**Infrastructure As Code**

Every application runs inside an **environment.**

An environment is a stack of software and hardware components that supports the application.

* Building an environment involves many activities such as:
  + Bringing up components.
  + Provisioning and configuring components based on need.

**IaC** is an important DevOps important feature that enables you to define, automatically manage, and provision infrastructure through source code.

Infrastructure is managed like a software system.

**IaC** is frequently referred to as Programmable Infrastructure.

**Why IaC?**

* Everything required to run the services are connected, deployed and monitored together.
* Faster delivery of quality services
* Built-in automation.
* Changes can be made faster than manual.

**What would happen if something fails on a server?**

**Traditionally,**

* A support person from the IT team will physically reach the server and make the fix.
* Consumes workforce, which is a slower process.

**In IaC,**

* A tool will monitor the state of the infrastructure.
* A script will be sent automatically to fix the issue.
* Faster process, no/less human involvement.

**Challenges Addressed by IaC**

* **Server Sprawl** **-** Growth in the number of servers makes it difficult to manage.
* **Configuration drift** **-** Differences in version and configurations across servers.
* **Snowflake Server -** Requires special configuration and is different from other servers in the network.
* **Fragile infrastructure -** Infrastructure disrupted easily but is hard to fix.

**IaC Principles**

Principles you should keep in mind while designing your infrastructure

* Systems can be reproduced with fewer efforts
* Systems are replaceable
* Systems are consistent
* Processes are repeatable
* Ever evolving design

**Infrastructure that can be defined as a code is broadly categorized as:**

* Compute
* Databases, Caching, and Messaging
* Storage, Backup, and Content Delivery
* Networking
* Security and Identity
* Monitoring, Logging, and Analytics
* Management Tooling

**IaC Implementation Approaches**

**Declarative**

* Focuses on the desired end state of infrastructure (Functional).
* Tools perform the necessary actions to reach that state.
* Examples are **Terraform, CloudFormation, and Puppet.**

**Imperative**

* Focuses on how to achieve the desired state (Procedural).
* Examples are **Chef, Ansible.**

**Declarative or Imperative?**

Imperative model has the following issues over declarative:

* The **state** of the infrastructure **is not fully captured**, so you may **need to know the order** in which you applied the changes.
* You should ***remember the full history of changes*** that has ever happened to the infrastructure.
* The ***code bases tend to grow large and get complicated over time*** because of the inherent limitation of re-usability.

**Mutable and Immutable**

\*\*\*Mutable Infrastructure\*\*\*:

* When a **change is required**, the system in production is **patched**.
* Sometimes it might be confusing.
* Bigger differences between systems in production and development.

\*\*\*Immutable Infrastructure\*\*\*:

* When a **change is required**, it **completely replaces** the servers.
* It increases the predictability.
* Only **a minor difference** between the system in testing and system in **production**.

**Disposability principle** - \*\*\*Do not fix it, throw it away\*\*\*.

**Dynamic Infrastructure Platform**

* Dynamic Infrastructure Platform (DIP) lets you **create, destroy, replace, resize, and move** the resources readily, which **enhances disposability**.
* **Software must continue running even when servers are added, removed or resized.**

Requirements

* \*\*\*On Demand\*\*\*:

1. The resources can be created or destroyed dynamically.
2. Plan your budget to be growing on demand.

* \*\*\*Programmable\*\*\*:

1. The API must be scriptable.

* \*\*\*Self-service\*\*\*:

1. Increases the availability and performance.

**Dynamic Infrastructure platform (DIP) resource offers three key blocks.**

* \*\*\*Compute Resources\*\*\*: Server instance that makes the **server management** task more **convenient** and **powerful**.
* \*\*\*Network Resources\*\*\*: The platform should be able to **enable connectivity** between **internal** as well as **external terminals**.
* \*\*\*Storage Resources\*\*\*: As your **infrastructure is dynamic**, your required storage too must be dynamic.

**Choosing a platform, consider all these factors:**

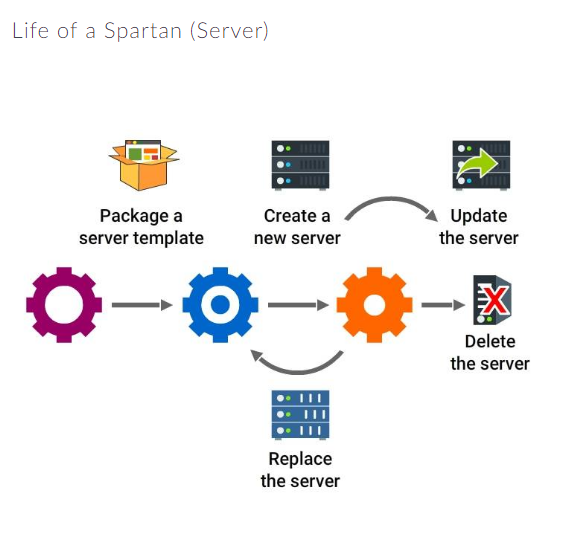
* Security and data protection
* Requirements for Variable capacity
* Legal Constraints on hosting locations
* Build Cost
* Commodity or differentiator

**Before Picking a Cloud Provider:**

* Supports various services
* **Dependency Resolution** - Downloading all the dependency related information from the master once a dependency is identified in the configuration file.
* **Idempotency** - The same configuration files should yield the same result every time it gets called
* **Existing** tool integration
* Platform **agnostic** templates
* **Extendibility**

Benefits of Server-less Configurations

* Auto-scale
* Pay-as-you-use
* Supports Micro services



**Tools**

1. **Terraform**

* An open source tool developed by HashiCorp
* Written in Go.
* Used to build, change and combine infrastructures safely and efficiently.
* Detects the state changes in an infrastructure and builds the \*terraform plan\*.

1. **CloudFormation**

* Proprietary tool, developed by Amazon
* Supports all the AWS resources
* Easy to configure AWS resources through CloudFormation
* CloudFormation Designer tool may be used to generate the code designing the infrastructure by drag and drop option.

1. **OpenStack Heat**

* Heat is an infrastructure definition tool developed by OpenStack.
* Its orchestration engine takes templates as a base and creates multiple composite cloud apps.
* Written in Python.
* Uses text files to define the templates.
* Supports many existing templates of CloudFormation tool.

1. **Chef**

* A tool written in Ruby and Erlang to automate the configuration management tasks.
* Pure Ruby, DSL (Domain Specific Language) are the main ingredients of all her Recipes.

**Chef - The actual definition**

* Chef is a server configuration tool.
* You can write Recipes to describe your server applications and a group of Recipes makes a Cookbook.
* HP Public Cloud, Facebook and Prezi are using Chef for their automation.