Infrastructure as Code

What is IaC, Provision and Configuration Management tools

Technical Trainers

SoftUni Team

Software University

SoftUni

https://softuni.bg

Software University

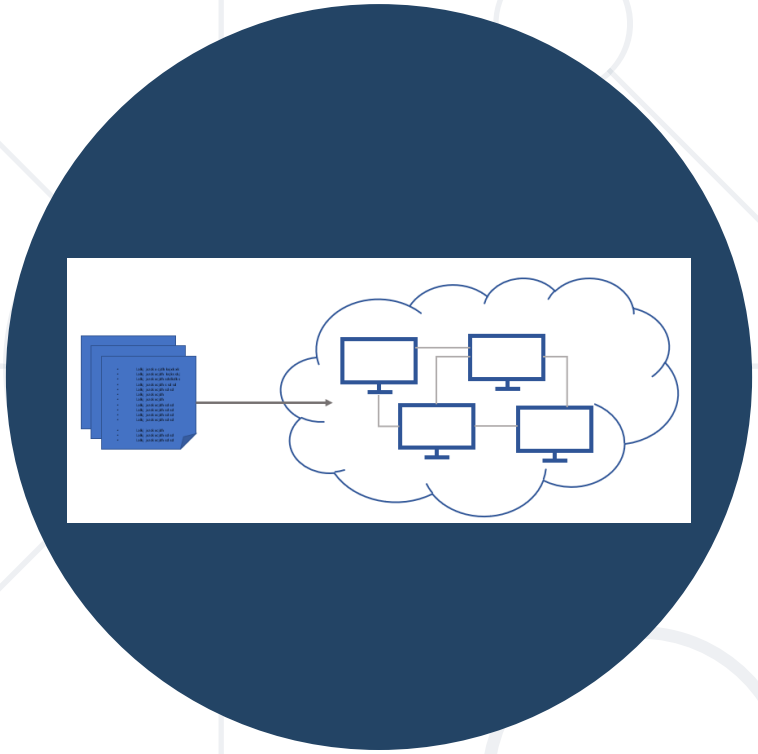
Have a Question?

Software University

sli.do

#Dev-Ops

Table of Content

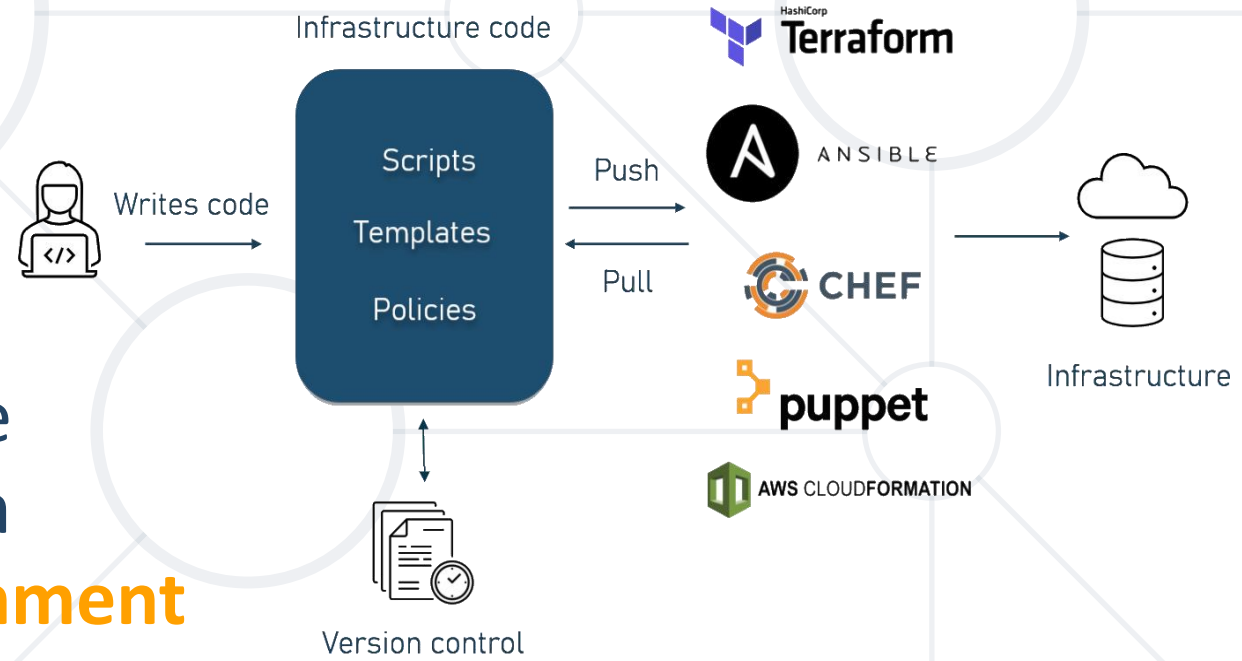


Infrastructure as Code

Automating Infrastructure Management Using Code

What is IaC?

- **Infrastructure as Code** (**IaC**) is the managing and provisioning of **infrastructure through code** instead of through manual processes
  - As VMs, networks, OS servers, storage, etc.
- **IaC** involves
  - Writing **code** to define the desired state of an **infrastructure environment**
  - Using **tools** to **automatically** deploy and configure the environment based on the code

