Deploying Terraform via Azure DevOps with some awesome features

Thomas Thornton & Gregor Suttie

Speaker Intro – Thomas Thornton

- **10+ years experience in IT**
- **Senior Ops Engineer – Kainos in Belfast**
- **Azure Certified**
  - **Azure Solutions Architect**
  - **DevOps Engineer**
  - **Azure Security Engineer**
  - **Azure Administrator**
- **https://thomasthornton.cloud/ - Blogging all things Azure**
- **Twitter:- @tamstar1234**

Speaker Intro – Gregor Suttie

- 23 years' experience in IT
- Developer and DevOps background
- Azure Architect for Intercept in the Netherlands
- Azure MVP and a Microsoft Certified Trainer
- https://gregorsuttie.com/ - Azure Blog
- Twitter:- @gregor_suttie

Agenda

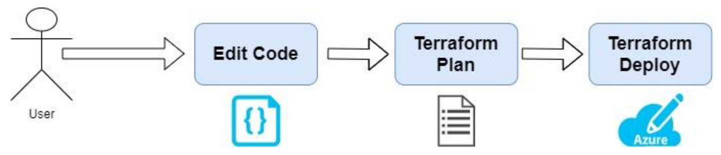
- What is Terraform?
- Terraform Workflow
- Azure DevOps
- Deploying Terraform into Azure using Azure DevOps
- Demos
- Questions?

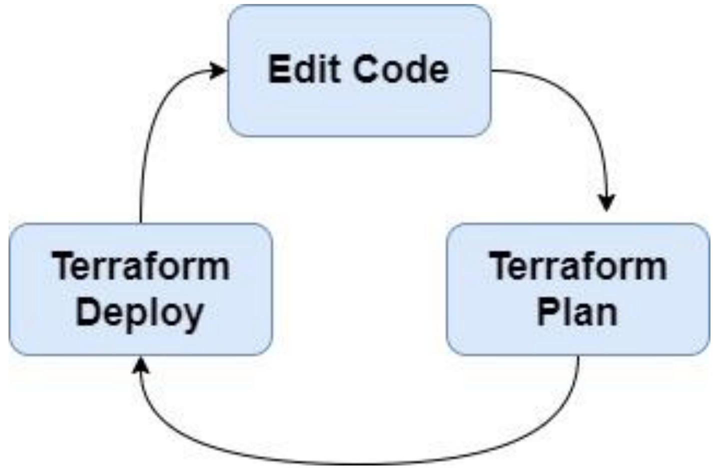
What is Terraform?

- A way to manage Azure
- Easy to read and write
- Declarative
- Driven via the Azure API
- OpenSource Free
- Disposable Environments
- Lowers the potential for human errors while deploying and managing infrastructure.

Terraform Workflow

- **terraform init -** Initialize a Terraform working directory
- **terraform plan -** Generate and show an execution plan
- **terraform apply -** Builds or changes infrastructure
- **terraform output -** Read an output from a state file
- **terraform destroy -** Destroy Terraform-managed infrastructure





Terraform Teminology

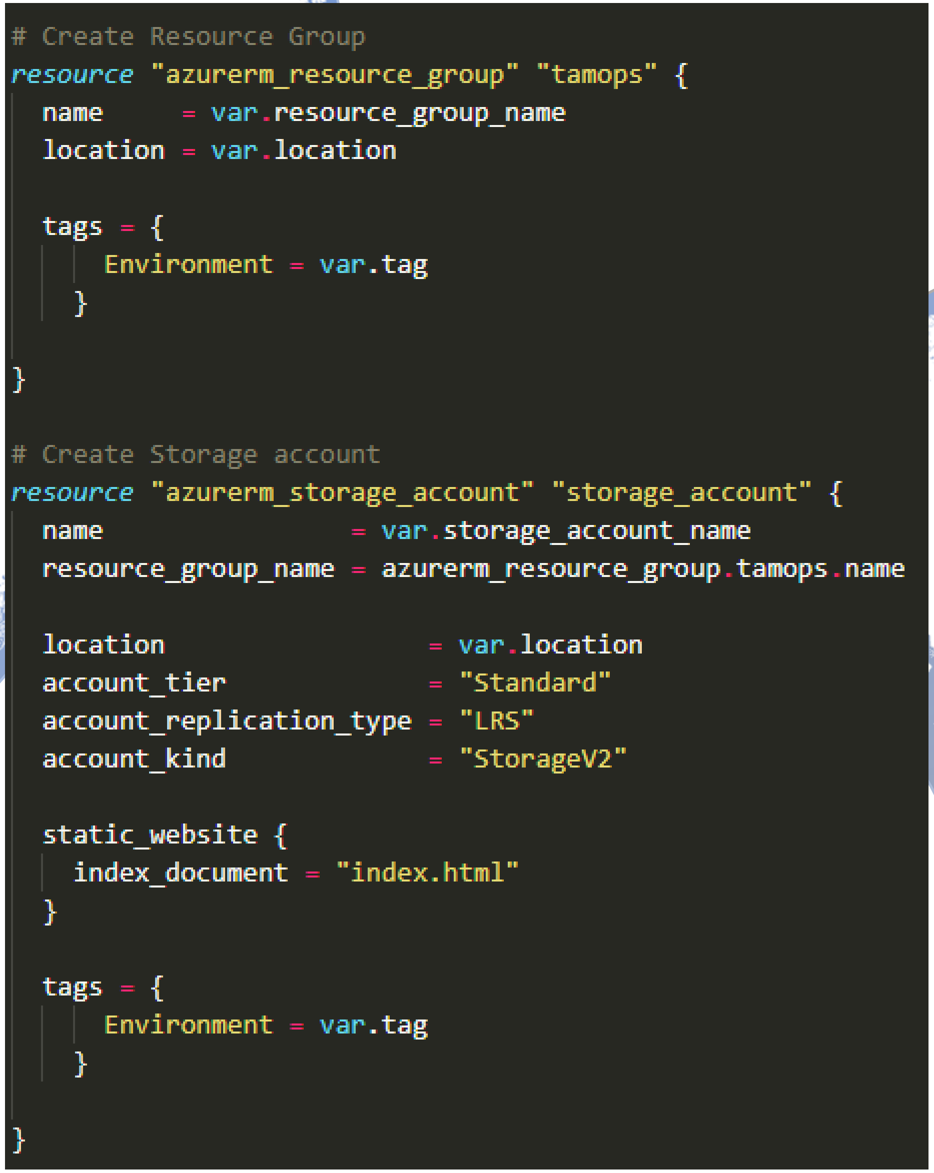
Providers represent a cloud provider or a local provider

