Terraform State:

Administración y Buenas Prácticas

Robert Rozas Navarro

Software Engineer at Microsoft

https://github.com/AshWilliams/
https://stackoverflow.com/users/1987838/hackerman?tab=profile

In this talk

- Infrastructure as Code (IaC)
- Terraform
- Terraform State
- Demo

$ whoami - Robert

Robert Rozas Navarro

robert.rozas.n@outlook.com

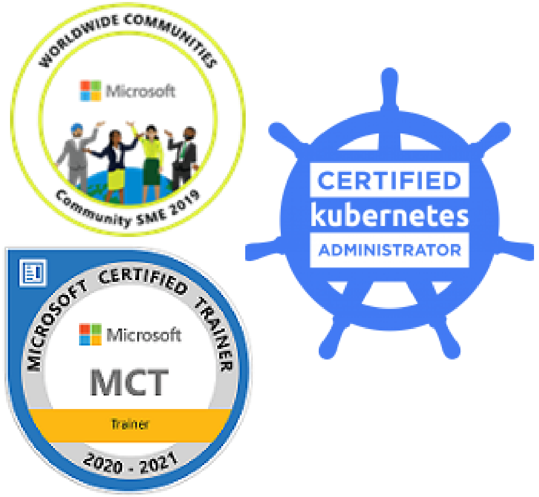
Software Engineer at Microsoft

Open Source Advocate

Microsoft SME

Github: @ashwilliams

StackOverflow: @hackerman

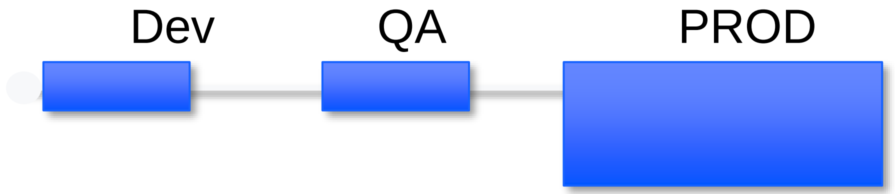


Infrastructure as Code

What is Infrastructure as Code (IaC)

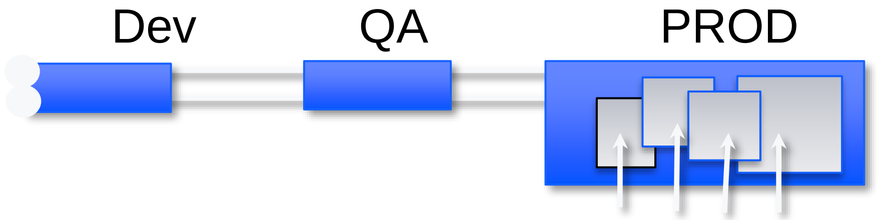
- Build the infrastructure for an App all at once through automation
- Not just for Cloud, Software Defined Data Center
- Embedded Documentation
- Source Control
- Flexible Build Process

DevOps Confronts the Agile Challenge



Circa 2010

Select Single Applications | Virtual Machines | Infrastructure as Code | Continuous Integration



Circa 2014

Shared Private/Hybrid Cloud Environments | Multiple enterprise applications | Orchestration and co-ordination is needed

