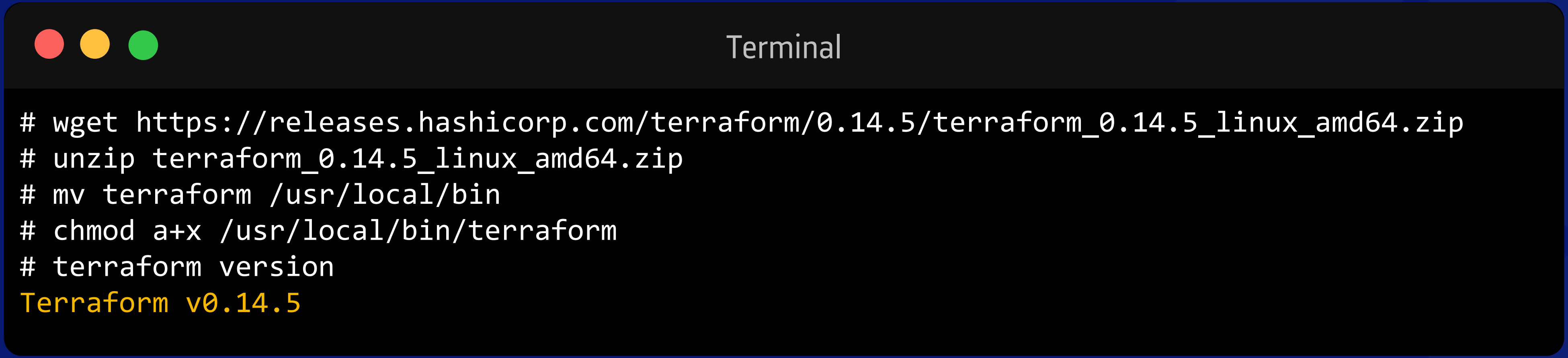
Installing Terraform

Installing Terraform

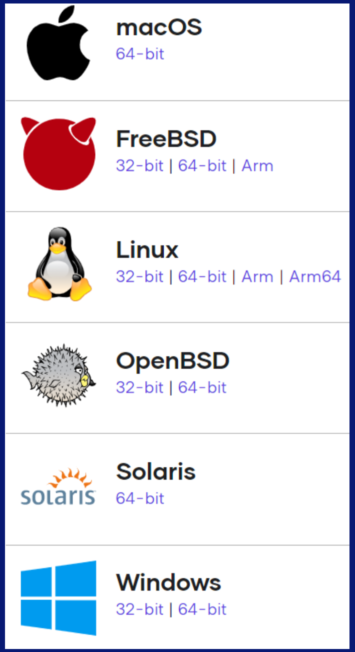
- Download the latest version for your OS from https://www.terraform.io/downloads.html

For linux

```
wget https://releases.hashicorp.com/terraform/0.14.5/terraform_0.14.5_linux_amd64.zip
unzip terraform_0.14.5_linux_amd64.zip
mv terraform /usr/local/bin
chmod a+x /usr/local/bin/terraform
```



```
# wget https://releases.hashicorp.com/terraform/0.14.5/terraform_0.14.5_linux_amd64.zip
# unzip terraform_0.14.5_linux_amd64.zip
# mv terraform /usr/local/bin
# chmod a+x /usr/local/bin/terraform
# terraform version
Terraform v0.14.5
```