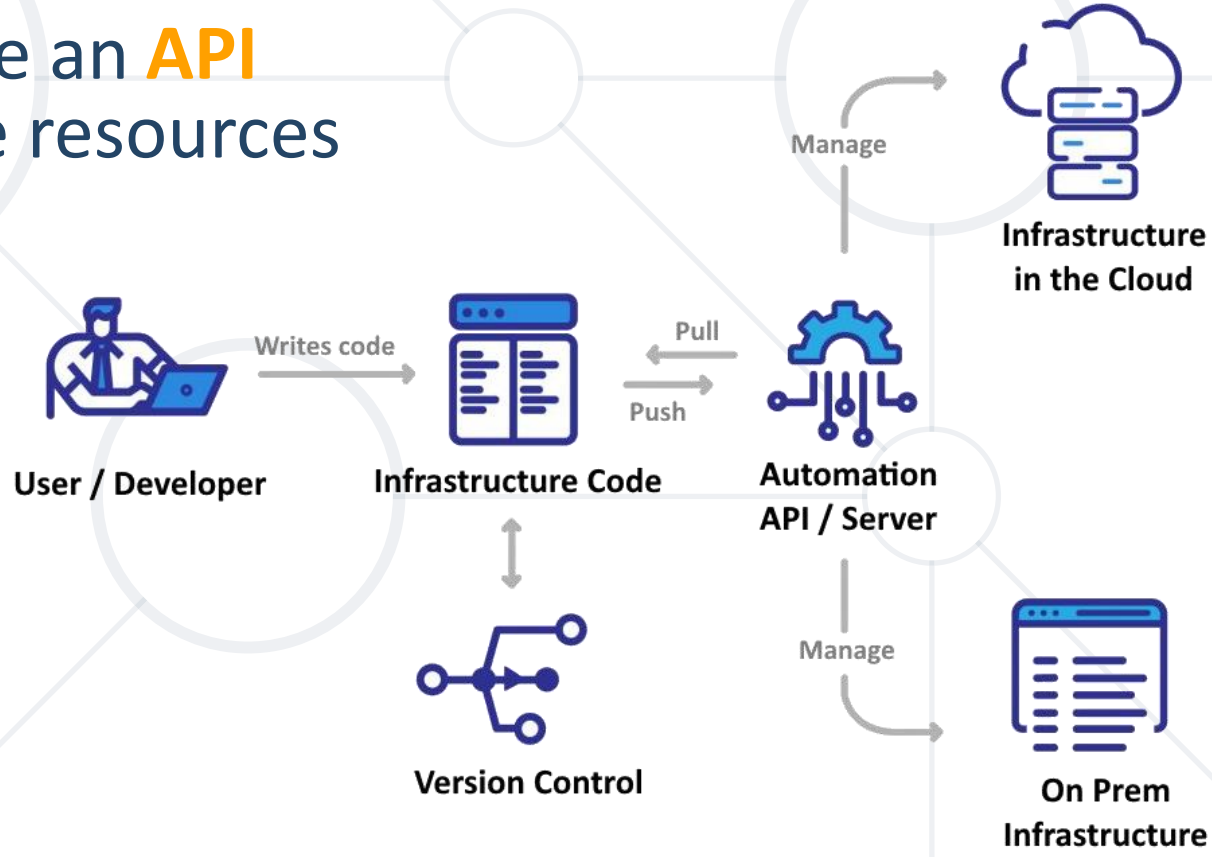


IaC Configuration Files

- **IaC** is a form of configuration management that **codifies infrastructure resources** into text files
- **Configuration files** are created with your infrastructure specifications
  - Should be **version controlled** and **tested** (unit, integration, ... tests)
  - Ensure that you provision the **same environment every time**
  - Allow you to divide your infrastructure into **modular components** and combine them through automation
  - Should contain always **up-to-date infrastructure documentation**

What Do You Need for IaC?

- **Remote accessible hosting** or **IaaS cloud hosting platform**
  - IaC tools connect and modify remote host
  - IaaS cloud hosting platforms have an **API** for modification of infrastructure resources
- **Provisioning tool**
  - Automates the infrastructure deploy and management
- **Configuration management tool**
  - Manages infrastructure state
- **Version control system**
  - Stores text files used by the CM platform

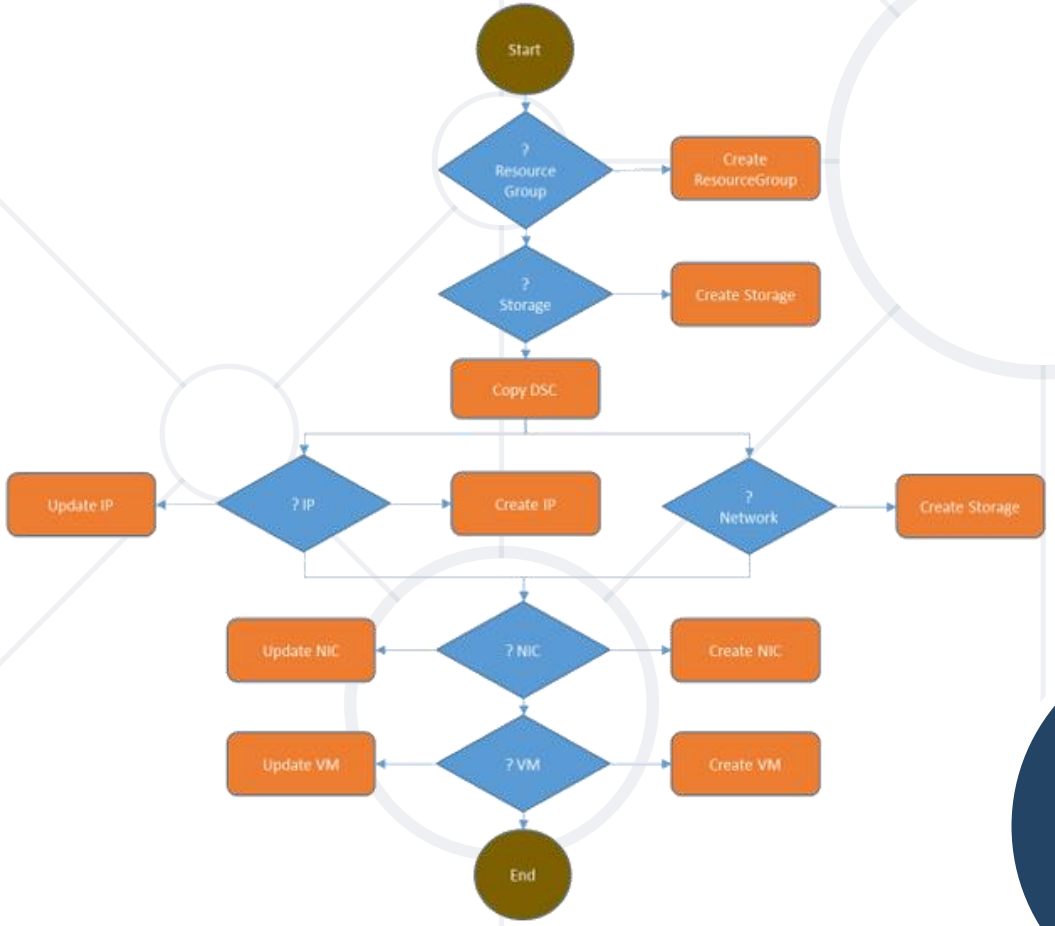


Approaches to IaC

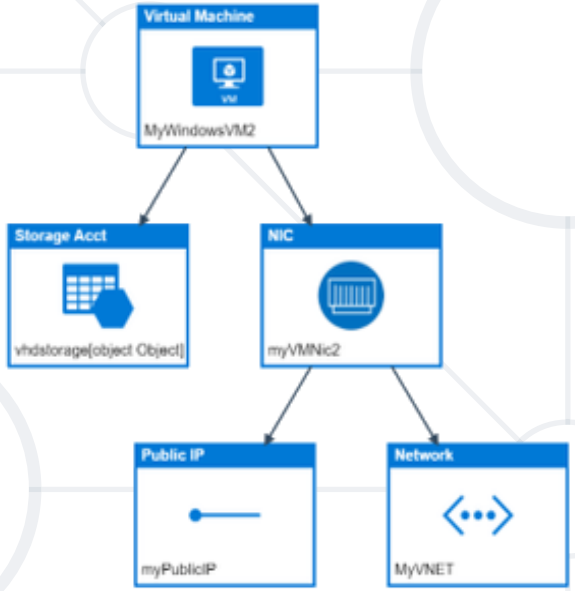
- **Imperative approach**
  - Tell the system **how** to do something every step of the way
  - Defines the **specific commands** to be executed in a **specific order** for the **desired configuration**
- **Declarative approach**
  - Tell the system **what** you want and let it figure out how to do it
  - Defines the **desired state** of the **system** – resources, their properties and an IaC tool for configuration