Resources can be invoked to create/update infrastructure locally or on the cloud.

State is representation of the infrastructure created/updated by terraform.

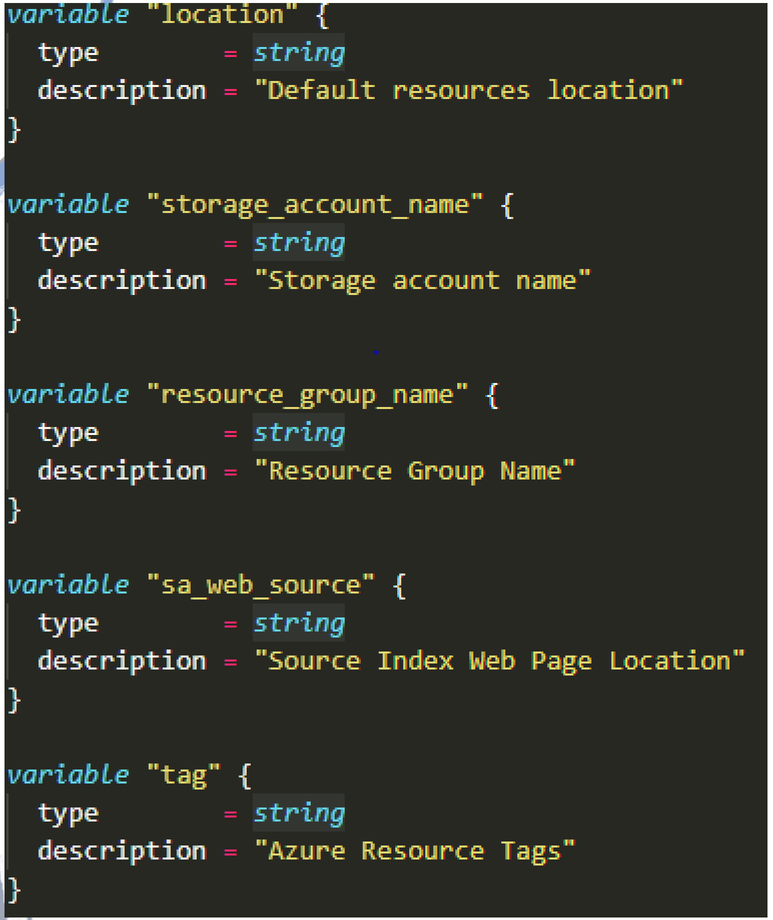
Data Sources are "read-only" resources

Terraform templates



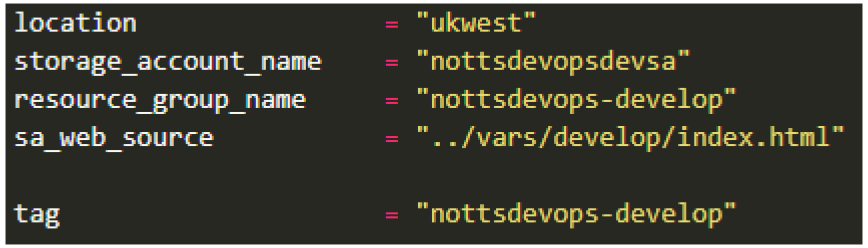
```
# Create Resource Group
resource "azurerm_resource_group" "tamops" {
  name     = var.resource_group_name
  location = var.location

  tags = {
    Environment = var.tag
  }

}

# Create Storage account
resource "azurerm_storage_account" "storage_account" {
  name                = var.storage_account_name
  resource_group_name = azurerm_resource_group.tamops.name

  location                 = var.location
  account_tier             = "Standard"
  account_replication_type = "LRS"
  account_kind             = "StorageV2"

  static_website {
    index_document = "index.html"
  }

  tags = {
    Environment = var.tag
  }

}
```