macOS
64-bit

FreeBSD
32-bit | 64-bit | Arm

Linux
32-bit | 64-bit | Arm | Arm64

OpenBSD
32-bit | 64-bit

Solaris
64-bit

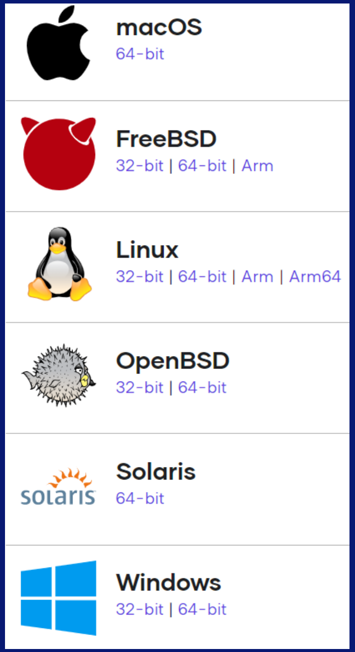
Windows
32-bit | 64-bit

ℹ Run `terraform version` to get the version of Terraform installed

Installing Terraform

For Windows

- Download the appropriate package from https://www.terraform.io/downloads.html
- Extract the package to the folder C:\Program Files (x86) or to any other location in your local system. Ex: C:\terraform
- Update the path environment variable to include the folder where your Terraform executable is located
  1. On a Windows 10 system, Go to Control Panel -> System and Security -> System
  2. click Advanced system settings on the left and then Advanced tab of the System Properties
  3. Click Environment Variables near the bottom of the window
  4. In the System variables pane, click Path and then click Edit
  5. Click New. Add the path to the folder where your Terraform executable is located. Ex: C:\terraform
  6. Click OK to save your changes and then click OK to exit the Environment Variables windows. Then click OK again to exit the System Properties window
  7. To verify your installation and check the version, launch Windows PowerShell and enter: `terraform version`



macOS
64-bit

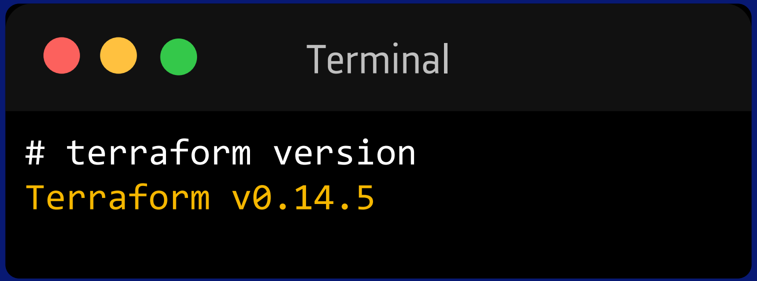
FreeBSD
32-bit | 64-bit | Arm

Linux
32-bit | 64-bit | Arm | Arm64

OpenBSD
32-bit | 64-bit

Solaris
64-bit

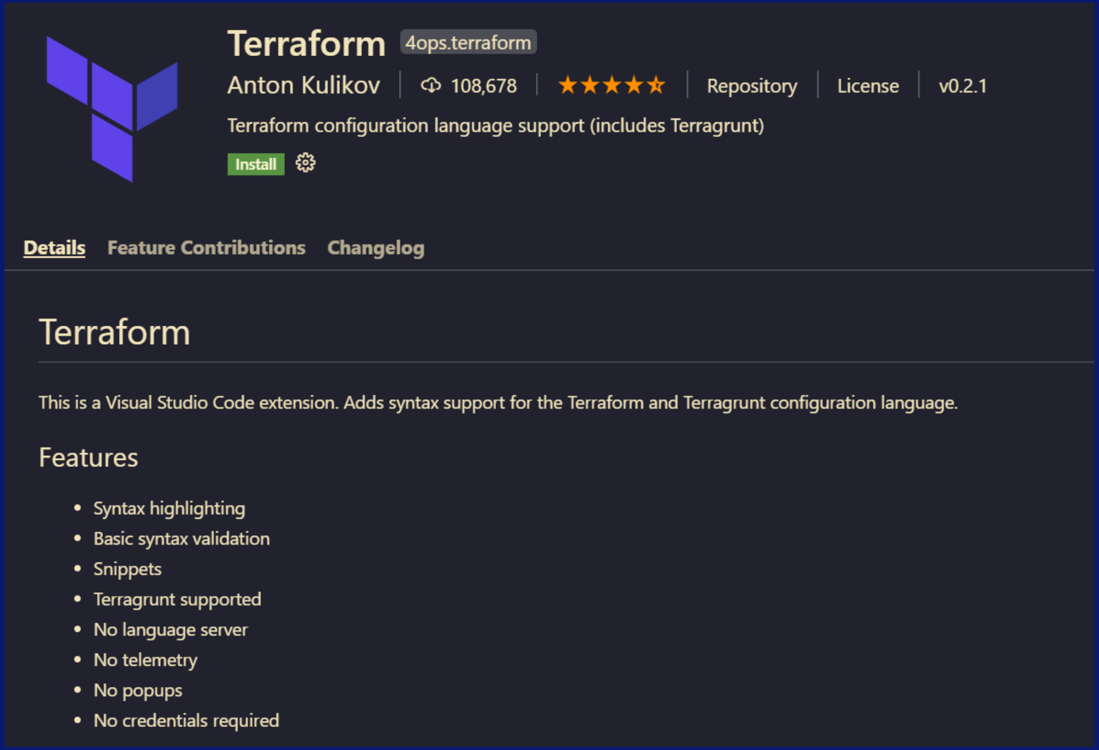
Windows
32-bit | 64-bit



```
# terraform version
Terraform v0.14.5
```

