Microsoft
tech·days
Helsinki 2019

HashiCorp
Terraform

Peter De Tender
CEO & Lead Technical Trainer – PDTIT | 007FFFLearning

@pdtit @007FFFLearning info@007FFFLearning.com

# Tervetuloa Helsinkiin

Welcome to TechDays 2019 - Helsinki

## Azure Loves Terraform Loves Azure

Intro to Infrastructure as Code

Settings the scene

for TerraForm

Terraform

Loves Demos

Q & A

## About Your Trainer

- **CEO** and **Lead Technical Trainer** at PDTIT & 007FFFLearning
- **Microsoft Cloud Architect & Trainer**
- Microsoft Certified Trainer – **MCT**
- Microsoft Learning Regional Lead
- Microsoft **Azure MVP** (2013-2019)
- Ex-Microsoft Corp Azure Engineering PM

- Book author for Packt Publishing & Apress
- Microsoft Learning Courseware Author and Trainer
- Technical Writer, Content Creator

- **peter@pdtit.be** **info@007FFFLearning.com**
- **@pdtit** **@007FFFLearning**

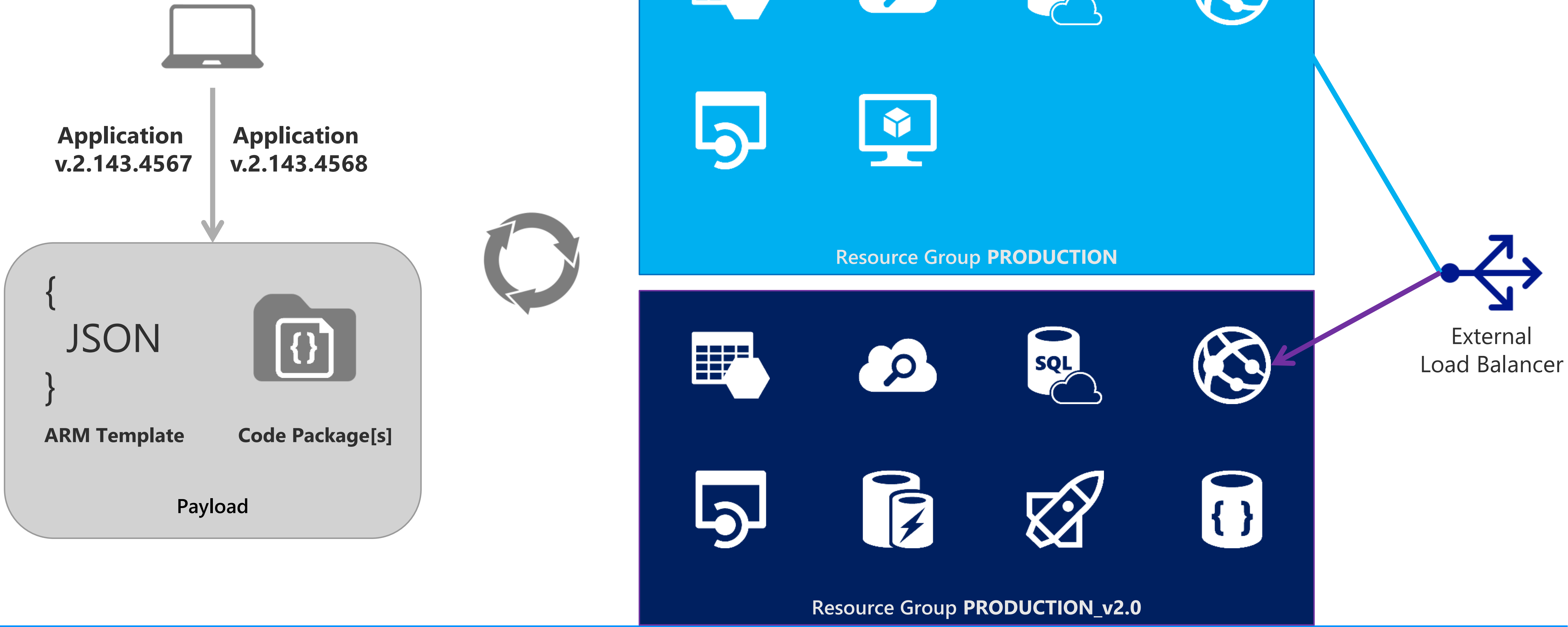
# “Lights Out”, Immutable Deployments

- **Lights Out” Deployment**
  - Deploy your application without need for manual intervention
  - Production application should not have access endpoints including:
    - SSH
    - RDP
    - Remote PowerShell
    - Etc.

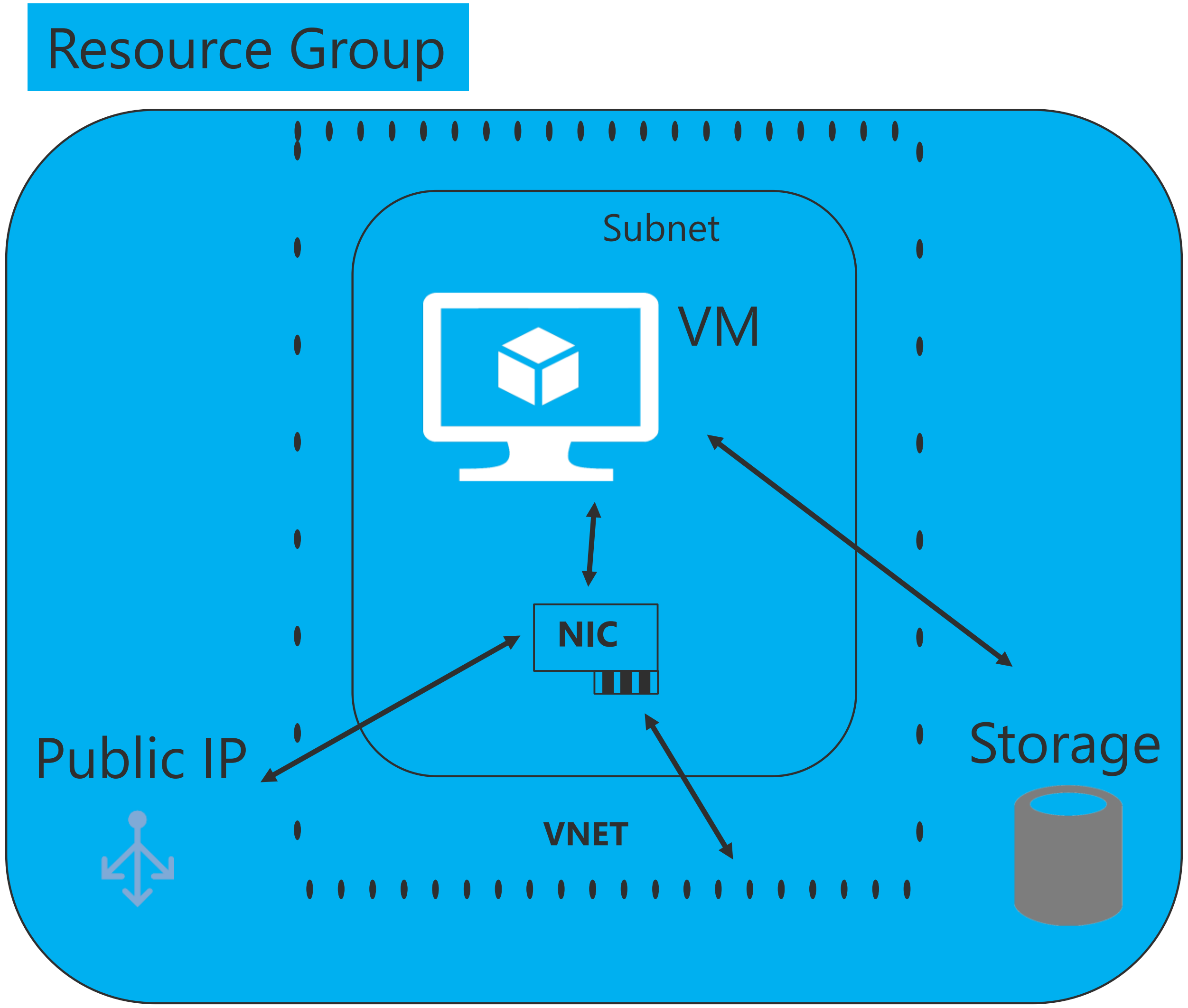
- **Immutable Deployment**
  - You should be able to deploy your environment unlimited times without side-effects
  - Environment deployment should be identical each time
-