main.tf



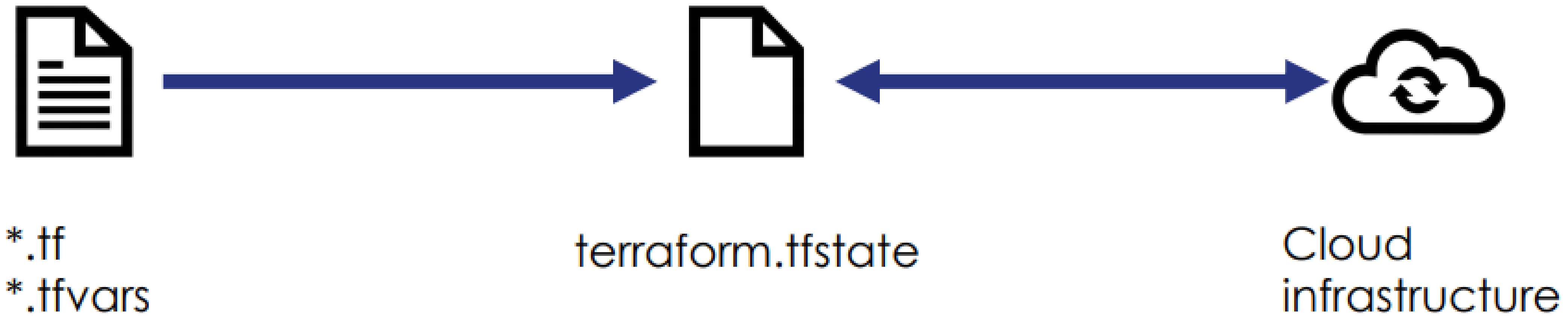
```
variable "location" {
  type        = string
  description = "Default resources location"
}

variable "storage_account_name" {
  type        = string
  description = "Storage account name"
}

variable "resource_group_name" {
  type        = string
  description = "Resource Group Name"
}

variable "sa_web_source" {
  type        = string
  description = "Source Index Web Page Location"
}

variable "tag" {
  type        = string
  description = "Azure Resource Tags"
}
```

variables.tf



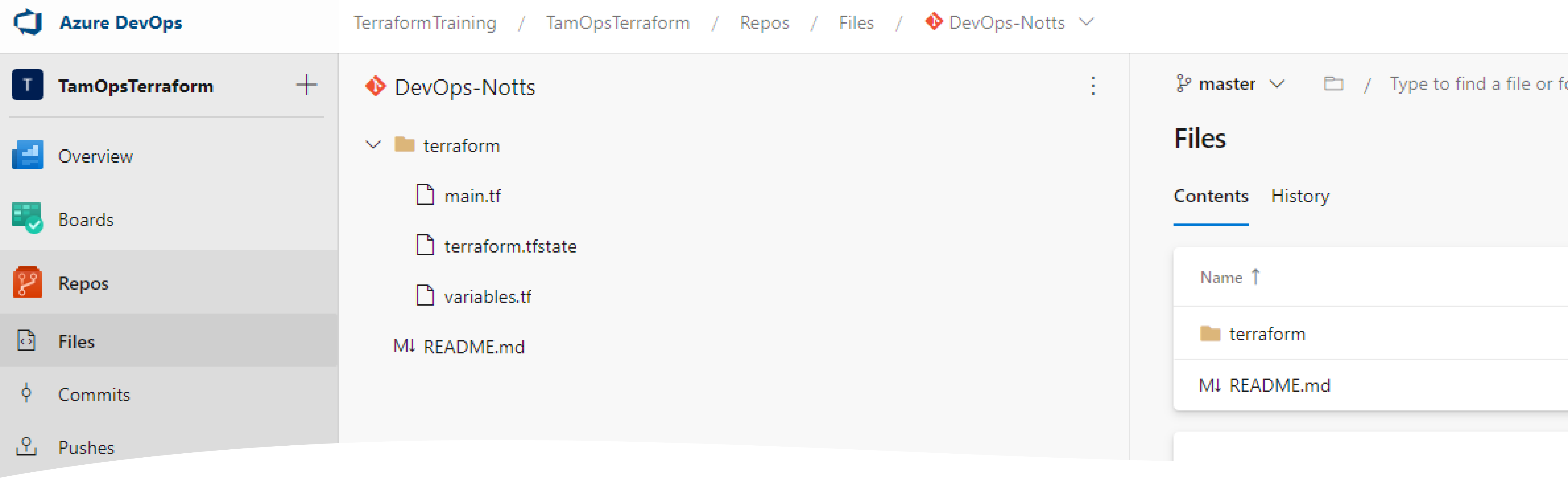
```
location                 = "ukwest"
storage_account_name     = "nottsdevopsdevsa"
resource_group_name      = "nottsdevops-develop"
sa_web_source            = "../vars/develop/index.html"

tag                      = "nottsdevops-develop"
```

develop.tfvars



Terraform State

- Terraform must store state about your managed infrastructure and configuration.
- This state is used by Terraform to map real world resources to your configuration, keep track of metadata, and to improve performance for large infrastructures.
- This state is stored by default in a local file named "terraform.tfstate", but it can also be stored remotely, which works better in a team environment.


Azure DevOps   TerraformTraining / TamOpsTerraform / Repos / Files / DevOps-Notts

TamOpsTerraform
- Overview
- Boards
- Repos
- Files
- Commits
- Pushes

DevOps-Notts
- terraform
  - main.tf
  - terraform.tfstate
  - variables.tf
- README.md

master   / Type to find a file or fo

Files

Contents   History

Name ↑
terraform
README.md

Azure Repos

- Part of Azure DevOps
- A Git repository that will be used to store your Terraform code
- Deploying your Pipeline will reference an Azure Repo

Plan smarter, collaborate better and ship faster with **Azure DevOps** Services © Azure DevOps

Azure Pipelines

Azure Pipelines is a cloud service that you can use to automatically build and test your code project and make it available to other users. It works with just about any language or project type.

Azure Pipelines combines continuous integration (CI) and continuous delivery (CD) to constantly and consistently test and build your code and ship it to any target.

