

Infrastructure as Code: ARM vs Terraform

Matteo Tumiati

Senior DevOps Engineer & Microsoft MVP

matteot@aspitalia.com | @xtumiox

Morgan Pizzini

Senior Consultant @icubedsrl

morgan.pizzini@icubed.it | @morwalpiz

Container &
DevOps Day

Why infrastructure as code?

Follow the same software lifecycle best-practices

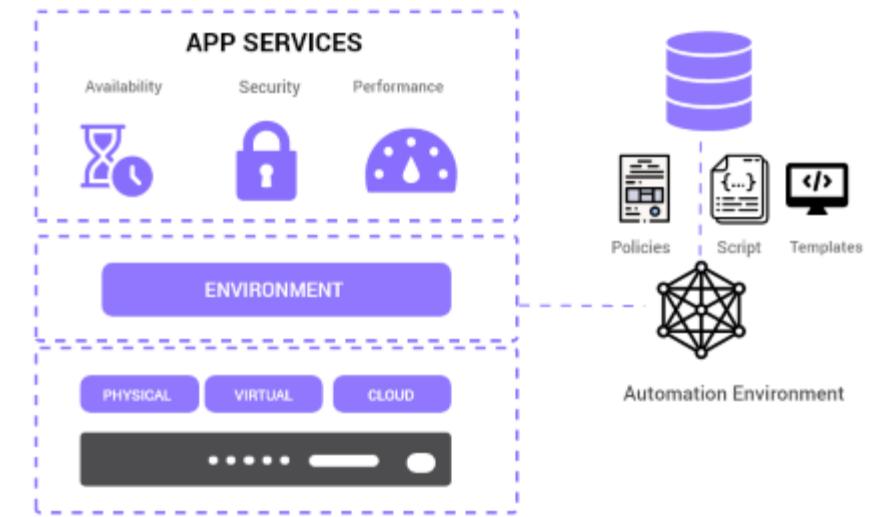
Automate (de)provisioning process

Faster and better deployments

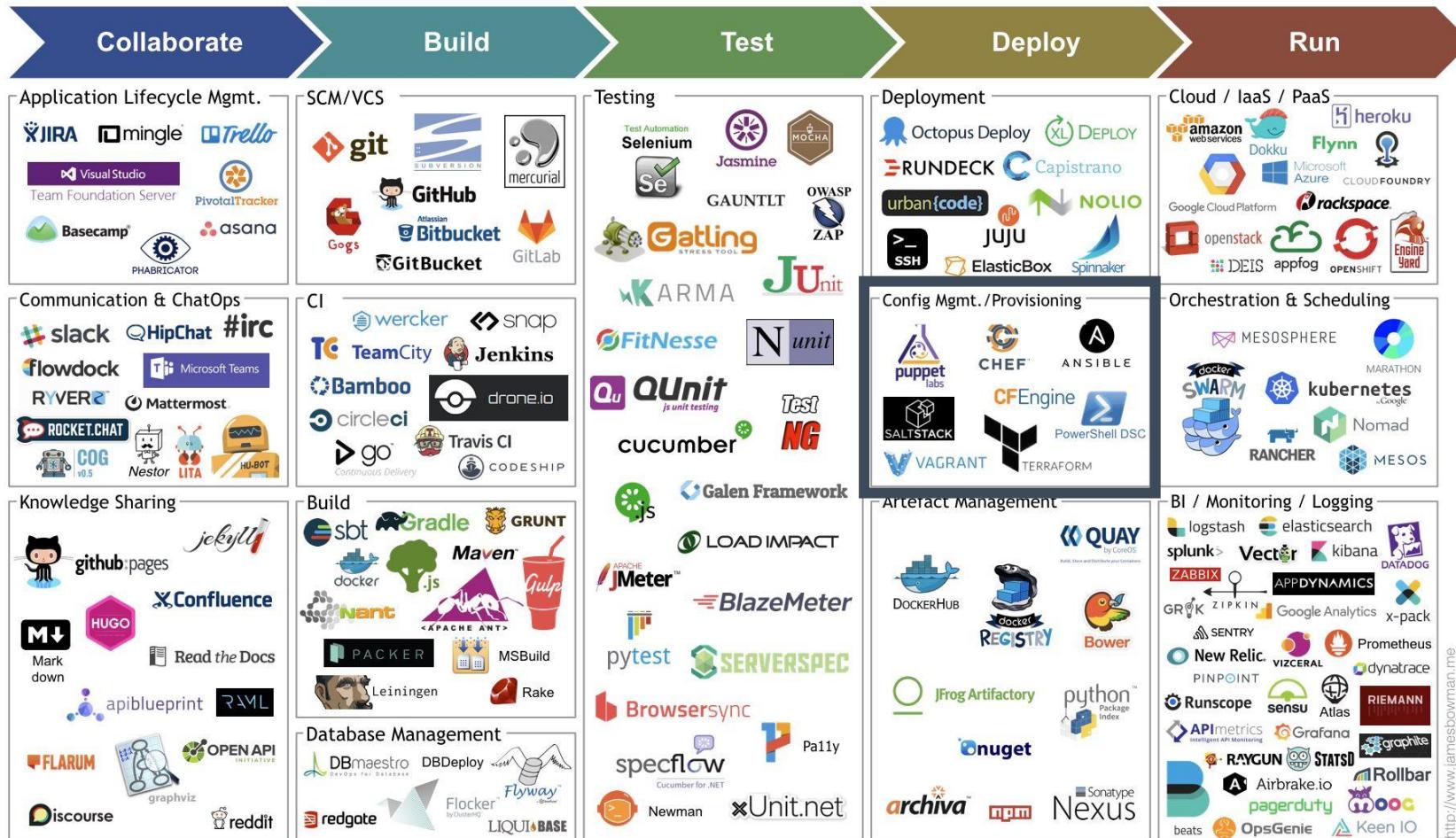
Reduce risks

Distribute and integrate with third party dependencies

Cost analysis and cost reduction



Landscape



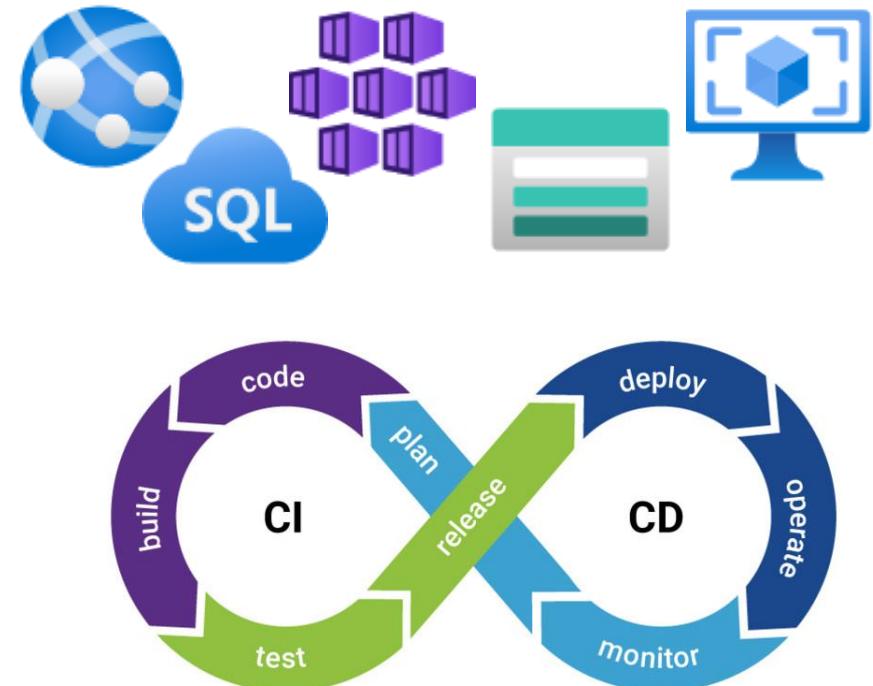
Why ARM? The old way

The image shows two screenshots of the Microsoft Azure portal. The left screenshot is titled 'Create Web App' and shows the 'Basics' tab selected. It includes fields for 'Subscription' (Visual Studio Enterprise - MPN), 'Resource Group' (Create new), 'Name' (Web App name), 'Publish' (Code selected), 'Runtime stack' (Select a runtime stack, Linux selected), 'Operating System' (Windows selected), and 'Region' (Central US). The right screenshot is the 'Dashboard' for a web app named 'devopsday'. It shows various management options like 'Overview', 'Activity log', and 'Tags'. Below these are sections for 'Essentials' (Resource group: delete, Status: Running, Location: West Europe, Subscription: Visual Studio Enterprise - MPN) and 'Diagnose and solve problems' (with a link to Application Insights). At the bottom are two charts: 'Http 5xx' and 'Data In'.