Imperative vs Declarative Approach

- Imperative Approach

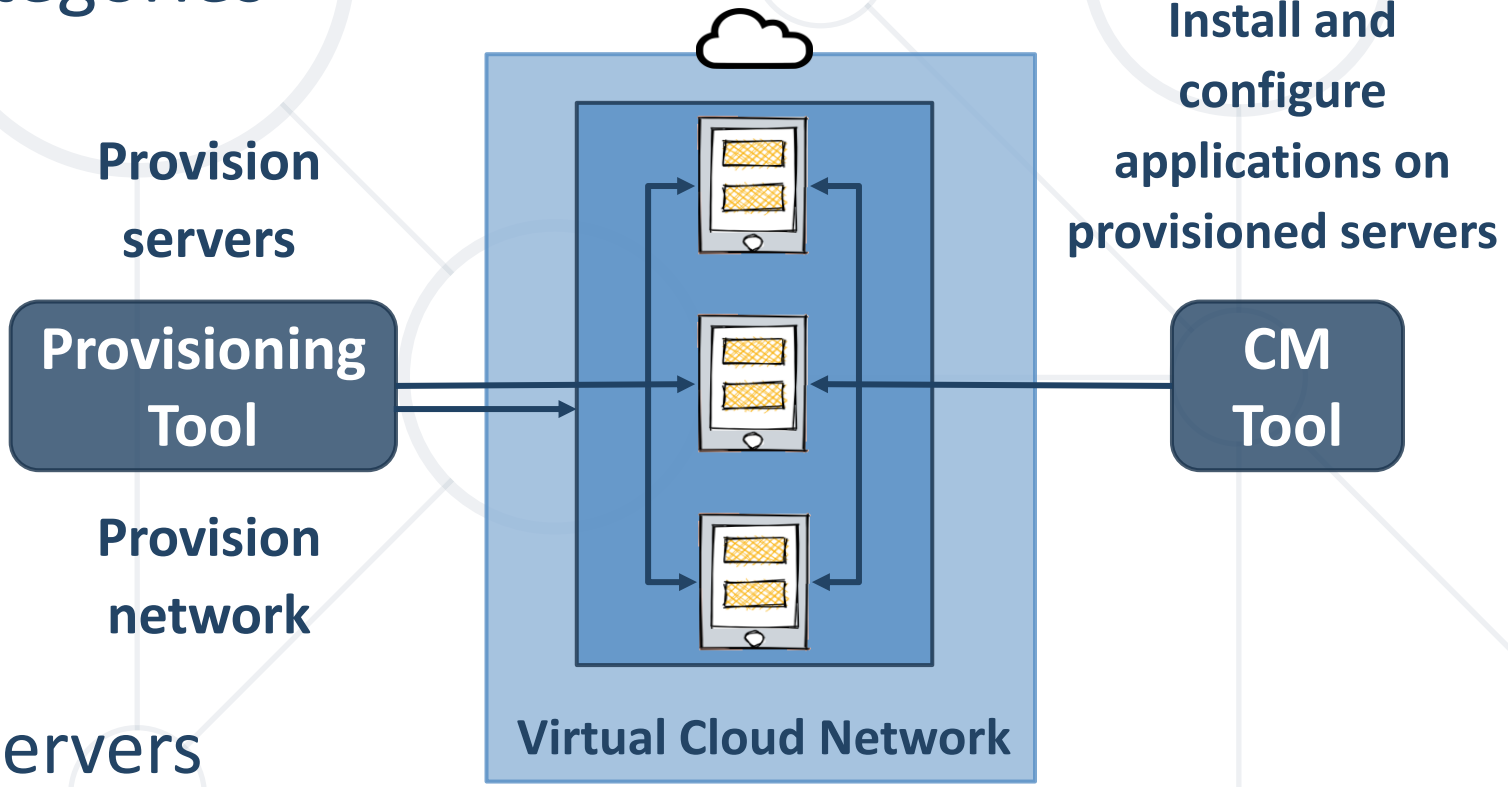


- Declarative Approach



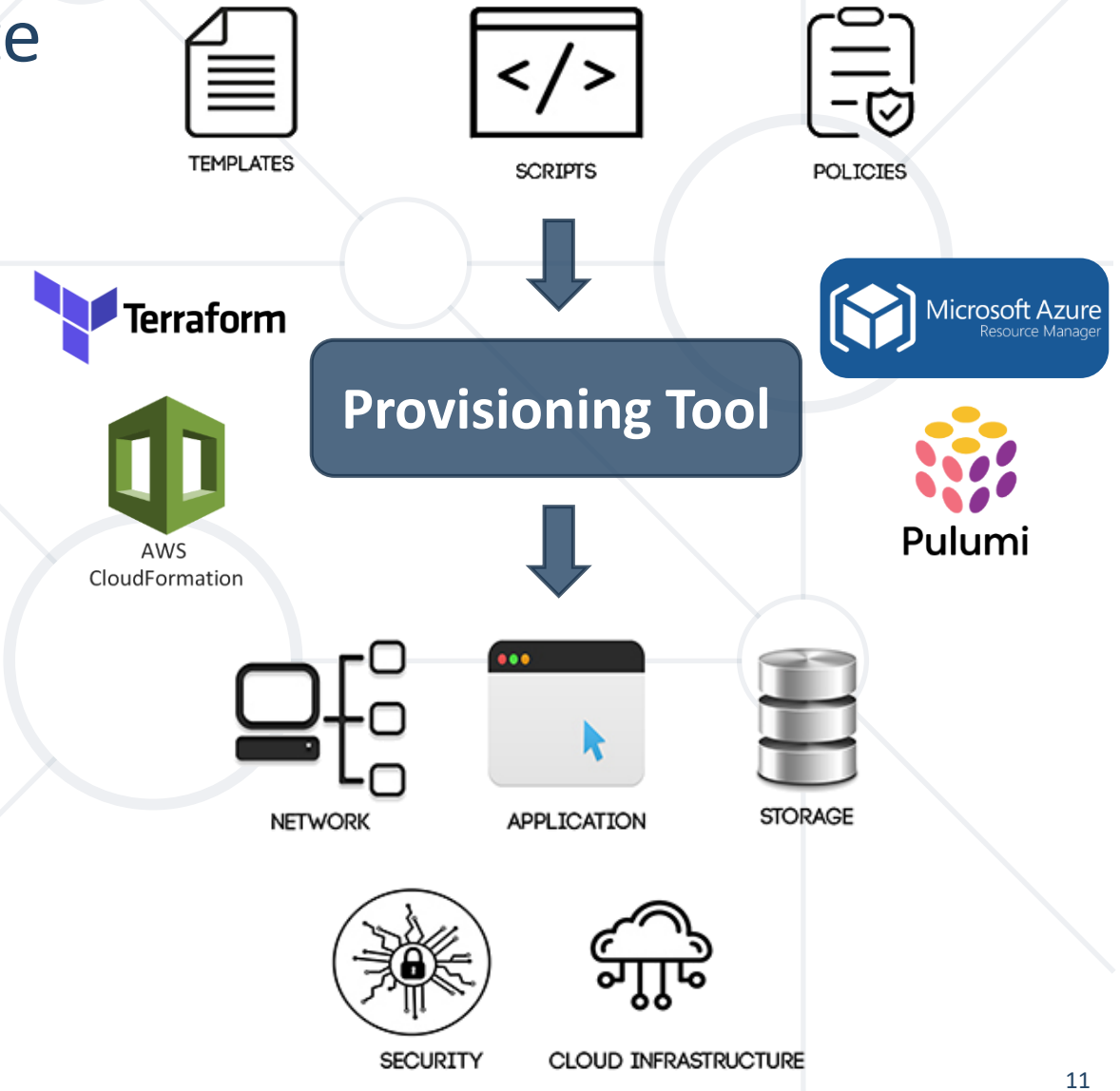
IaC Tools

- The primary goal of **IaC tools** is to **bring the infrastructure component** to the **desired state** declared by the user
- **IaC tools** fall into two categories
  - **Infrastructure provisioning tools** – create infrastructure components
  - **Configurations management tools** – configure provisioned servers



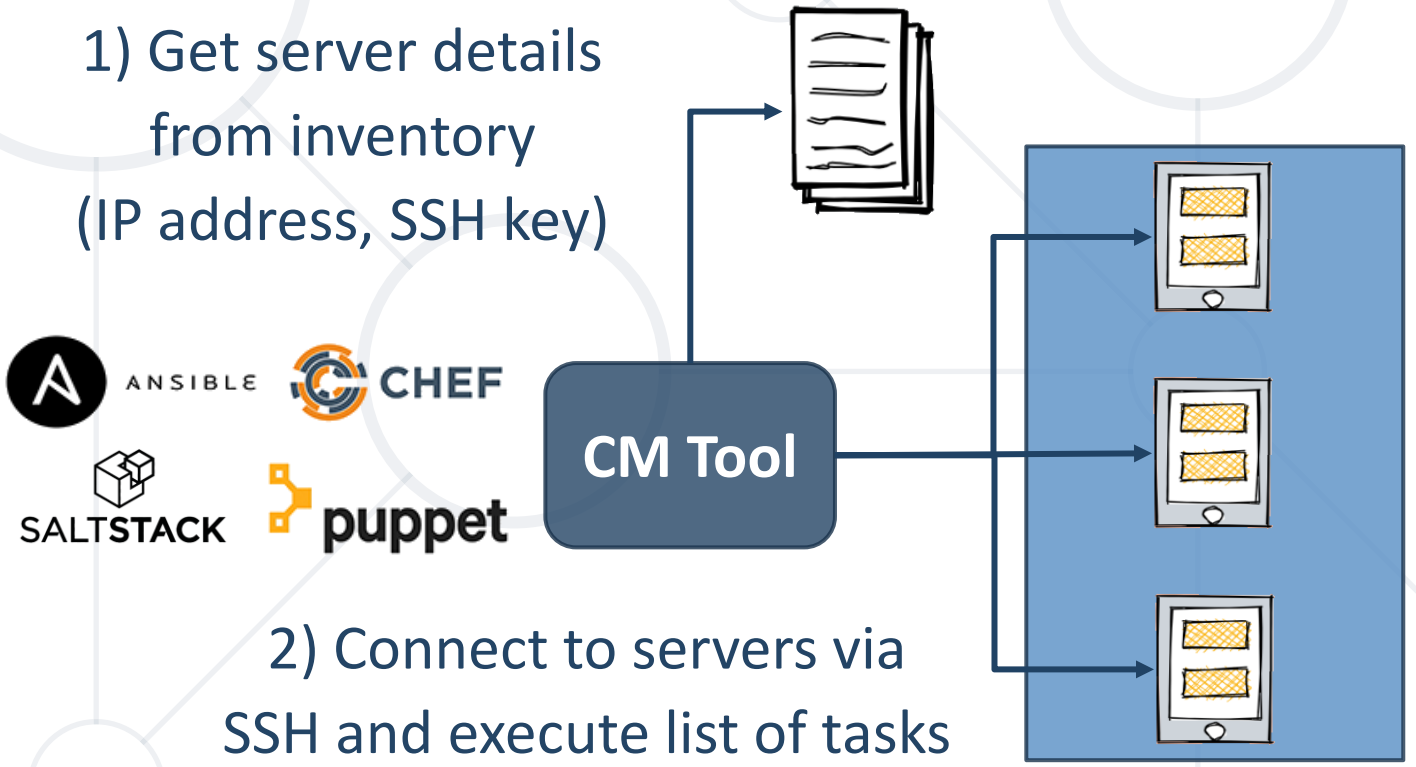
Infrastructure Provisioning Tools

- **Infrastructure provisioning** == create infrastructure resources like virtual servers, storage, networking, cloud managed services, etc.
