PRAIRIE DEV CON

WEB | DEV | CLOUD | AI

THE DEVELOPER'S SWISS ARMY KNIFE FOR INFRASTRUCTURE


PRAIRIE
DEV CON
WEB | DEV | CLOUD | AI


NORTHFIELD


Johnston
group


Lotlinx


online
business systems


LAMBERT
NEMEC
GROUP
EXCELLENCE IN RECRUITMENT

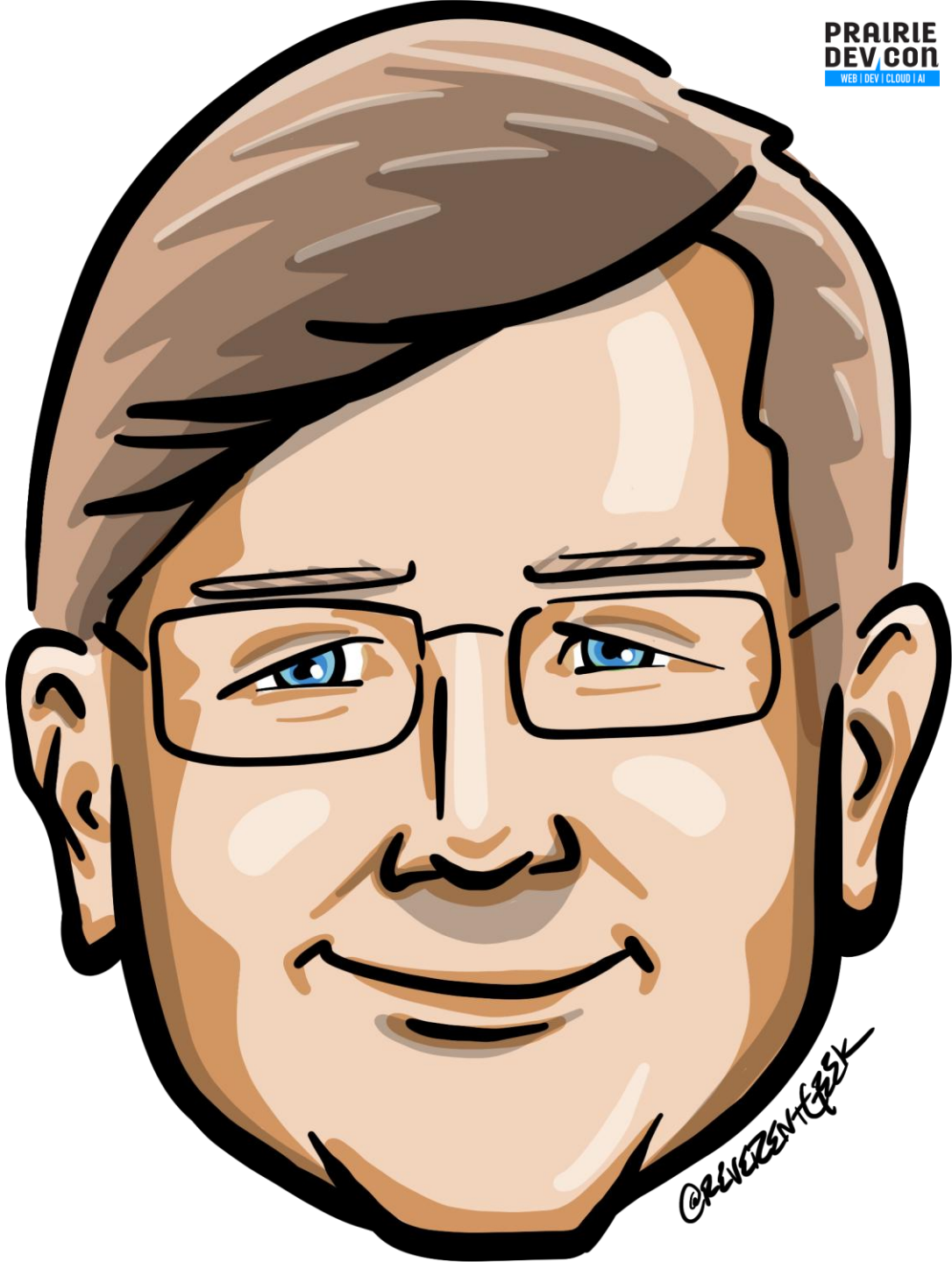

CONQUEST


ATLASSIAN

Who is Chad Green?