- Manual Process
- Config can change through time
- Can be deleted
- Not sustainable when applications raise
- BORING

Why ARM?

- Create as much resources as needed in seconds
- Keep configuration consistent through time
- Re-apply default configuration when needed
- Fits perfectly with CI/CD process
- 1 JSON and 1 command
- SOLID



ARM template

ARM = Azure Resource Manager

- JSON file that allow us to create resources on Azure

Minimum model requirements

- `$schema`: JSON location for template language
- `contentVersion`: template language API version
- `resources`: array of resource to be deployed

ARM template

```
[ "resources": [
    {
        "name": "storageaccount1",
        "type": "Microsoft.Storage/storageAccounts",
        "apiVersion": "2019-06-01",
        "location": "[resourceGroup().location]",
        "tags": {
            "displayName": "storageaccount1"
        },
        "kind": "StorageV2",
        "sku": {
            "name": "Premium_LRS",
            "tier": "Premium"
        }
    }
]]
```

The diagram illustrates the structure of an ARM template. A dashed box encloses the 'resources' array. Inside, a single resource object is shown. Four arrows point from the right side to specific fields: 'Name' points to 'name: "storageaccount1"', 'Type' points to 'type: "Microsoft.Storage/storageAccounts"', 'Resource group' points to 'location: "[resourceGroup().location]"', and 'SKU' points to 'sku: { "name": "Premium_LRS", "tier": "Premium" }'.

- Name

- Type

- Resource group

- SKU

ARM template

Parameters

Custom value provided during deployment

Variables

Variables used in resource definition

Function

User-defined functions

Outputs

Output of deployment

```
parameters": {  
    "location": {  
        "type": "string",  
        "defaultValue": "[resourceGroup().location]",  
        "metadata": {  
            "description": "The location resource."  
        }  
    }  
},  
variables": {  
    "appServicePlanName": "[concat(parameters('location'), '-sp')]"  
},  
functions": [{  
    "namespace": "sample",  
    "members": {  
        "concat": {  
            "parameters": [  
                {  
                    "name": "name",  
                    "type": "string"  
                },  
                {  
                    "output": {  
                        "type": "string",  
                        "value": "[concat('hello',toLower(parameters('name')))]"  
                    }  
                }  
            ]  
        }  
    }  
}]
```

Demo

Work with ARM Templates

Container &
DevOps Day

What is Terraform?

OSS created in 2014 by HashiCorp (same as Vault, Vagrant, Consul and others)

Written in GO

Like ARM templates, you can build, change, version infrastructure through config files

Why Terraform?

