Applied Virtual Networks

COMP 4912

Instructor: **Dawood Sajjadi**

PhD, SMIEEE, CISSP

ssajjaditorshizi@bcit.ca

Winter-Spring 2025

Week #7

BCIT™

Emergence of Public Clouds

RECAP

Spacelift report (Mar 2024)

Gartner report (Nov 2023)

BCIT™

Key advantages of Public Clouds

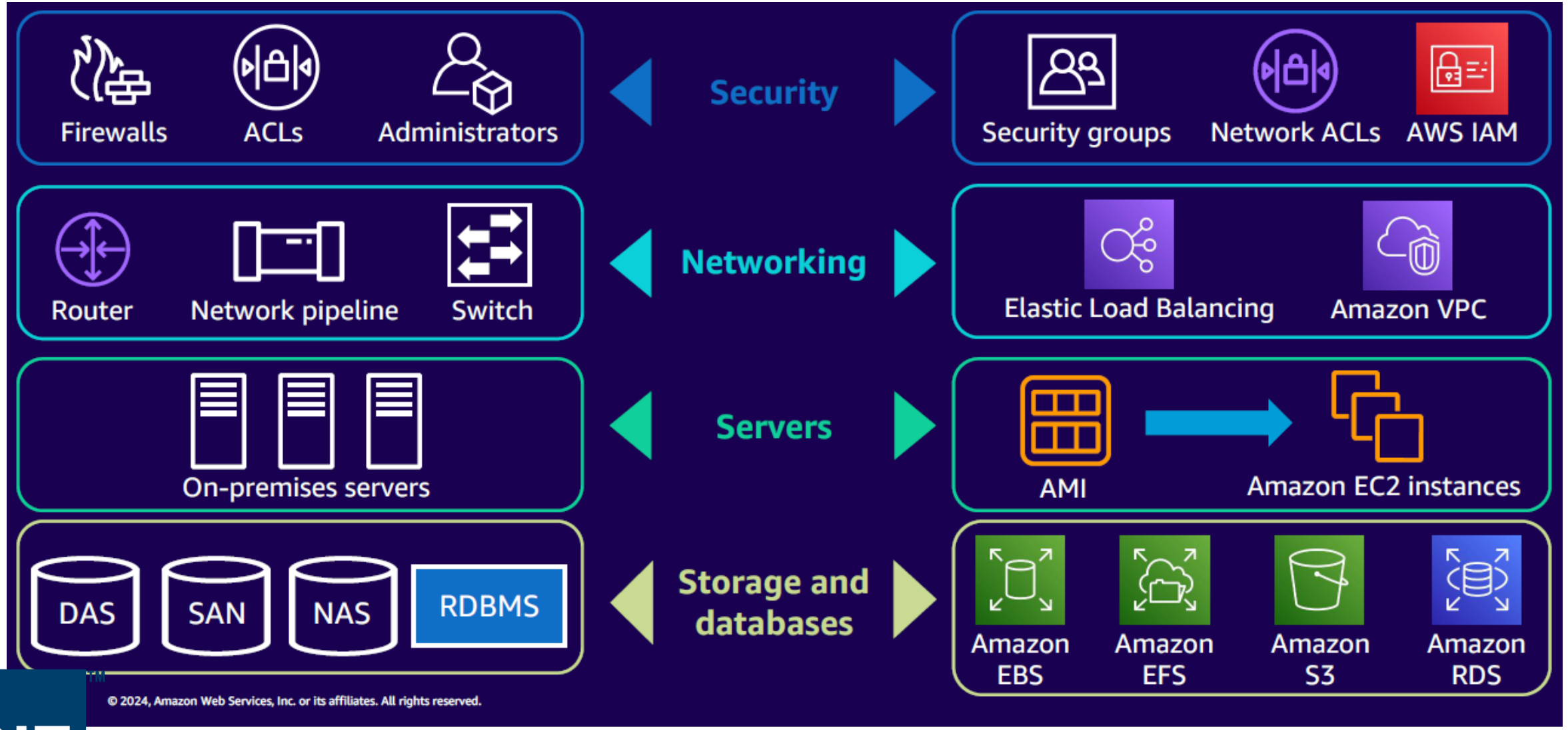
RECAP

BCIT™

Reference: AWSome Day, Michalynn J. Koziol (2024)

AWS Core Infrastructure-as-a-Service (IaaS)

RECAP



BCIT

Reference: AWSome Day, Michalynn J. Koziol (2024)

AWS storage options

RECAP

Amazon S3
Scalable, highly durable object storage in the cloud

AWS Storage Gateway
Hybrid cloud storage service that gives you on-premises access to virtually unlimited cloud storage.

Amazon S3 Glacier
Low-cost, highly durable archive storage in the cloud

Amazon EBS
Network-attached volumes that provide durable block-level storage for Amazon EC2 instances

Amazon EFS
Scalable network file storage for Amazon EC2 instances

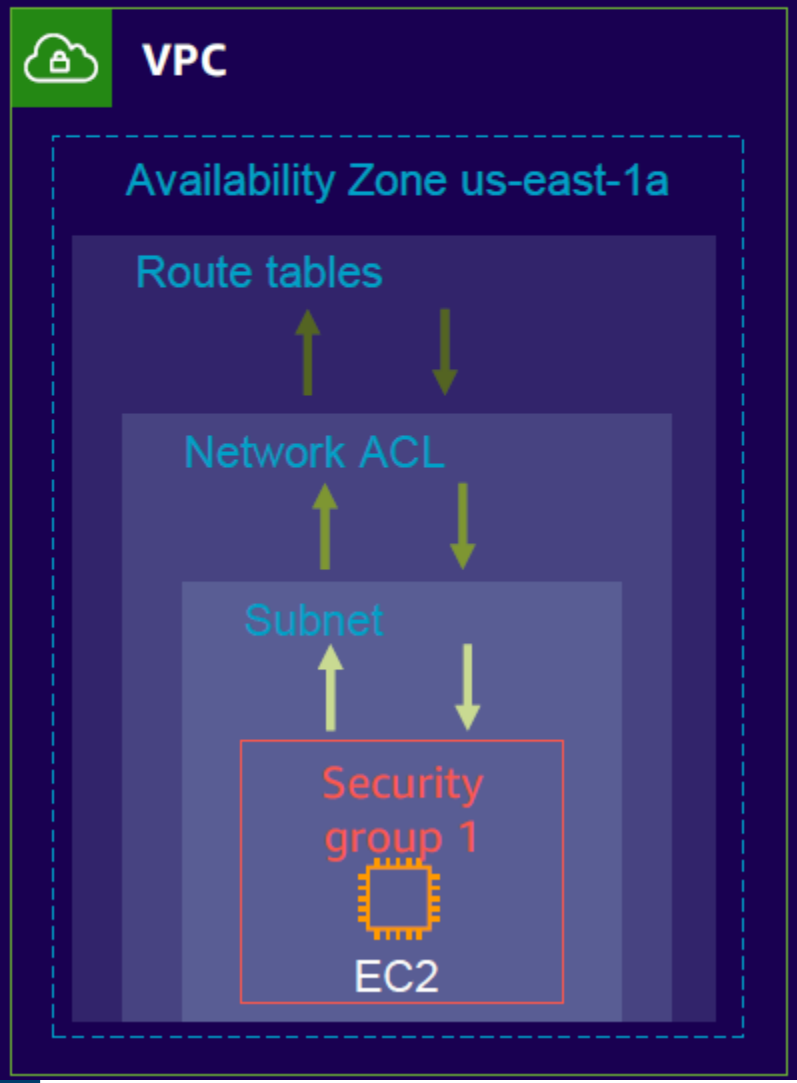
Amazon FSx
Fully managed, cost-effective file storage offering the capabilities and performance of popular commercial and open-source file systems

BCIT

Reference: AWSome Day, Michalynn J. Koziol (2024)