- chadgreen@chadgreen.com
- TaleLearnCode
- ChadGreen.com
- ChadGreen & TaleLearnCode
- ChadwickEGreen

Microsoft® MVP Most Valuable Professional



PRAIRIE DEV CON WEB | DEV | CLOUD | AI

The Tale of the Master Builder's Blueprint

Terraform: The Developer's Swiss Army Knife for Infrastructure

The Tale of the Master Builder’s Blueprint

Terraform: The Developer’s Swiss Army Knife for Infrastructure

The Tale of the Master Builder’s Blueprint

Terraform: The Developer’s Swiss Army Knife for Infrastructure

The Terraform Scroll

The Terraform Scroll

- Defining Infrastructure
- Provisioning Resources
- Managing Changes

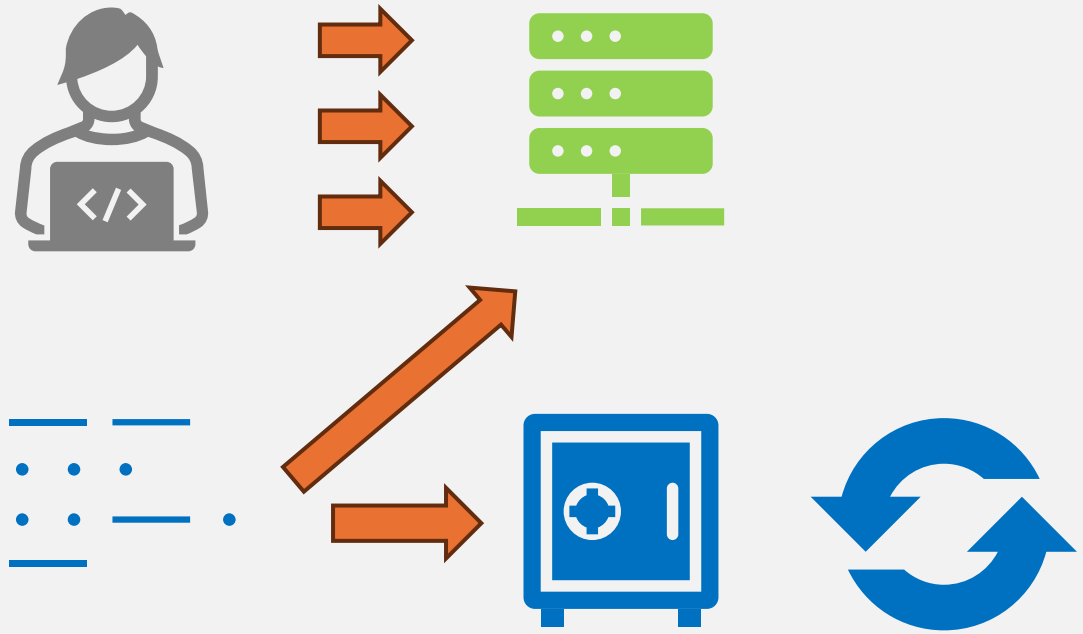
The Role of Eirik the Enlightened

The Benefits

- Infrastructure as Code
- Automated Provisioning
- Scalability and Flexibility

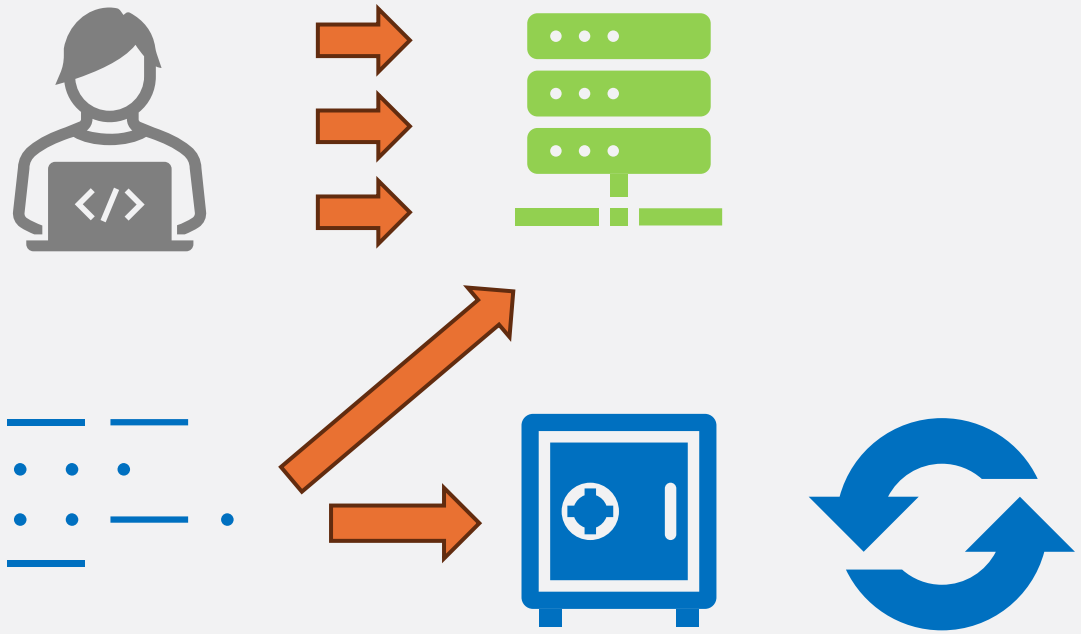
Infrastructure as Code (IaC)

Infrastructure as Code (IaC)