Using providers, you can deploy infrastructure and much more

- Manage Azure DevOps, GitHub or any other service
- Create blog posts on Medium [joatmon08/terraform-provider-medium \(github.com\)](https://joatmon08/terraform-provider-medium.github.com)

Ensures creation and consistency of resources

Uses an API-agnostic DSL

Multi-cloud/provider support

Benefits in using Terraform

Reliability and consistency through template validation

Changes are applied incrementally

Infrastructure is stored in a state file (with locking)

Preview changes before deployment

Scaling, locking and other common scenarios are easy to implement

Can work with any other configuration management tool

Can integrate third party scripts

Good integration (w/ IntelliSense) in VSCode

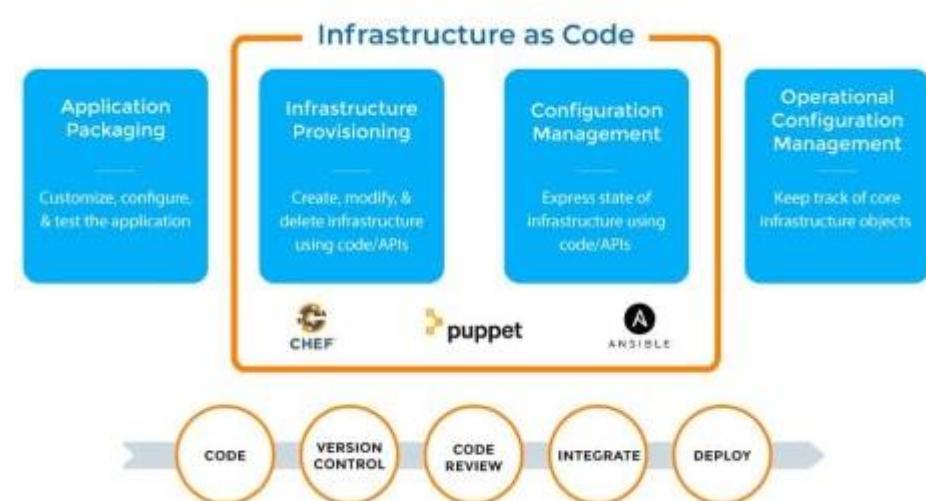
Different ways of doing IaC

Chef, Puppet, Ansible and others are configuration management tools

- Used to configure infrastructure
- Mutable infrastructure
- Procedural deployment => configuration drift
- Require server to host configuration

ARM and Terraform are provisioning tools

- Used to create/manage/destroy infrastructure
- Immutable infrastructure
- Declarative deployment
- Creates a state file



Project structure

There's not any required structure, but one was *commonly* established
All the infrastructure can be defined in a single "test.tf" file

```
-- README.md
-- main.tf
-- variables.tf
-- outputs.tf
-- ...
-- modules/
  |-- moduleA/
  |  |-- README.md
  |  |-- variables.tf
  |  |-- main.tf
  |  |-- outputs.tf
  |-- moduleB/
  |-- .../
```

main contains all the resources that will be deployed
variables defines all the input variables used to deploy components defined in *main*
outputs defined all the output variables created post-deployment

Modules are used to "organize" infrastructure code in components
Structure and content of each module depends on the context