- Primary goal → **keep the infrastructure in its desired state** and reproduce or update it
- Tools: Terraform, AWS Cloudformation, Azure Resource Manager (ARM) Templates, Pulumi
- They can also **trigger CM tools**



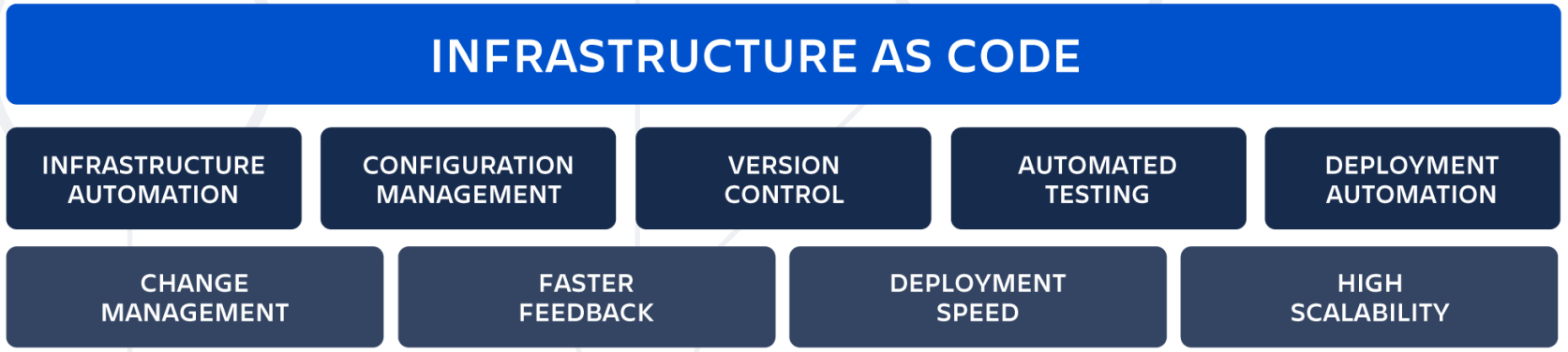
Configuration Management Tools

- **Configuration management** == configuring infrastructure resources
  - E.g., configuring a server with required applications or configuring a firewall device
- Primary goal → **configure the server**
- Tools: Ansible, Chef, Puppet, SaltStack, etc.
- In **cloud environments**, tools use an **API-based dynamic inventory** to get the server details