Terraform

VS Code Extensions



Terraform 4ops.terraform

Anton Kulikov | 108,678 | ★★★★☆ | Repository | License | v0.2.1

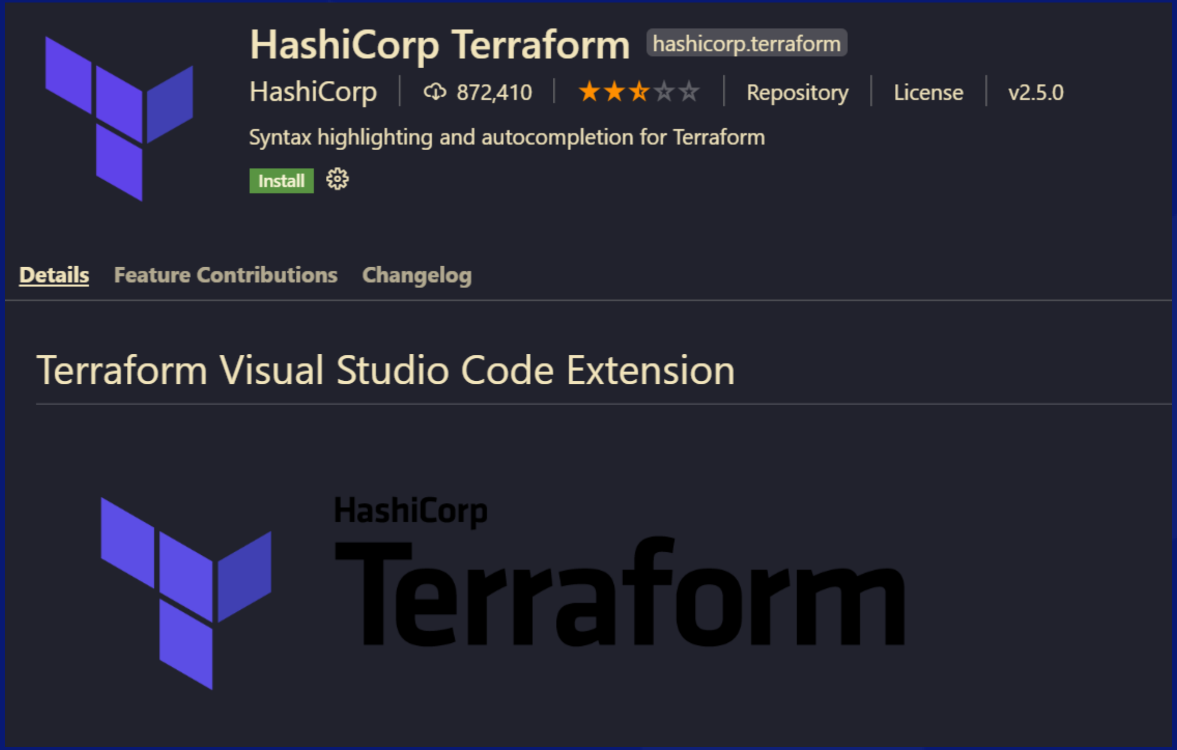
Terraform configuration language support (includes Terragrunt)

Terraform

This is a Visual Studio Code extension. Adds syntax support for the Terraform and Terragrunt configuration language.