Let's start from the language

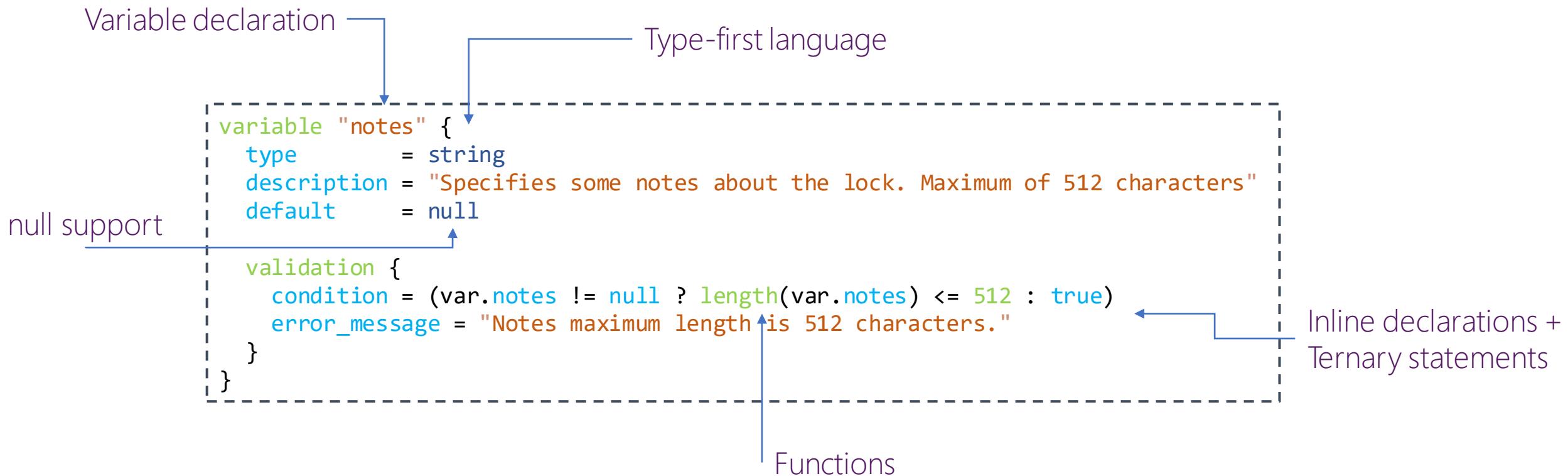
```
{  
    "$schema": "https://schema.management.azure.com/schemas/2015-  
01-01/deploymentTemplate.json#",  
    "contentVersion": "1.0.0.0",  
    "parameters": {  
        "storageAccountType": {  
            "type": "string",  
            "defaultValue": "Standard_GRS"  
        },  
        "storageAccountName": {  
            "type": "string",  
            "defaultValue": "myRG"  
        },  
        "location": {  
            "type": "string",  
            "defaultValue": "West Europe"  
        }  
    },  
    "resources": [  
        {  
            "type": "Microsoft.Storage/storageAccounts",  
            "apiVersion": "2019-06-01",  
            "name": "[parameters('storageAccountName')]",  
            "location": "[parameters('location')]",  
            "sku": {  
                "name": "[parameters('storageAccountType')]"  
            },  
            "kind": "StorageV2",  
            "properties": {  
                "supportsHttpsTrafficOnly": true  
                "accessTier": "Hot"  
            }  
        }  
    ]  
}
```

```
resource "azurerm_resource_group" "ASPItaliaRG" {  
    name      = "myRG"  
    location  = "West Europe"  
}  
  
resource "azurerm_storage_account" "storage" {  
    name          = "myStorageAccount"  
    resource_group_name = azurerm_resource_group.ASPItaliaRG.location  
    account_tier   = "Standard"  
}
```