IaC Benefits for DevOps

- **IaC** is an important part of implementing **DevOps practices**
  - **Version control**, **test** and **deploy** of infrastructure code changes
  - **Improved collaboration** – Ops team can participate in writing IaC templates together with Dev team, as IaC uses simple, text-based files
  - **Automation** of creation and management of infrastructure resources
  - **Consistency and reliability across environments** is achieved as IaC generates the same environment every time



Terraform

IaC Tool for Infrastructure Provisioning Automation

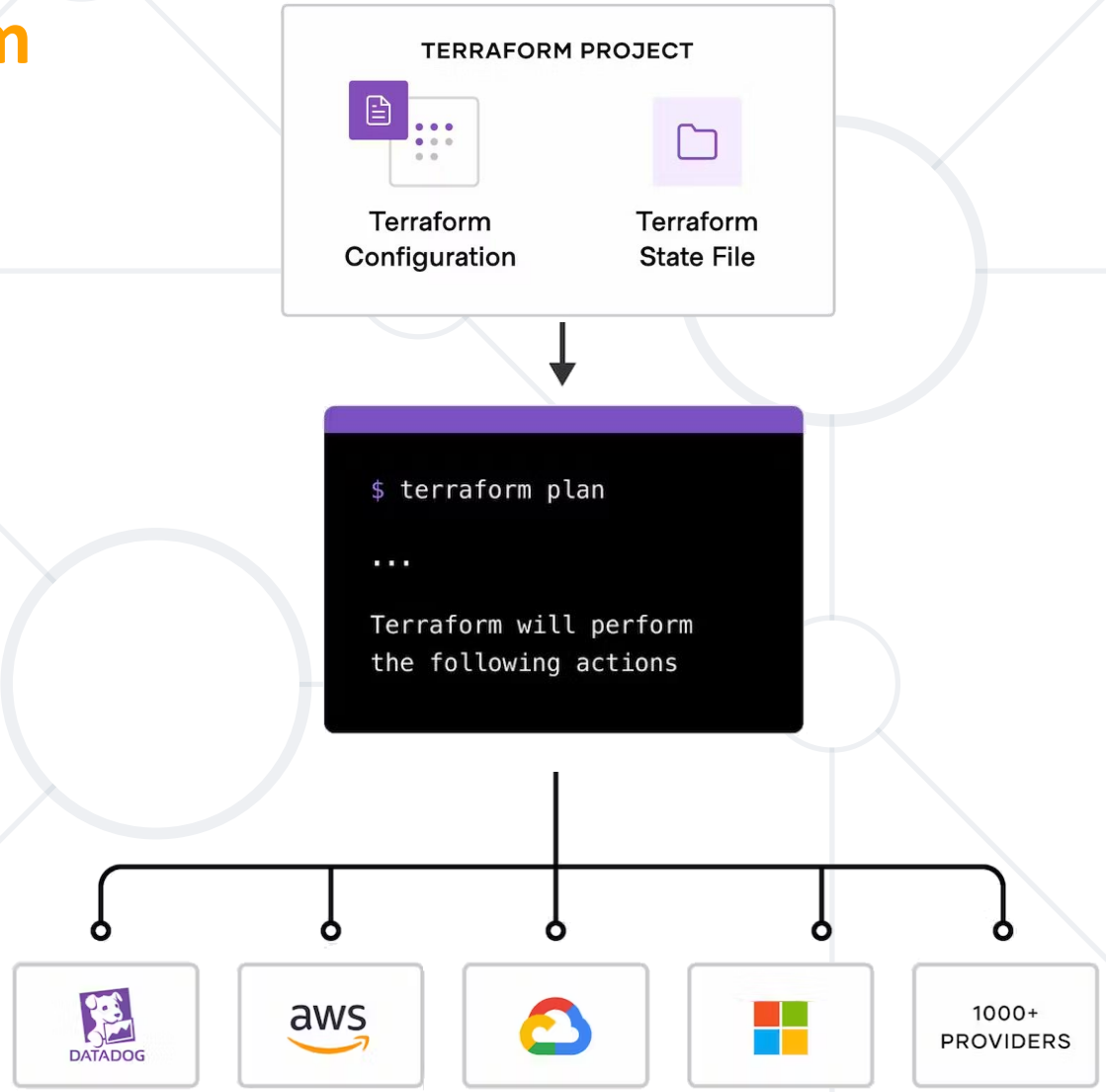
Terraform Overview

Software University

- Open-source **IaC** tool
  - Used for **Provisioning**, **managing** and **deploying** infrastructure resource
  - Written in Golang
- Allows managing infrastructure for applications across **multiple cloud providers** – AWS, Azure, GCP, etc.
  - Through their application programming interfaces (**APIs**)
- Uses **declarative syntax** – you define desired infrastructure state, Terraform figures out the best way to achieve it

Terraform Workflow

- To **deploy infrastructure with Terraform**
  - Scope – **identify the infrastructure** for your project
  - Author – define **infrastructure** in **configuration files**
  - Initialize – **install the plugins** Terraform needs to manage the infrastructure
  - Plan – **preview the changes** Terraform will make to match your configuration
  - Apply – Terraform **provisions the infrastructure** and **updates state file**



Terraform Configuration File

Software University

- To **create an infrastructure**, a **Terraform Configuration file** (**.tf**) should be executed
- Executed with the help of **Terraform CLI** or other executors
- Written in HashiCorp Configuration Language (**HCL**) or **JSON syntax**