Securing your infrastructure

RECAP

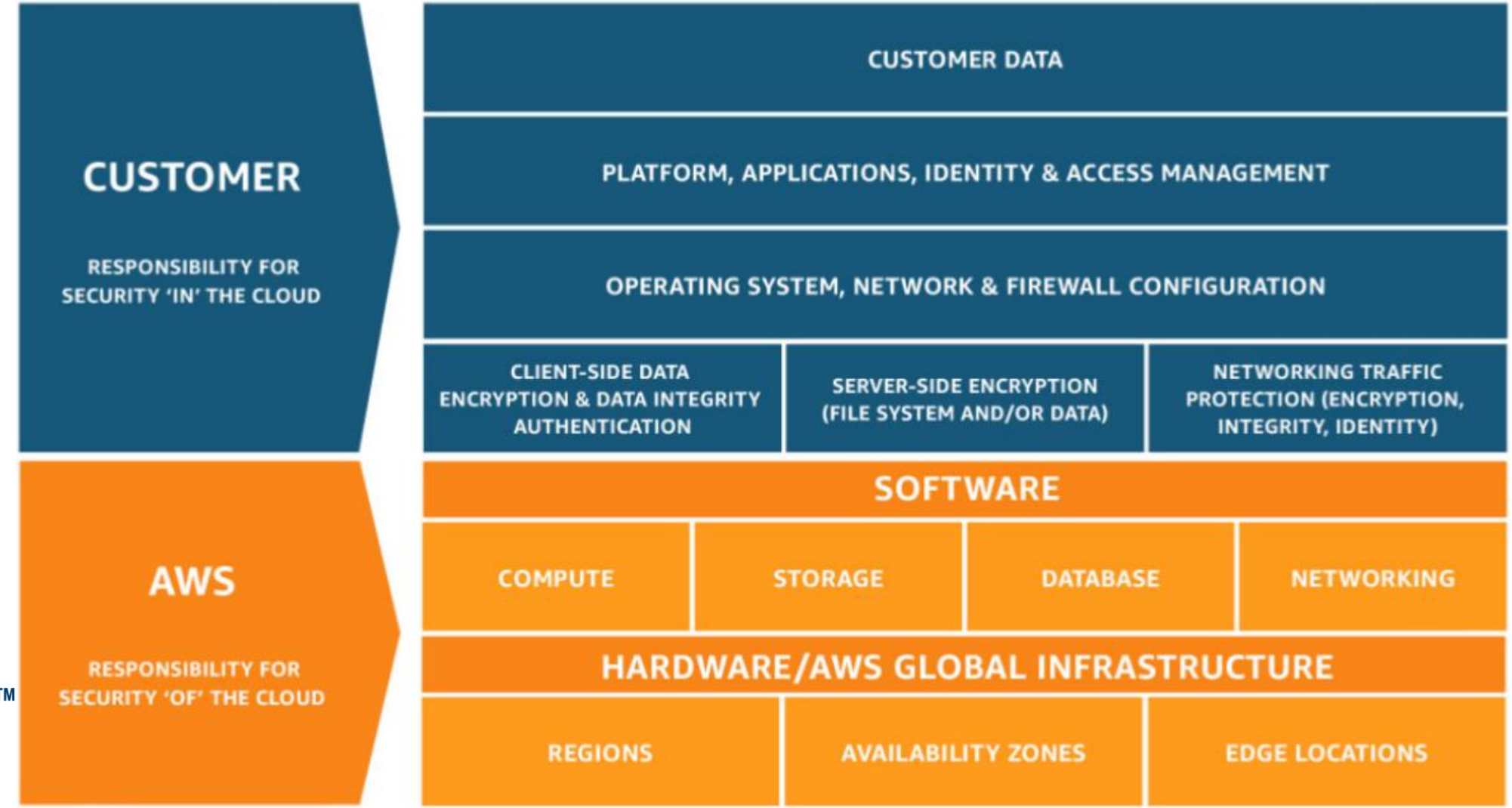


- **Route tables**
  - Contains a set of rules, called routes, that are used to determine where network traffic is directed
  - Routes tables can have association with VPC, gateways, and subnets
- **Network access control lists (network ACLs)**
  - Allow or deny traffic in and out of subnets
  - Hardens security as a secondary level of defense at the subnet level
- **Security groups**
  - Used to allow traffic to and from at the network interface (instance) level
  - Usually administered by application developers

Reference: AWSome Day, Michalynn J. Koziol (2024)

UNDERSTANDING AWS SHARED RESPONSIBILITY MODEL

RECAP



https://allcloud.io/blog/aws-security-panel-summary/

BCIT™

Learning Outcomes of Week #7

1. Understand the core concepts of Automation and Infrastructure-as-Code (IaC).
2. Learn how to use Ansible to Automate configuration management and software deployment.
3. Explore Terraform for provisioning and managing infrastructure across cloud providers.
4. Gain hands-on experience writing simple Ansible playbooks and Terraform configuration files.
5. Compare and use Ansible and Terraform for end-to-end automation workflows.
6. Develop skills to implement scalable, repeatable, and efficient automation solutions.

BCIT™

The **Ansible** project is an open source community sponsored by Red Hat. It uses YAML to perfectly describe IT application environments in Ansible **Playbooks**. **Ansible Engine** is a supported product built from the **Ansible** community project.

What Ansible Does?

- ✓ Overview of Configuration Management and Automation
- ✓ Implementing an agentless architecture to manage resources

Key Features

- ✓ Simple, YAML-based Playbooks
- ✓ Agentless Approach
- ✓ Scalability and Performance
- ✓ Extensibility with Modules

BCIT™

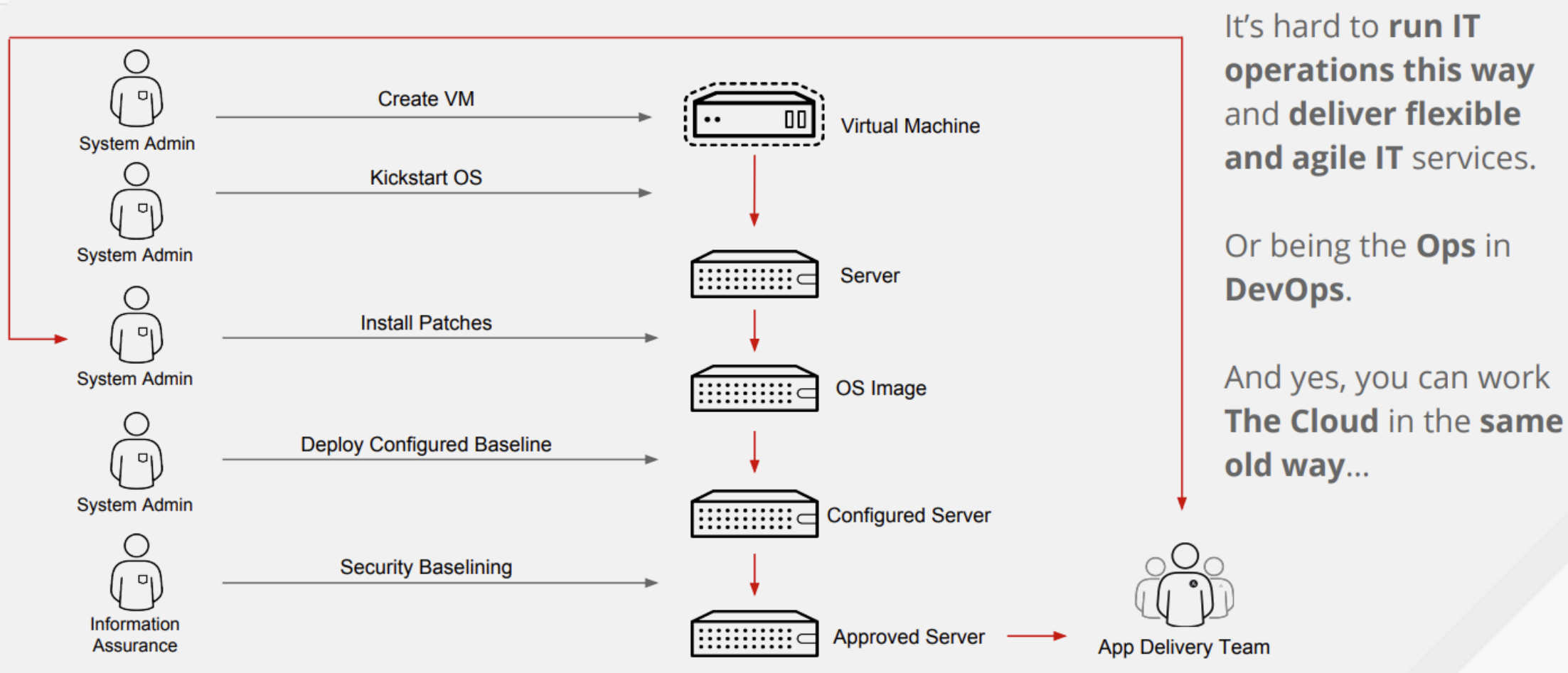
Script vs Ansible Playbook

```
#!/bin/bash
# Script to add a user to Linux system
if [ $(id -u) -eq 0 ]; then
   $username=johndoe
   read -s -p "Enter password : " password
   egrep "^$username" /etc/passwd >/dev/null
   if [ $? -eq 0 ]; then
      echo "$username exists!"
      exit 1
   else
      useradd -m -p $password $username
      [ $? -eq 0 ] && echo "User has been added
to system!" || echo "Failed to add a user!"
   fi
fi
```

```
- hosts: all_my_web_servers_in_DR
  tasks:
    - user:
        name: johndoe
```