Terraform

ARM

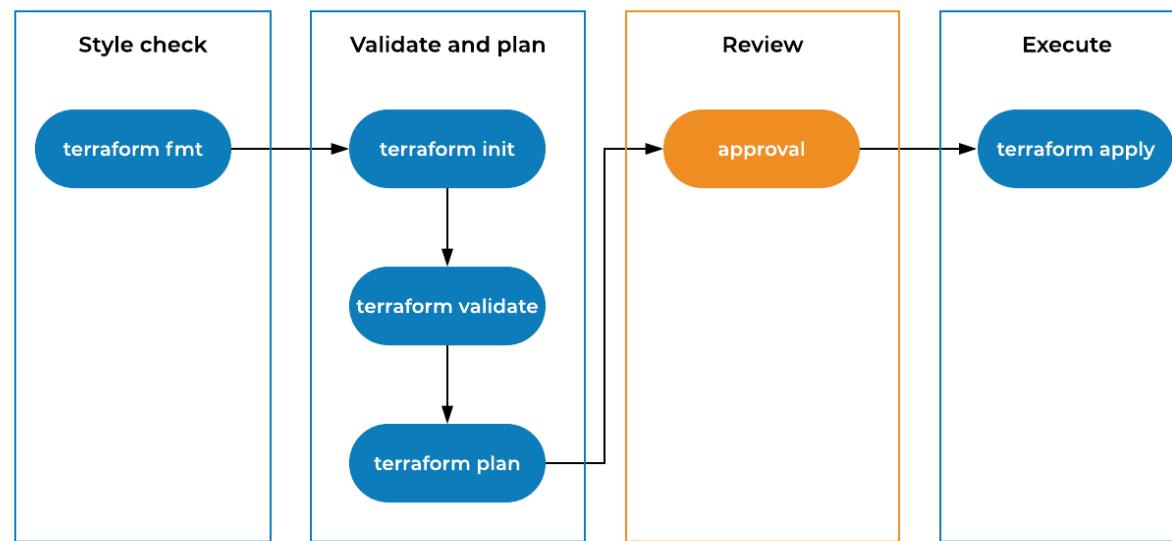
Template validation



Development process

Everything is done via the CLI

<https://www.terraform.io/downloads.html>



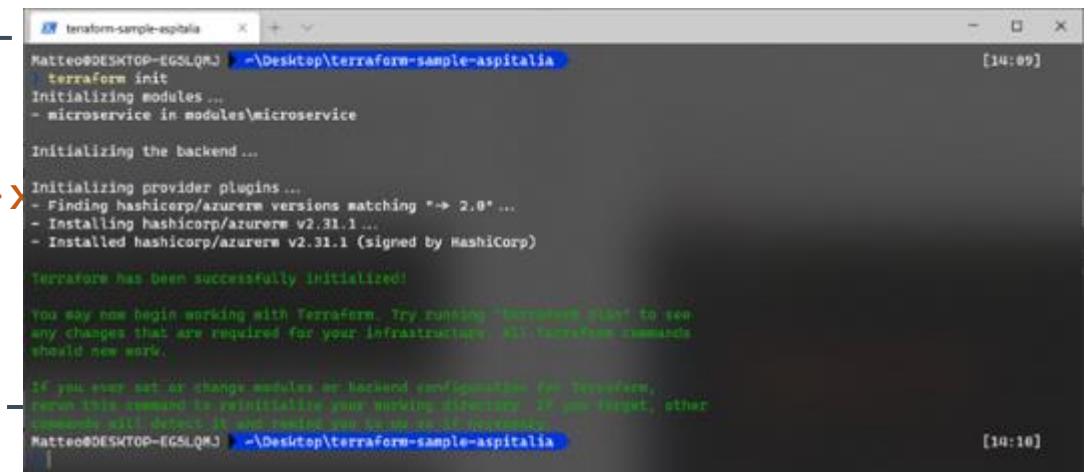
terraform init

Initializes a working directory containing Terraform configuration files

The chosen backend is initialized using the given configuration settings

Searches the configuration for both direct and indirect references to providers and attempts to install the plugins for those providers

```
provider "azurerm" {
  version = "=2.41.0"
  subscription_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxx-xxxx"
  skip_provider_registration = true
  features {}
}
```



A screenshot of a Windows Command Prompt window titled "terraform-sample-aspitalia". The window shows the output of the "terraform init" command. It starts with "Initializing modules ...", then "Initializing the backend ...", followed by "Initializing provider plugins ...". It lists the installation of "hashicorp/azurerm v2.31.1" from HashiCorp. Finally, it displays the message "Terraform has been successfully initialized!" and instructions for running "terraform plan". The terminal window has a dark theme and is located on a desktop with a blurred background.

terraform validate & plan

Validate checks the configuration files in a directory (syntactically valid and internally consistent), referring only to the configuration and not accessing any remote services such as remote state, provider APIs, etc.