# “Lights Out”, Immutable Deployments

- Using ARM



# Resource Manager: Building a VM



**virtualMachine**

- hardwareProfile
- osProfile
- storageProfile
- networkProfile

**networkInterface**

- privateIPAllocation Method

**publicIPAddress**

- allocationMethod
- domainNameLabel

**storageAccount**

- accountType

**virtualNetwork**

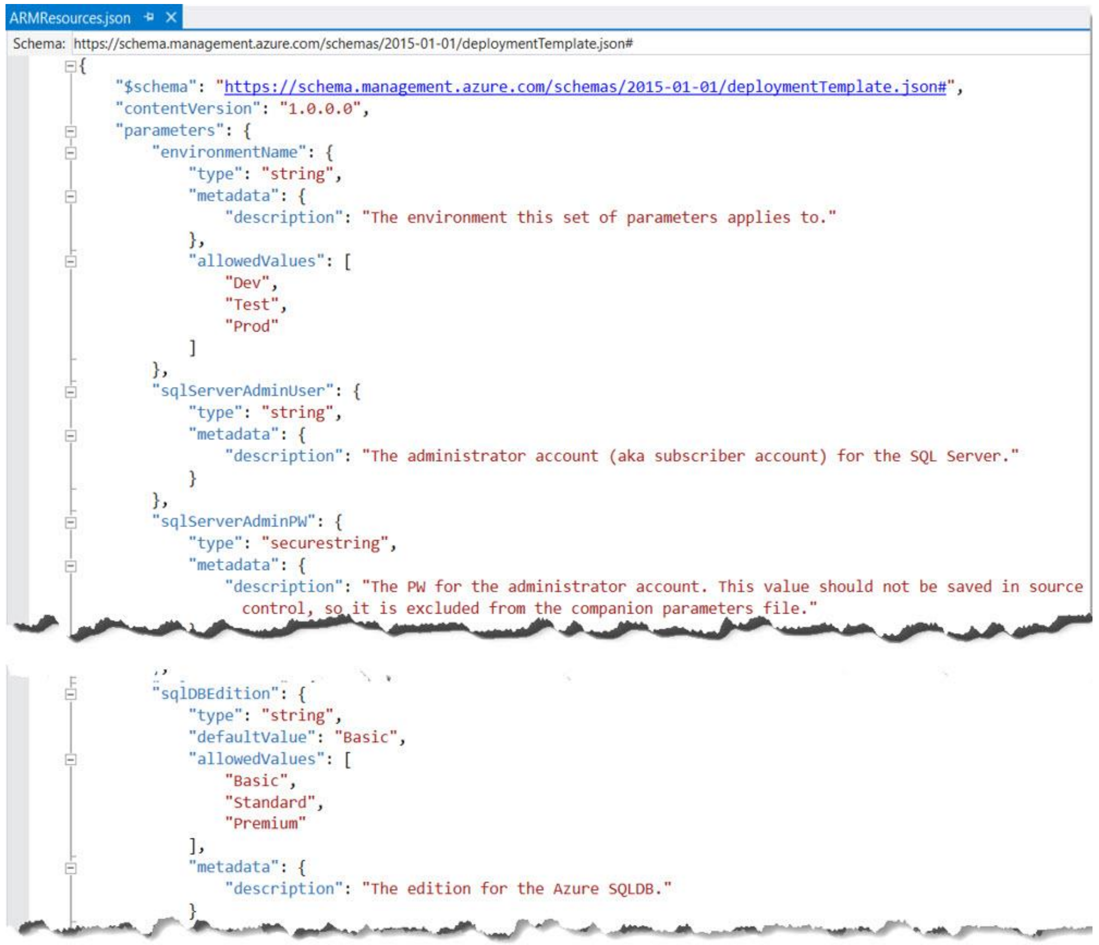
- addressSpace
- Subnet
  - addressPrefix

## Ways to create an ARM Template

1. From the automation script available from the Azure Portal (which is imperfect – more on that in a moment), or
2. An ARM template in Visual Studio, or
3. QuickStart Templates from GitHub (there's a lot to choose from – the templates that start with 101 are less complex), or
4. Create from the ground up, or
5. Start with a combination of 1 and 2 or 3, customizing the way you like it

# Basic Template Structure: Parameters

- This is where you store the definitions of all parameters used throughout the ARM template
- **Type**
  - Most common types are string or int;
  - Password = securestring;
- **MetaData / Description**
  - Helpful info to remember what it's used for, or any other notes you want to leave for yourself and your team.

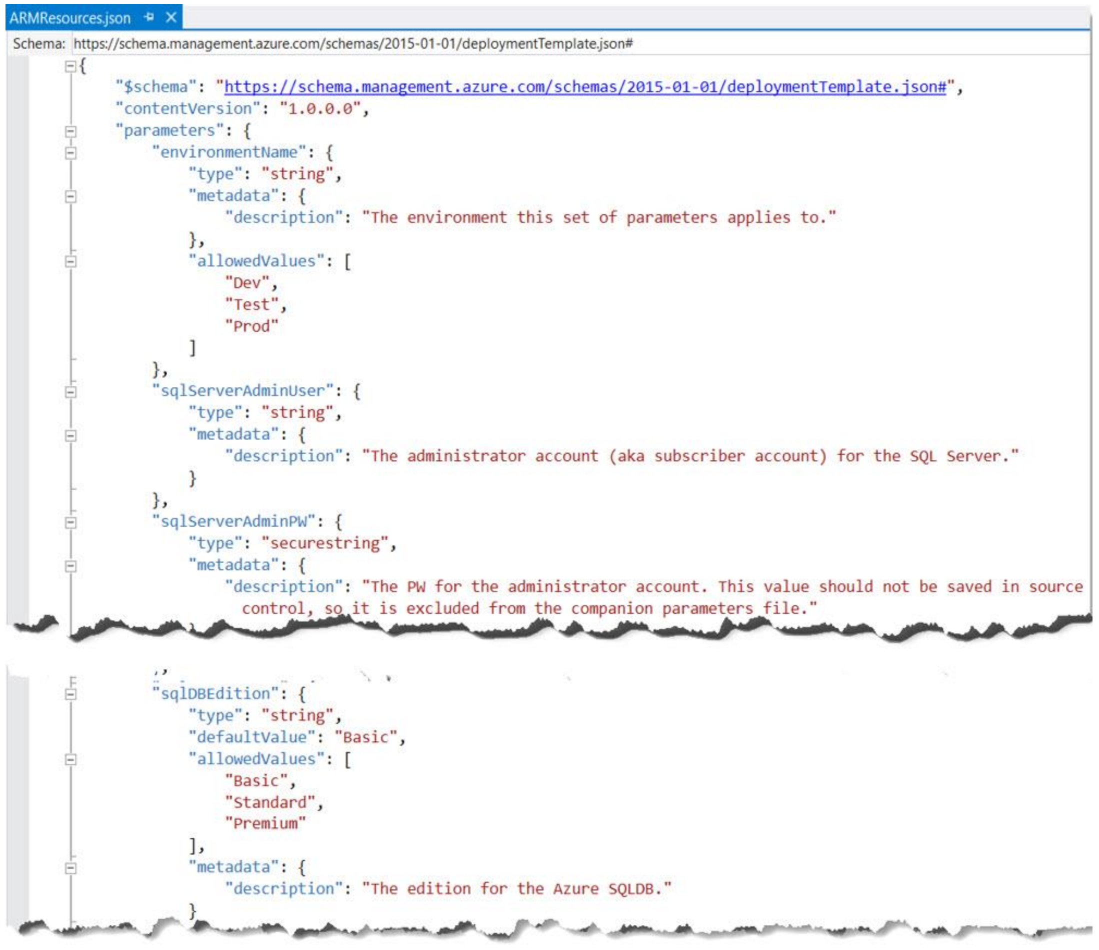

ARMResources.json
Schema: https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "environmentName": {
            "type": "string",
            "metadata": {
                "description": "The environment this set of parameters applies to."
            },
            "allowedValues": [
                "Dev",
                "Test",
                "Prod"
            ]
        },
        "sqlServerAdminUser": {
            "type": "string",
            "metadata": {
                "description": "The administrator account (aka subscriber account) for the SQL Server."
            }
        },
        "sqlServerAdminPW": {
            "type": "securestring",
            "metadata": {
                "description": "The PW for the administrator account. This value should not be saved in source
                  control, so it is excluded from the companion parameters file."
```