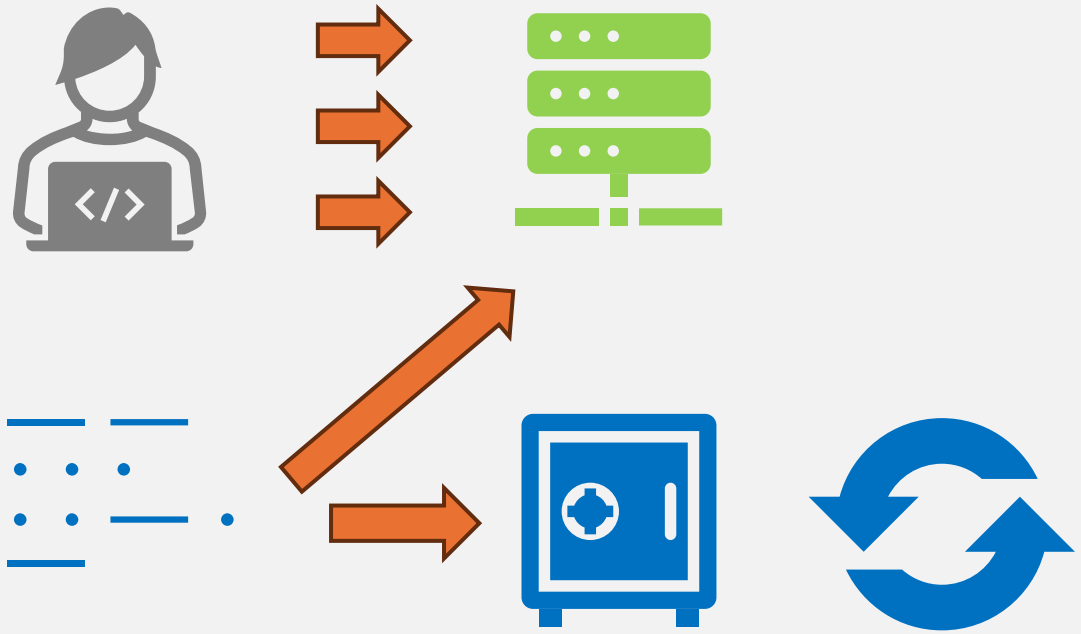
Infrastructure as Code (IaC)

Automation



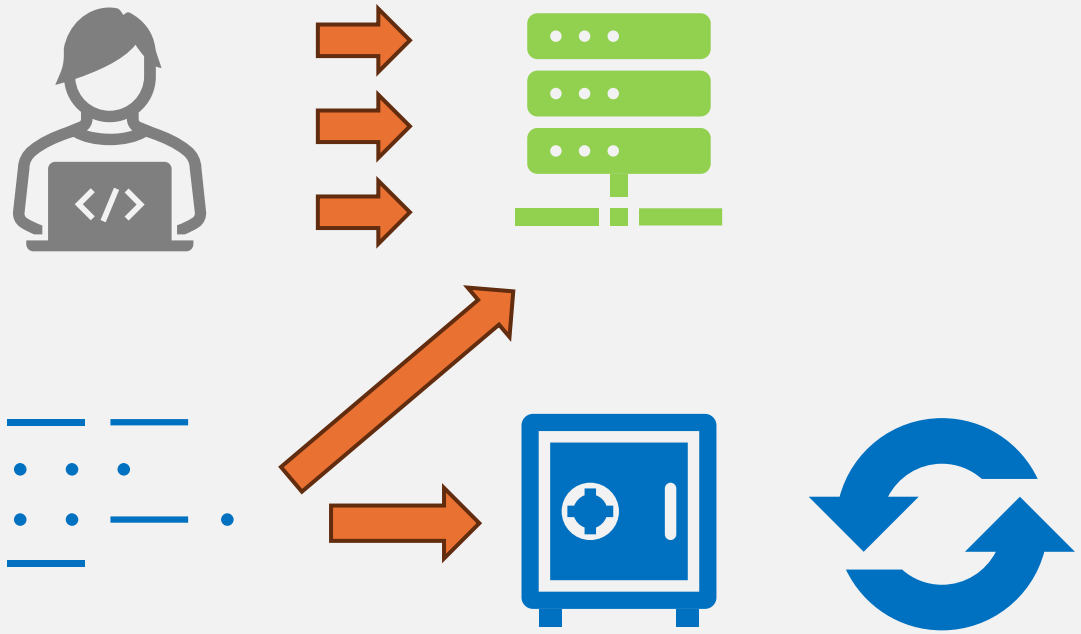
Infrastructure as Code (IaC)