BCIT™

Setting up a Server in the Old Days



BCIT™

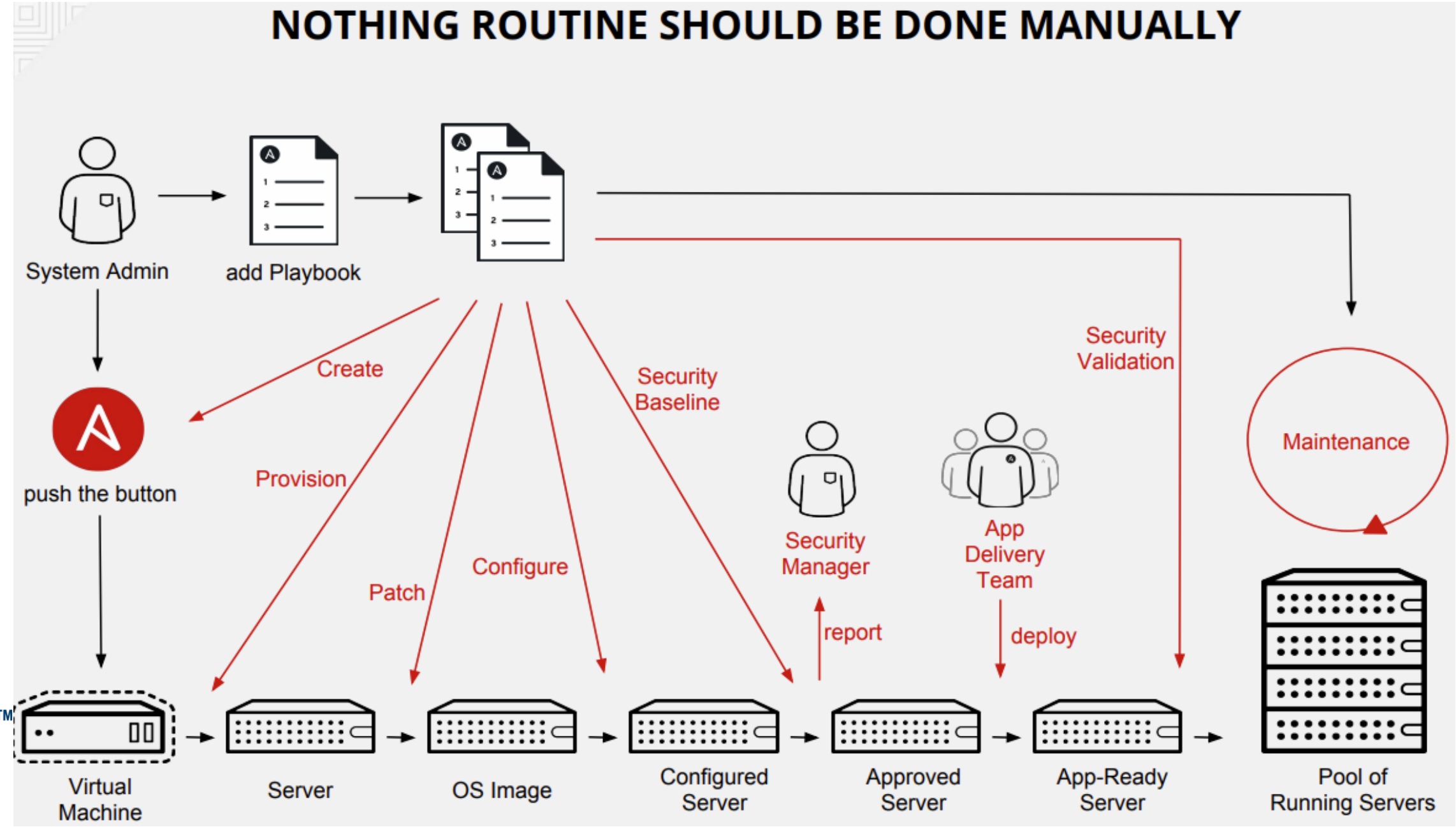
https://people.redhat.com/mkoch/training/1806-vienna/presentations/05%20-%20Ansible%20and%20Ansible%20Tower%20Introduction.pdf

Eliminating Toil

https://sre.google/sre-book/eliminating-toil/

BCIT™

NOTHING ROUTINE SHOULD BE DONE MANUALLY



BCIT™

The Importance of Automation in SRE and DevOps

- Consistency and Efficiency in Operations
- Accelerating Deployment Cycles
- Reducing Manual Error

Core Components of Ansible

- **Playbooks**
- **Inventory**
- **Modules**
- **Roles**

BCIT™

How Ansible Works

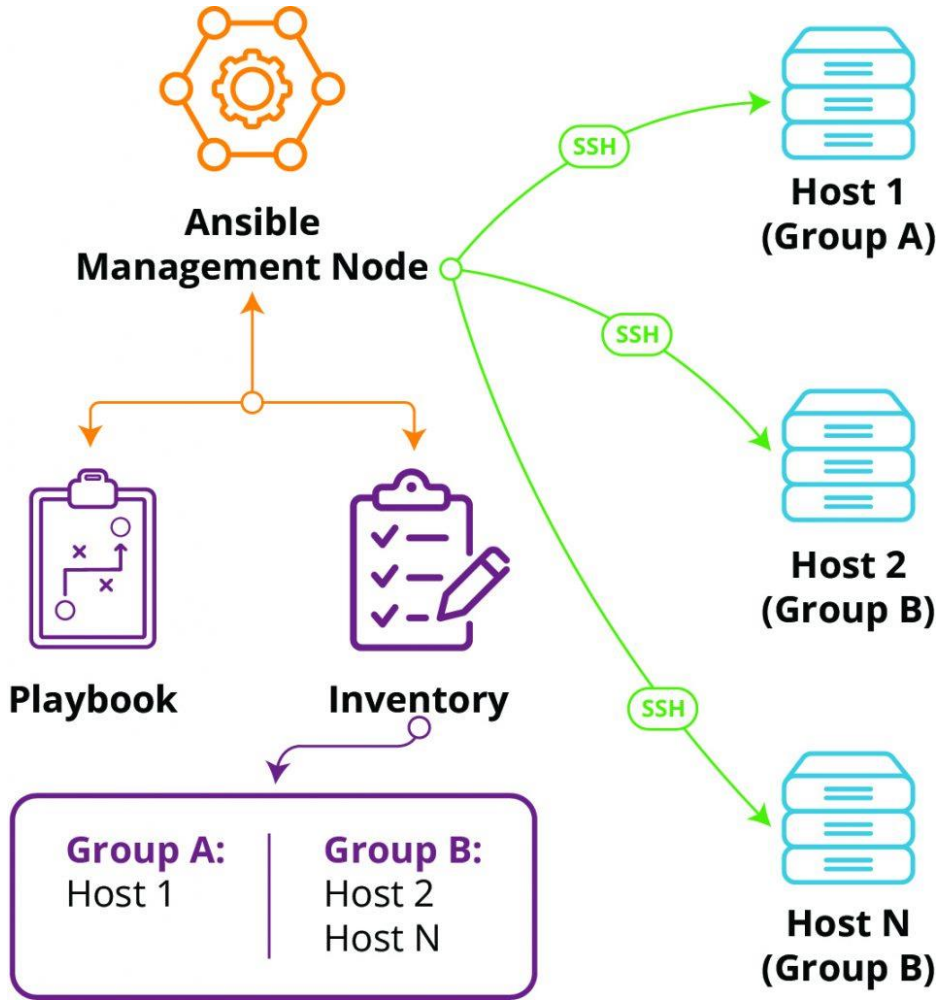
Execution Flow of an Ansible Playbook

→ Communication via SSH
→ Mostly written in Python
→ It is Agentless (no service running)