How to Get Started

People → Process → Products

Process

- Simplicity
- Modular
- Flexible
- Versioning

Products

- Powershell/Bash
- VS Code
- GitHub
- Azure Automation, Ansible, Terraform

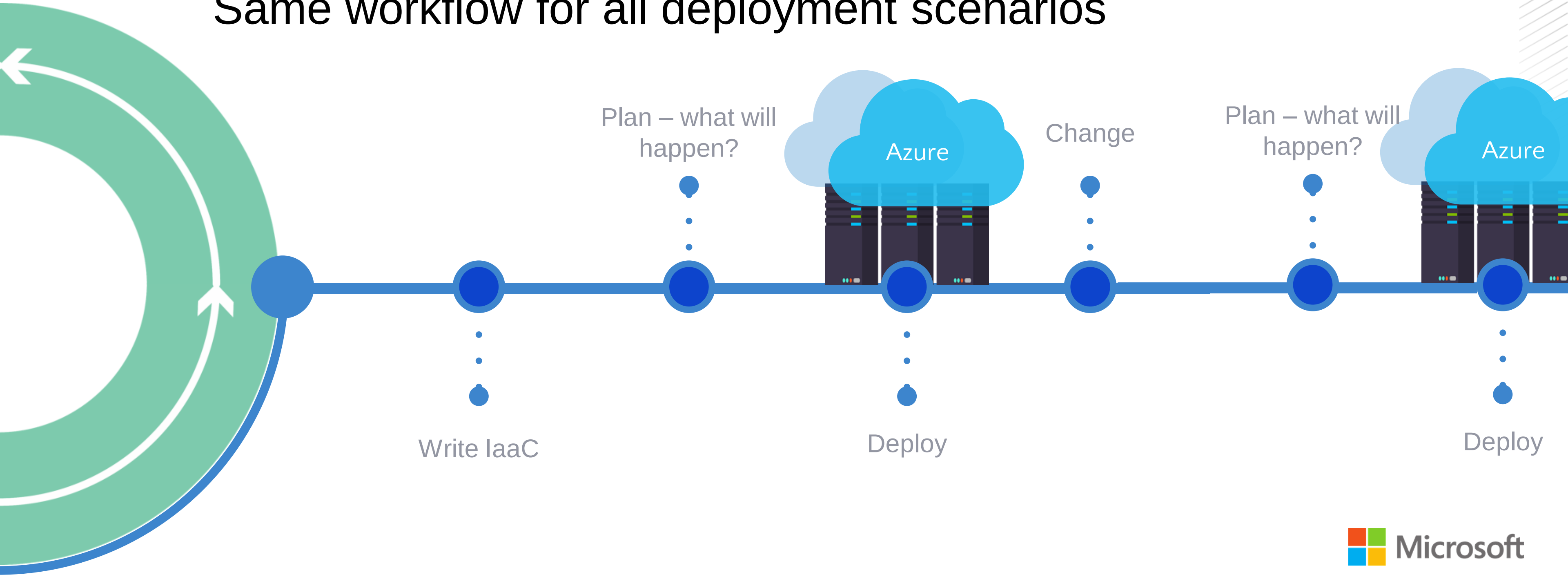
Steps to Implement IaC

- Find something easy to automate – low effort, low risk
- Set the right expectations – experimentation is necessary
- Prove that it works – show the time savings and effort needed
- Don't be shy about it – advocate
- Do it again