Plan is used to create an execution plan. Terraform determines what actions are necessary to achieve the desired state specified in the configuration files

- It's a convenient way to check whether the execution plan for a set of changes matches expectations without making any changes to real resources or to the state



The screenshot shows a terminal window titled "terraform-sample-apply" with the following content:

```
[MatteoDESKTOP-EGLQMJ] ->\Desktop\terraform-sample-aspitalia
Terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
+ create

Terraform will perform the following actions:

# azurerm_resource_group.ASPitaliaRG will be created
+ resource "azurerm_resource_group" "ASpitaliaRG" {
    + id          = (known after apply)
    + location    = "westeurope"
    + name        = "ASpitaliaRG-SampleTerraform"
}

# module.microservice.azurerm_app_service.app_service will be created
+ resource "azurerm_app_service" "app_service" {
    + app_service_plan_id      = (known after apply)
    + app_settings              = {
        + "SOME_KEY" = "some-value"
    }
    + affinity_type            = (known after apply)
    + auto_heal                = (known after apply)
    + cors {
        + allowed_origins     = (known after apply)
        + support_credentials = (known after apply)
    }
    + https_port               = (known after apply)
    + kind                      = (known after apply)
    + min_tls_version          = (known after apply)
    + php_version               = (known after apply)
    + python_version            = (known after apply)
    + remote_debugging_enabled = (known after apply)
    + remote_debugging_version = (known after apply)
    + scm_ip_restriction        = (known after apply)
    + scm_type                  = (known after apply)
    + scm_use_main_ip_restriction = (known after apply)
    + use_32_bit_worker_process = (known after apply)
    + websockets_disabled       = (known after apply)
    + windows_fx_version        = (known after apply)
}

Plan: 0 to add, 0 to change, 0 to destroy.

Note: You didn't specify an "-out" parameter to save this plan, so Terraform
can't guarantee that exactly these actions will be performed if
"terraform apply" is subsequently run.

[14:13]
```

terraform apply

Applies the changes required to reach the desired state of the configuration, or the pre-determined set of actions generated by a terraform plan execution plan

Requires explicit approval (can be skipped in CI/CD with `--auto-approve`)

Will NOT apply locking to the state file (locally), unless specified with `--lock=true`

Will enable max supported parallelism to complete the deployment as fast as possible, but can be limited via `--parallelism=n`

```
Matteo@DESKTOP-EG5LQMJ ~\Desktop\terraform-sample-aspitelia
terraform apply
azurerm_resource_group.ASPItaliaRG: Creating ...
azurerm_resource_group.ASPItaliaRG: creation complete after 1s [id=/subscriptions/[REDACTED]/resourceGroups/ASPItaliaRG-SampleTerraform]
#module.microservice.azurerm_app_service_plan.appservice_plan: Creating ...
#module.microservice.azurerm_app_service_plan.appservice_plan: Still creating ... [10s elapsed]
#module.microservice.azurerm_app_service_plan.appservice_plan: Creation complete after 11s [id=/subscriptions/65f-4664-9ead-31a3b977b178/resourceGroups/ASPItaliaRG-SampleTerraform/providers/Microsoft.Web/serverFarms/ASPItaliaASP]
#module.microservice.azurerm_app_service.app_service: Creating ...
#module.microservice.azurerm_app_service.app_service: Still creating ... [10s elapsed]
#module.microservice.azurerm_app_service.app_service: Still creating ... [20s elapsed]
#module.microservice.azurerm_app_service.app_service: Still creating ... [30s elapsed]
#module.microservice.azurerm_app_service.app_service: Creation complete after 35s [id=/subscriptions/[REDACTED]/resourceGroups/ASPItaliaRG-SampleTerraform/providers/Microsoft.Web/sites/ASPItaliaWithTerraform]
#module.microservice.azurerm_app_service_slot.app_service_slot: Creating ...
#module.microservice.azurerm_app_service_slot.app_service_slot: Still creating ... [10s elapsed]
#module.microservice.azurerm_app_service_slot.app_service_slot: Still creating ... [20s elapsed]
#module.microservice.azurerm_app_service_slot.app_service_slot: Creation complete after 25s [id=/subscriptions/[REDACTED]/resourceGroups/ASPItaliaRG-SampleTerraform/providers/Microsoft.Web/sites/ASPItaliaRM/slots/staging]

Apply complete! Resources: 0 added, 4 changed, 0 destroyed.
Matteo@DESKTOP-EG5LQMJ ~\Desktop\terraform-sample-aspitelia
```

Demo

Deploy resources with Terraform

Container &
DevOps Day

What's next?