- Automation
- Consistency



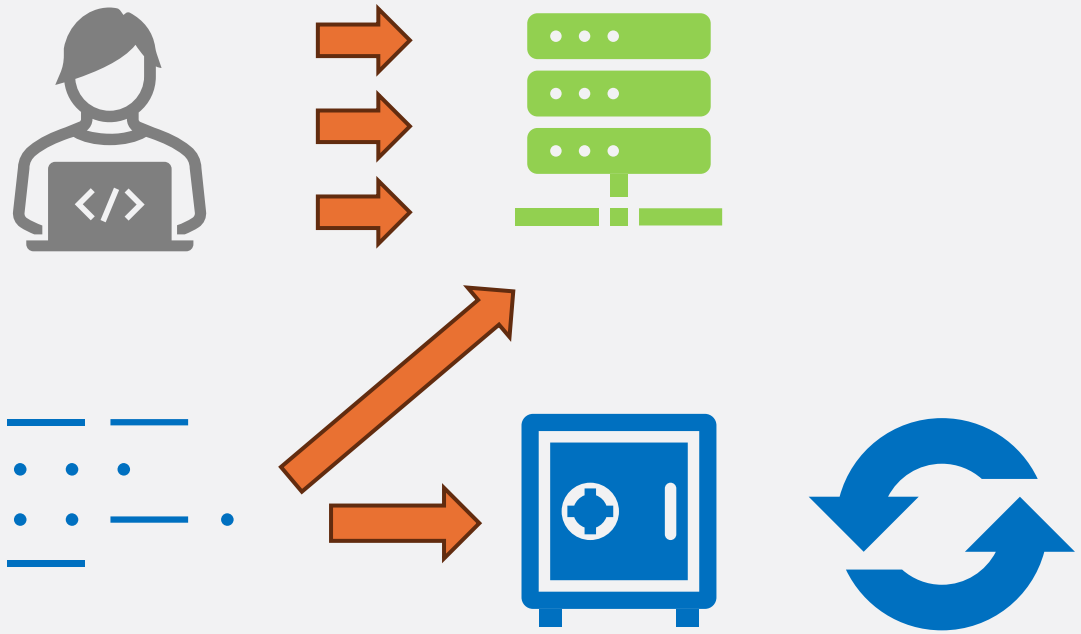
Infrastructure as Code (IaC)

- Automation
- Consistency
- Version Control



Infrastructure as Code (IaC)

- Automation
- Consistency
- Version Control
- Documentation

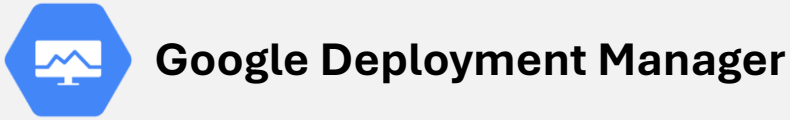


IaC Tools

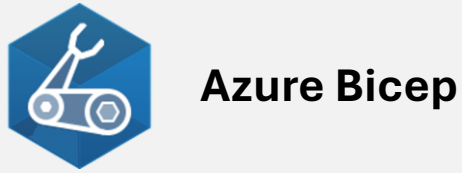

OpenTofu


Pulumi


AWS CLOUDFORMATION


Google Deployment Manager


aws
Cloud
Development
Kit


Azure Bicep


HashiCorp
Terraform

Terraform

Terraform

Declarative Language

Terraform

Declarative Language

Providers

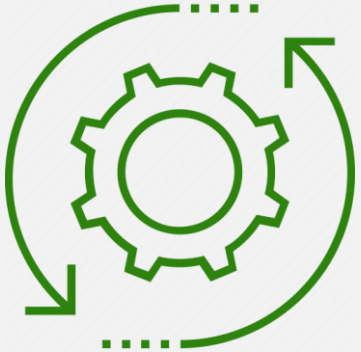
Terraform

- Declarative Language
- Providers
- State Management

Terraform

- Declarative Language
- Providers
- State Management
- Plan and Apply

Write



Initialize

Plan

Apply

Terraform

- Declarative Language
- Providers
- State Management
- Plan and Apply
- Modularity

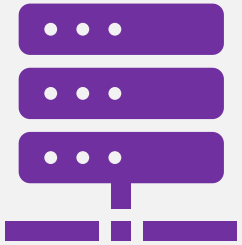
Core Concepts

Core Concepts

Providers

Core Concepts

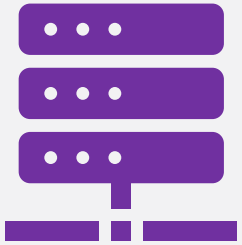
Providers



Resources

Core Concepts

Providers

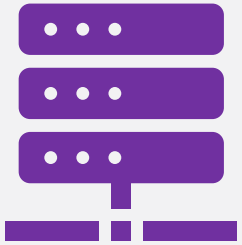


Resources

Modules

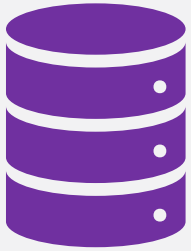
Core Concepts

Providers



Resources

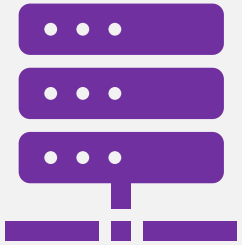
Modules



State

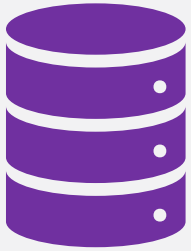
Core Concepts

Providers

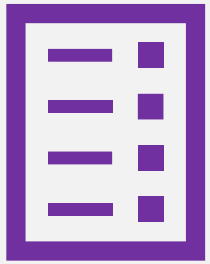


Resources

Modules



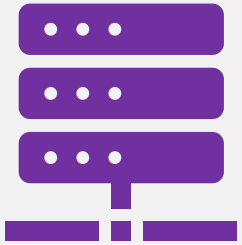
State



Plans

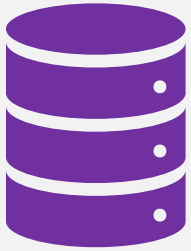
Core Concepts

Providers

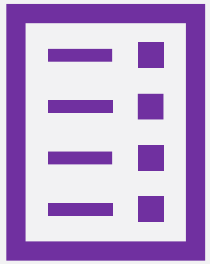


Resources

Modules



State



Plans

Configuration Files

Configuration Files

Common File Extension

- .tf – main configuration files
- .tfvars – variables
- .tfstate – state files
- .tfconfig – provider configuration