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

resource "aws_s3_bucket" "bucket_name" {
  bucket = "bucket-name"
}
```

Terraform Configuration File (HCL)

Terraform CLI

Infrastructure Automation/Provisioning with Terraform

Terraform State File

- Terraform stores **state** about managed infrastructure and configuration
- **State** allows us to have a point-in-time **view of our infrastructure** and **compare**
  - **Desired state** (our code)
  - **Perceived state** (the state file)
  - **Reality** (the resources within the platform / service)
- This state is stored in a **local file "terraform.tfstate"**
- State file is recommended to be **kept in cloud**
- State file format is **JSON**, but should not be edited directly

Ansible

What is Ansible?

- Open source infrastructure **automation tool**
  - Written in Python
- Focuses on **security** and **reliability**
  - Uses OpenSSH
- Easy to read and write
  - Uses **YAML**
  - Structured
- **Agentless**
  - No agents, repositories, etc.

Key Features

- Powerful tool for **managing Infrastructure as Code**
- **Declarative**
- **Idempotent**
  - Run an operation multiple times, without changing the initial state of the application
- Three major **use cases**
  - Inventory (Provision)
  - Configuration management
  - Application deployment

Inventory

- **Inventory** (**Provision**) == description of the nodes that can be accessed by Ansible
- Described by a configuration file
  - Default location is `./etc/ansible/hosts`
  - List of the **IP address**/**hostname** of each node
- Each host is assigned to a **group**
  - Web server
  - Database server
  - Etc.

Configuration Management

- **Configuration management** == definition and enforcement of the desired state of the infrastructure
  - Running a service, installing service on a VM, etc.
- **Playbook** == a set of tasks, stored in YAML
  - Ansible **Playbook** contains **plays**
  - Plays map **hosts** to tasks
  - Each **play** can contain **multiple tasks**
  - Tasks call **modules**

Modules

- **Modules** == Ansible features that perform the **actual work**
- Executed in two ways
  - Manually, using the **`ansible`** command
  - In batches with **ansible-playbook**
- Also known as task plugins or library plugins
- Organized in **categories**

App Deployment

- **Deploy** applications into the VMs in the infrastructure
- Using **playbooks**, you can manage the entire lifecycle of an application
- **Multi-environment** support
  - Define different playbooks for different environments
    - Easily manage the deployment process for each environment

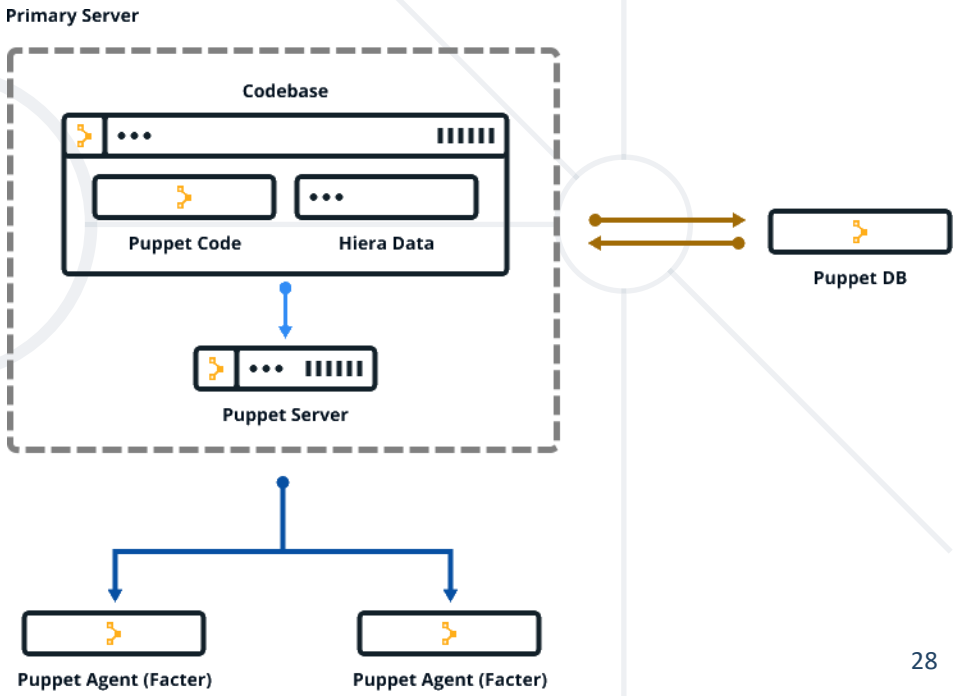