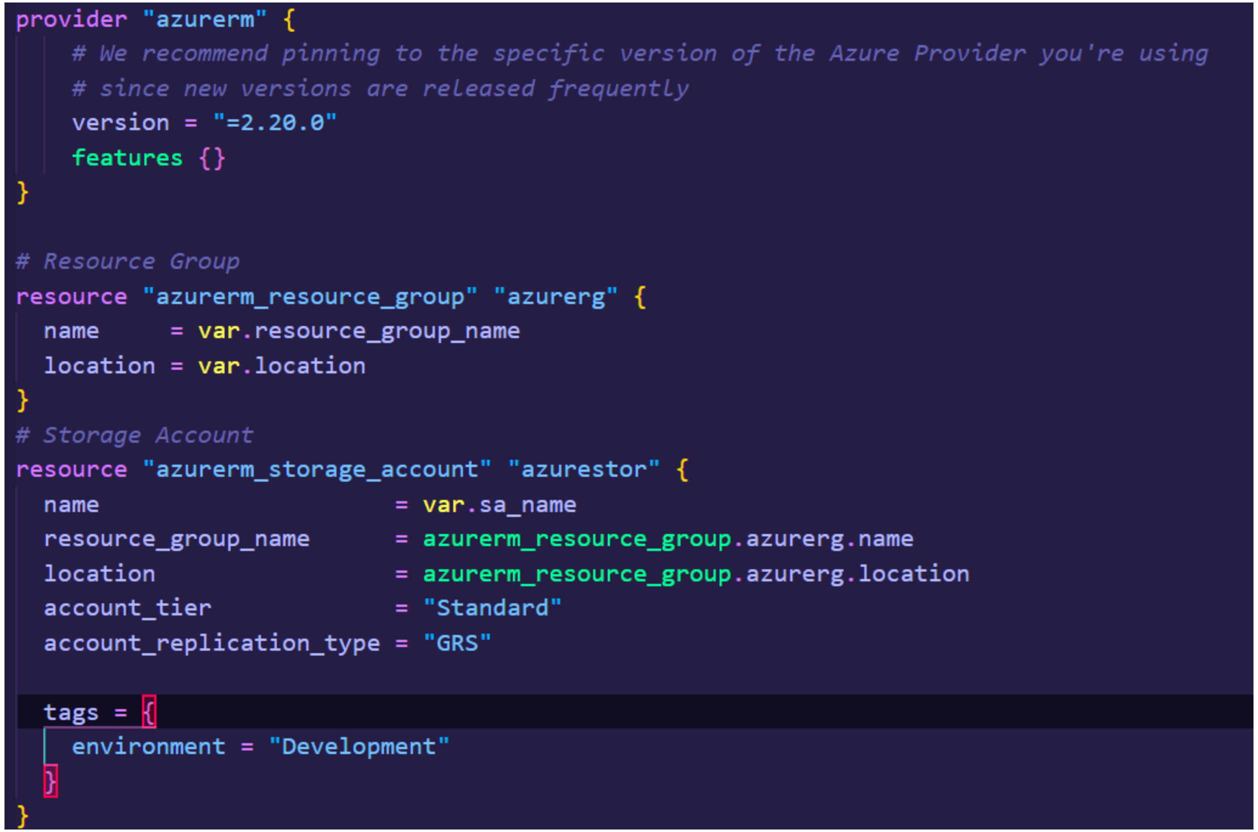
Terraform

Terraform

Write, *plan* and create infrastructure as code
Same workflow for all deployment scenarios



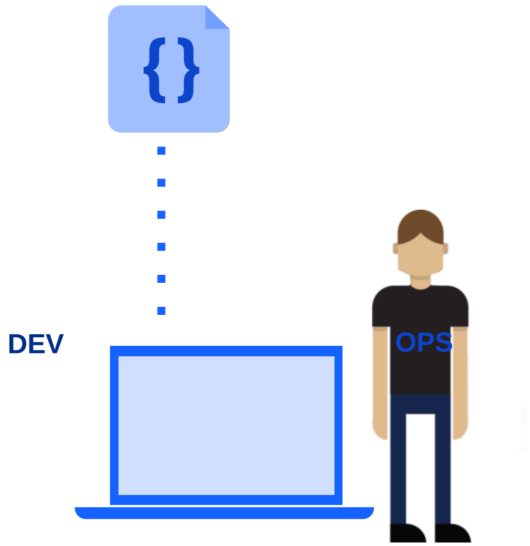
Creating Terraform Templates

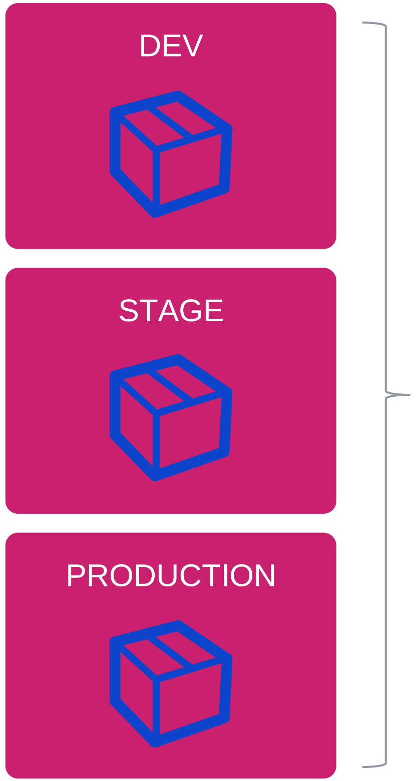


```
provider "azurerm" {
    # We recommend pinning to the specific version of the Azure Provider you're using
    # since new versions are released frequently
    version = "=2.20.0"
    features {}
}

# Resource Group
resource "azurerm_resource_group" "azurerg" {
  name     = var.resource_group_name
  location = var.location
}
# Storage Account
resource "azurerm_storage_account" "azurestor" {
  name                     = var.sa_name
  resource_group_name      = azurerm_resource_group.azurerg.name
  location                 = azurerm_resource_group.azurerg.location
  account_tier             = "Standard"
  account_replication_type = "GRS"

  tags = {
    environment = "Development"
  }
}
```

Microsoft

Environment Parity – Idempotency





Same Terraform Code

Replace Dedicated Staging

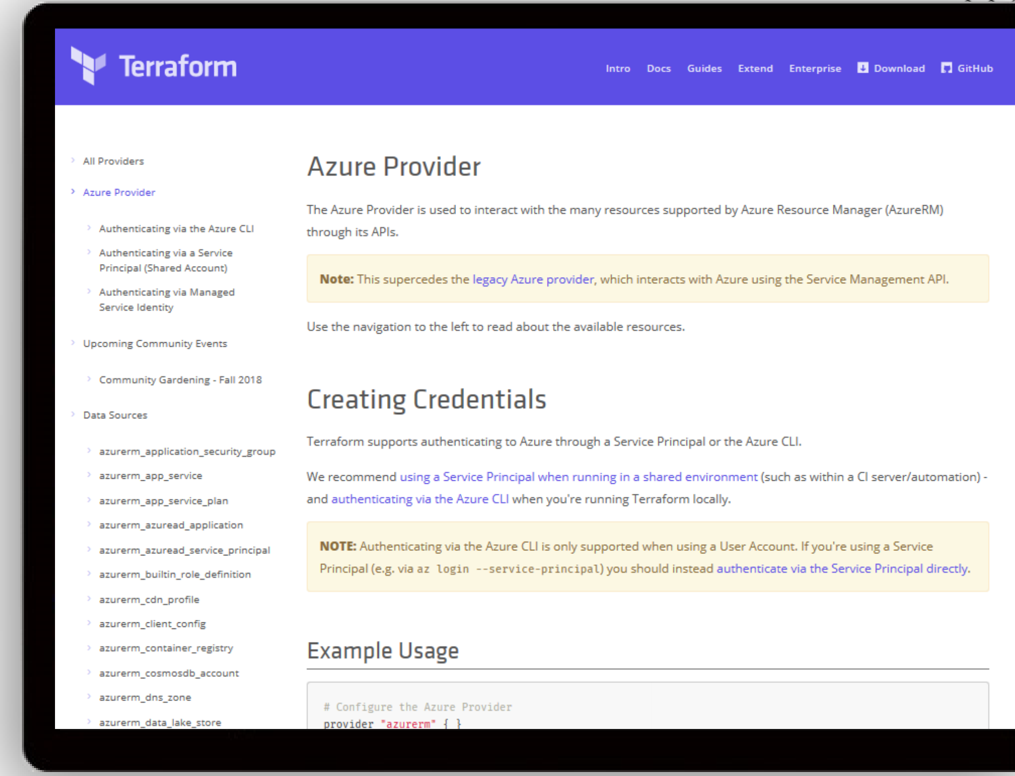
Speed Up Infra provisioning

Azure DevOps Tool Integrations

Bringing native Azure support for customers using Terraform

- [Documentation Hub for Terraform]
- [Terraform in Azure Cloud Shell]
- Azure Resource Provider
- [Azure Module Registry]
- [Azure Cloud Shell Integration]

docs.microsoft.com/azure/terraform


Terraform
Intro
Docs
Guides
Extend
Enterprise
Download
GitHub
All Providers
Azure Provider
Authenticating via the Azure CLI
Authenticating via a Service Principal (Shared Account)
Authenticating via Managed Service Identity
Upcoming Community Events
Community Gardening - Fall 2018
Data Sources
azurerm_application_security_group
azurerm_app_service
azurerm_app_service_plan
azurerm_azuread_application
azurerm_azuread_service_principal
azurerm_builtin_role_definition
azurerm_cdn_profile
azurerm_client_config
azurerm_container_registry
azurerm_cosmosdb_account
azurerm_dns_zone
azurerm_data_lake_store

Azure Provider

The Azure Provider is used to interact with the many resources supported by Azure Resource Manager (AzureRM) through its APIs.

Note: This supercedes the legacy Azure provider, which interacts with Azure using the Service Management API.

Use the navigation to the left to read about the available resources.

Creating Credentials

Terraform supports authenticating to Azure through a Service Principal or the Azure CLI.

We recommend using a Service Principal when running in a shared environment (such as within a CI server/automation) - and authenticating via the Azure CLI when you're running Terraform locally.

NOTE: Authenticating via the Azure CLI is only supported when using a User Account. If you're using a Service Principal (e.g. via `az login --service-principal`) you should instead authenticate via the Service Principal directly.

Example Usage

```
# Configure the Azure Provider
provider "azurerm" { }
```

Terraform State

Microsoft

Terraform Backends

- Local
- Remote
- Workspaces (former known as environments)
- Locking
- Encryption at rest
- Versioning
- Note: Backend configuration doesn't support interpolations.