Typical Structure

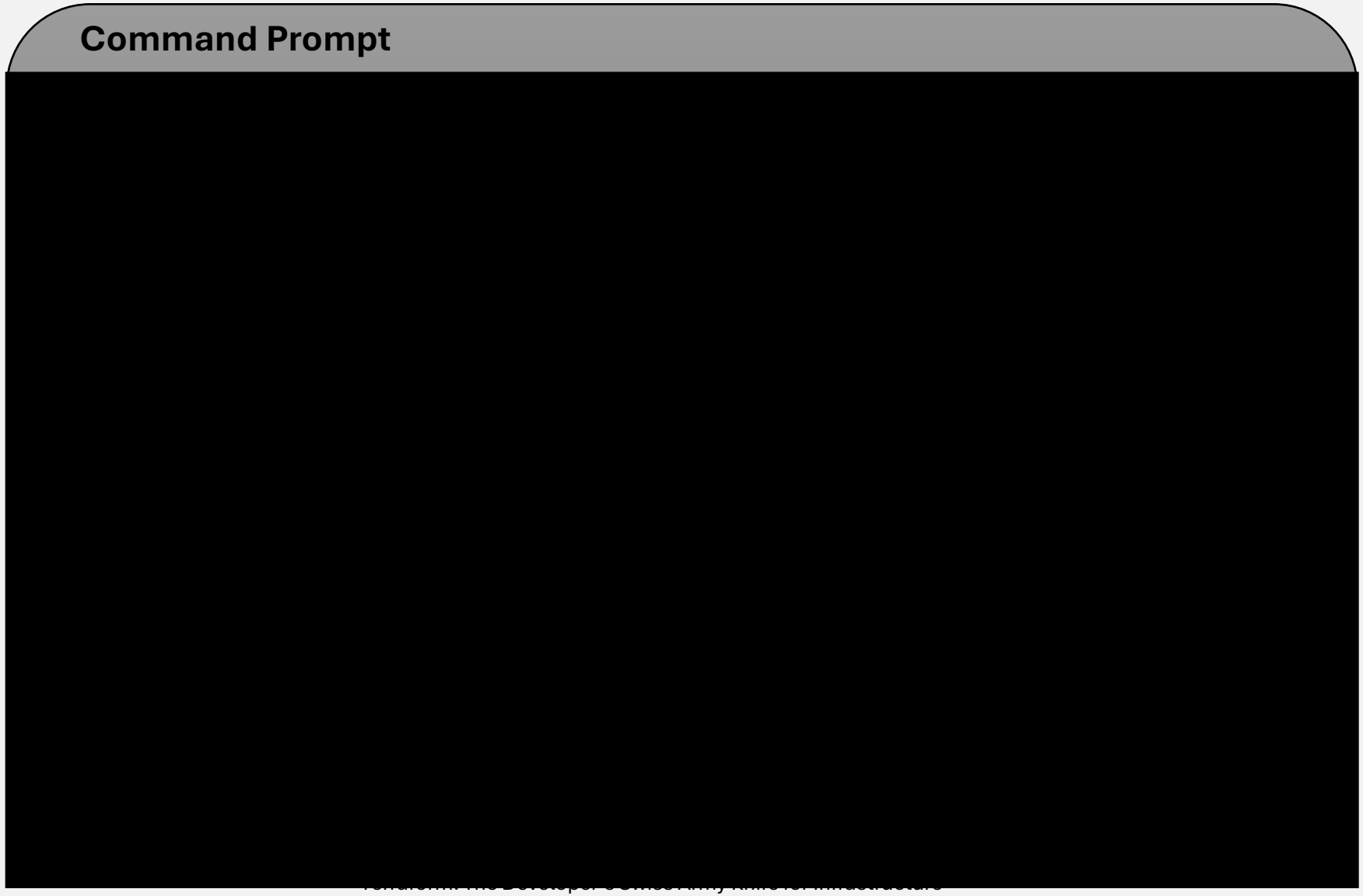
- main.tf
- variables.tf
- outputs.tf
- terraform.tfvars