Within our environment, we will be using Azure Pipelines to deploy our Terraform code

You define pipelines using the YAML syntax or through the Azure DevOps portal – we will be using YAML

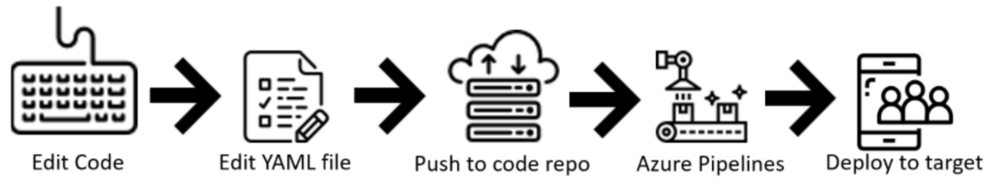
Azure Pipelines via Code

- You define your pipeline in a YAML file within your repo, **azure-pipelines.yml** for example

- The YAML pipeline is versioned the same way as your Terraform code.

- It will follow the same branching structure allowing you to have a pull-request process for any changes to any Pipelines that you may make

- Will look at some cool pipeline/repo additions



Azure Pipelines:- Additional Info

We’ve covered the basics of Azure Pipelines and how they are created using YAML. Some more info on additional components that will be used within Azure Pipelines

How do the Pipelines run?

- Can be manual within Azure DevOps Portal or create a trigger to tell a pipeline to run.

Variable Groups

- Used to store values that you want to control and make available across multiple pipelines.

Approval Gates

- Can be used for Pre-Deployment and Post-deployment conditions within a Pipeline. For example, when terraform plan is successful; before terraform apply is ran you can create an Approval gate to allow review of the Terraform plan etc.

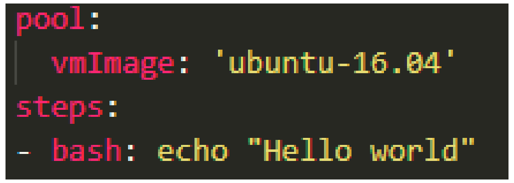
Trigger

- A trigger is something that's set up to tell the pipeline when to run.

Azure Pipelines Breakdown

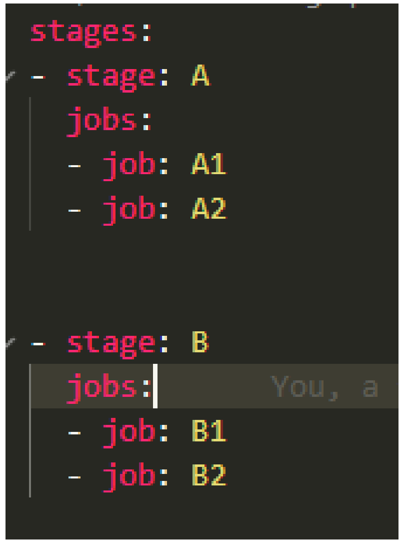
- You can organise your pipeline into jobs.
- Every pipeline has at least one job.
- A job is a series of steps that run sequentially as a unit
- Your pipeline may have multiple stages, each with multiple jobs. In that case, use the **stages** keyword.

1. Single Step



```yaml
pool:
  vmImage: 'ubuntu-16.04'
steps:
- bash: echo "Hello world"
```

2. Multi-stage

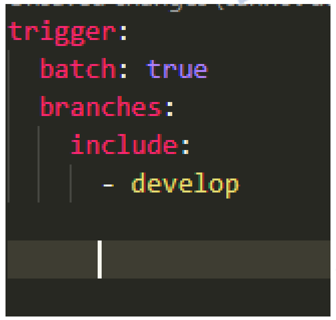


```yaml
stages:
- stage: A
  jobs:
  - job: A1
  - job: A2


- stage: B
  jobs:            You, a
  - job: B1
  - job: B2
```

Azure Pipelines Breakdown:- Triggers

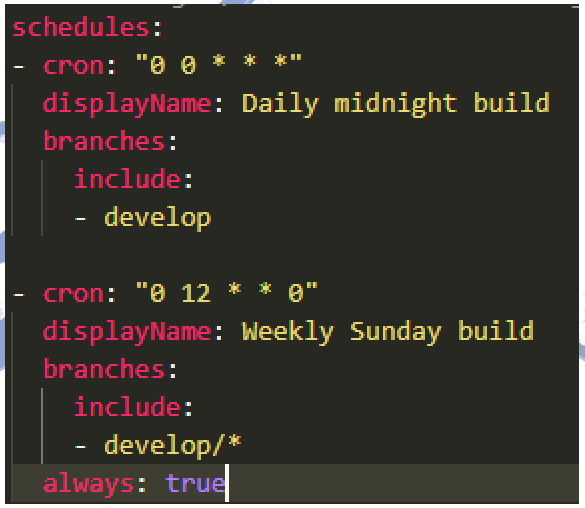
- Use triggers to run a pipeline automatically.
- Azure Pipelines supports many types of triggers.
- Common Triggers
  - PR Triggers
  - Scheduled Triggers
  - Pipeline Trigger

1. PR Trigger



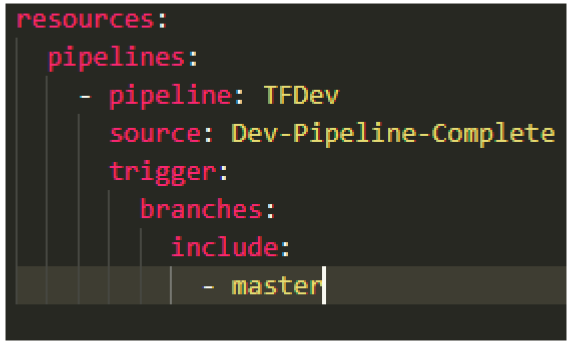
```yaml
trigger:
  batch: true
  branches:
    include:
      - develop
```

