Getting started with Terraform Pipelines

ThoughtWorks®

Michael Lihs

Infrastructure Consultant

father of 👶 and 👧

🏡 Stuttgart

🚴 cycling

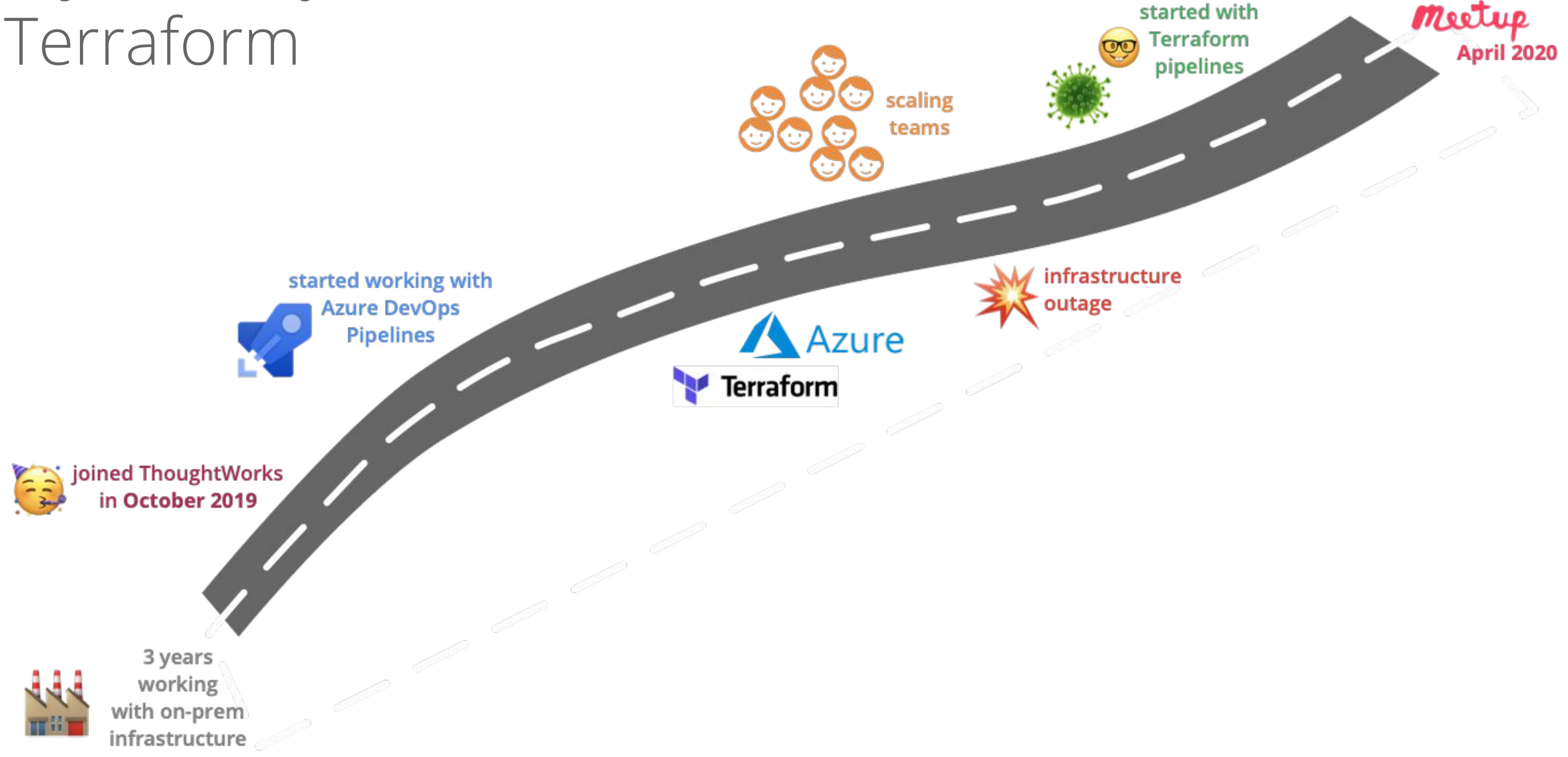
🎸 guitar

🐦 @kaktusmimi

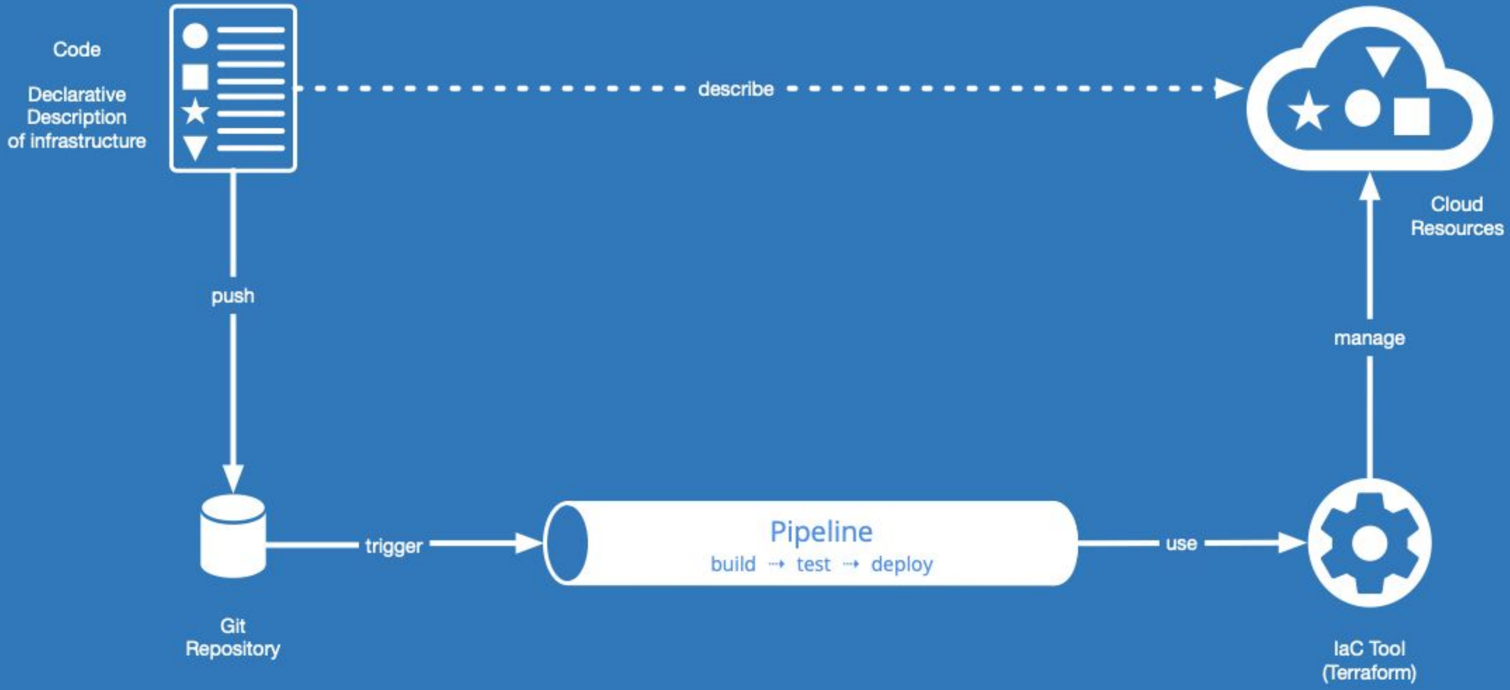
Agenda

- Project Context
- Motivation (why Terraform in Pipelines)
- Terraform in a Nutshell
- Blast Radius & Nuking Infrastructure
- Terraform in Pipelines
- Summary & Outlook

My Journey into Terraform


3 years working with on-prem infrastructure
joined ThoughtWorks in October 2019
started working with Azure DevOps Pipelines
Azure
Terraform
scaling teams
infrastructure outage
started with Terraform pipelines
Meetup
April 2020