Common Use Cases

→ Configuration Management
→ Application Deployment
→ Orchestration of Complex Workflows

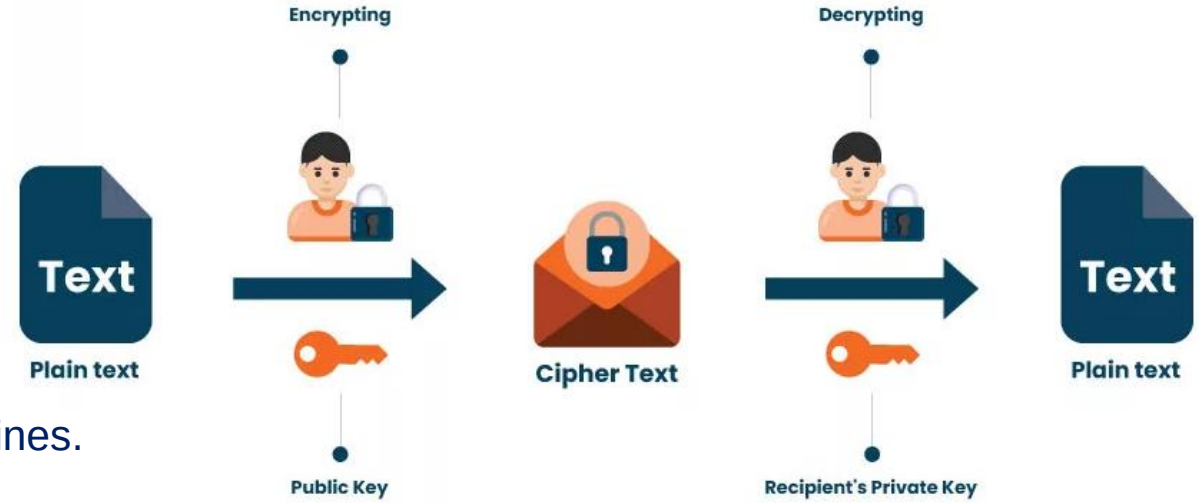
Architecture Overview



BCIT™

What is SSH (Secure Shell)?

Secure Shell (SSH) is a cryptographic network protocol used to securely access and manage remote systems over an unsecured network.



- ✓ Encrypts data to prevent eavesdropping.
- ✓ Enables secure login and command execution on remote machines.
- ✓ Supports SCP & SFTP for secure file transfer.

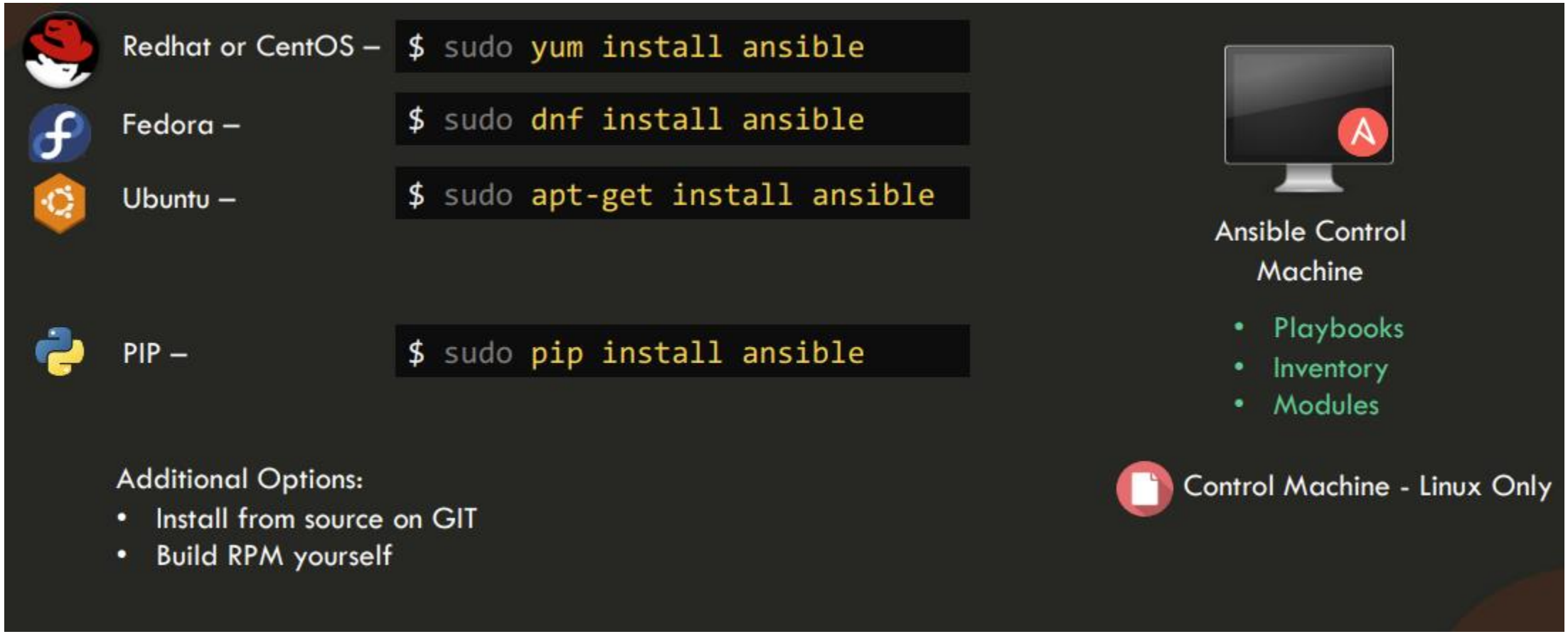
Default Port is 22 TCP

$ ssh username@<remote-host>

$ ssh ubuntu@10.11.12.1

BCIT™

How Ansible Works



BCIT™

Setting Up Ansible

- ✓ Installation Requirements
- ✓ Setting Up the Control Node
- ✓ Configuring Inventory