```
        "sqlDBEdition": {
            "type": "string",
            "defaultValue": "Basic",
            "allowedValues": [
                "Basic",
                "Standard",
                "Premium"
            ],
            "metadata": {
                "description": "The edition for the Azure SQLDB."
            }
```

## Basic Template Structure: Parameters

- This is where you store the definitions of all parameters used throughout the ARM template
- **Allowed Values**
  - If there are only certain options which will work successfuly (such as the VM Size), this is really helpful.
- **Default Value**
  - Makes sense in certain circumstances. If there's not a default value, then the value needs to be in the separate parameters file, or typed in at deployment time.



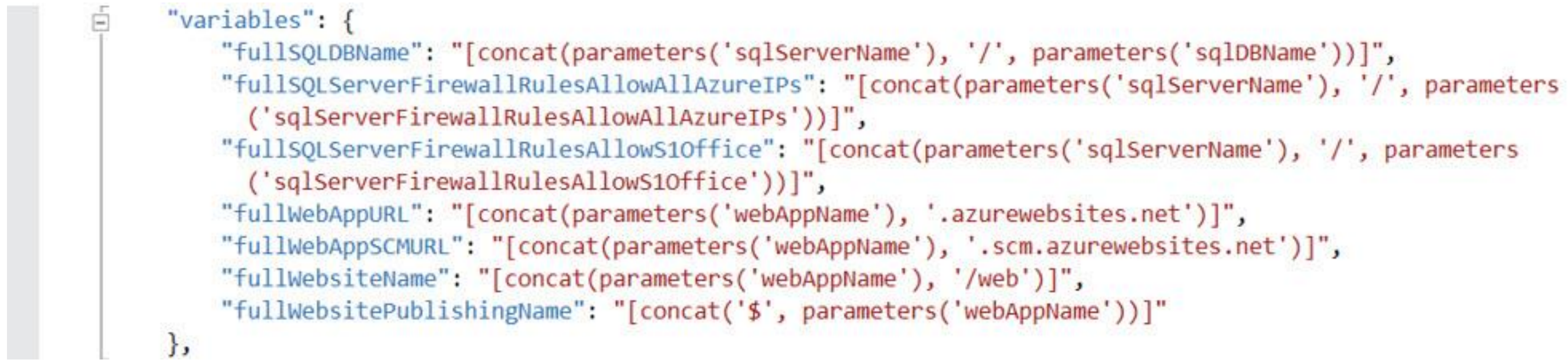
Schema: https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "environmentName": {
            "type": "string",
            "metadata": {
                "description": "The environment this set of parameters applies to."
            },
            "allowedValues": [
                "Dev",
                "Test",
                "Prod"
            ]
        },
        "sqlServerAdminUser": {
            "type": "string",
            "metadata": {
                "description": "The administrator account (aka subscriber account) for the SQL Server."
            }
        },
        "sqlServerAdminPW": {
            "type": "securestring",
            "metadata": {
                "description": "The PW for the administrator account. This value should not be saved in source
                 control, so it is excluded from the companion parameters file."
```

```
        "sqlDBEdition": {
            "type": "string",
            "defaultValue": "Basic",
            "allowedValues": [
                "Basic",
                "Standard",
                "Premium"
            ],
            "metadata": {
                "description": "The edition for the Azure SQLDB."
            }
```

## Basic Template Structure: Variables

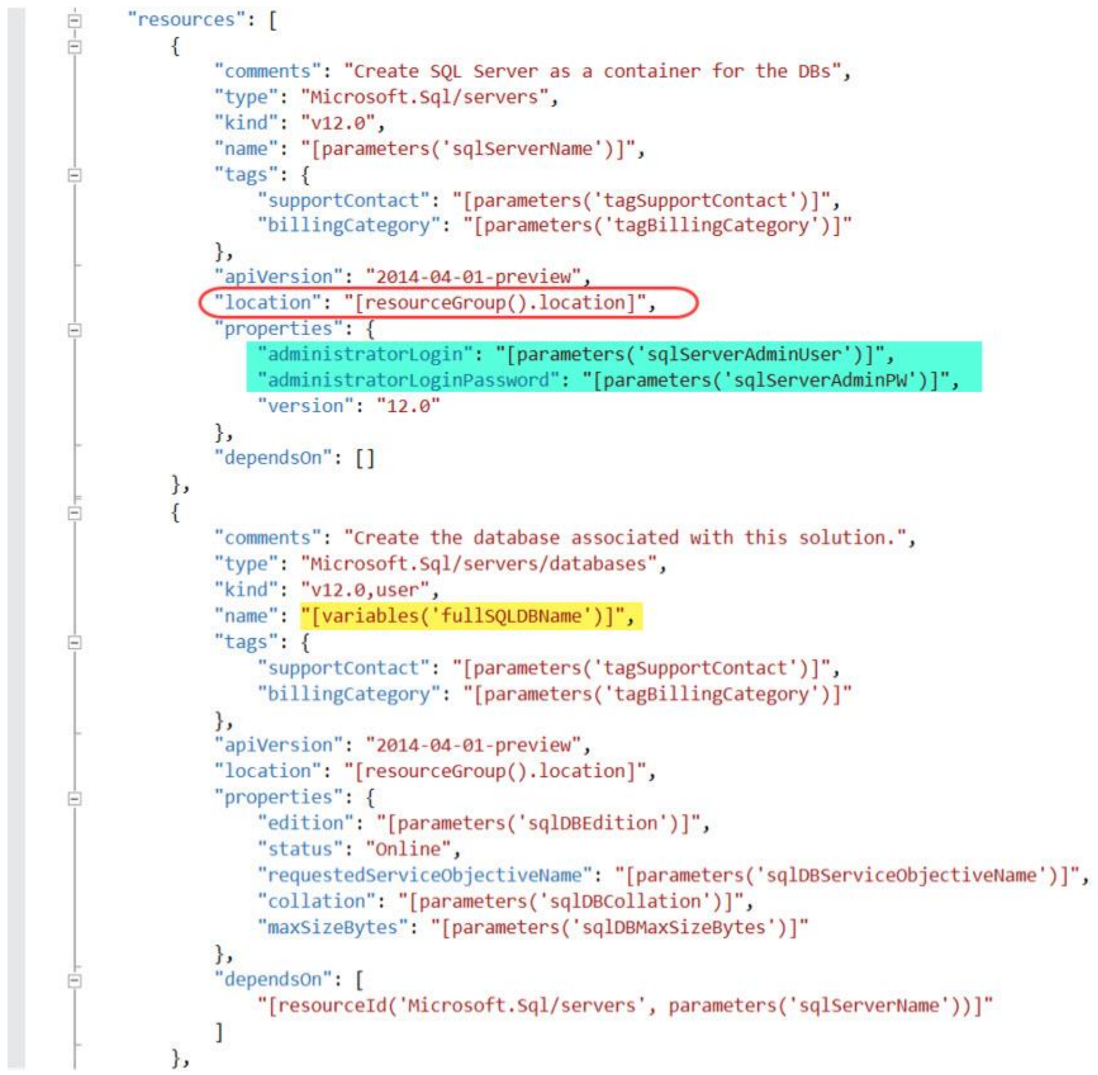
- The variables section can be optionally used for things that do vary but can be worked out systematically via JSON fragments. The idea here is to simplify what’s in the resources section. Note that variables do not have definitions (allowed values, default values, etc) the way parameters do.



```
"variables": {
    "fullSQLDBName": "[concat(parameters('sqlServerName'), '/', parameters('sqlDBName'))]",
    "fullSQLServerFirewallRulesAllowAllAzureIPs": "[concat(parameters('sqlServerName'), '/', parameters
      ('sqlServerFirewallRulesAllowAllAzureIPs'))]",
    "fullSQLServerFirewallRulesAllowS1Office": "[concat(parameters('sqlServerName'), '/', parameters
      ('sqlServerFirewallRulesAllowS1Office'))]",
    "fullWebAppURL": "[concat(parameters('webAppName'), '.azurewebsites.net')]",
    "fullWebAppSCMURL": "[concat(parameters('webAppName'), '.scm.azurewebsites.net')]",
    "fullWebsiteName": "[concat(parameters('webAppName'), '/web')]",
    "fullWebsitePublishingName": "[concat('$', parameters('webAppName'))]"
},
```