Infrastructure as Code



Terraform - Basic Building Blocks

Terraform Binary

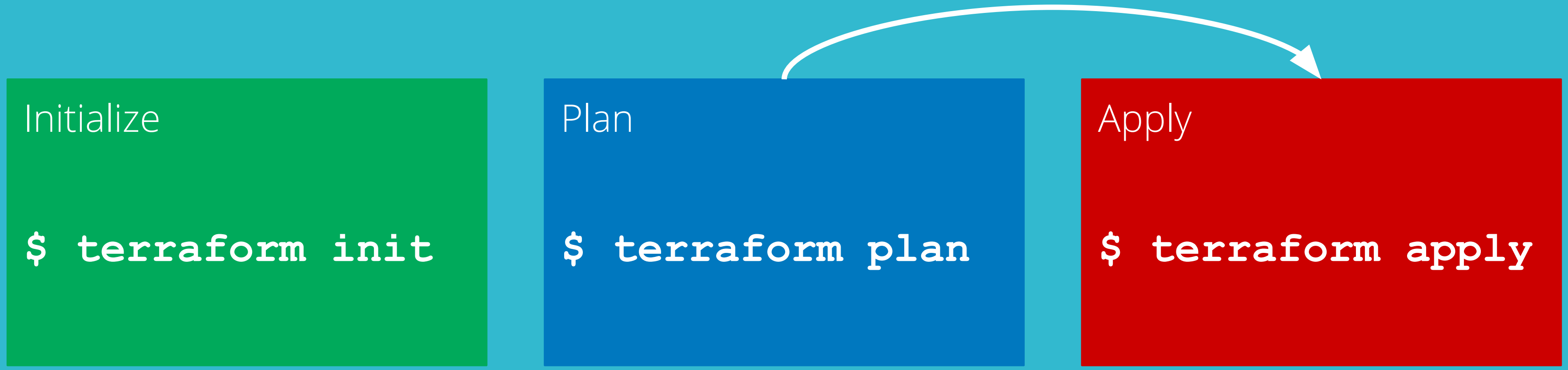
```
$ terraform
```

Resources

Providers

Terraform State

Terraform Workflow



```
$ terraform init
```

```
$ terraform plan
```

```
$ terraform apply
```