Ansible Inventory

- ✓ Static vs. Dynamic Inventory
- ✓ Defining Hosts and Groups

Writing the first playbook

- ✓ Structure of a Playbook
- ✓ Example: Installing a Package

playbook.yml

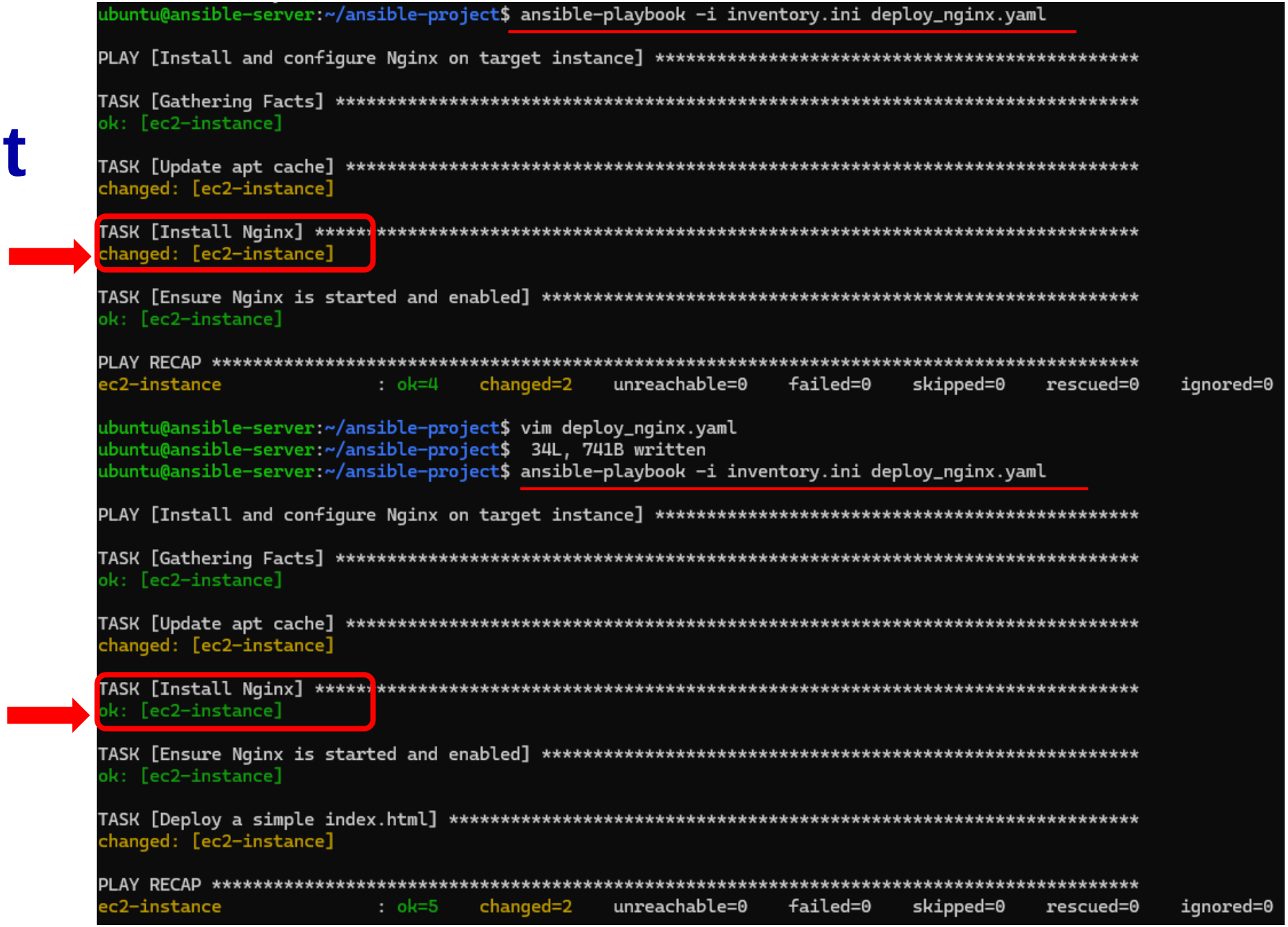
```
-
  name: Play 1
  hosts: localhost
  tasks:
   - name: Execute command 'date'
     command: date

   - name: Execute script on server
     script: test_script.sh

   - name: Install httpd service
     yum:
      name: httpd
      state: present

   - name: Start web server
     service:
      name: httpd
      state: started
```

https://kodekloud.com/wp-content/uploads/2020/10/Ansible-for-Beginners-KodeKloud.pdf

BCIT™

Ansible is Idempotent

Repeated runs produce same result without changes.



BCIT™

Ansible Galaxy

https://galaxy.ansible.com/

Ansible Galaxy is an online repository where users can find, share, and reuse Ansible roles, modules, and collections.

$ ansible-galaxy collection install **cisco.ios**

Collections in this yaml file indicates that the playbook is using the Cisco IOS collection (downloaded from Ansible Galaxy). This collection provides the **ios_config** module and other Cisco-specific modules that enable configuration of Cisco devices.

Community-Driven: Access to thousands of community-contributed roles.
Reusability: Roles and collections help modularize & reuse automation code.

```yaml
- name: Configure Cisco IOS Device
  hosts: cisco
  connection: network_cli
  gather_facts: no
  collections:
    - cisco.ios
  tasks:
    - name: Set device hostname
      ios_config:
        lines:
          - hostname MyNewCiscoDevice

    - name: Configure interface GigabitEthernet1
      ios_config:
        parents: interface GigabitEthernet1
        lines:
          - description Configured by Ansible
```

BCIT™

Check These Resources for Ansible

1. https://github.com/ansible-community/awesome-ansible
2. https://ansible.puzzle.ch/ (slides/labs)
3. https://www.env0.com/blog/the-ultimate-ansible-tutorial-a-step-by-step-guide

Ansible Roles

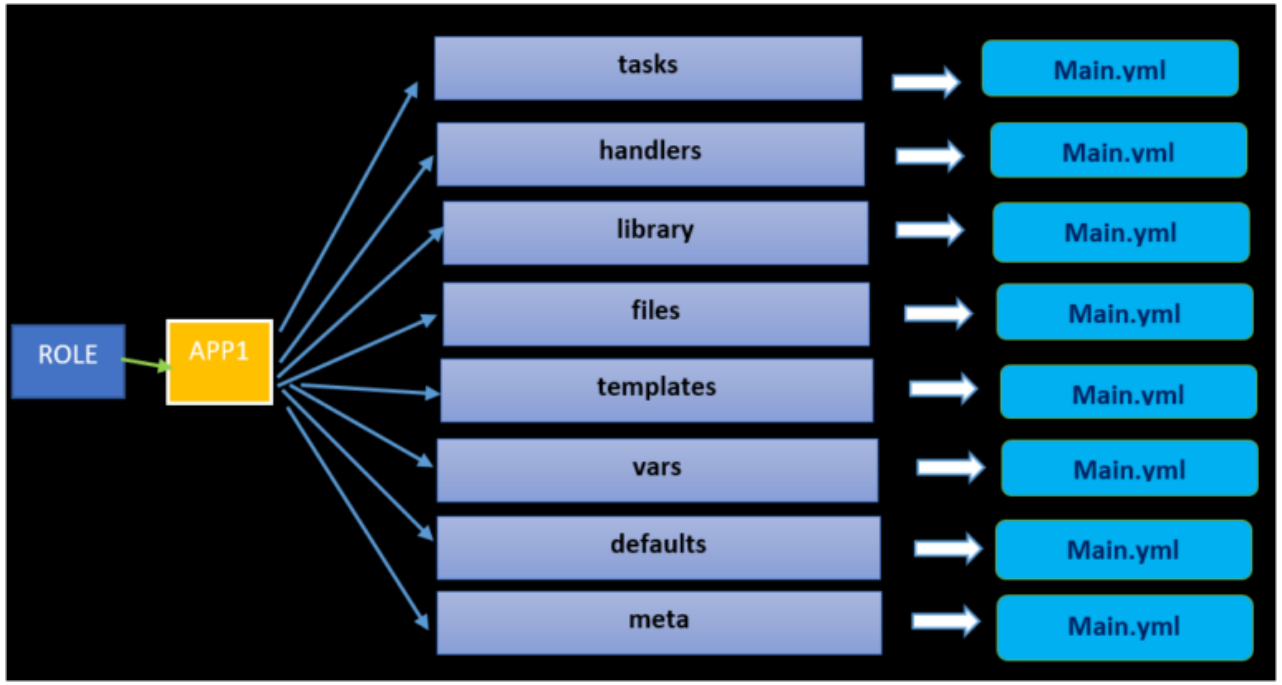
You can try the following on **Ansible** Control node to setup an **Nginx Server** using an **Ansible Galaxy** role.