# Basic Template Structure: Resources

- The resources section defines each resource to be deployed, with references to parameters and variables as necessary. The elements which are defined vary based on the kind of resource which is being deployed.



```
"resources": [
    {
        "comments": "Create SQL Server as a container for the DBs",
        "type": "Microsoft.Sql/servers",
        "kind": "v12.0",
        "name": "[parameters('sqlServerName')]",
        "tags": {
            "supportContact": "[parameters('tagSupportContact')]",
            "billingCategory": "[parameters('tagBillingCategory')]"
        },
        "apiVersion": "2014-04-01-preview",
        "location": "[resourceGroup().location]",
        "properties": {
            "administratorLogin": "[parameters('sqlServerAdminUser')]",
            "administratorLoginPassword": "[parameters('sqlServerAdminPW')]",
            "version": "12.0"
        },
        "dependsOn": []
    },
    {
        "comments": "Create the database associated with this solution.",
        "type": "Microsoft.Sql/servers/databases",
        "kind": "v12.0,user",
        "name": "[variables('fullSQLDBName')]",
        "tags": {
            "supportContact": "[parameters('tagSupportContact')]",
            "billingCategory": "[parameters('tagBillingCategory')]"
        },
        "apiVersion": "2014-04-01-preview",
        "location": "[resourceGroup().location]",
        "properties": {
            "edition": "[parameters('sqlDBEdition')]",
            "status": "Online",
            "requestedServiceObjectiveName": "[parameters('sqlDBServiceObjectiveName')]",
            "collation": "[parameters('sqlDBCollation')]",
            "maxSizeBytes": "[parameters('sqlDBMaxSizeBytes')]"
        },
        "dependsOn": [
            "[resourceId('Microsoft.Sql/servers', parameters('sqlServerName'))]"
        ]
    },
```

DEMO

## ARM Templates

## What’s wrong with ARM Templates?

## What’s wrong with ARM Templates?

- Complicated to author
- JSON is not human friendly, not even for developers
- It’s Azure-only
- Lacking tools to make writing ARM templates easier
- Azure QuickStart Templates on GitHub are way behind
- ARM templates is just 1 step in the full DevOps cycle
- …

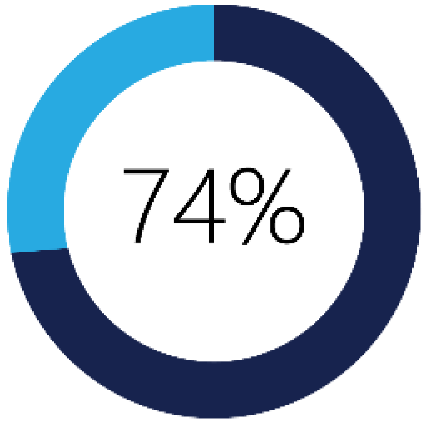
## Why not just relying on Azure Resource Manager Templates?

## It’s not just Azure…

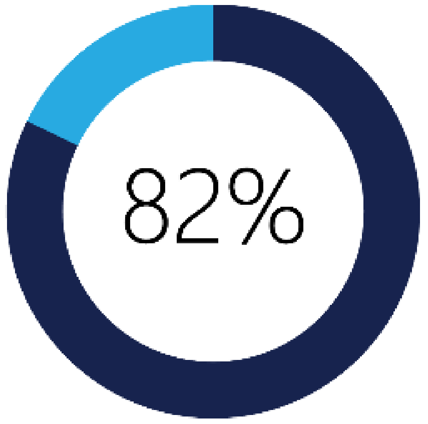
### Hybrid cloud, a reality today

**Workload requirements**

- ✓ Regulation
- ✓ Data Sensitivity
- ✓ Customization
- ✓ Latency
- ✓ Legacy support



Enterprise believe a hybrid cloud will enable business growth[1]



Enterprise have a hybrid cloud strategy, up from 74 percent a year ago[2]

Sources: 1. Avanade, Global Study: Hybrid Cloud-From Hype to Reality (Dec 2014); 2. IDC Cloud Prediction for 2015 (Dec 2014).

# Hybrid Cloud, a reality today

77% planning to deploy to multiple cloud[1]

91% of respondents planning to deploy to cloud[1]

Explosion in multi-cloud OSS tools

70% of enterprises to have multi cloud[2] strategy by 2019

1. Dimensional Research study
2. Gartner Study of Future of Datacenter in Cloud Era

## Why Terraform

- Terraform is a product to provision infrastructure and application resources across private cloud, public cloud, and external services using a common workflow
- Multi cloud
- Easy to describe JSON-alike format called HCF
- Support for both on-premises and clouds

Provision Any Infrastructure For Any Application

## Why Terraform

- HCL feels like a natural language
- Faster Deployments
- Refresh – Deploy – Update – Cleanup
- Several Azure Resources can be deployed with TerraForm, but not with ARM Templates

Provision Any Infrastructure For Any Application

HashiCorp
Terraform

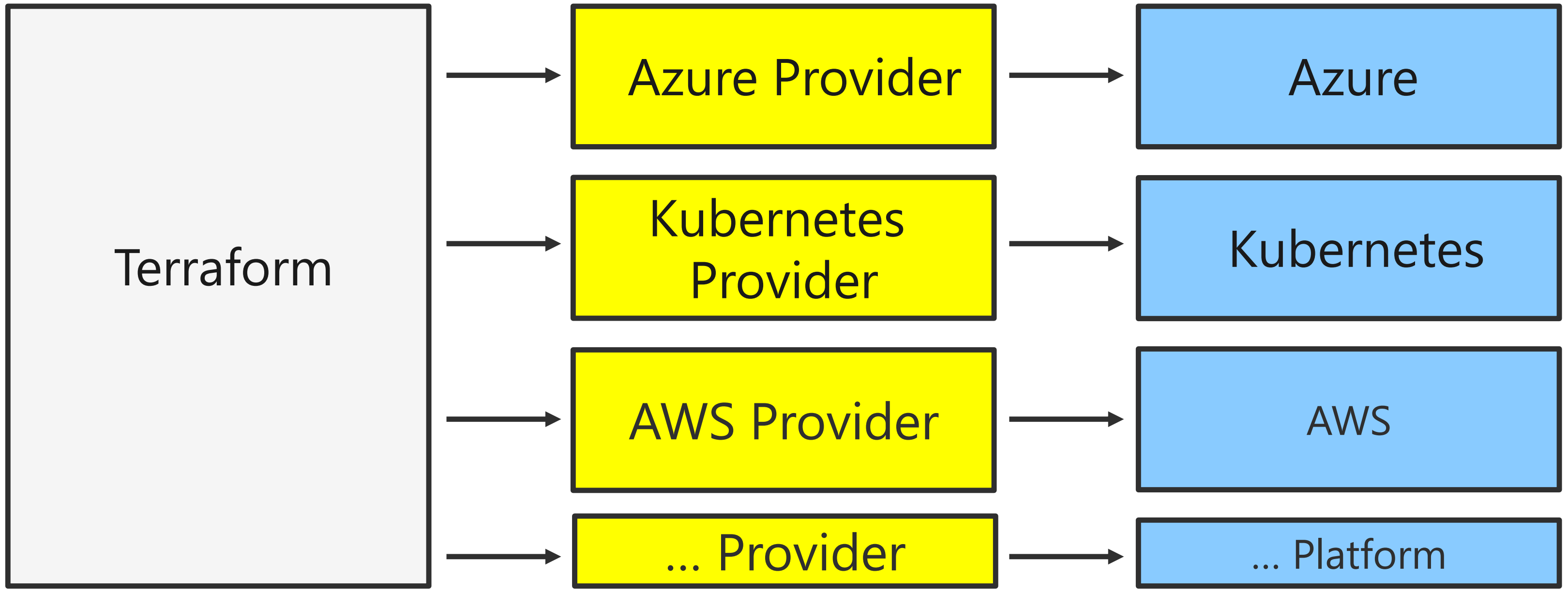
# ARM JSON vs HCL JSON

```
{
    "$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 "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"
}
```