Initialize

- install plugins (providers)
- initialize remote state backend

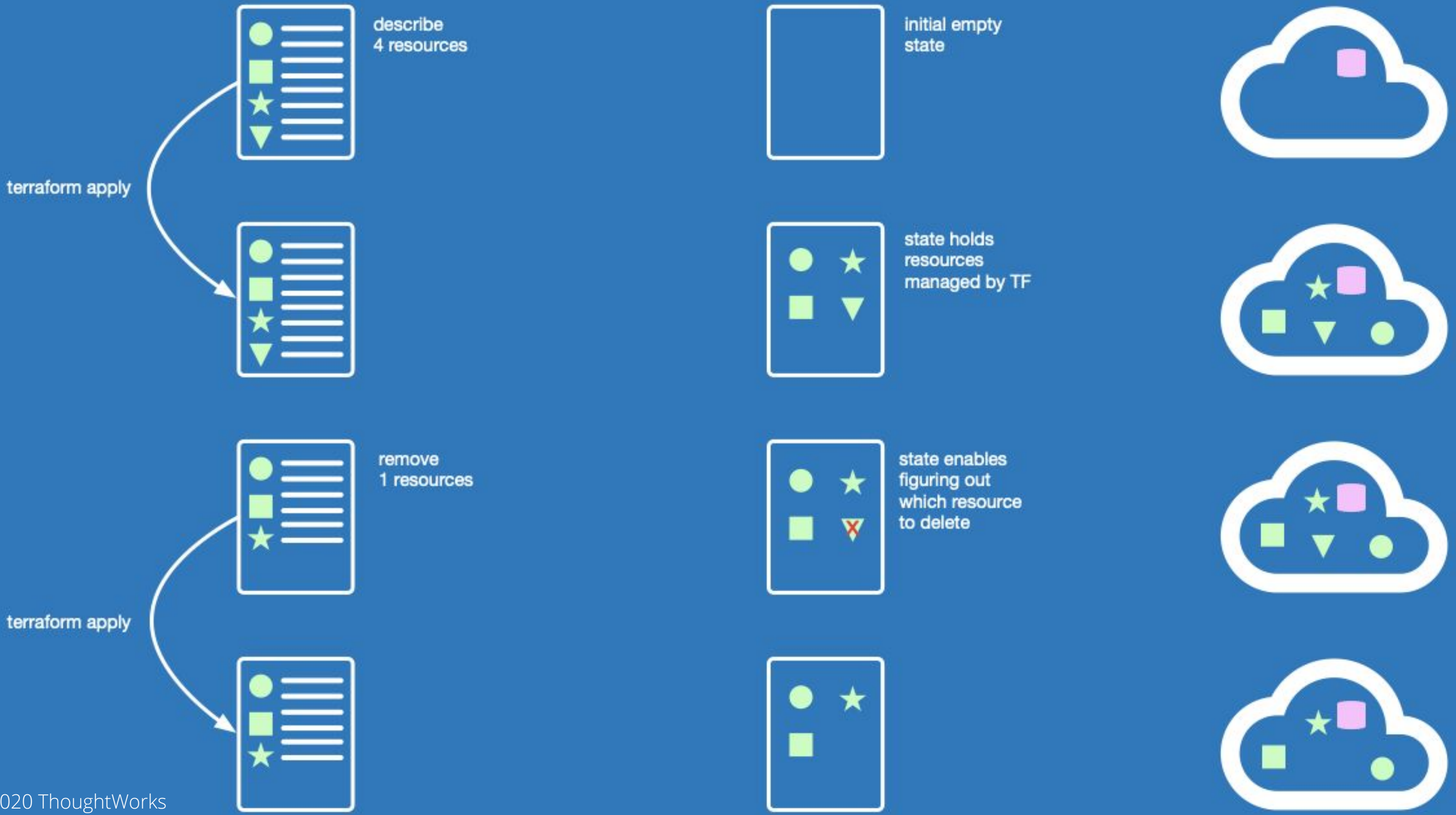
Plan

- compare current state with expected state
- create a plan of necessary changes