Example Configuration

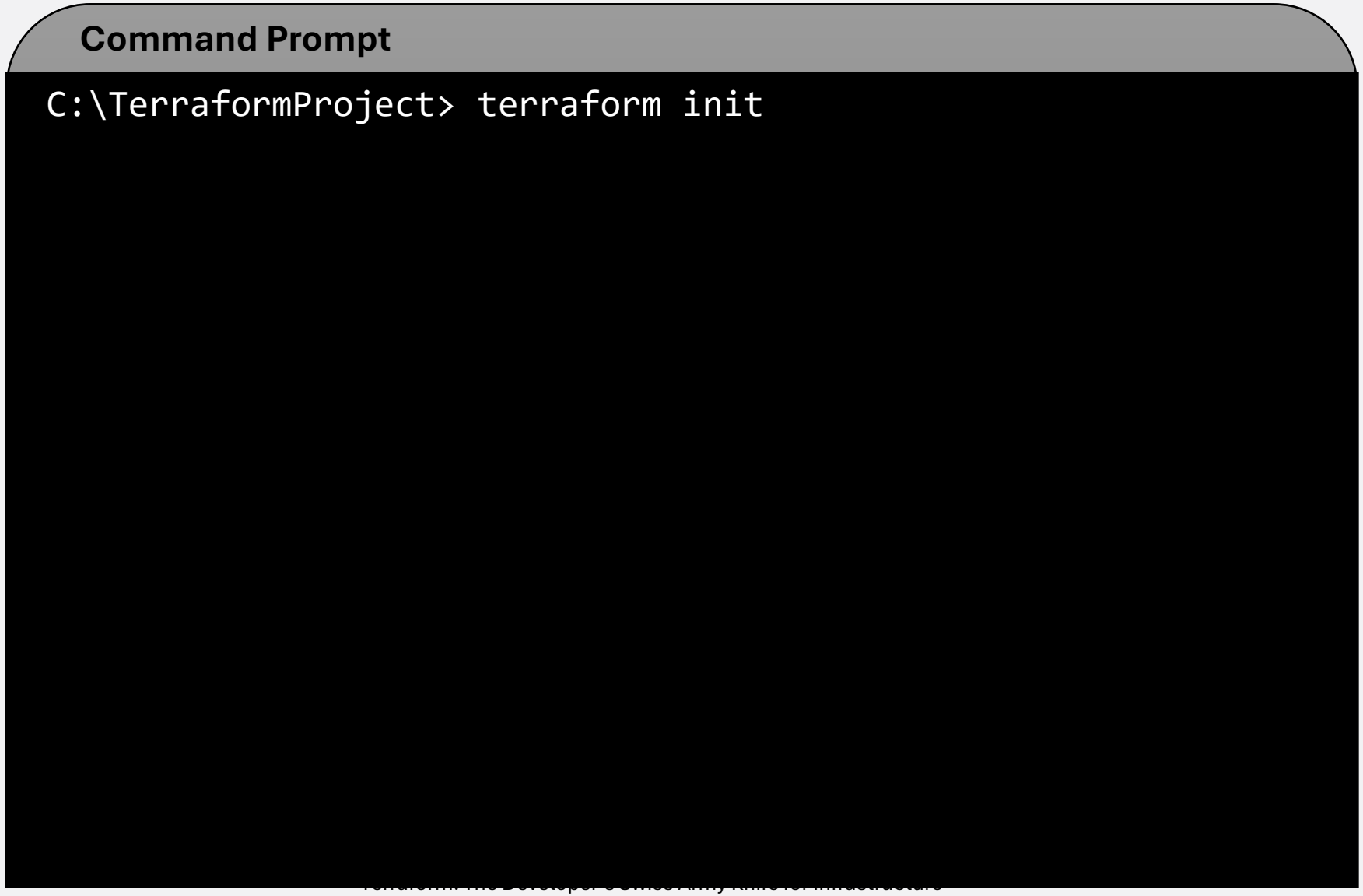
main.tf

```
provider "azurerm" {
  features {}
}

variable "location" {
  description = "The Azure region to deploy resources in"
  default     = “East US 2"
}


resource "azurerm_resource_group" "example" {
  name     = "example-resources"
  location = “East US 2"
}

output "resource_group_name" {
  value = azurerm_resource_group.example.name
}
```

Terraform CLI



Terraform CLI



Terraform CLI

Command Prompt

```
C:\TerraformProject> terraform init

C:\TerraformProject> terraform plan
```

Terraform CLI

Command Prompt

```
C:\TerraformProject> terraform init

C:\TerraformProject> terraform plan

C:\TerraformProject> terraform apply
```

Terraform CLI

Command Prompt

```
C:\TerraformProject> terraform init

C:\TerraformProject> terraform plan

C:\TerraformProject> terraform apply

C:\TerraformProject> terraform fmt
```

Terraform CLI

Command Prompt

```
C:\TerraformProject> terraform init

C:\TerraformProject> terraform plan

C:\TerraformProject> terraform apply

C:\TerraformProject> terraform fmt

C:\TerraformProject> terraform validate
```

Terraform CLI

Command Prompt

```
C:\TerraformProject> terraform init

C:\TerraformProject> terraform plan

C:\TerraformProject> terraform apply

C:\TerraformProject> terraform fmt

C:\TerraformProject> terraform validate
C:\TerraformProject> terraform show
```