# TerraForm Providers

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



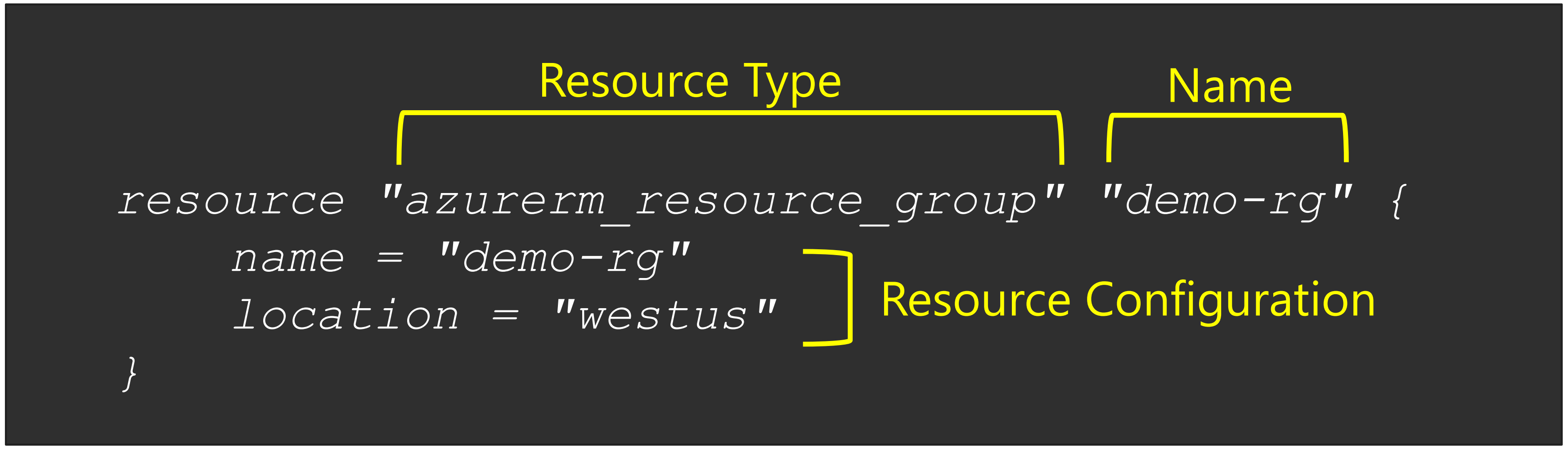
# TerraForm Template – AWS vs Azure (VPC <> Vnet)

```
# Configure the AWS Provider

  provider "aws" {

    region = "us-east-1"

  }


  Variable "vpc_id" {}

  data "aws_vpc" "selected" {
    id = "${var.vpc_id}"
  }
  resource "aws_subnet" "example" {
    vpc_id = "${data.aws_vpc.selected.id}"
    availability_zone = "us-west-2a"
    cidr_block =
"${cidrsubnet(data.aws_vpc.selected.cidr_block, 4, 1)}"

 }
```

```
# Configure the Azure Provider

  provider "azurerm" {

  version = "=1.22.0"

 }


Variable "vnet_id" {}

data "azurerm_virtual_network" "test" {
  name = "production"
  resource_group_name = "networking"
}

resource "virtual_network_id" {
    vnet = "${data.azurerm_virtual_network.test.id}"
    subnet = "${data.azurerm_virtual_network.subnet.id
}
```

## Basic resource creation

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



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

DEMO

## TerraForm Template

## Basic Terraform commands

| INIT | PLAN | APPLY | DESTROY |
|---|---|---|---|
| Initialize the working folder | Pre-flight validation | Actual Deploy | Removes Resources |

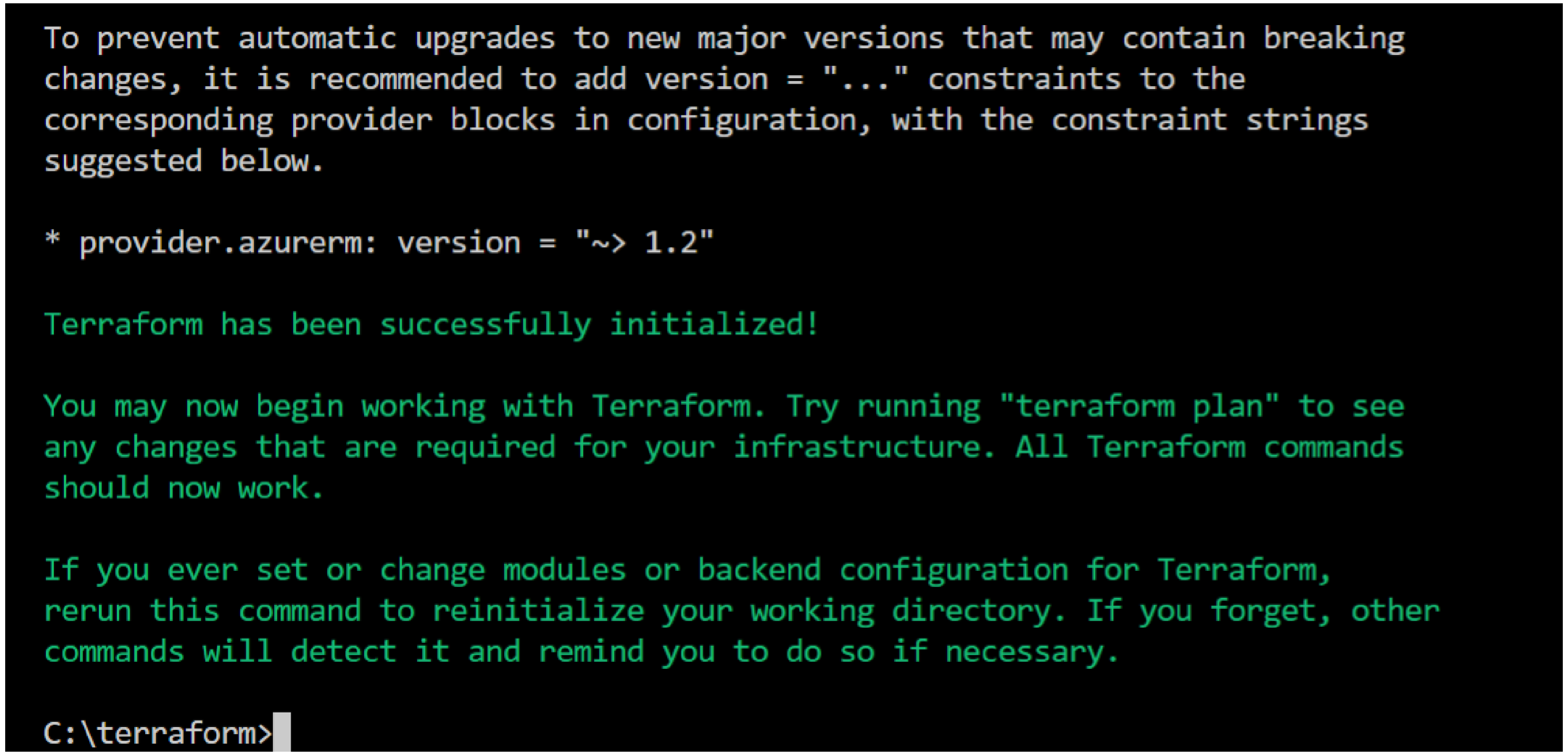
That's it ☺ !!