2. Schedule Triggers



```yaml
schedules:
- cron: "0 0 * * *"
  displayName: Daily midnight build
  branches:
    include:
    - develop

- cron: "0 12 * * 0"
  displayName: Weekly Sunday build
  branches:
    include:
    - develop/*
  always: true
```

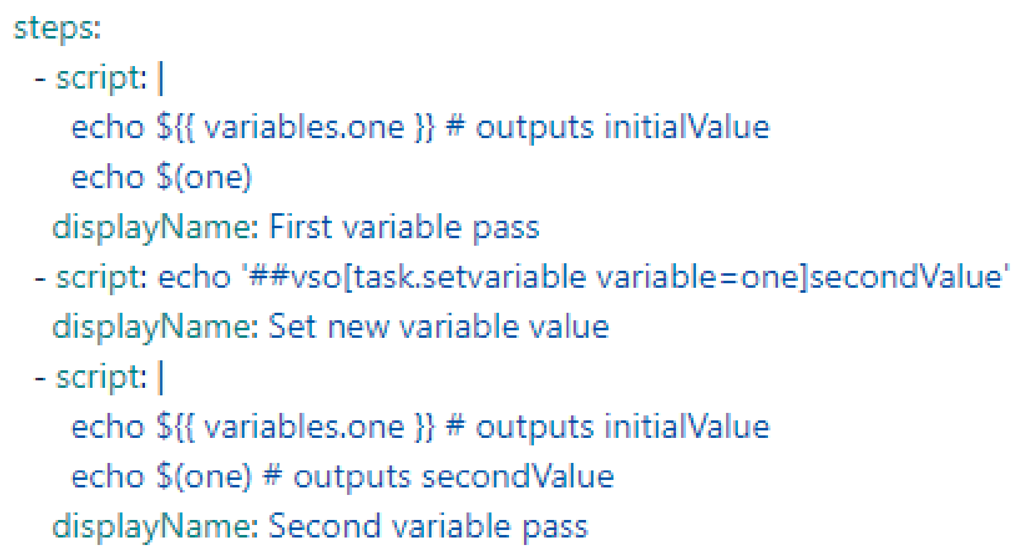
3. Pipeline Trigger



```yaml
resources:
  pipelines:
    - pipeline: TFDev
      source: Dev-Pipeline-Complete
      trigger:
        branches:
          include:
            - master
```

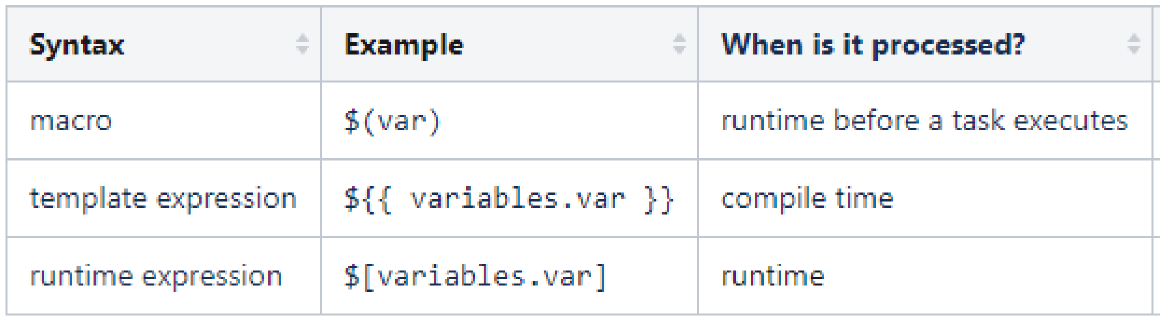
Azure Pipelines Breakdown:- Variables

- Variables give you a convenient way to get key bits of data into various parts of the pipeline.
- Azure Pipelines supports three different ways to reference variables: macro, template expression, and runtime expression. Each syntax can be used for a different purpose and has some limitations.



```
steps:
 - script: |
    echo ${{ variables.one }} # outputs initialValue
    echo $(one)
   displayName: First variable pass
 - script: echo '##vso[task.setvariable variable=one]secondValue'
   displayName: Set new variable value
 - script: |
    echo ${{ variables.one }} # outputs initialValue
    echo $(one) # outputs secondValue
   displayName: Second variable pass
```

Example of macro, template and runtime variables

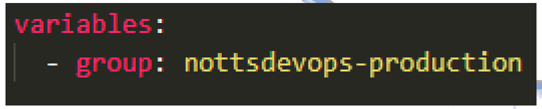


Syntax	Example	When is it processed?
macro	`$(var)`	runtime before a task executes
template expression	`${{ variables.var }}`	compile time
runtime expression	`$[variables.var]`	runtime