Features

- Syntax highlighting
- Basic syntax validation
- Snippets
- Terragrunt supported
- No language server
- No telemetry
- No popups
- No credentials required



HashiCorp Terraform hashicorp.terraform

HashiCorp | 872,410 | ★★★☆☆ | Repository | License | v2.5.0

Syntax highlighting and autocompletion for Terraform

Terraform Visual Studio Code Extension

More content on the way . . .

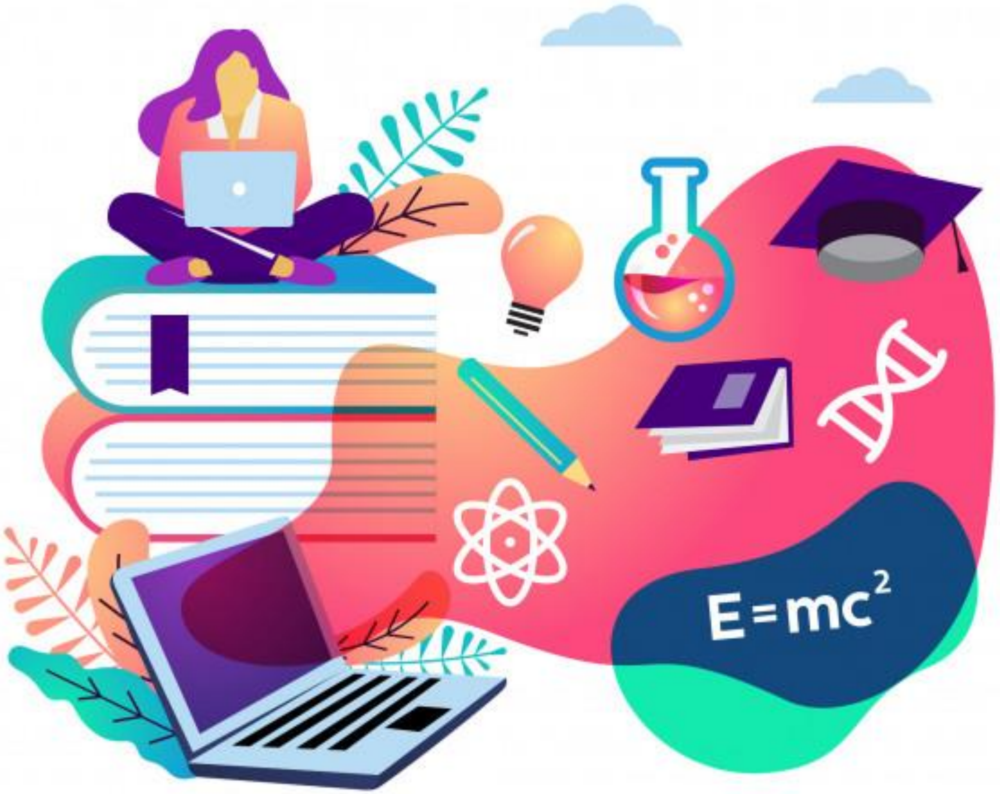
Subscribe to my Facebook page:
https://www.facebook.com/vikram.devops