$ ansible-galaxy install geerlingguy.nginx

$ vim website.ymal

```
- name: Setup Nginx on Web Servers
  hosts: web
  become: yes
  roles:
    - geerlingguy.nginx
```

$ ansible-playbook -i inventory.ini website.yaml



BCIT™

Visit the following link for further details about this role

https://github.com/geerlingguy/ansible-role-nginx

Ansible docs/exercise

To get more information about Ansible, you may visit https://docs.ansible.com/

Create two AWS instances (Ubuntu) and install Ansible in one of them.
Call these instances Ansible-N1 and Ansible-N2, respectively.
Install Ansible on Ansible-N1 using 'apt update && apt install ansible -y'

Visit the following link and create these file in Ansible-N1 as shown in the class.
https://github.com/5tuxnet/courses/tree/main/comp4912/lab4
Once the file created and SSH access verified, run 'ansible-playbook -i inventory.ini deploy-nginx.yml'

Try running the following commands from node with Ansible installed:

$ ansible all -i inventory.ini -m ping
$ ansible ec2-instance -i inventory.ini -m shell -a "hostname && uptime"
$ ansbile ec2-instance -i inventory.ini -m reboot --become

BCIT™

Terraform, the de facto standard for IaC

“Like the principle that the same source code generates the same binary, an Infrastructure as Code (IaC) model generates the same environment every time it is applied.”

Sam Guckenheimer (Product Owner, Azure DevOps)

Infrastructure as Code (IaC)

- ✓ Support more collaboration between Dev and Ops
- ✓ More automation, less human error
- ✓ Traceability and Integrity
- ✓ Repeatability and agility
- ✓ Increase Transparency

BCIT™

HashiCorp Terraform

Terraform, the de facto standard for IaC

Terraform is used to create, manage, and update infrastructure resources such as virtual networks, VMs, security rules, containers, domains and more. Almost any infrastructure type can be represented as a resource in Terraform.

- Declarative
- Written in Go
- Multiplatform
- Free Software & Open Source
- Freemium (Premium options: GUI, support, …)
- HashiCorp Product (https://developer.hashicorp.com/terraform/intro)

It is important to note that Terraform is primarily an IaC tool, designed to provision and manage infrastructure such as servers, networks, and storage, but **not to install or configure software on those servers**.

Terraform does NOT replace configuration management tools like Ansible.

BCIT™

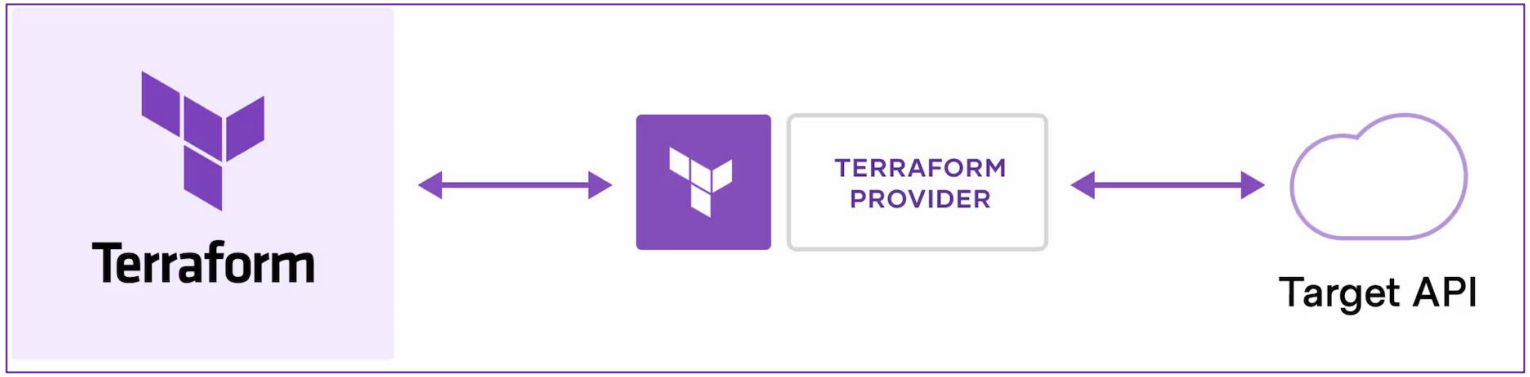
HashiCorp Terraform

Key benefits of Terraform

1. Declarative
2. Automate changes
3. Expediting DR process
4. Minimizing Human Errors
5. Standardize configurations
6. Stateful (No Agent needed)
7. Saving resources and money
8. Version Control for Infrastructure
9. Multi-cloud support (AWS/GCP/Azure)
10. Automating building Cloud Infrastructure

HashiCorp Terraform

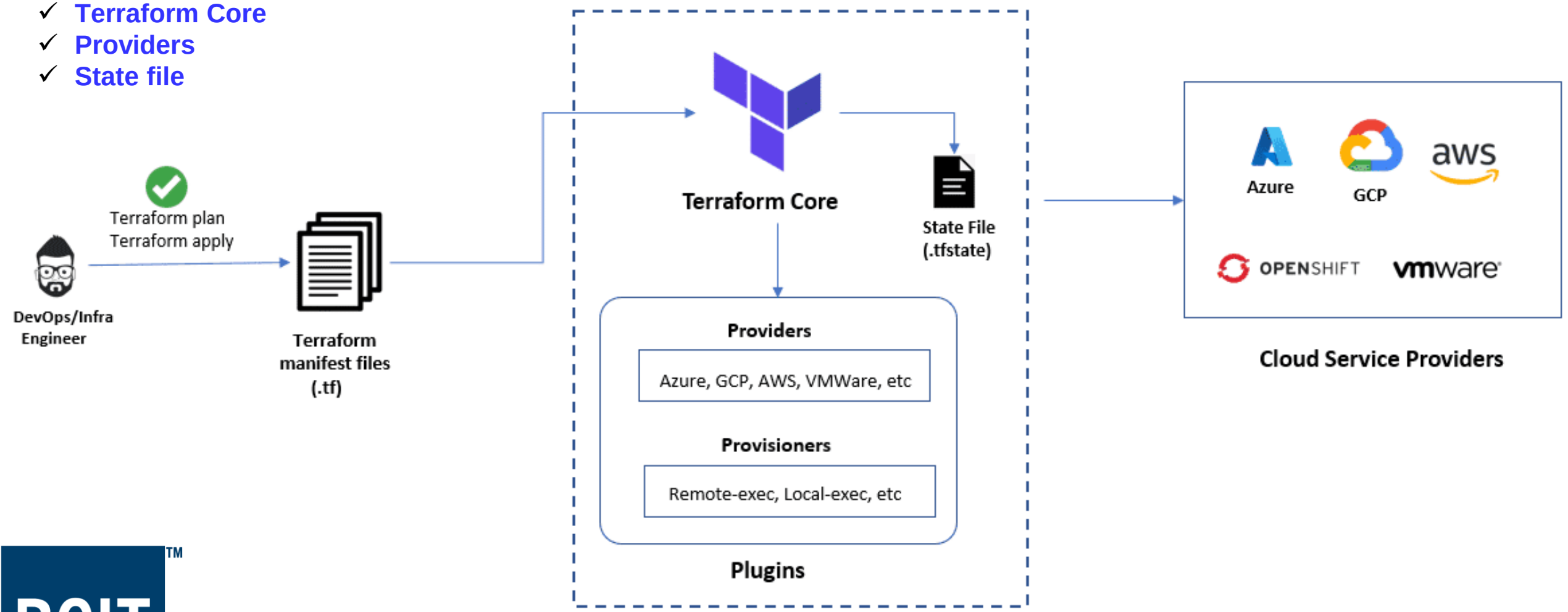
Terraform creates and manages resources on cloud platforms and other services through their APIs.
Providers enable Terraform to work with virtually any platform or service with an accessible API.
You can find all publicly available Providers on the Terraform Registry, e.g., AWS/GCP/GitHub.