Azure Pipelines Breakdown:- Variable Groups

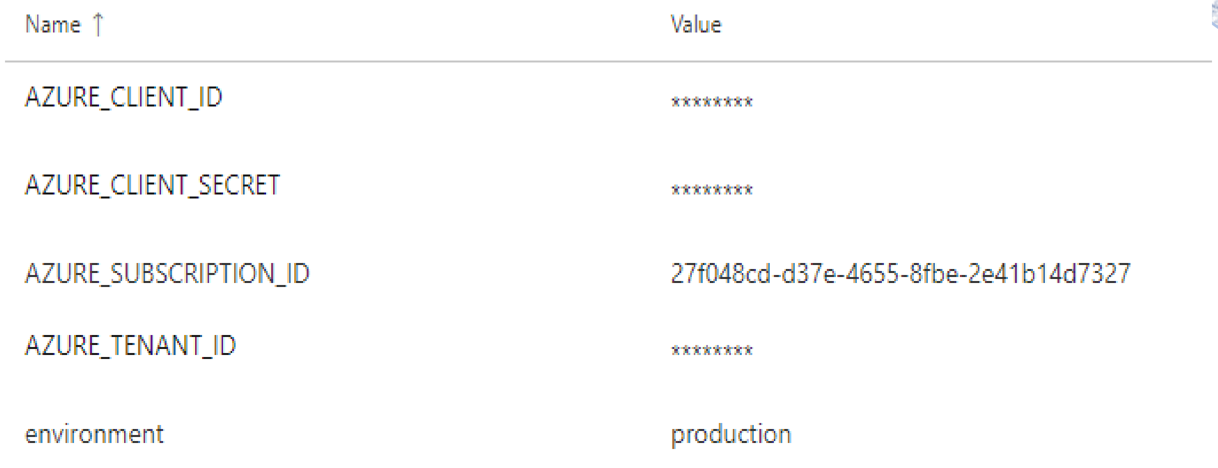
- Variable groups are great; used throughout our Pipelines
- Use a variable group to store values that you want to control and make available across multiple pipelines
- You can also use variable groups to store secrets and other values that might need to be passed into a YAML pipeline.

YAML Variable Group



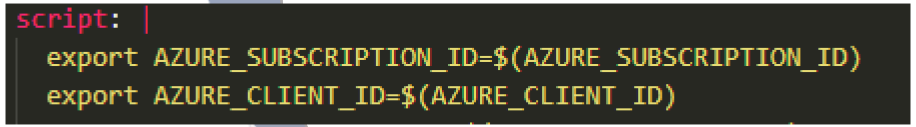
```yaml
variables:
  - group: nottsdevops-production
```

Variable Group variables



Name ↑	Value
AZURE_CLIENT_ID	********
AZURE_CLIENT_SECRET	********
AZURE_SUBSCRIPTION_ID	27f048cd-d37e-4655-8fbe-2e41b14d7327
AZURE_TENANT_ID	********
environment	production

Reference variable in pipeline



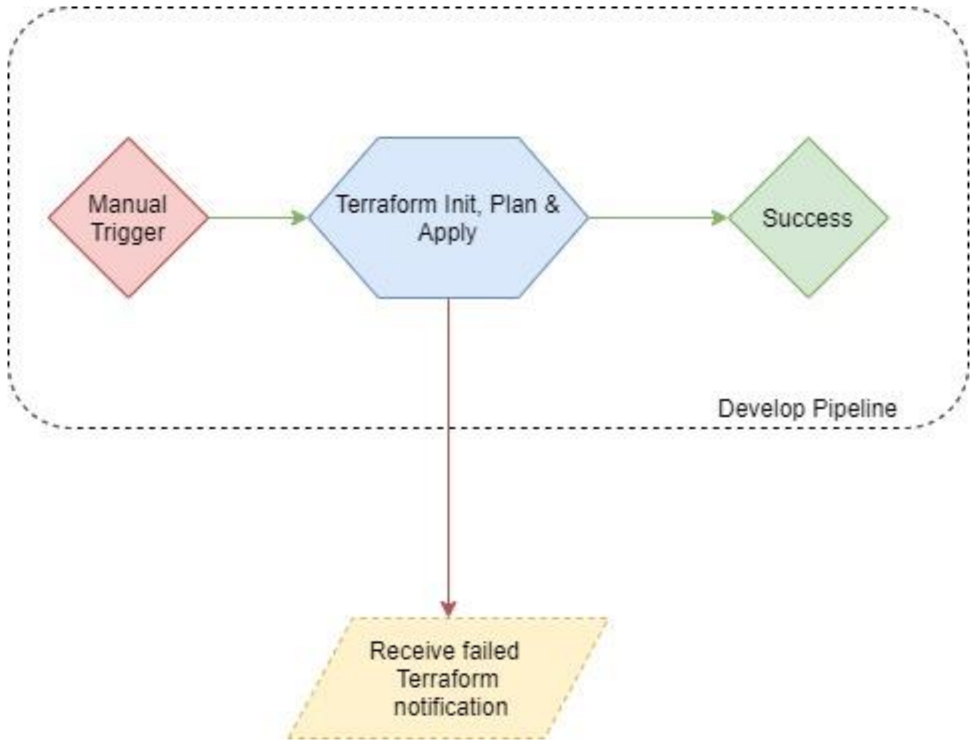
```yaml
script: |
  export AZURE_SUBSCRIPTION_ID=$(AZURE_SUBSCRIPTION_ID)
  export AZURE_CLIENT_ID=$(AZURE_CLIENT_ID)
```

Demo Time

What will be covered?