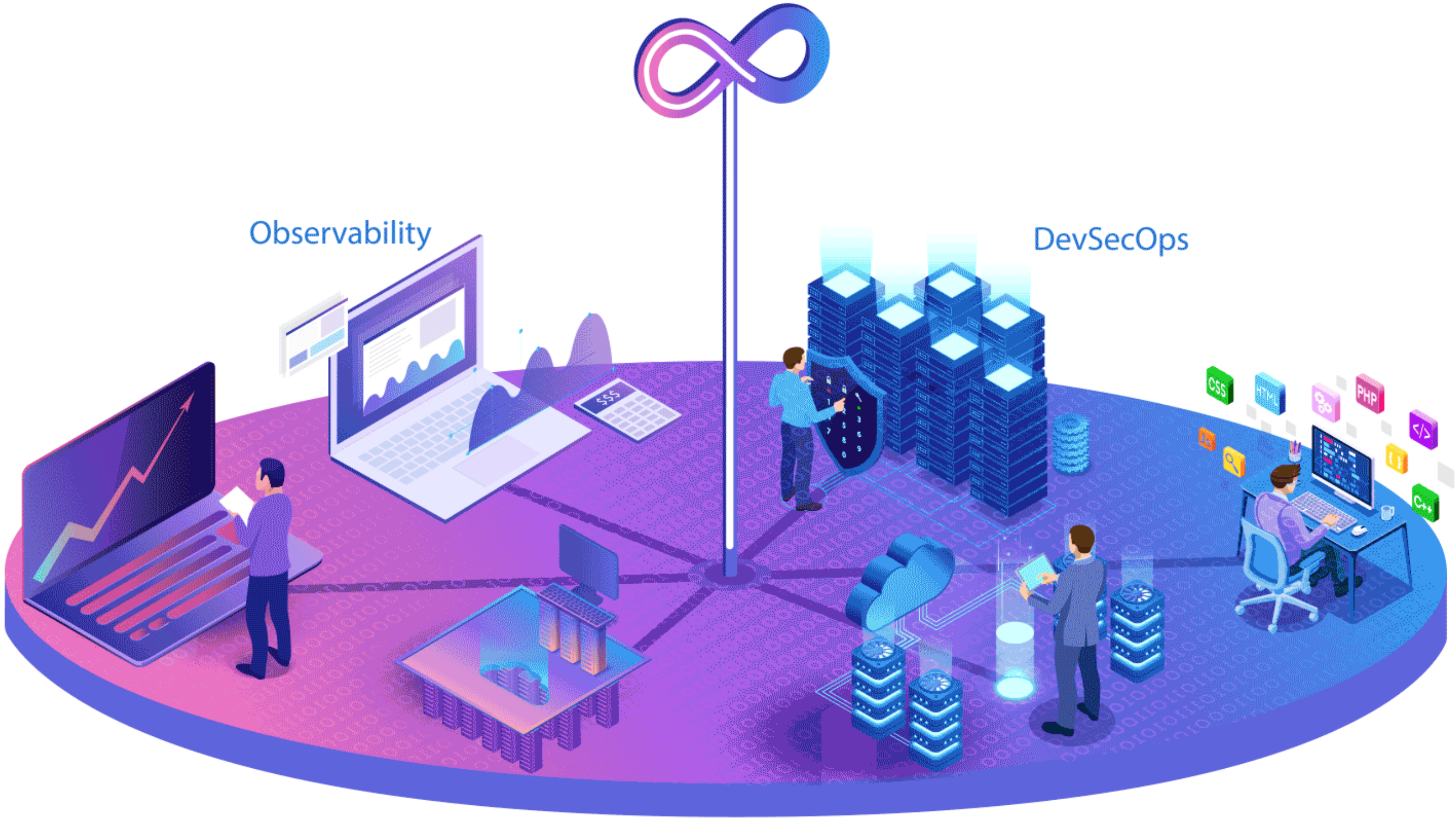
and join my group:
https://www.facebook.com/groups/171043094400359

for all latest updates on DevOps

References

- https://www.terraform.io/docs/index.html
- https://www.terraform.io/docs/cli/index.html
- https://learn.hashicorp.com/collections/terraform/aws-get-started?utm_source=WEBSITE&utm_medium=WEB_IO&utm_offer=ARTICLE_PAGE&utm_content=DOCS
- https://medium.com/faun/infrastructure-as-code-917d245ce35e
- https://www.terraform-best-practices.com/code-structure
- https://shahadarsh.com/2020/07/12/principles-patterns-and-practices-for-effective-infrastructure-as-code/