Project Bicep!

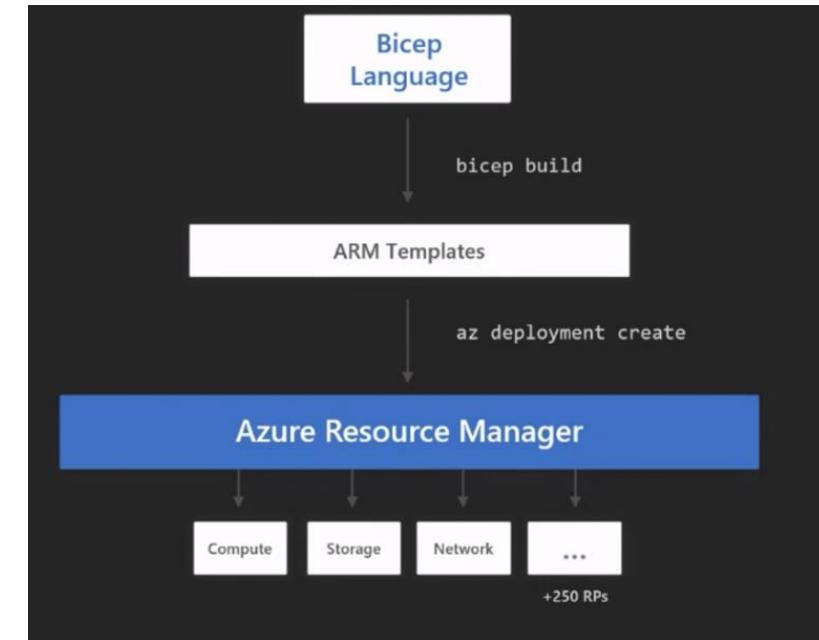
Currently it's an experimental language with the aim of becoming the next generation of ARM Templates

Bicep is a transparent abstraction over ARM and ARM templates

Bicep compiles down to standard ARM Template JSON files, which means the ARM JSON is effectively being treated as an Intermediate Language (IL)

It compiles, but also decompiling works

Solves many issue with linking related resources



Project Bicep

```
resource "azurerm_storage_account" "storage" {
    name                = "myStorageAccount"
    resource_group_name = azurerm_resource_group.ASPItaliaRG.location
    account_tier        = "Standard"
}
```

Terraform

```
resource storage 'Microsoft.Storage/storageAccounts@2019-06-01' = {
    name          : "myStorageAccount"
    resource_group_name : ASPItaliaRG.location
    sku : {
        name: 'Standard'
    }
}
```

Bicep

Cross-reference resources

bicep build

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": { ... },
    "resources": [
        {
            "type": "Microsoft.Storage/storageAccounts",
            "apiVersion": "2019-06-01",
            "name": "[parameters('storageAccountName')]",
            "location": "[parameters('location')]",
            "sku": {
                "name": "[parameters('storageAccountType')]"
            },
            "kind": "StorageV2",
            "properties": {
                "supportHttpsTrafficOnly": true
                "accessTier": "Hot"
            }
        }
    ]
}
```

```
resource web 'Microsoft.Web/sites@2020-06-01' = {
    name          : "myWebSite"
    properties : {
        siteConfig: {
            appSettings: [
                {
                    name: 'PrimaryConnectionString'
                    value: '${storage.properties.primaryEndpoints.blob}'
                }
            ]
        }
    }
}
```

@xtumiox
matteot@aspitalia.com

@morwalpiz
morgan.pizzini@icubed.it

Slide e materiali su
<https://aspit.co/ContainerDevOpsDay-21>

Container &
DevOps Day