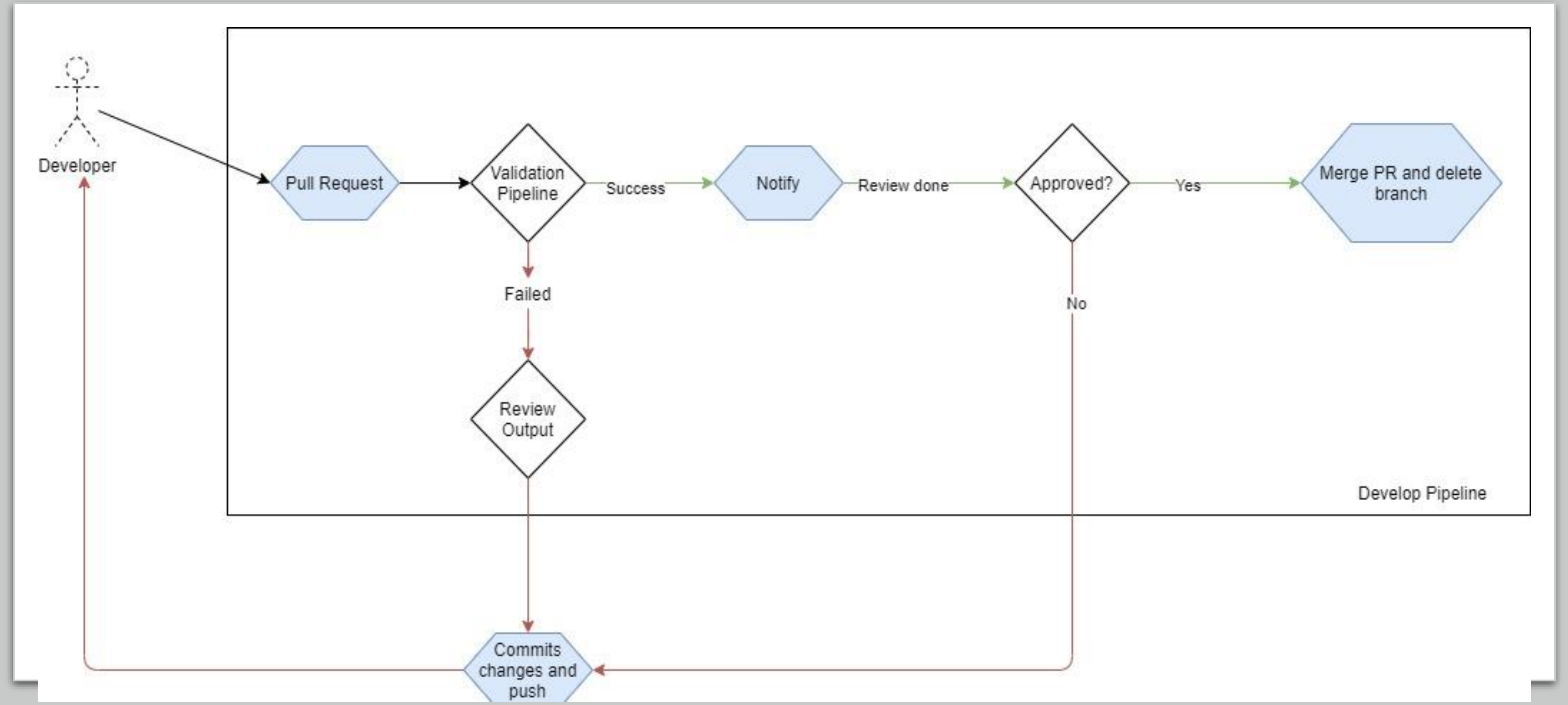
- Azure Pipeline deployment with Terraform
- Multi-branch triggers
- I don’t want to validate my code in the main pipeline, what can I do?
- Can I test what I have deployed? Ofcourse, we will touch briefly on Inspec-Azure!

Where did we start?


Manual Trigger
Terraform Init, Plan & Apply
Success
Develop Pipeline
Receive failed Terraform notification

We added a build validation pipeline!

- Keeps the main pipeline clean!
- Can validate Terraform code before it reaches an environment pipeline
- Can review the plan during PR

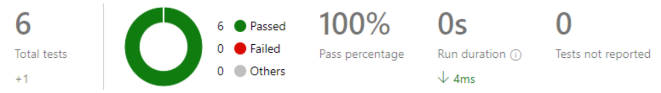


Testing what you deployed

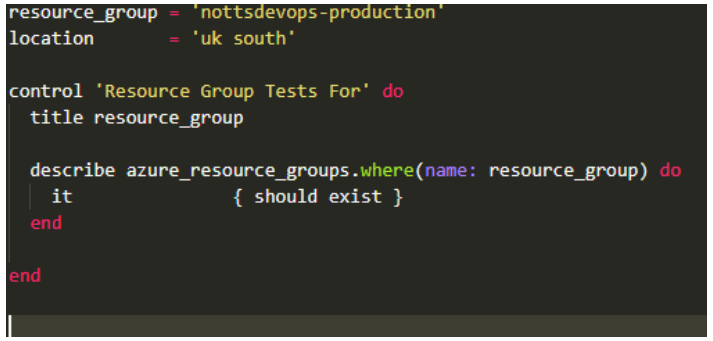
- Inspec-Azure is a resource pack provided by Chef that uses the Azure REST API, to allow you to write tests for resources that you have deployed in Microsoft Azure.