BCIT™

Terraform Architecture

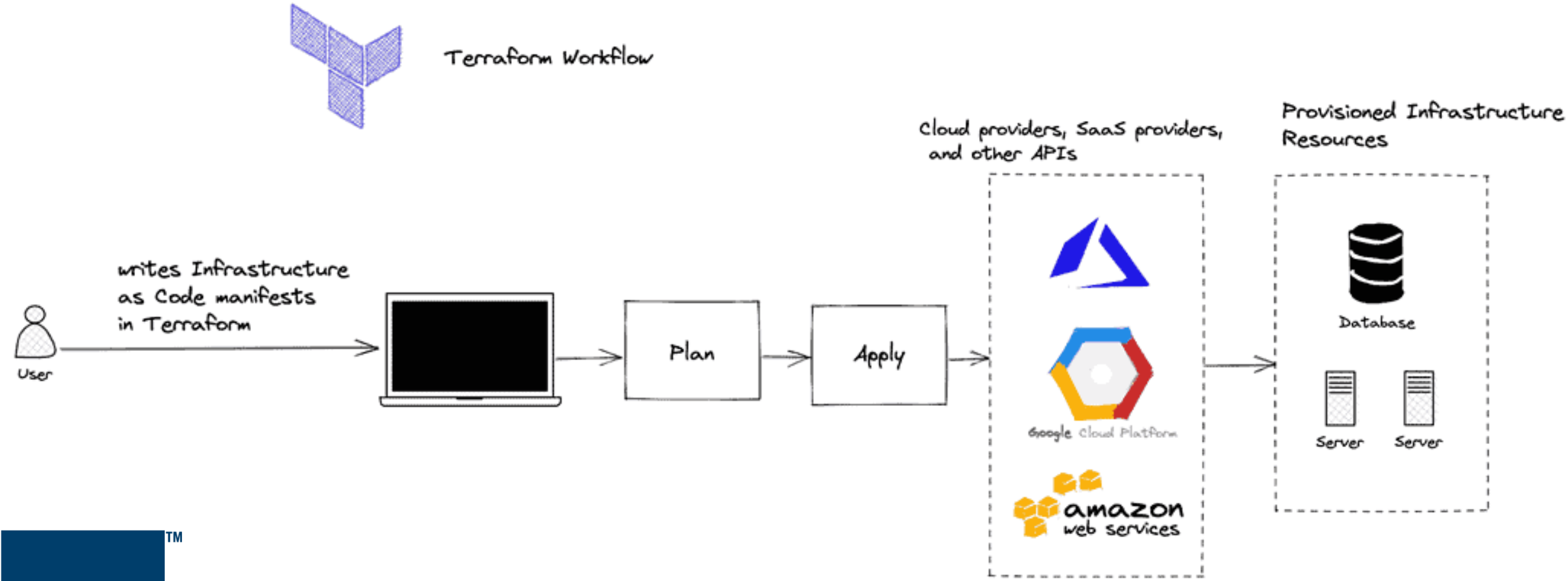
- ✓ Terraform Core
- ✓ Providers
- ✓ State file



BCIT™

https://spacelift.io/blog/terraform-architecture

How Terraform Works?



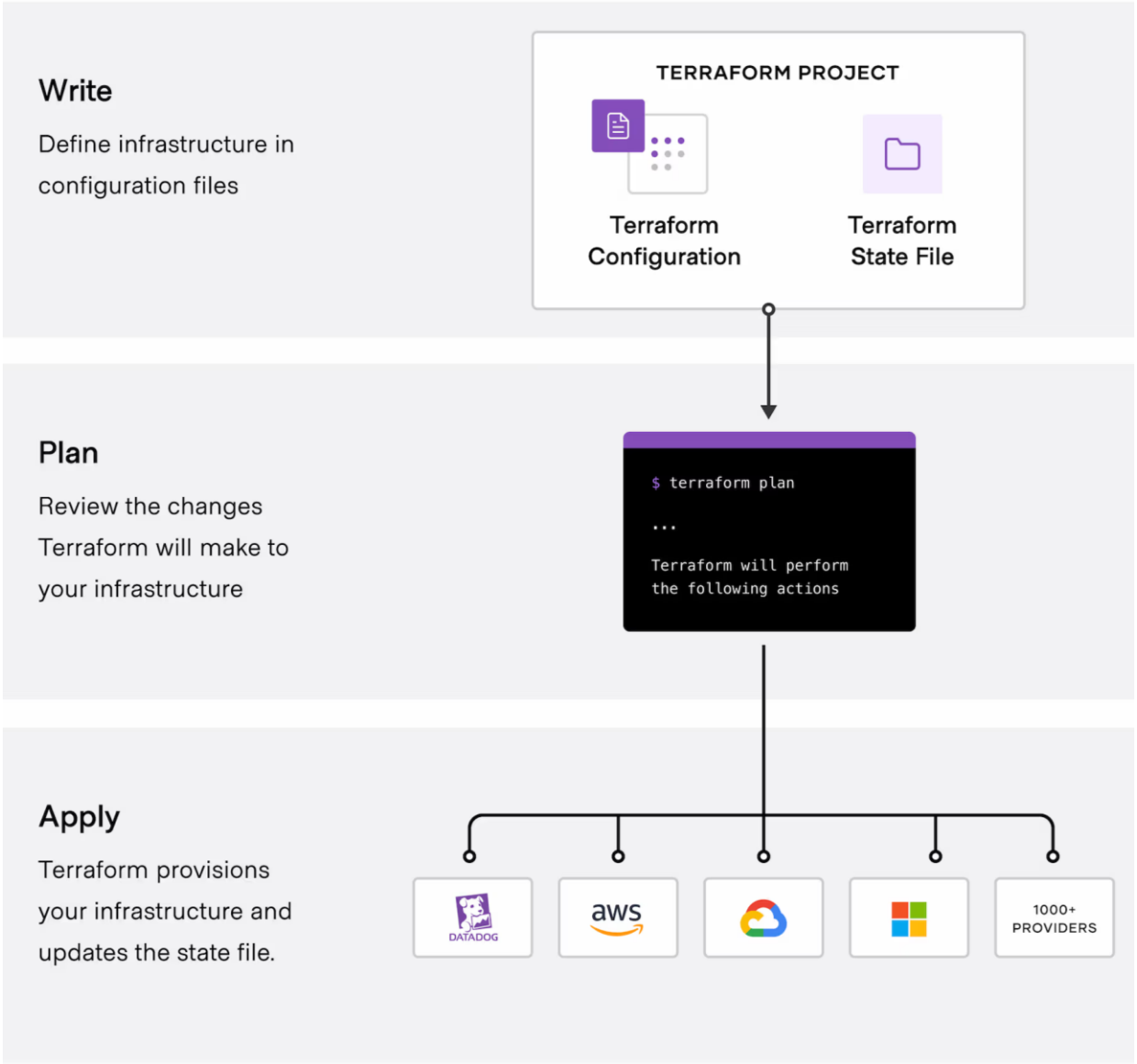
https://k21academy.com/terraform-iac/important-terraform-concepts/

How Terraform Works?

- ✓ **Terraform init** initializes the (local) Terraform environment. Usually executed only once per session.
- ✓ **Terraform plan** compares the Terraform state with the as-is state in the cloud, build and display an execution plan. This does not change the deployment (read-only).
- ✓ **Terraform apply** executes the plan. This potentially changes the deployment.
- ✓ **Terraform destroy** deletes all resources that are governed by this specific terraform environment.

BCIT™

The core Terraform workflow consists of three stages:



Terraform Structure

Here's a **Terraform configuration example** that provisions an **AWS EC2 instance**, demonstrating how each block is used in Terraform's structure.

Terraform Block

```
# 1 Terraform Block: Defines Terraform settings & backend storage
terraform {
  required_version = ">= 1.3.0"
  backend "s3" {
    bucket         = "my-terraform-state-bucket"
    key            = "ec2-instance/terraform.tfstate"
    region         = "us-west-2"
    encrypt        = true
  }
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
  }
}
```

Provider Block
Resource Block

```
# 2 Provider Block: Configures the AWS provider
provider "aws" {
  region = var.aws_region
}

# 3 Resource Block: Creates an EC2 instance
resource "aws_instance" "web_server" {
  ami           = data.aws_ami.latest_amazon_linux.id
  instance_type = var.instance_type
  subnet_id     = module.network.public_subnet_id

  tags = {
    Name = "Terraform-EC2-Instance"
  }
}
```

You can define many more parameters including number of CPU cores, networking details and security groups for your EC2 instance here.