Yes, 4 commands…

DEMO

## TerraForm Initialize

# TerraForm Initialize



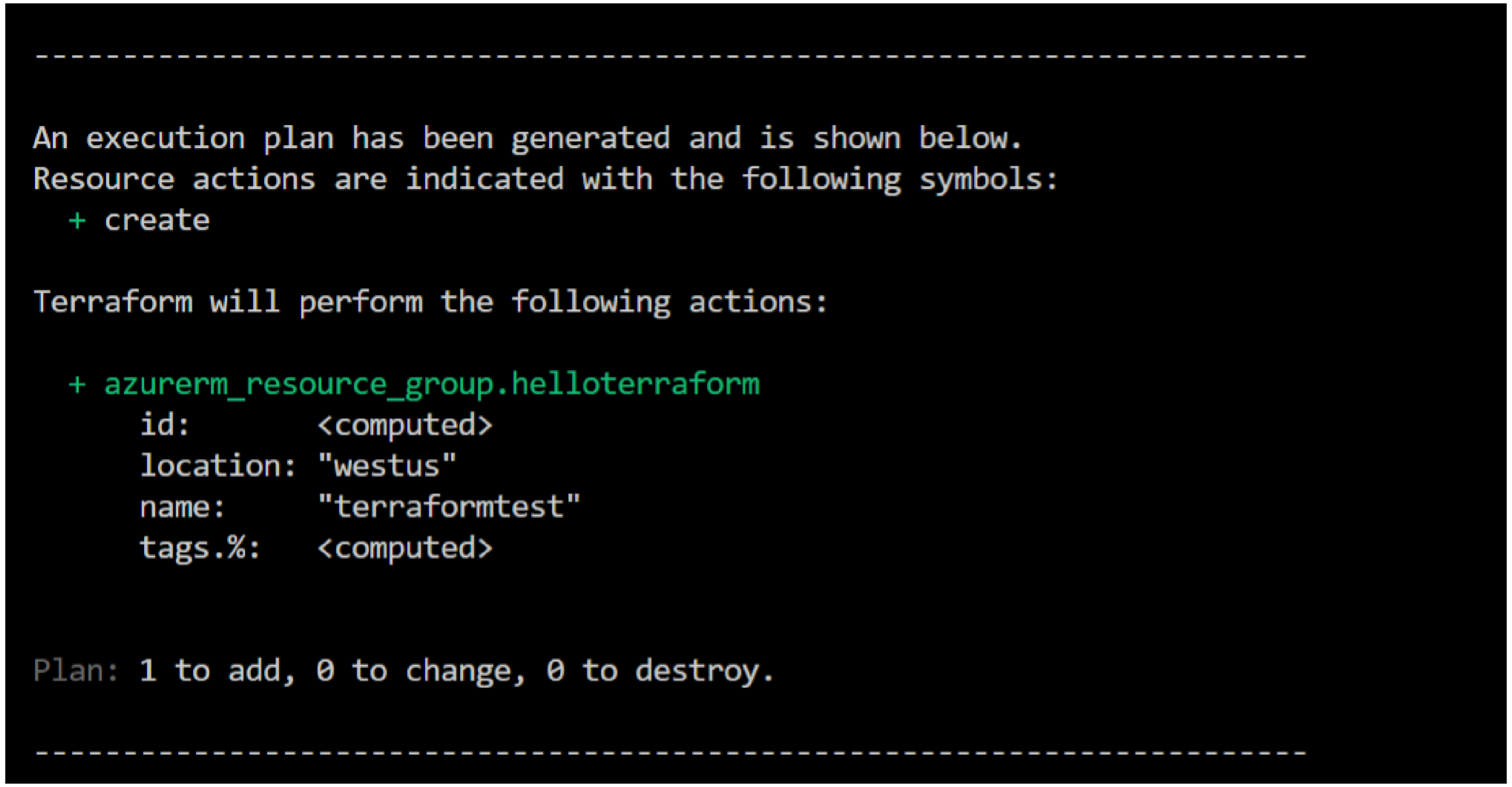
```
To prevent automatic upgrades to new major versions that may contain breaking
changes, it is recommended to add version = "..." constraints to the
corresponding provider blocks in configuration, with the constraint strings
suggested below.

* provider.azurerm: version = "~> 1.2"

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.

C:\terraform>
```

*"… I now know what platform you want me to talk to…"*

DEMO

## TerraForm Plan

# TerraForm Initialize



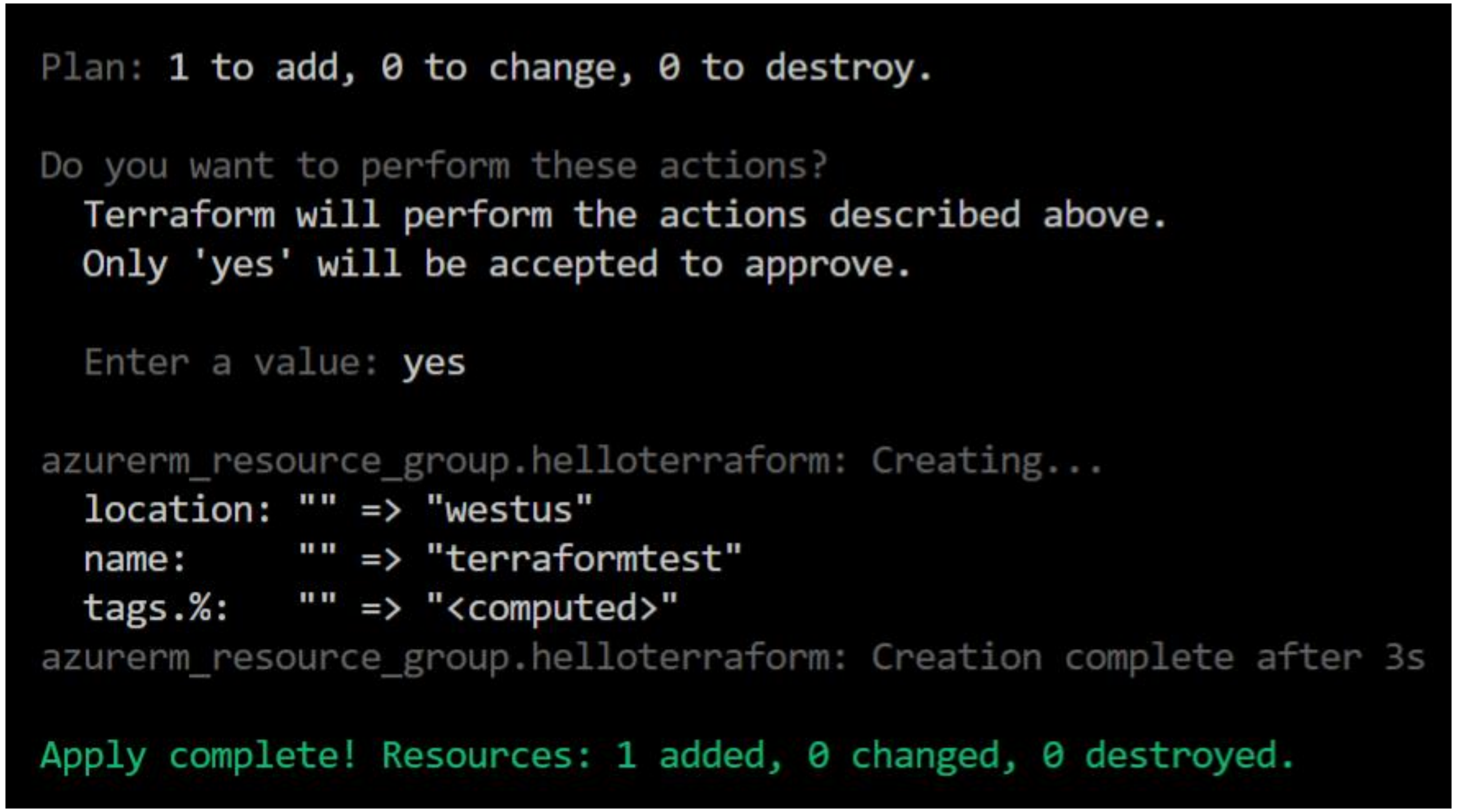
```
------------------------------------------------------------------------

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.helloterraform
      id:        <computed>
      location:  "westus"
      name:      "terraformtest"
      tags.%:    <computed>


Plan: 1 to add, 0 to change, 0 to destroy.

------------------------------------------------------------------------
```