- does not yet change any resources

Apply

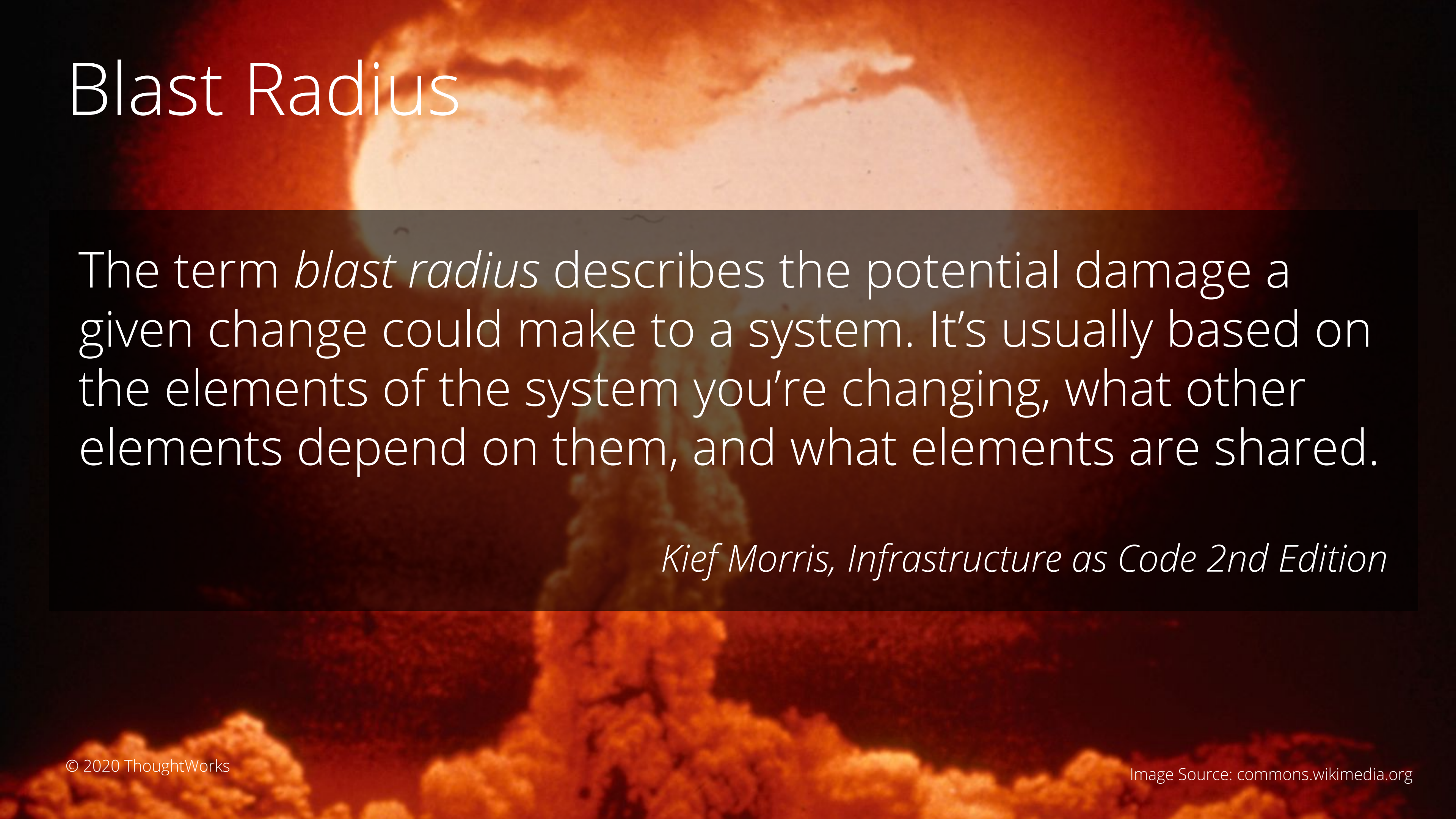
- creates, updates or deletes resources
- makes sure that actual state of infrastructure matches expected state
- updates the state