Puppet

What is Puppet?

- **Configuration management** tool for **servers**
  - Ensures all systems are **configured** to the **desired states**
- Also used as a **deployment** tool
- Uses **server-agent model**
- Configurations are written in Puppet code
  - Ruby DSL
- Open Source and Enterprise

Puppet Platform

- **Puppet Server**
  - Controls configuration for one or more managed nodes
  - Communicate via HTTPS with the agents
  - Has a built-in certificate authority
  - Runs an agent to configure itself
- **Puppet Agent**
  - **Facter** → gather information about a node
  - **Hiera** → separate the data from the code
- **Puppet DB**
  - Stores facts, catalog, reports, etc.



Components (1)

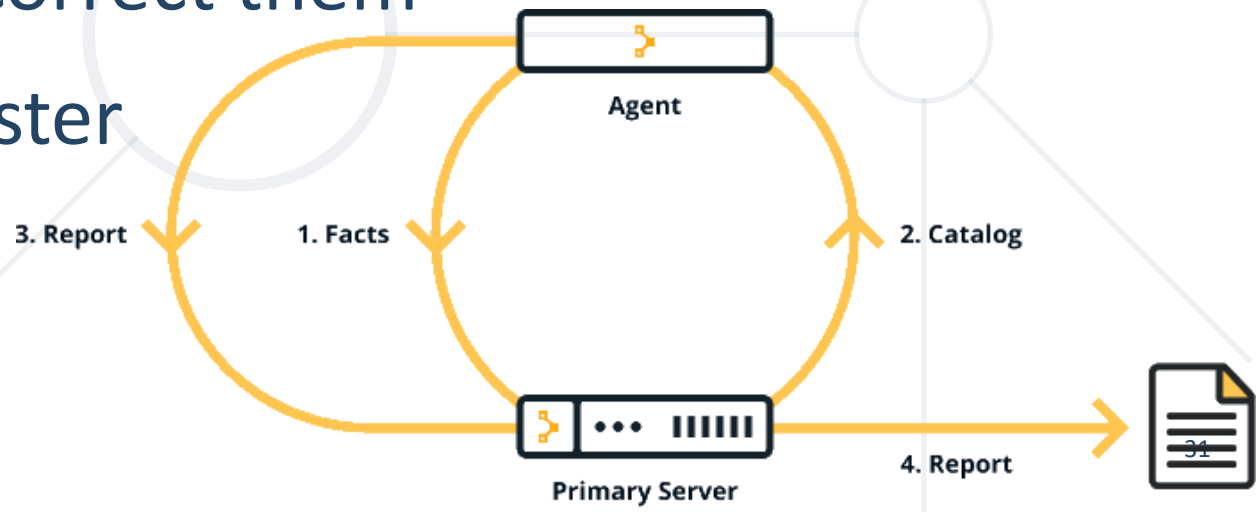
- **Resources**
  - Fundamental unit for modeling system configurations
  - Describe aspects of a system
- **Manifest** == files, containing a set of instructions
- **Classes** == code blocks, which can be called from a code elsewhere

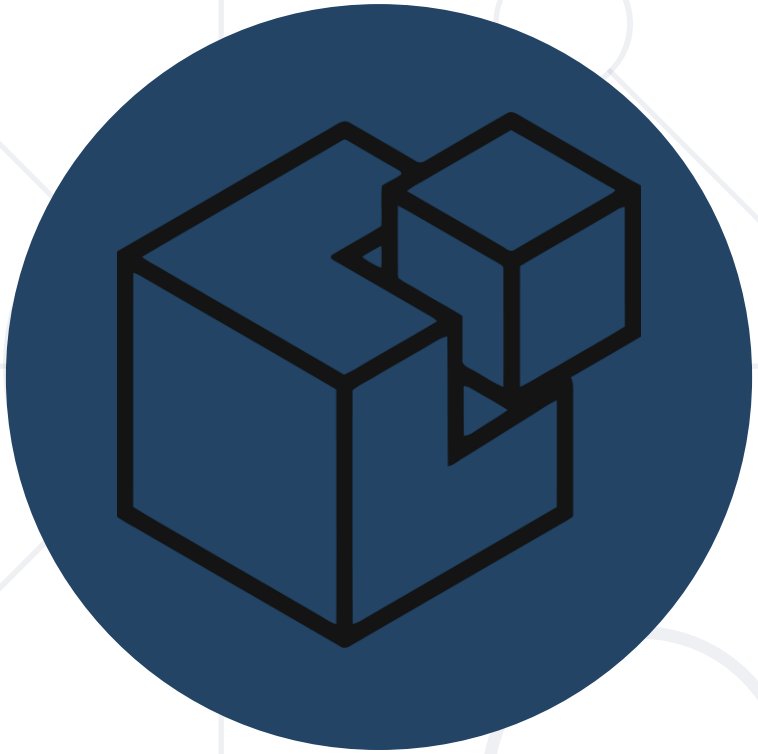
Components (2)

- **Facts**
  - Per-node data, available in Puppet manifest files as variables
  - Built-in, custom and external
  - Legacy and modern facts
- **Modules**
  - Each module manages a specific task
  - Basic building blocks
  - Install from Puppet Forge or create your own

Workflow

- An **agent node** sends facts to the master and requests a catalog
- The **master compiles** and **returns** the node's catalog
- The **agent applies** the catalog to the node by checking each resource the catalog describes
- If the agent finds resources that are not in their desired state, it makes the changes necessary to correct them
- The agent reports back to the master





SaltStack

What is SaltStack?

- **Management** tool
  - Used for **configuration management**, data-driven orchestration and remote execution
- Two **operation modes**
  - With agents (minions)
  - Agent-less
- Management instructions in **YAML**

Salt Master

- **Salt master** == the machine that **controls** the infrastructure and dictates policies for the servers it manages
  - Operates as
    - A **repository** for **configuration data**
    - A **control center**
      - Initiates **remote commands**
      - Ensures the **state** of other machines

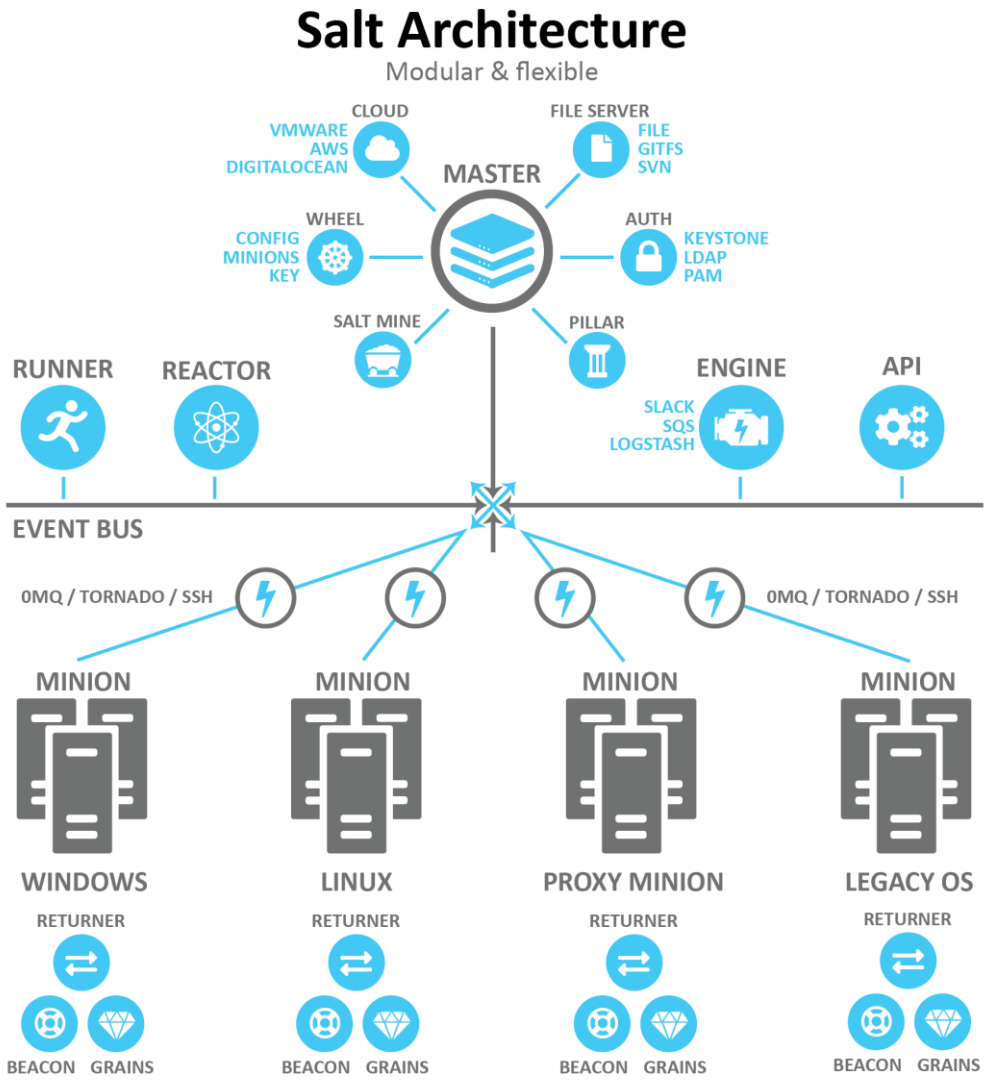
Minions

- **Minions** == **Servers** that Salt manages
  - Responsible for
    - **Executing the instructions** sent by the master
    - **Reporting** on the success of jobs
    - **Providing data** about the underlying host

Artifacts

- **Execution modules** == sets of related **functions** that perform work on minions