Terraform CLI

Command Prompt

```
C:\TerraformProject> terraform init

C:\TerraformProject> terraform plan

C:\TerraformProject> terraform apply

C:\TerraformProject> terraform fmt

C:\TerraformProject> terraform validate
C:\TerraformProject> terraform show

C:\TerraformProject> terraform destroy
```

Live Demonstration

Best Practices

Best Practices

Version Control

Things to Keep in Mind

- Use Branches
- Commit Regularly
- Pull Requests/Merge Requests
- Tagging and Releases

Best Practices

Version Control

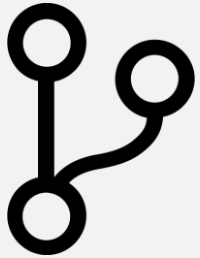
Things to Keep in Mind

- Use Branches
- Commit Regularly
- Pull Requests/Merge Requests
- Tagging and Releases

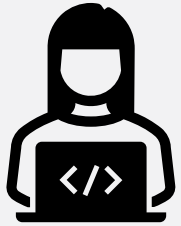
Best Practices

Version Control

Workflow

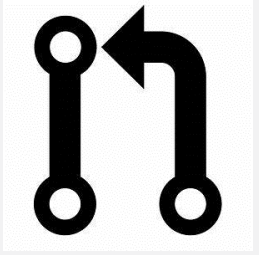


Create a Branch

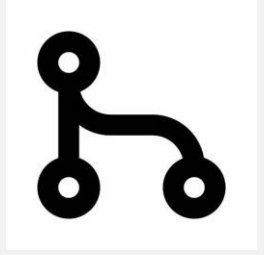


Make Changes

Commit Changes



Open Pull Request



Merge

Deploy

Best Practices

Version Control | **Modules**

Why use modules?

- Promote reusability of code.
- Improve Organization and manageability.
- Enable separation of concerns.

Best Practices

- Keep modules simple and focused.
- Document modules thoroughly.
- Version control your modules
- Use semantic versioning for module releases.

Best Practices

- Version Control
- Modules
- Organization

Best Practices

Version Control

Modules

File Name Conventions

- main.tf
- variables.tf
- outputs.tf
- provider.tf

Best Practices

Version Control | Modules | Organization

File Name Conventions

- main.tf
- variables.tf
- outputs.tf
- provider.tf

Modules

Best Practices

Version Control | Modules | Organization

File Name Conventions

- main.tf
- variables.tf
- outputs.tf
- provider.tf

Modules

Workspaces

Best Practices

Version Control | Modules | Organization

File Name Conventions

- main.tf
- variables.tf
- outputs.tf
- provider.tf

Modules

Workspaces

State Management

Best Practices

Version Control | Modules | Organization

File Name Conventions

- main.tf
- variables.tf
- outputs.tf
- provider.tf

Modules

Workspaces

State Management

Directory Structure

Common Issues

Frequent Challenges and Solutions

State Management Issues

Problem: Terraform state files becoming corrupted or inconsistent.

Solution: Use remote state storage with locking mechanisms.

```
terraform {
  backend "azurerm" {
    resource_group_name  = "rg_name"
    storage_account_name = "storage_name"
    container_name       = "container_name"
    key                  = "terraform.tfstate"
  }
}
```

Resource Deletion by Mistake

Problem: Resources accidentally deleted when running terraform apply.

Solution: Use the terraform plan command to review changes before applying. Implement resource protection by setting *prevent_destroy*.

```
resource "azurerm_resource_group" "example" {
  name     = "example-resources"
  location = "West Europe"

  lifecycle {
    prevent_destroy = true
  }
}
```

Module Versioning

Problem: Using incompatible or outdated module versions.

Solution: Specify module versions and use version constraints.

```
module "network" {
  source  = "terraform-aws-modules/vpc/aws"
  version = "~> 2.0"
  ...
}
```

Handling Sensitive Data

Problem: Exposing sensitive information in configuration files.

Solution: Use environment variables, secrets management services, and the sensitive attribute in Terraform.

```
variable "db_password" {
  description = "The password for the database"
  type        = string
  sensitive   = true
}
```

Networking Configuration Errors

Problem: Incorrectly configured networking resources leading to connectivity issues.

Solution: Validate and test networking configurations. Use tools like terraform validate and terraform plan to catch errors early.

```
terraform validate
terraform plan
```