BCIT™

Terraform Structure

Data Block

```
# 4 Data Block: Fetches the latest Amazon Linux 2 AMI
data "aws_ami" "latest_amazon_linux" {
  most_recent = true
  owners      = ["amazon"]

  filter {
    name   = "name"
    values = ["amzn2-ami-hvm-*-x86_64-gp2"]
  }
}
```

Module Block

```
# 5 Module Block: Calls a reusable VPC module
module "network" {
  source = "terraform-aws-modules/vpc/aws"
  version = "5.0"

  name = "my-vpc"
  cidr = "10.0.0.0/16"

  azs             = ["us-west-2a", "us-west-2b"]
  public_subnets  = ["10.0.1.0/24", "10.0.2.0/24"]
}
```

Locals Block

```
variable "instance_type" {
  description = "EC2 instance type"
  type        = string
  default     = "t2.micro"
}

# 8 Locals Block: Defines reusable local values
locals {
  env_prefix = "dev"
  instance_name = "${local.env_prefix}-web-server"
}
```

Output Block

```
# 6 Output Block: Displays the public IP of the EC2 instance
output "ec2_public_ip" {
  description = "The public IP of the EC2 instance"
  value       = aws_instance.web_server.public_ip
}

# 7 Variable Block: Defines configurable input variables
variable "aws_region" {
  description = "The AWS region to deploy resources in"
  type        = string
  default     = "us-west-2"
}
```

Providers

```
# 9 Providers: Specifies required providers
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
  }
}
```

BCIT™

Terraform Structure

1. Terraform Block
2. Provider Block
3. Resource Block
4. Data Block
5. Module Block
6. Output Block
7. Variable Block
8. Locals Block
9. Providers

Explanation of Each Block

Block Type	Purpose
Terraform Block	Specifies Terraform settings, backend for state storage, and required providers.
Provider Block	Defines the AWS provider and its configuration (e.g., region).
Resource Block	Creates an AWS EC2 instance using AMI and instance type variables.
Data Block	Retrieves the latest Amazon Linux AMI dynamically.
Module Block	Uses a pre-built Terraform module to create a VPC and subnets.
Output Block	Displays the public IP of the created EC2 instance.
Variable Block	Defines input variables for configurability (e.g., region, instance type).
Locals Block	Stores reusable values like environment prefixes.
Providers	Specifies required Terraform providers (same as in `terraform {}` block).

BCIT™

Terraform Configuration Files

main.tf
Primary configuration file where you define resources (e.g., EC2 instances, networking, databases).

providers.tf
Contains provider blocks (e.g., AWS, Azure, GCP) and associated configuration.

variables.tf
Defines input variables (their names, types, and optional descriptions).

terraform.tfvars
Provides default values for the variables defined in variables.tf.

outputs.tf
Contains output blocks to return values after terraform apply (e.g., instance IPs).

modules/ (Directory)
Contains reusable Terraform modules, each with its own main.tf, variables.tf, and outputs.tf.

.terraform.lock.hcl (Generated)
Auto-generated file that locks provider versions to ensure reproducible builds.

terraform.tfstate
It helps Terraform figure out what's already been created, what needs to change, and what should be destroyed. Indeed, it is the Source of Truth for your entire Infrastructure created using Terraform.

Terraform docs/exercise

To get more information about Ansible, you may visit:
https://developer.hashicorp.com/terraform/docs

Check the following link to see details for AWS Provider (ec2/aws_instances):

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance

For creation of your own instance, you may use **AMI-Catalog** to find the **AMI ID**. Ensure to select a right **Region** before search for AMI ID.

main.tf > resource "aws_instance" "web_server"

```
provider "aws" {
  profile = "default"
  region = "us-west-1"
}

resource "aws_instance" "web_server" {
  ami           = "ami-07d2649d67dbe8900"
  instance_type = "t2.micro"

  tags = {
    Name = "Terraform_instance"
  }
}
```

HashiCorp Terraform

BCIT™

Ansible + Terraform

Check This Video (Red Hat)

https://www.youtube.com/watch?v=vQSDWa8MlN8

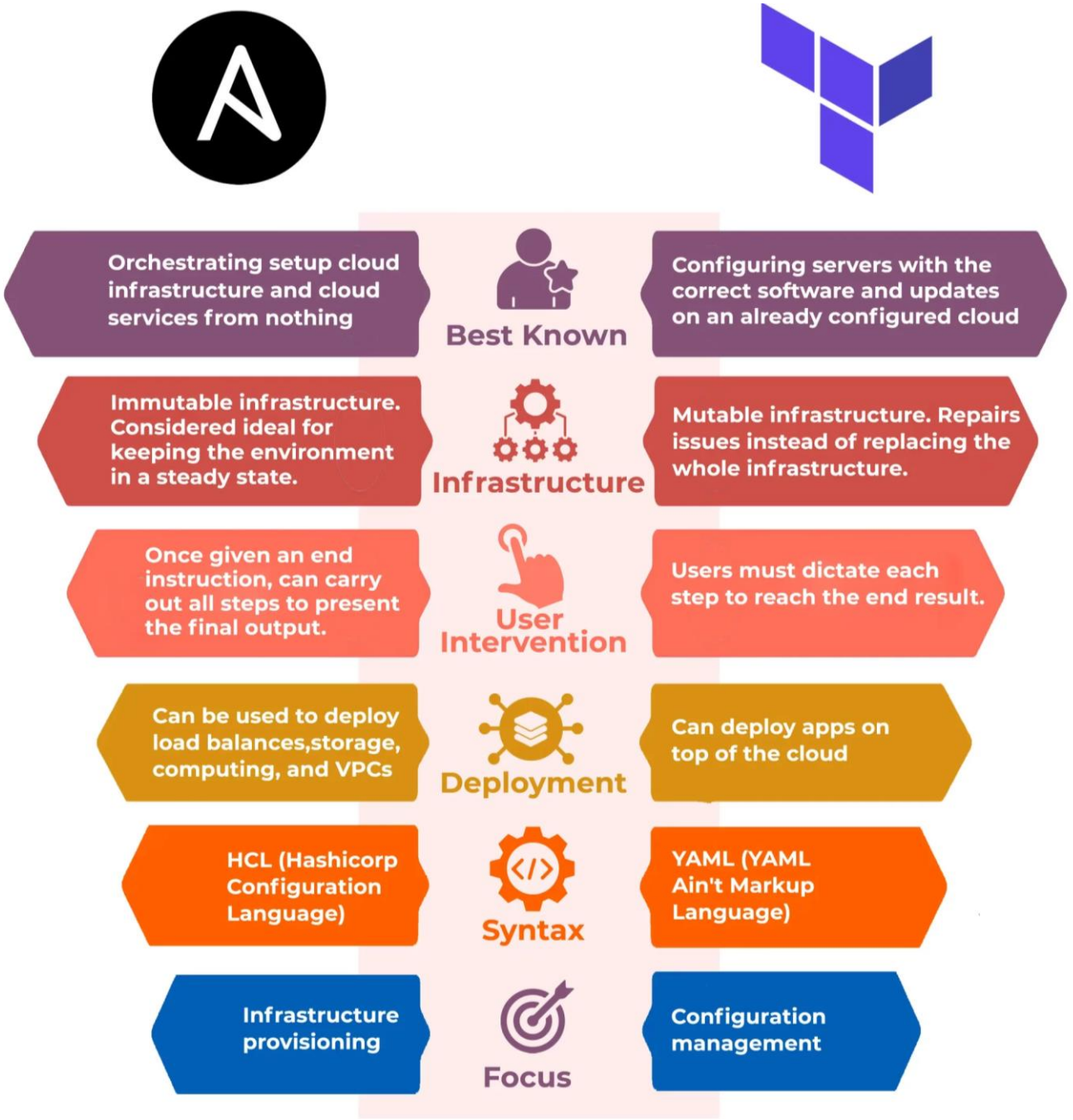
NetworkChuck (Ansible)

https://www.youtube.com/watch?v=5hycyr-8EKs

John Hammond (Terraform)

https://www.youtube.com/watch?v=a3vVUiLzm8w

BCIT™



End of Lecture #7

THANK

Y☺U

- Dawood Sajjadi

BCIT™