- The configuration management portion of Salt is primarily implemented using the **state** system, which uses **state modules**
- Salt **formulas** == **sets of state module calls**, arranged with the aim of producing a certain result
- Configuration management is handled by **SLS files**, written in **YAML**, that describe how a system should look once the formula has been applied
- Salt **grains** are pieces of information, gathered by and maintained by a minion, primarily concerning its underlying host system

General Salt Architecture



Chef

What is Chef?

- **Configuration Management** tool
  - Written in Ruby and Erlang
  - Uses pure-Ruby DSL
- Works with system configuration "**recipes**"
- Used for **configuring** and **maintaining** servers
- Can be integrated with **cloud-based platforms** to automatically provision and configure new machines
- **Chef Infra** → **configure** and **manage** infrastructure

Chef Infra

Software University

- Policy-based **configuration management** tool
  - Define and enforce desired state of systems
- Uses the **master-agent** model
- "**Recipes**" are contained in "**cookbooks**"
  - Manage configuration, software installations and system updates

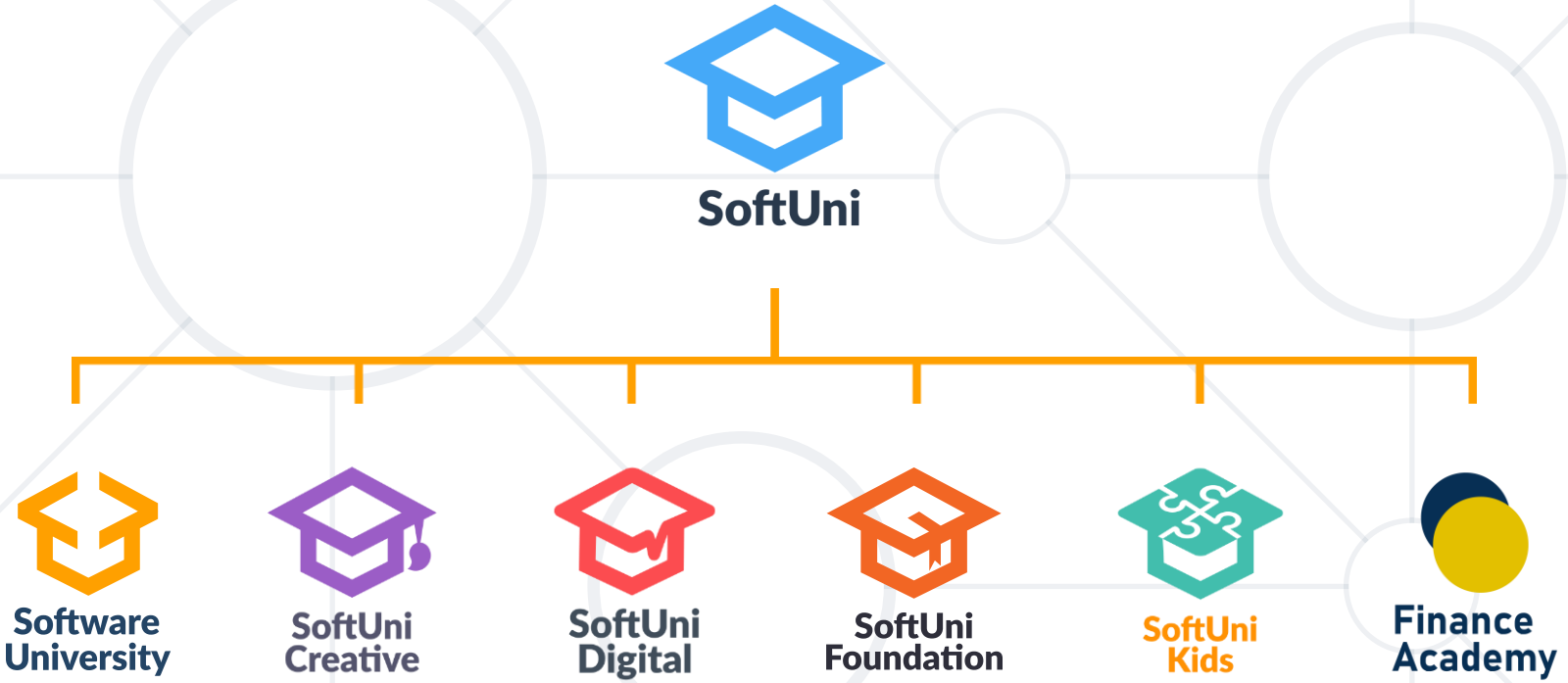
Components and Workflow

- **Chef workstation** == the point where users can author and test cookbooks and interact with the Chef server
- **Chef client nodes** == the machines that are managed by Chef
- **Chef client** → installed on each node and is used to configure the node to its desired state
- **Chef server** == a hub for cookbooks, policies, and metadata
- Nodes use the Chef client to ask the Chef server for configuration details, such as recipes, templates, and file distributions

Summary

- **Infrastructure as Code** (**IaC**) uses DevOps practices and versioning with a descriptive model to **define** and **deploy infrastructure**
- **Terraform** is an IaC **provisioning** tool used to create infrastructure
- **Ansible**, **Puppet**, **Salt** and **Chef** are **configuration management** tools used to configure provisioned servers

Questions?



SoftUni Diamond Partners

Software University

SUPER HOSTING .BG

Coca-Cola HBC Bulgaria

POKERSTARS
POKER | CASINO | SPORTS
a Flutter International brand

INDEAVR
Serving the high achievers

AMBITIONED

DRAFT KINGS

SOFTWARE GROUP

createX

Postbank
Решения за твоето утре

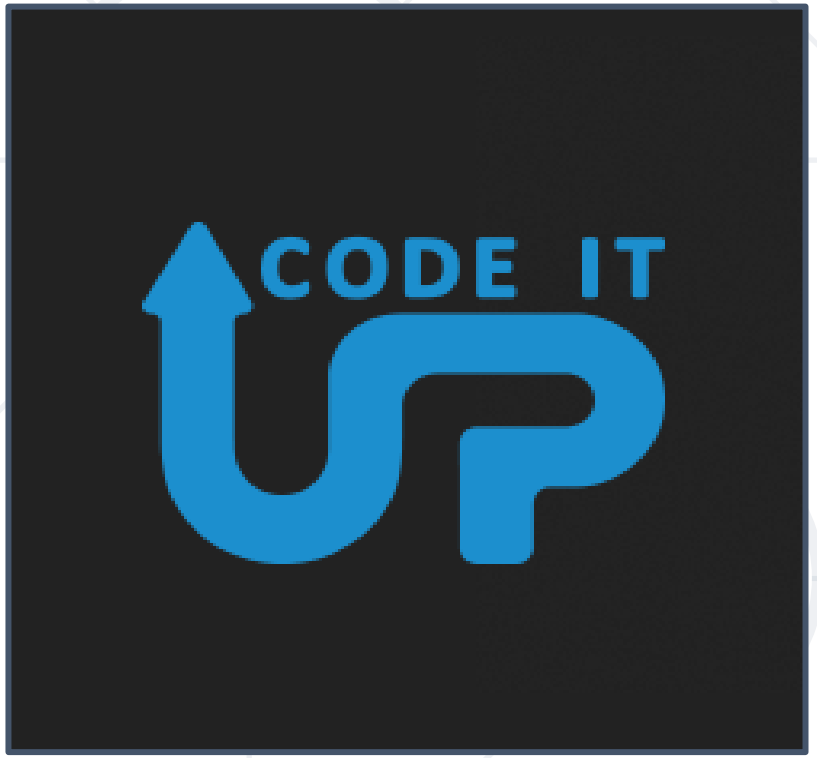
BOSCH

DXC TECHNOLOGY

SmartIT

Educational Partners

Software University



License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg/
- © Software University – https://softuni.bg

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg