Terraform 101

Landscaping your infrastructure garden

Erwin Staal

@erwin_staal

Erwin Staal

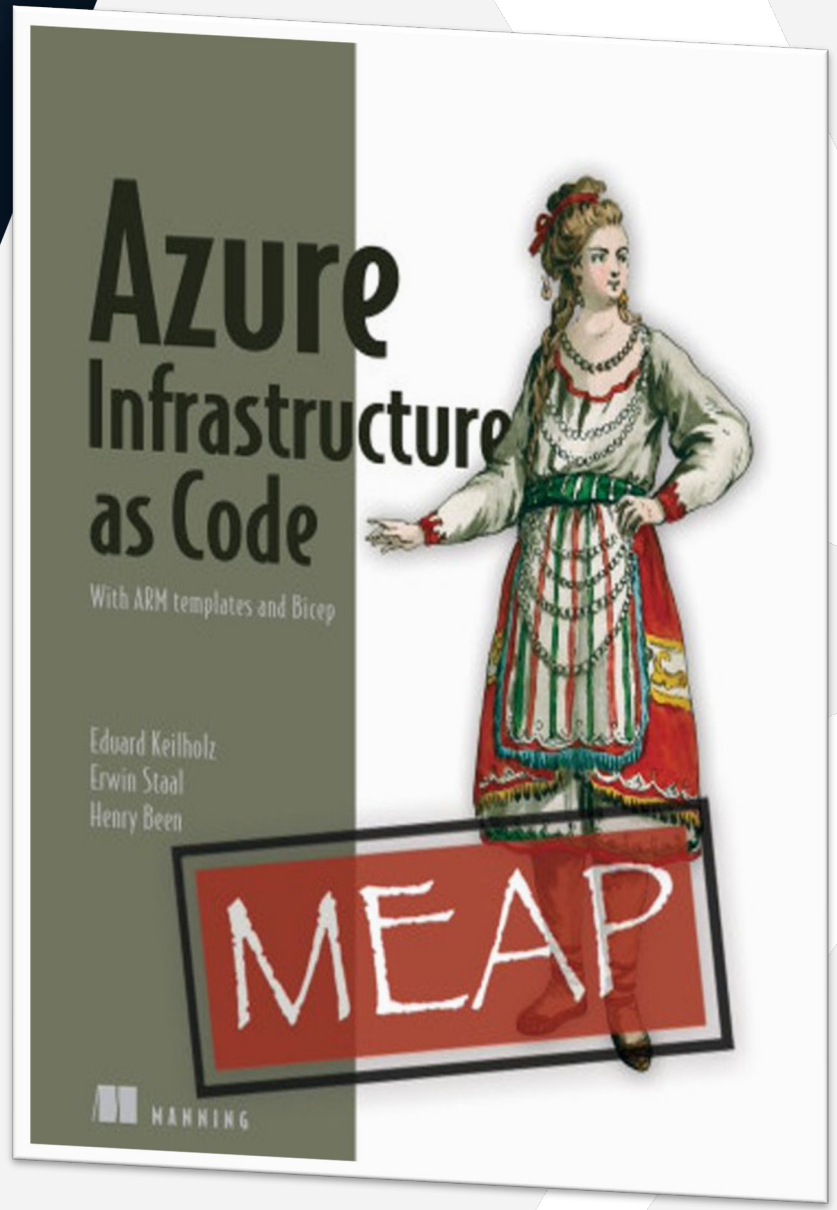
AZURE ARCHITECT

DEVOPS CONSULTANT

Xpirit

Erwin Staal

- AZURE ARCHITECT
- DEVOPS CONSULTANT



Terraform

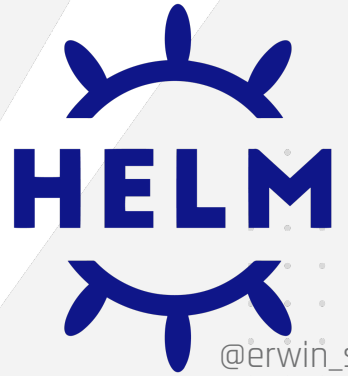
"Terraform is a tool for building, changing, and versioning infrastructure safely and efficiently. Terraform can manage existing and popular service providers as well as custom in-house solutions."