- These tests can be used to validate the Azures resources that were deployed via code using Terraform or even Azure RM templates

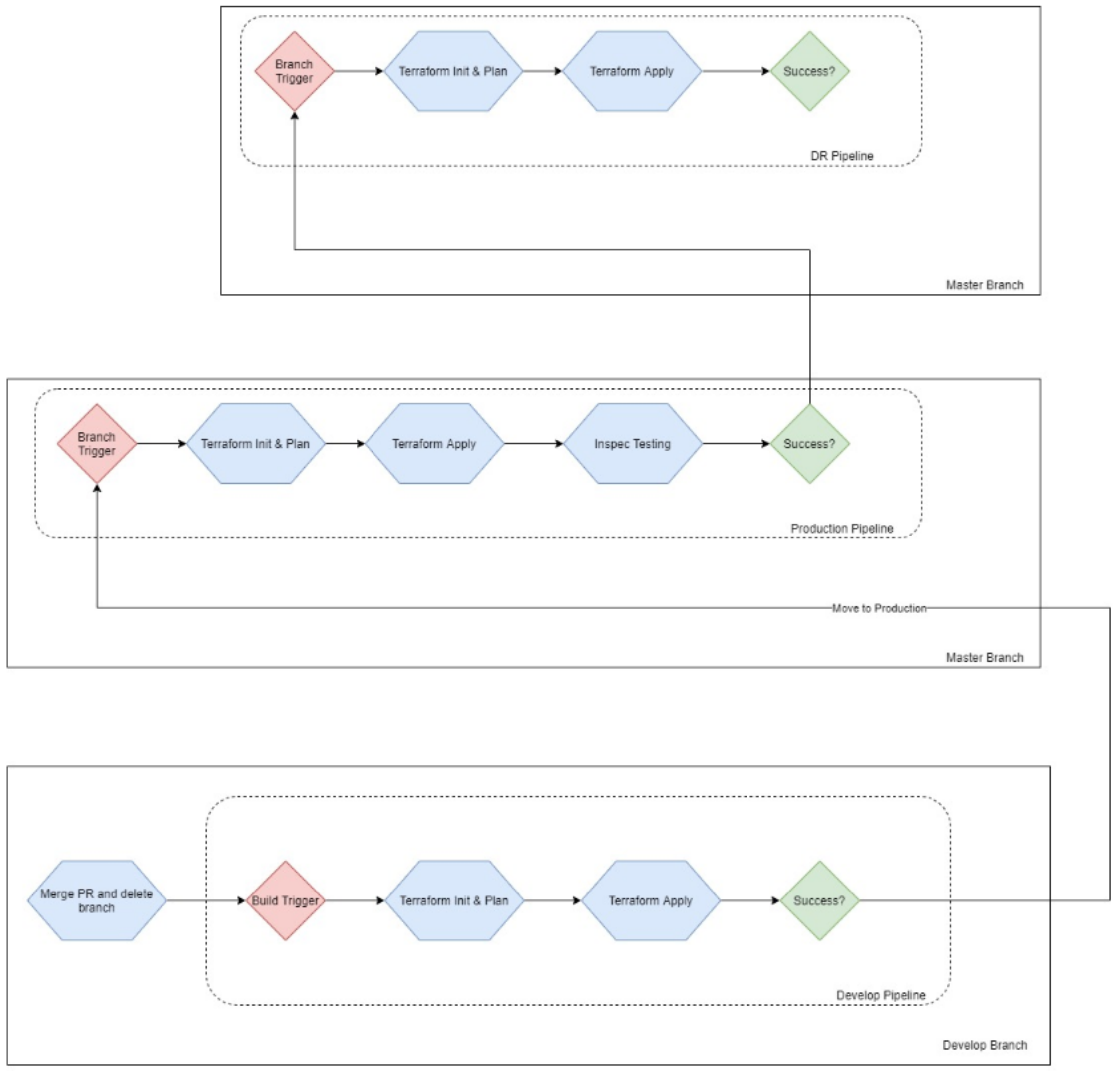
- Inspec is an open source framework that is used for testing and auditing your infrastructure



```
6            6 ● Passed   100%              0s                 0
Total tests  0 ● Failed   Pass percentage   Run duration ⓘ     Tests not reported
+1           0 ● Others                     ↓ 4ms
```



```
resource_group = 'nottsdevops-production'
location       = 'uk south'

control 'Resource Group Tests For' do
  title resource_group

  describe azure_resource_groups.where(name: resource_group) do
    it                { should exist }
  end

end
```



The finish:- Multibranch pipeline!

- From one environment to another
- Pipeline triggers are great
- Test that infrastructure!

Key Takeaways

- Terraform is readable and quite user friendly
- The beginning of CI/CD deployments
- Use of triggers for automation
- Test outside of your pipeline
- Inspec is a fun additional, worth checking out further
- The urge to see even more of Azure DevOps & what it can do! :D

Recommended blog posts

1. **Deploy Terraform using Azure DevOps** https://thomasthornton.cloud/2020/07/08/deploy-terraform-using-azure-devops/

2. **Validating Terraform Code During a Pull request in Azure DevOps** https://thomasthornton.cloud/2020/07/29/validating-terraform-code-during-a-pull-request-in-azure-devops/

3. **Deploying Terraform from Develop to Production consecutively using Azure DevOps** https://thomasthornton.cloud/2020/09/22/deploying-terraform-from-develop-to-production-consecutively-using-azure-devops/

4. **Inspec-Azure in Azure DevOps Pipeline** https://thomasthornton.cloud/2020/08/27/inspec-azure-in-azure-devops-pipeline/

5. **Azure DevOps – 6 Recommendations to help with securing your branches** https://thomasthornton.cloud/2020/09/03/azure-devops-6-recommendations-to-help-with-securing-your-branches/

6. GitHub for demo/slides https://github.com/thomast1906/talks-and-demos/tree/main/Azure-Notts-DevOps

Community Resources

- Azure Blog’s
- YouTube
- User Groups & Virtual Conferences
- GitHub Resources
- CloudFamily.info #cloudfamily
- Community #azurefamily

Questions

- **Twitter: @tamstar1234 https://twitter.com/tamstar1234**
- **LinkedIn: https://bit.ly/34b2kvi**
- **Blog: https://thomasthornton.cloud/**