Observability

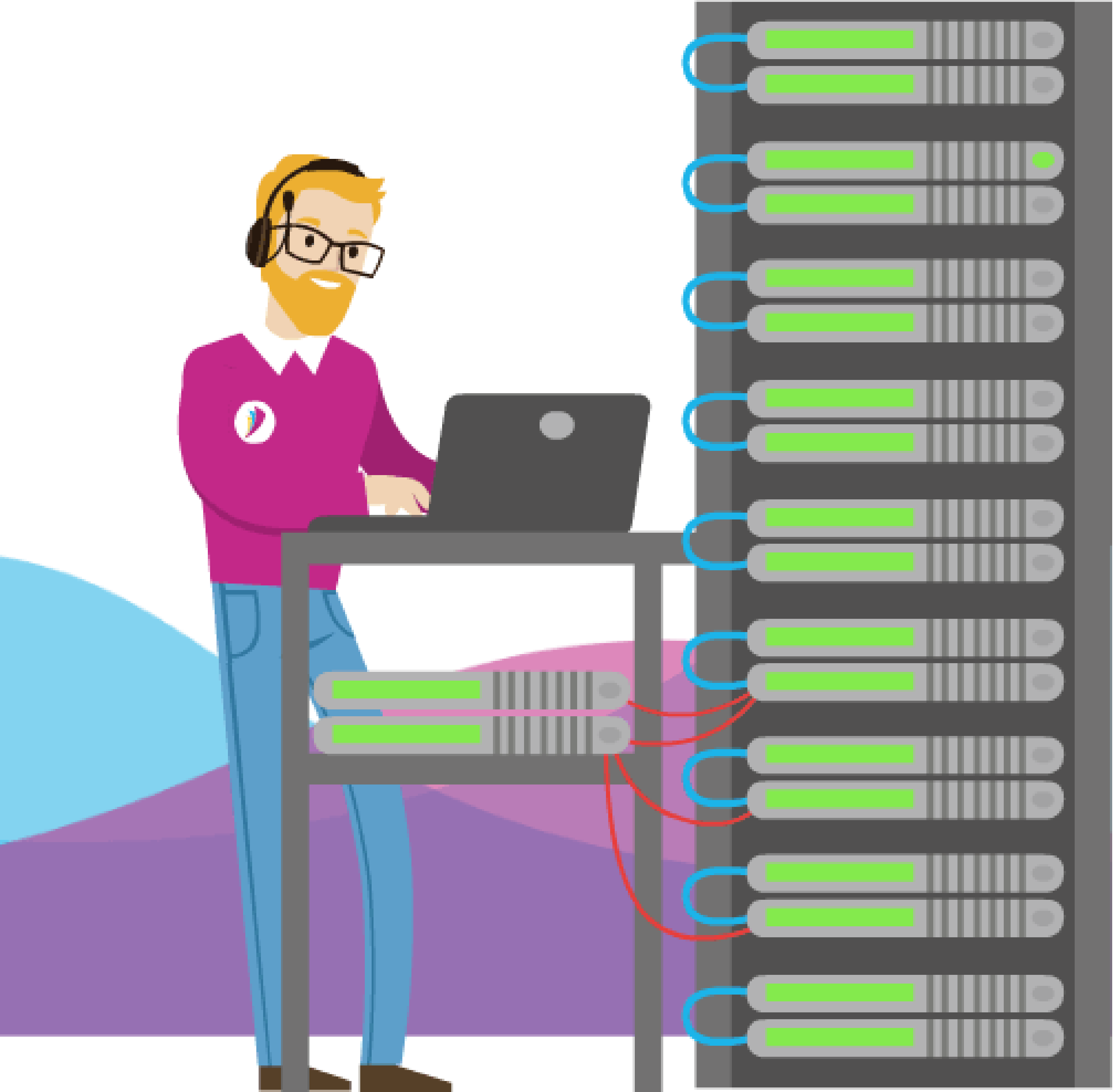
DevSecOps

Cloud Native Development (Microservices)

SRE & Chaos Engineering

Cloud Automation

DevOps Services



/kunchalavikram1427