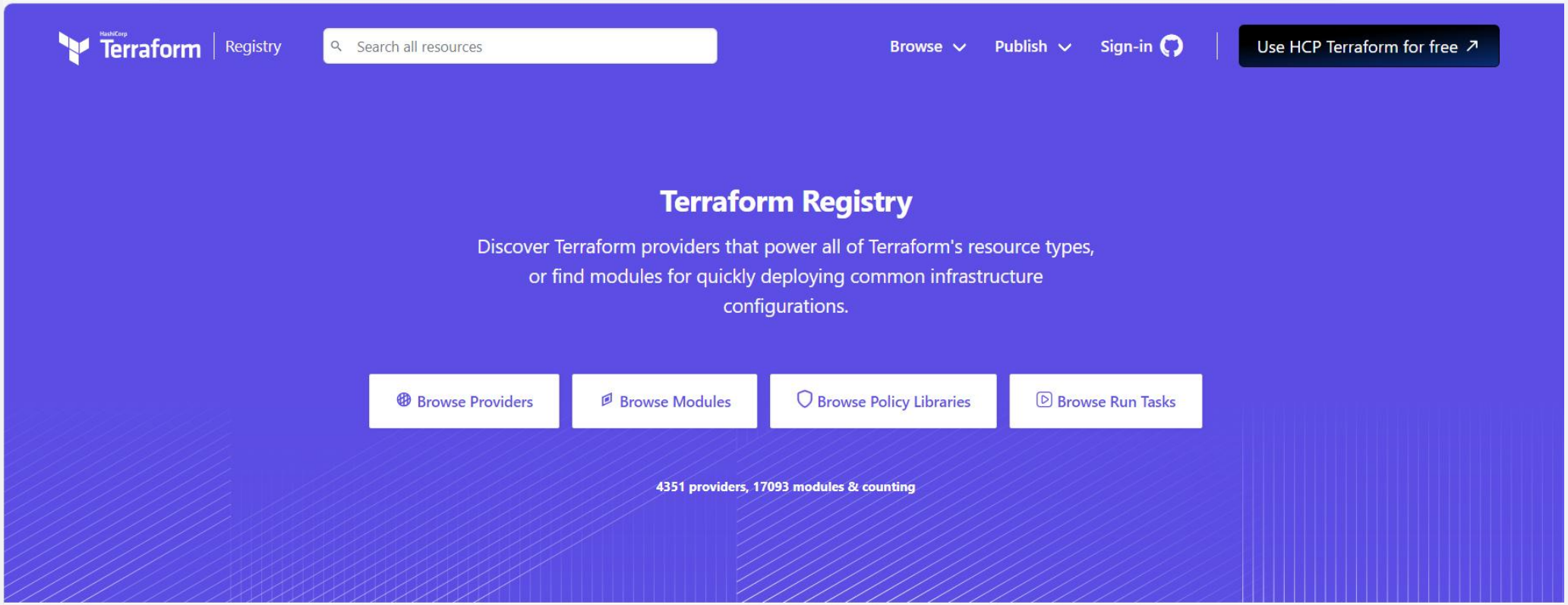
Lessons Learned

From Implementing Terraform

Get to know registry.terraform.io

Lesson: Utilize the Terraform Registry for pre-built modules and providers

Experience: Saved time and effort by reusing community-vetted modules and providers.



HCL is not a programming language

Lesson: Recognize that HCL is declarative, not imperative.

Experience: Improved clarity by focusing on defining the desired state of infrastructure rather than procedural logic.

HCL Does Have Programming Capabilities

Lesson: Utilize HCL's interpolation and conditionals for dynamic configurations.

Experience: Enhanced configurations with dynamic and reusable code patterns.

```
variable "enable_monitoring" {
  description = "Whether to enable monitoring"
  type        = bool
  default     = true
}

resource "azurerm_monitor_diagnostic_setting" "example" {
  count = var.enable_monitoring ? 1 : 0
  name  = "example-diagnostics"
  target_resource_id = azurerm_storage_account.example.id
}
```

Start Small and Iterate

Lesson: Begin with small, manageable configs and incrementally add complexity.

Experience: Avoided overwhelming initial setups and allowed for gradual learning and troubleshooting.

Embrace Remote State

Lesson: Store state files remotely to ensure consistency and enable team collaboration.

Experience: Prevented state file conflicts and allowed for reliable state management across the team.

Use Key Vault Secrets

Lesson: Store and manage sensitive information using Azure Key Vault

Experience: Enhanced security and simplified management of secrets.

```
variable "sensitive_value" {
  description = "A sensitive value, can be null"
  type        = string
  default     = null
}

data "azurerm_key_vault" "example" {
  name                = "your-key-vault-name"
  resource_group_name = "your-resource-group-name"
}

data "azurerm_key_vault_secret" "example_secret" {
  name         = "your-secret-name"
  key_vault_id = data.azurerm_key_vault.example.id
}

locals {
  value_to_use = coalesce(var.sensitive_value, data.azurerm_key_vault_secret.example_secret.value)
}

resource "azurerm_resource_group" "example" {
  name     = "example-resources"
  location = "West Europe"
  tags = {
    sensitive_tag = local.value_to_use
  }
}

output "value_to_use" {
  value = local.value_to_use
}
```

Modularize Your Configurations

Lesson: Use modules to encapsulate and reuse infrastructure components.

Experience: Simplified configurations, promoted reuse, and improved maintainability.

Document Everything

Lesson: Maintain thorough documentation of your Terraform configurations and processes.

Experience: Facilitated onboarding, troubleshooting, and knowledge sharing within the team.

Implement Code Reviews

Lesson: Conduct regular code reviews for Terraform configurations.

Experience: Caught errors early and improved the overall quality of the infrastructure code. Reduce the possibility of spinning up incorrectly configured expensive services.

Summary

Session Recap

Recap

- Infrastructure as Code
- Introduction to Terraform
- Live Demonstration
- Best Practices
- Common Issues
- Lessons Learned