Deploying to Azure Using Terraform

Robert Rozas Navarro
Premier Field Engineer
Apps Domain

Microsoft

Agenda

High level Azure deployment methods

Template based deployments

Terraform based deployment

Methods for deploying stuff in Azure

Three over-arching methods.

- Azure Portal (manual)
- Scripts / SDKs (automation)
- Template based deployments (automation)

Methods for Azure resource deployments

Azure Portal (manual)

Pros:

- Browser based, quick setup, no fuss
- Nice for exploration and visual inspection
- Fully featured

Cons:

- Everything is performed manually
- Error prone
- Lack of process integration (DevOps, ITSM)

Methods for Azure resource deployments

Scripts / SDKs (automation)

Pros:

- Process integration (DevOps / ITSM)
- Removes human / less error prone
- Unopinionated / total flexibility

Cons:

- Requires scripting knowledge / environment
- Complex logic needs to be hand built

Methods for Azure resource deployments

Template based deployments (automation)

Pros:

- Process integration (DevOps / ITSM)
- Removes human / less error prone
- Handles some complex logic
- Options for state management

Cons:

- Requires templating knowledge / environment
- Opinionated and lack of full flexibility

Template based deployments

Digging deeper on template based deployments.

- Azure Resource Manager templates or Terraform
- Declaration of desired infrastructure
- JSON or JSON like syntax
- Deploy, update, delete

Azure Resource Manager Templates

What are Azure Resource Manager Templates?

- Written in JSON
- Tooling for Visual Studio and Visual Studio Code
- Native Azure portal integration
- Generated directly from REST / Swagger

Azure Resource Manager Template Example

```
{
    "$schema": "https://schema.management.azure.com/..json#",
    "contentVersion": "1.0.0.0",
    "parameters": {},
    "variables": {},
    "resources": [{
            "type": "Microsoft.Resources/resourceGroups",
            "apiVersion": "2018-05-01",
            "location": "eastus",
            "name": "demo-storage",
            "properties": {}
        },
        {
            "type": "Microsoft.Storage/storageAccounts",
            "name": "demo-storage",
            "apiVersion": "2018-02-01",
            "location": "eastus",
            "sku": {
                "name": "Standard_LRS"
            },
            "kind": "Storage",
            "properties": {}
        }
    ]
}
```

Resource Group

Storage Account

Terraform

What is Terraform?

- Open source project
- Cross computing environment templating language
- Provision, Update, and Delete resources
- Authored in HashiCorp Configuration Language (HCL) or JSON

Terraform Example

```
resource "azurerm_resource_group" "testrg" {
    name = "resourceGroupName"
    location = "westus"
}

resource "azurerm_storage_account" "testsa" {
    name = "storageaccountname"
    resource_group_name = "testrg"
    location = "westus"
    account_tier = "Standard"
    account_replication_type = "GRS"
}
```

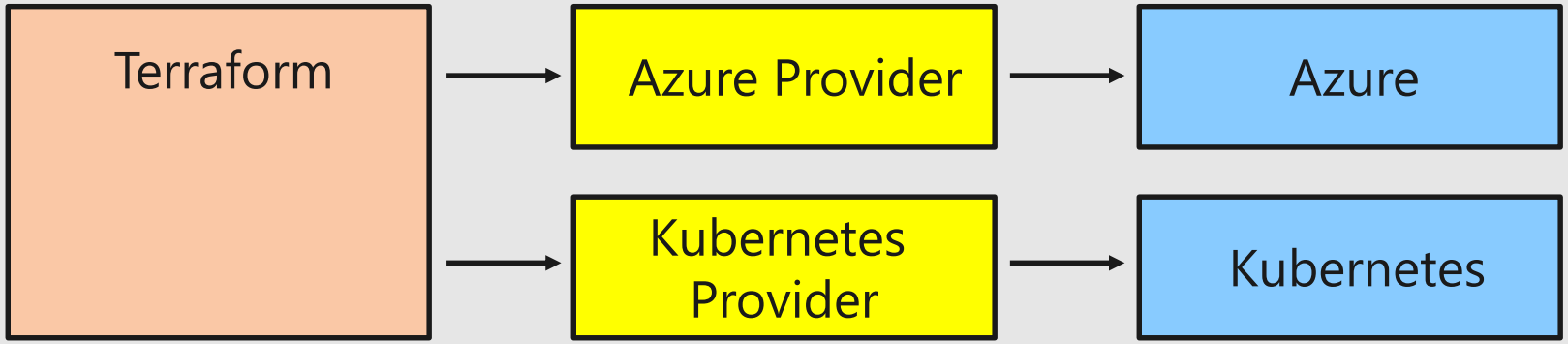
Resource Group

Storage Account

Providers

What is a Terraform provider?

- Terraform ‘extensions’ for deploying resources
- Manages cloud / endpoint specific API interactions
- Available for major clouds and other platforms
- Hand authored (azurerm)



Basic resource creation

Deployment foundations.

- Resource Type: required provider
- Name: internal name
- Configuration: deployment details

Resource Type

Name

```
resource "azurerm_resource_group" "demo-rg" {
    name = "demo-rg"
    location = "westus"
}
```

Resource Configuration

Basic Terraform commands

Once we have authored, how do we deploy?

- Terraform init – initializes working directory
- Terraform plan – pre-flight validation
- Terraform apply – deploys and updates resources
- Terraform destroy – removes all resources defined in a configuration

Variables and output

- Input variables: parameters for Terraform modules
- Environment variables: TF_VAR_azureclientid
- Output: Displayed and retrieved from state

```
$ TF_VAR_azureclientid = “00000000-0000-0000-0000-000000000000”

variable “azureclientid” {}
```

String Interpolation

Interpolation: the insertion of something of a different nature into something else.

- Variables
- Other resources
- Functions: ${count.index + 1}
- Others (Docs)

```
resource "azurerm_container_group" "demo-aci" {
    name = "demo-aci"
    location = "${azurerm_resource_group.demo-rg.location"
}
```

from resource

Dependencies

How are resource dependencies managed?

- Implicit – derived from interpolation
- Explicit – hard coded / explicit dependency

```
resource "azurerm_container_group" "demo-aci” {
    name = “demo-aci”

    depends_on = [“azure_cosmosdb_account.vote-db”]
}
```

State / Backend

What is Terraform state and why store it remotely?

Issues with local state:

- No collaboration
- Easy to delete / loose
- State files include secrets

Alternative:

- Store state in a backend (Azure Storage)

Data Sources

What is a Terraform data source?

- External data source for Terraform configuration
- Uses a provider just like in resource creation

Data Source Provider

Name

```
data "terraform_remote_state" "azurerm" {
    <configuration goes here>
}
```

```
"${data.terraform_remote_state.azurerm.resource-group}"
```

Automation and process integration

Once we are cooking, many opportunities for automation and process integration.

- Terraform Backends
- Environment variables
- GitHub
- Web Hooks
- Azure DevOps
- Etc.