- Created by HashiCorp in 2014
- Open Source
- Runs on Linux, MacOS & Windows
- Uses HCL to describe infrastructure (HCL = HashiCorp Configuration Language)

Providers

- Groups resources in logical units
- Official and Community providers
- Distributed as plugins

PagerDuty



Terraform Template

- Describe infrastructure using HCL
  - High-level settings
  - Activate provider
  - Define resources

```hcl
# Terraform block used to configure the high-level settings for Terraform
terraform {
  required_version = ">= 1.1.7"

  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "3.27.0"
    }
  }
}

# Configure the AzureRM Provider
provider "azurerm" {
  features {}
}

# Create a resource
resource "azurerm_resource_group" "rg" {
  name     = "rg-${local.project_name}-${var.environment}"
  location = local.location
}
```

State file (json)

- Terraform must store state about your managed infrastructure and configuration
- Stored, by default, in a local file named "terraform.tfstate”
- Can also be stored remotely, which works better in a team environment

Xpirit

Terraform uses state to:

Map real world resources to your configuration

Keep track of metadata

Performance

Synchronize

How to use state in teams?

Developers must:

Always have the latest state data before running Terraform

Make sure that nobody else runs Terraform at the same time

Xpirit

Remote state

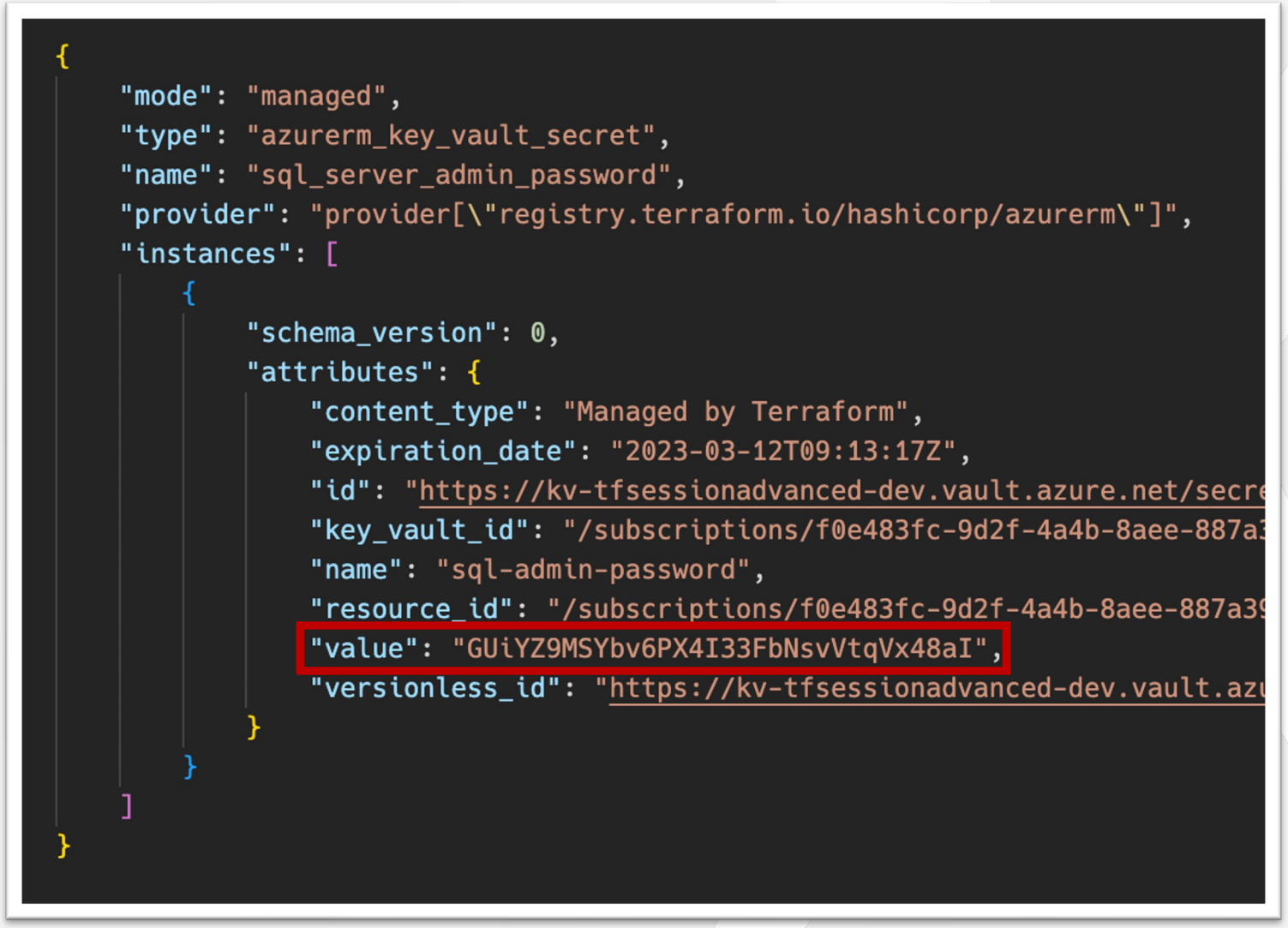
Implemented by a backend

- HashiCorp Consul, Amazon S3, Azure Blob Storage, Google Cloud Storage, Alibaba Cloud OSS, Terraform Cloud, and more

Locking

State might contain secrets

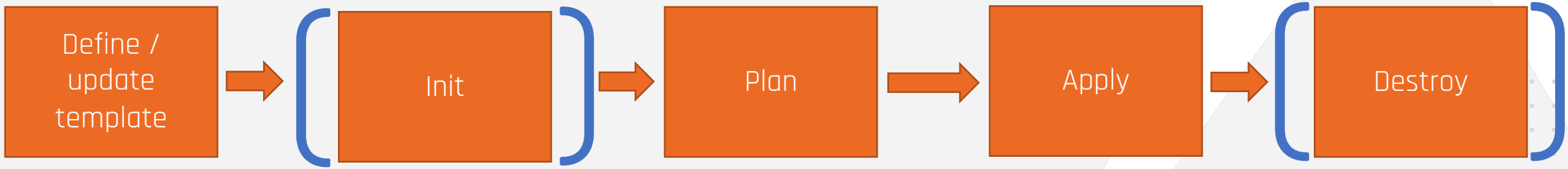
- Terraform state can contain sensitive data: password, private keys, etc.
- State is stored in plain-text JSON files
- Encryption at rest, IAM, TLS connection, IP whitelisting



Xpirit

Terraform workflow (*and Terraform CLI*)

- Define / update templates
- Init (*terraform init*)
  - Downloads providers and initializes backends
- Create a plan (*terraform plan*)
  - Creates an execution plan, by building a resource graph
- Apply (*terraform apply*)
  - Creates, updates and destroys resource to match the desired state
- Destroy (*terraform destroy*)
  - Destroys created resources



Demo

Tips and Tricks

When a resource is not supported or in preview

```
resource "azapi_resource" "automationAccount" {
  type      = "Microsoft.Automation/automationAccounts@2021-06-22"
  name      = "myAccount"
  parent_id = azurerm_resource_group.test.id

  location = azurerm_resource_group.test.location
  body = jsonencode({
    properties = {
      disableLocalAuth    = true
      publicNetworkAccess = false
      sku = {
        name = "Basic"
      }
    }
  })
}
```

```
resource "null_resource" "readcontentfile" {
  provisioner "local-exec" {
    command = "myscript.ps1"
    interpreter = ["PowerShell", "-Command"]
  }
}
```

Xpirit

When a resource already exists?

When you don't need to manage it: use it as a data source

```
data "azurerm_resource_group" "rg" {
  name     = "rg-${local.project_name}-${var.environment}"
}

resource "azurerm_service_plan" "app_service_plan" {
  name                = "asp-${local.project_name}-${var.environment}"
  resource_group_name = data.azurerm_resource_group.rg.name
  location            = data.azurerm_resource_group.rg.location
  os_type             = "Linux"
  sku_name            = "B1"
}
```

When you do need to manage it: use import

```
resource "azurerm_resource_group" "rg" {
  name     = "rg-${local.project_name}-${var.environment}"
  location = local.location
}
```

terraform import "azurerm_resource_group.rg" id-from-azure"

Ecosystem

- Terraform
  - fmt
  - validate
- TFLint
  - A static analysis tool for Terraform code that helps you detect and fix style issues, syntax errors, and best practices violations.
- TFSec
  - tfsec uses static analysis of your terraform code to spot potential misconfigurations.
- Terratest
  - Terratest is a Go library that provides patterns and helper functions for testing infrastructure
- Git Pre-commit

Xpirit

Thanks!

Erwin Staal

@erwin_staal

https://www.erwinstaal.nl - https://www.github.com/staal-it