Terraform on Azure
Delivering Infrastructure as Code

Daniel Neumann

Staff Software Engineer

April 25th 2020

Session objectives

LeanIX

- What is Infrastructure as Code?
- Terraform vs. Azure Resource Manager templates
- Terraform – from zero to hero

LeanIX

What is Infrastructure as Code?

What is Infrastructure as Code?

„Infrastructure is described using a high-level configuration syntax. This allows a blueprint of your datacenter to be versioned and treated as you would any other code. Additionally, infrastructure can be shared and re-used.“

Source: https://www.terraform.io/intro/index.html#infrastructure-as-code

Infrastructure as Code tools

Tool	Language / syntax	Cloud platform
Ansible	YAML	AWS, Azure, GCP, …
AWS CloudFormation	JSON or YAML	AWS
Azure Resource Manager templates	JSON	Azure
Google Cloud Deployment Manager	YAML (w/ Jinja2 or Python)	GCP
Pulumi	JavaScript, TypeScript, Python, Go and .NET Core	AWS, Azure or GCP
Terraform	HCL (HashiCorp Configuration Language) or JSON	AWS, Azure, GCP, …

LeanIX

Terraform vs. Azure Resource Manager templates

Azure Resource Manager templates

- Native Infrastructure as Code tooling for Azure
- Supports only Azure
- Does not store state
  - what-if functionality
  - https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/template-deploy-what-if
- Azure Deployment Manager adds additional capabilities

Terraform

- OSS 3rd party Infrastructure as Code tooling for Azure
- Supports multiple cloud platforms
- Stores state

Terraform vs.
Azure Resource Manager templates

	Terraform	ARM templates
Validate templates		
Plan deployments*		
Apply deployments		
Destroy deployments**		
State		
Modularization		

* what-if functionality for ARM templates
**Apply empty ARM template in `Complete` mode

LeanIX

Terraform – from zero to hero

Terraform – Basics: Providers

- Enable Terraform to interact with IaaS, PaaS or SaaS services
    - Official
        - https://www.terraform.io/docs/providers/index.html
    - Community
        - https://www.terraform.io/docs/providers/type/community-index.html
    - GitHub Repository
        - https://github.com/terraform-providers

Terraform – Basics: Elements

- Data sources
  - Allows fetching or computation of data

```
data "azuread_group" "aks" {
  name = var.aad_group_name
}
```

- Resources
  - Describes the objects or components like VMs, virtual networks or storage

```
resource "kubernetes_cluster_role_binding" "aks" {
  metadata {
    name = "aks-cluster-admins"
  }

  role_ref {
    api_group = "rbac.authorization.k8s.io"
    kind      = "ClusterRole"
    name      = "cluster-admin"
  }

  subject {
    api_group = "rbac.authorization.k8s.io"
    kind      = "Group"
    name      = data.azuread_group.aks.id
  }
}
```

Terraform – Basics: State

- Necessary requirement for Terraform
    - Mapping Terraform config to the real world
    - Metadata tracking like dependencies
    - Caching (`–refresh=false` or `–target`)
    - Syncing (Remote state / collaboration)

Terraform – Basics: Structure

- main.tf
  - Primary entrypoint
  - Contains data sources, locals, modules, providers and resources
- variables.tf
  - Contains variables
- output.tf
  - Contains output information like IP address, etc.

Terraform – Azure authentication

- Azure CLI

```
$ az login
```

```
provider "azurerm" {
  version >= "2.0.0"
}
```

Terraform – Azure authentication

- Azure Managed Identity

```
$ export ARM_USE_MSI=true
$ export ARM_SUBSCRIPTION_ID="00000000-0000-0000-0000-000000000000"
$ export ARM_TENANT_ID="00000000-0000-0000-0000-000000000000"
```

```
provider "azurerm" {
  version >= "2.0.0"
  use_msi = true
}
```

Terraform – Azure authentication

- Azure Service Principal

```
$ export ARM_CLIENT_ID="00000000-0000-0000-0000-000000000000"
$ export ARM_CLIENT_SECRET="00000000-0000-0000-0000-000000000000"
$ export ARM_SUBSCRIPTION_ID="00000000-0000-0000-0000-000000000000"
$ export ARM_TENANT_ID="00000000-0000-0000-0000-000000000000"
```

```
provider "azurerm" {
  version >= "2.0.0"
}
```

Terraform – State management

- Backends
  - local
  - remote → Terraform Cloud or Terraform Enterprise
  - azurerm → Azure Blob Storage
  - consul → HashiCorp Consul
  - etcdv3 → etcd
  - pg → Postgres database
  - gcs → Google Cloud Storage
  - s3 → Amazon S3
- Locking support

Terraform – Modules

- Container for multiple resources that are used together
- Re-usable in other configurations
- .tf files in the working directory are called root module
  - Every Terraform configuration is a module

Tips & tricks

LeanIX

- Avoid local-exec usage
- Avoid whitespaces when creating Azure resources
  - Fun part when running `terraform import`. Whitespaces must then be replaced with `%20`
- For cross subscription deployments you should provide an alias for the provider in the nested module you are using. Otherwise the override from the root module might not function properly.

Terragrunt

- Keeps your Terraform code DRY and maintainable
  - DRY – Don‘t Repeat Yourself
  - E.g. state backend configuration
- https://blog.gruntwork.io/terragrunt-how-to-keep-your-terraform-code-dry-and-maintainable-f61ae06959d8

LeanIX

Demo

LeanIX

Thanks