*"... I now know what you want me to deploy..."*

DEMO

## TerraForm Apply

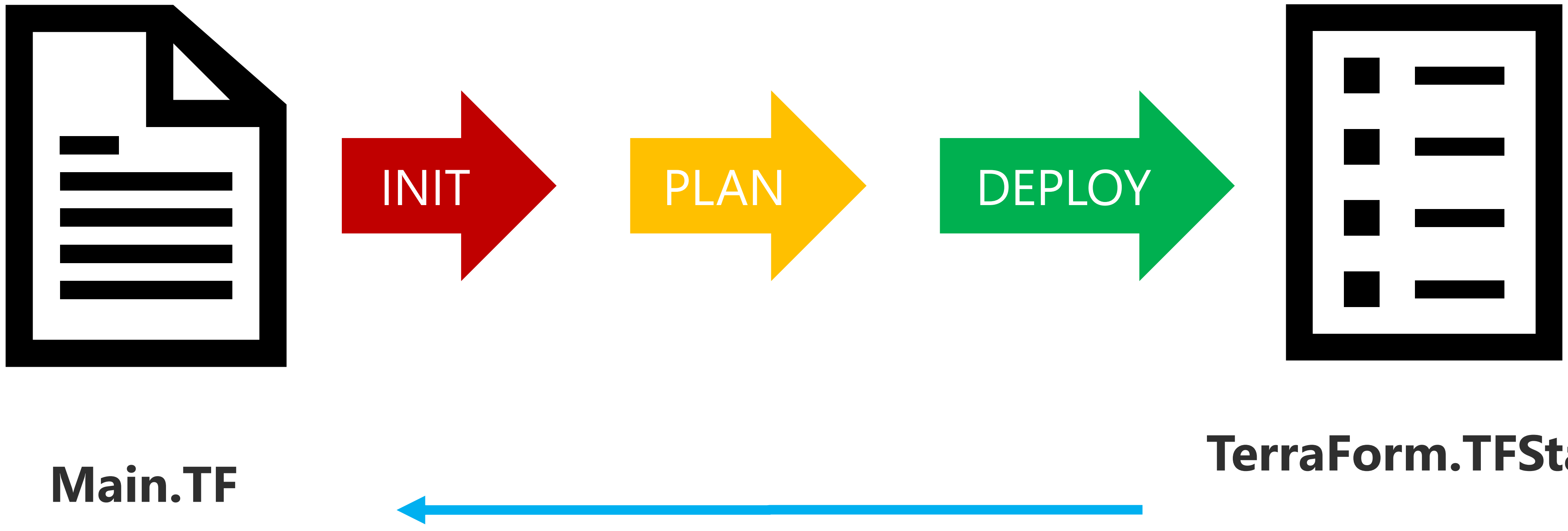
## TerraForm Initialize



```
Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

azurerm_resource_group.helloterraform: Creating...
  location: "" => "westus"
  name:     "" => "terraformtest"
  tags.%:   "" => "<computed>"
azurerm_resource_group.helloterraform: Creation complete after 3s

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

*"... I deployed stuff !! ..."*

## TerraForm TFState



**Main.TF**

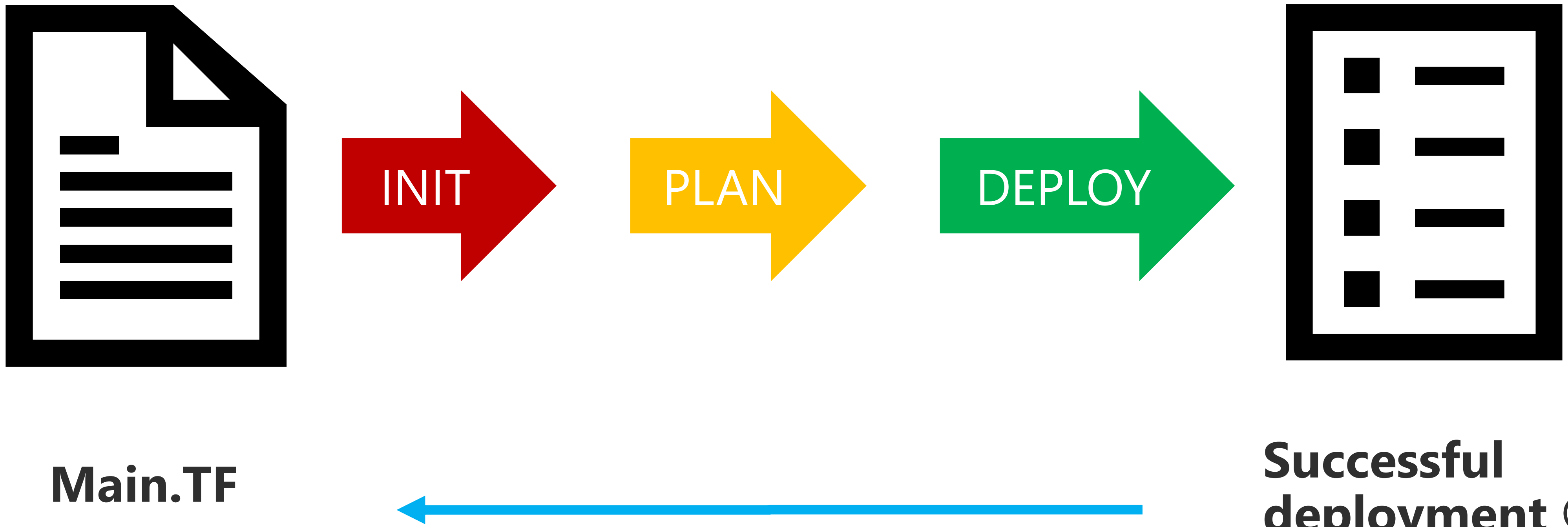
**= your definition**

**TerraForm.TFState**

**= the result state**

# Best of Both Worlds

# Deploying ARM Templates with TerraForm Templates



**Main.TF**

**Successful deployment ☺**

**= TerraForm snippets**
**+ ARM Template snippets**

DEMO

## TerraForm ARM deployment

# TerraForm Template Best Practices

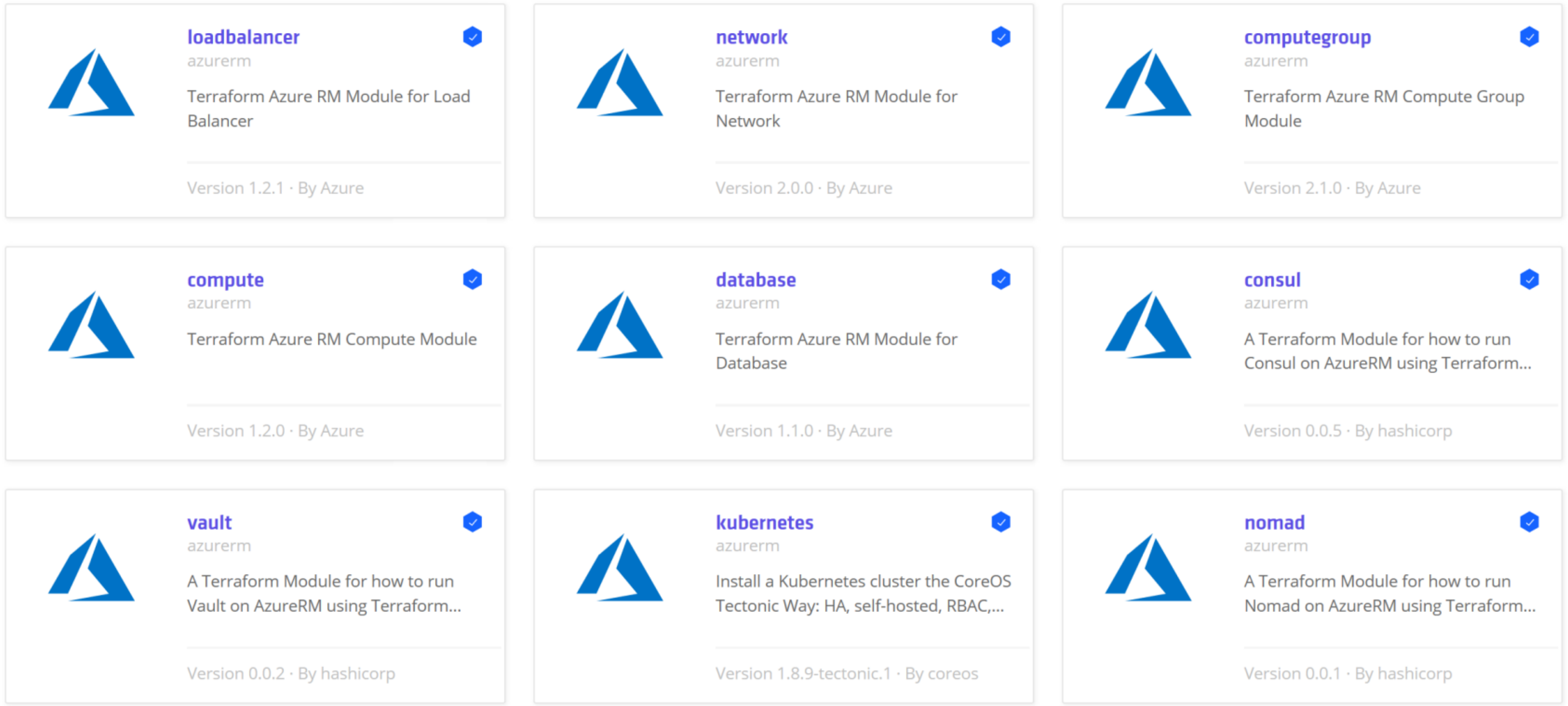
- Never store credentials in the Template
  (e.g. Azure Subscription ID, Service Principal ID, Tenant ID,…)
- Separate resource types in dedicated TerraForm Files
  (e.g. Network, Storage, Compute, Databases, Web Apps,… where each “team” provides its own template)
- Store the TerraForm.TFState in a shared, but safe place
  (e.g. GitHub, Azure DevOps, Azure Storage,…)
- Use TerraForm Modules where possible
  (e.g. Computer Group, more complex topologies,…)

DEMO

## TerraForm Template Best Practices

# TerraForm Modules

- Preconfigured Snippets of Code
  - Azure Load Balancer
  - Azure Vnet
  - Azure Key Vault
  - Kubernetes Services
  - Azure VM
  - …



loadbalancer
azurerm
Terraform Azure RM Module for Load Balancer
Version 1.2.1 · By Azure

network
azurerm
Terraform Azure RM Module for Network
Version 2.0.0 · By Azure

computegroup
azurerm
Terraform Azure RM Compute Group Module
Version 2.1.0 · By Azure

compute
azurerm
Terraform Azure RM Compute Module
Version 1.2.0 · By Azure

database
azurerm
Terraform Azure RM Module for Database
Version 1.1.0 · By Azure

consul
azurerm
A Terraform Module for how to run Consul on AzureRM using Terraform...
Version 0.0.5 · By hashicorp

vault
azurerm
A Terraform Module for how to run Vault on AzureRM using Terraform...
Version 0.0.2 · By hashicorp

kubernetes
azurerm
Install a Kubernetes cluster the CoreOS Tectonic Way: HA, self-hosted, RBAC,...
Version 1.8.9-tectonic.1 · By coreos

nomad
azurerm
A Terraform Module for how to run Nomad on AzureRM using Terraform...
Version 0.0.1 · By hashicorp

# TerraForm in DevOps

# DevOps in an Azure World

## Continuous integration (CI)

Take advantage of continuous integration to improve software development quality and speed. When you use Visual Studio Team Services or Jenkins to build apps in the cloud and deploy to Azure, **each time you commit code, it's automatically built and tested**— so bugs are detected faster.