HashiCorp Consul

etcd

Azure Blob

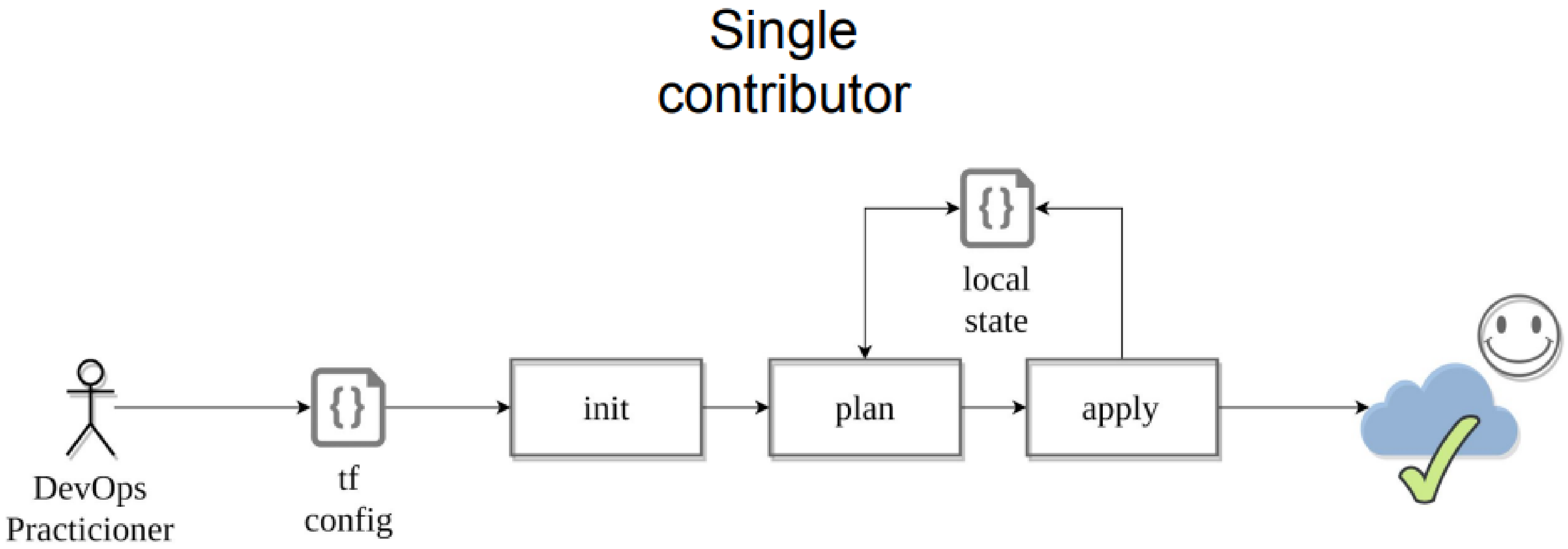
AWS S3

Terraform ENTERPRISE

Google Cloud Storage

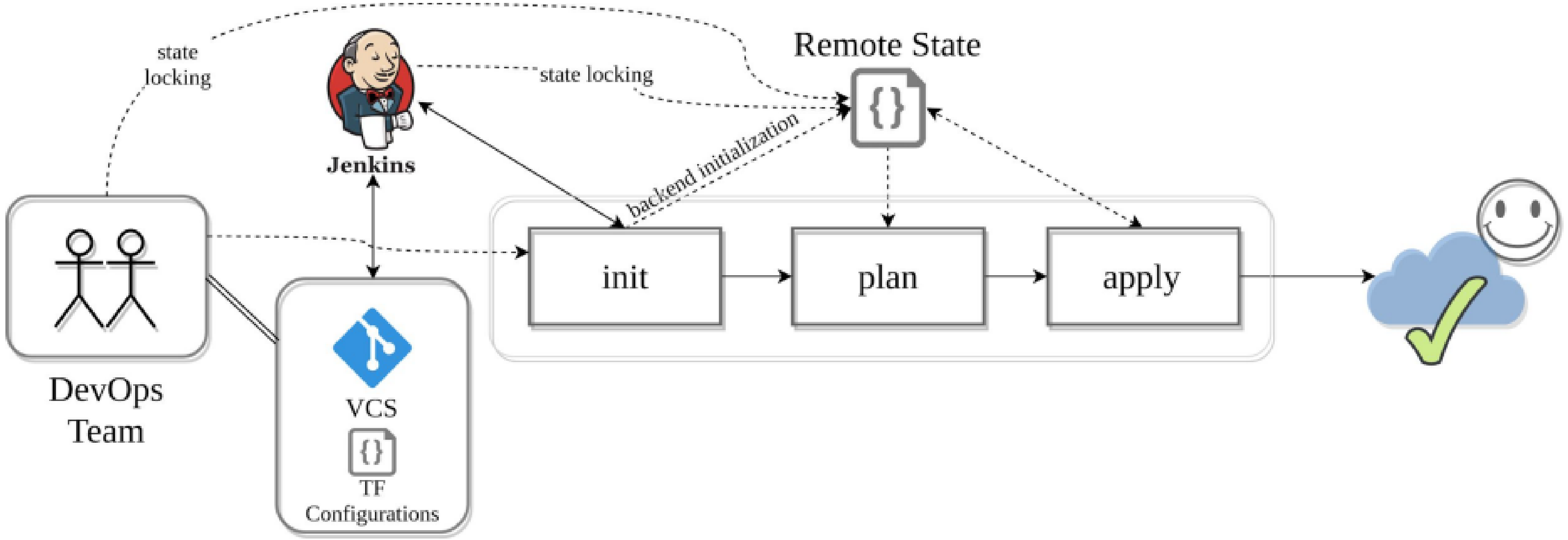
Microsoft

Workflow: Adoption stages

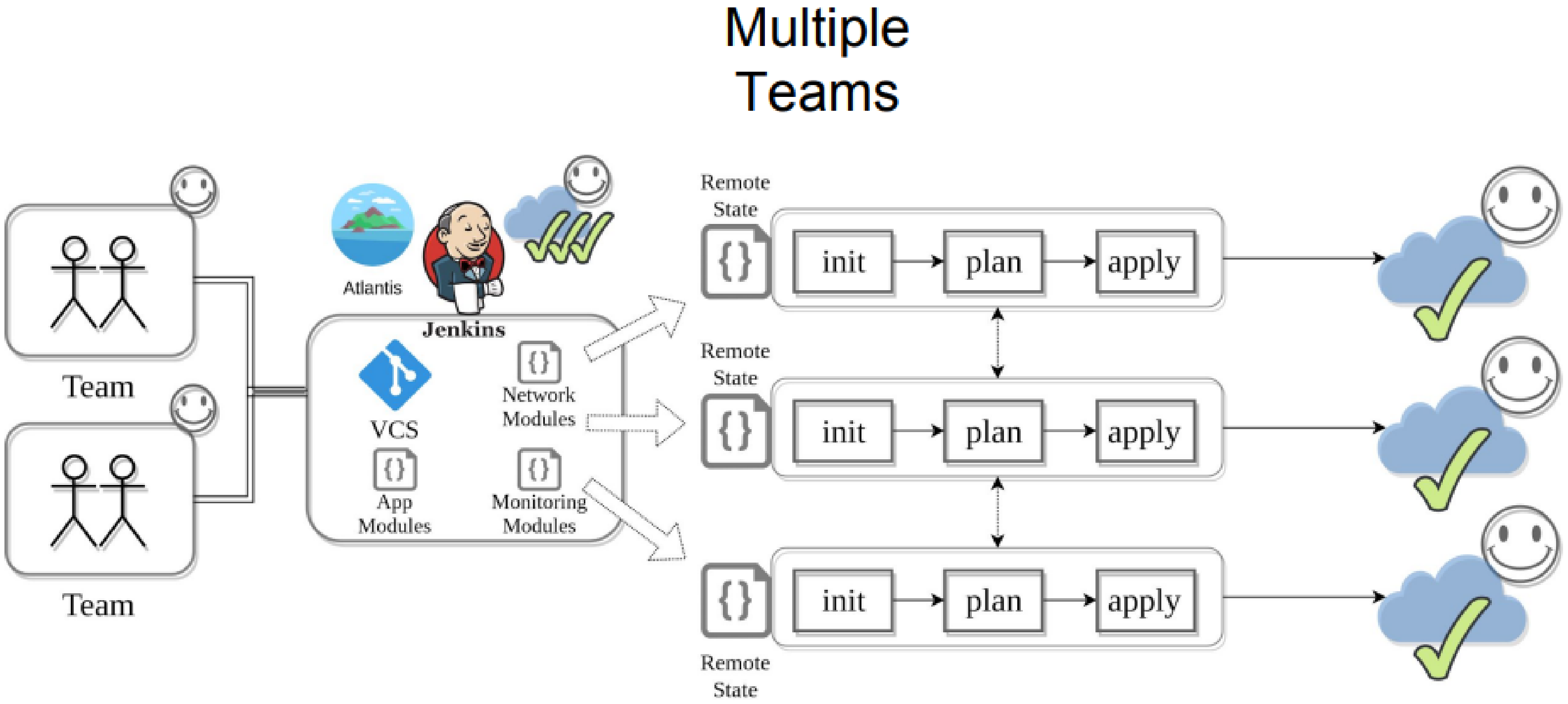


Workflow: Adoption stages

Team Collaboration



Workflow: Adoption stages



Terraform state file

- Backup your state files + use Versioning and Encryption
- Do Not edit manually!
- Main Keys: cat terraform.tfstate.backup | jq 'keys'
  - a. "lineage" - Unique ID, persists after initialization
  - b. "modules" - Main section
  - c. "serial" - Increment number
  - d. "terraform_version" - Implicit constraint
  - e. "version" - state format version
- Use “terraform state” command
  - a. mv - to move/rename modules
  - b. rm - to safely remove resource from the state. (destroy/retain like)
  - c. pull - to observe current remote state
  - d. list & show - to write/debug modules

Demo

https://github.com/AshWilliams/HashiTalksLatam2021

Thank You!

Microsoft