## Continuous Delivery (CD)

Ensure that code and infrastructure are always in a production-deployable state, with continuous delivery. **By combining continuous integration and infrastructure as code (IaC), you'll achieve identical deployments** and the confidence you need to manually deploy to production at any time.

## Continuous Deployment (CI/CD)

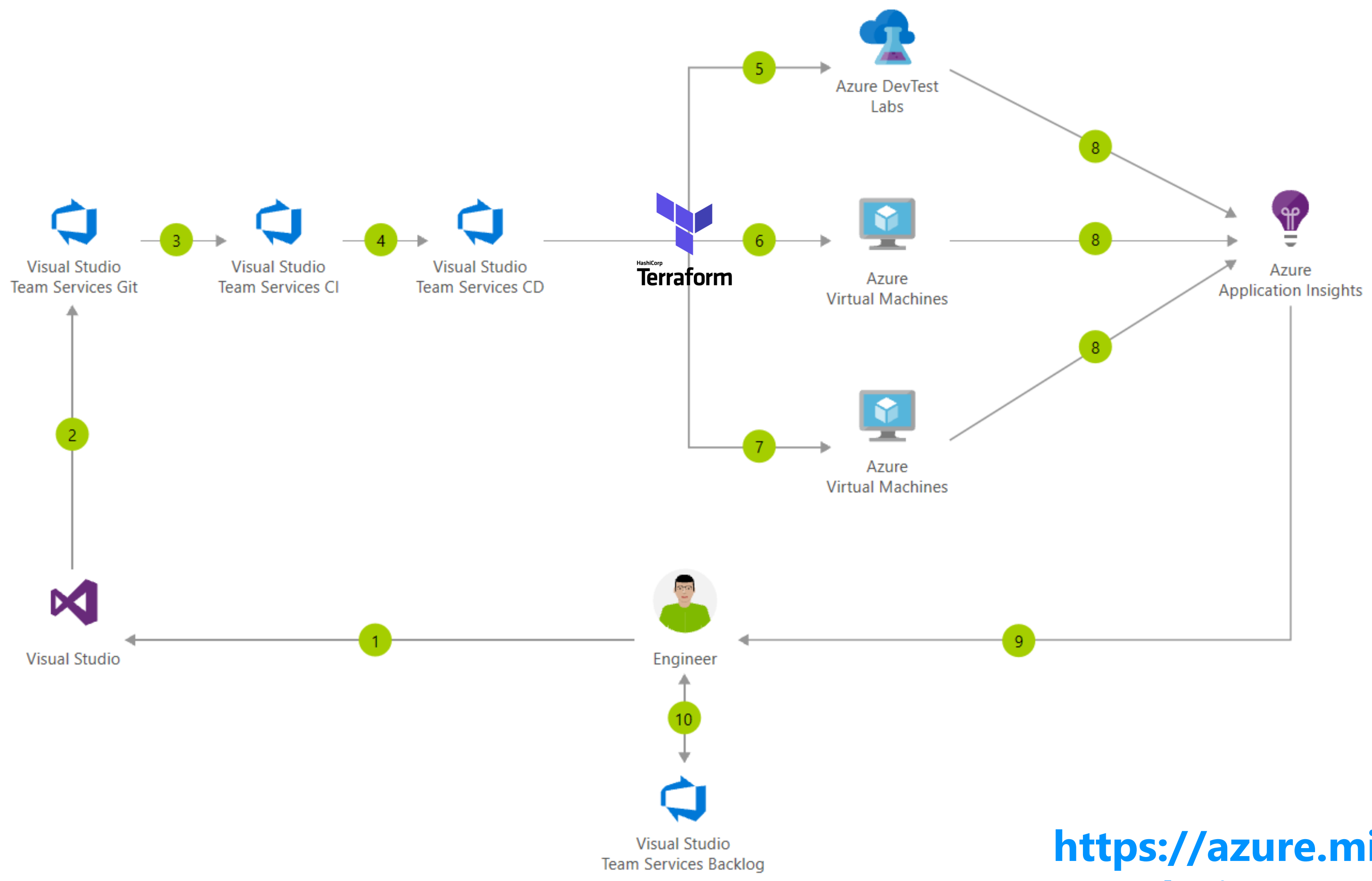
With continuous deployment, you can **automate the entire process from code commit to production** if your CI/CD tests are successful. Using CI/CD practices, paired with monitoring tools, you'll be able to safely deliver features to your customers as soon as they're ready.

## DevOps in an Azure World

- **Get the most from infrastructure automation and configuration management**
  - In addition to using Azure Resource Manager for infrastructure as code, you can provision and manage Azure infrastructure directly from your favorite third-party tools, such as Ansible, Chef, Puppet, and Terraform.

- **Gain clear guidance and example architectures**
- Azure services, third-party DevOps tools, and related products all work together to help meet the most common business needs and scenarios—including yours. Get started quickly with Azure DevOps solutions that give you access to architectures, tutorials, documentation, examples, templates, partners, and other resources.

# CI/CD for Azure using TerraForm



Azure DevTest Labs
Visual Studio Team Services Git
Visual Studio Team Services CI
Visual Studio Team Services CD
HashiCorp Terraform
Azure Virtual Machines
Azure Virtual Machines
Azure Application Insights
Visual Studio
Engineer
Visual Studio Team Services Backlog
1
2
3
4
5
6
7
8
8
8
9
10

1. Change application source code
2. Commit Application Code and Azure Resource Manager (ARM) Template
3. Continuous integration triggers application build and unit tests
4. Continuous deployment trigger orchestrates deployment of application artifacts with environment specific parameters
5. Deployment to QA environment
6. Deployment to staging environment
7. Deployment to production environment
8. Application Insights collects and analyses health, performance and usage data
9. Review health, performance and usage information
10. Update backlog item

**https://azure.microsoft.com/en-us/solutions/architecture/?solution=devops**

DEMO

## Azure DevOps with TerraForm

# Tervetuloa Helsinkiin

## Welcome to TechDays 2019 - Helsinki

## Azure Loves Terraform Loves Azure

Intro to Infrastructure as Code

Settings The scene for TerraForm

Terraform Loves Demos

Q & A

## Thank You

## You were an awesome audience

Microsoft
tech·days
Helsinki 2019

Get the slides and signup for the Finnish tech community newsletter at http://aka.ms/td-azure-community