Terraform State



Challenges with Infrastructure as Code

- Impact of things gone wrong
  - i.e. destruction of stateful resources
  - a.k.a. “Oops - I nuked the database 🤯”
- Infrastructure Code is hard to test
- Parts of our infrastructure might not be immutable
- Dev/Prod parity
- Long feedback cycles



Blast Radius

The term *blast radius* describes the potential damage a given change could make to a system. It's usually based on the elements of the system you're changing, what other elements depend on them, and what elements are shared.

Kief Morris, Infrastructure as Code 2nd Edition

Image Source: commons.wikimedia.org

Post Mortem

BLAME FREE!!!

Question 1

Why did this happen ("5 WHYs")?

Question 2

What did we miss?

Question 3

What can we do to prevent something like this from ever happening again?

Reducing Blast Radius

Staging (Environments)

Testing

Running Terraform in a Pipeline

Multiple Pipelines

Optimize for MTTR

Go for immutable infra if possible

Locking stateful Resources

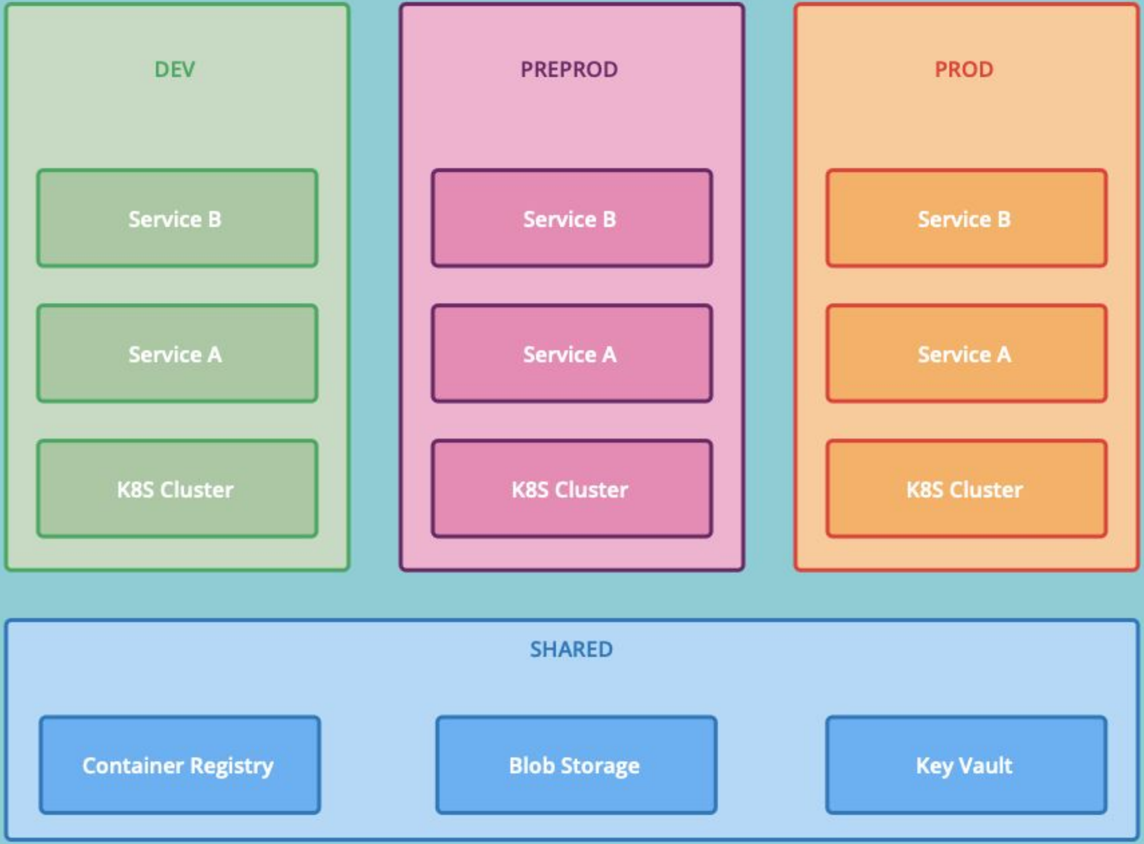
Modularizing TF Code

Splitting State

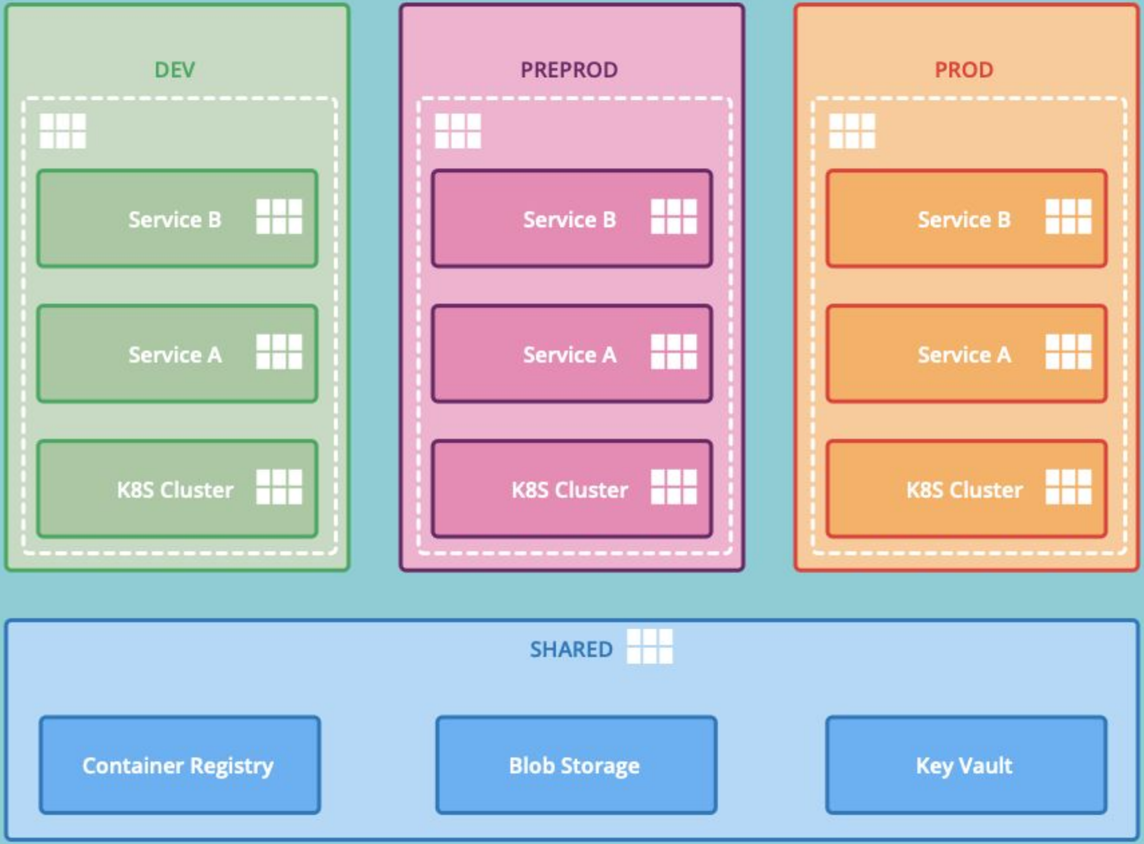
Backup & Recovery

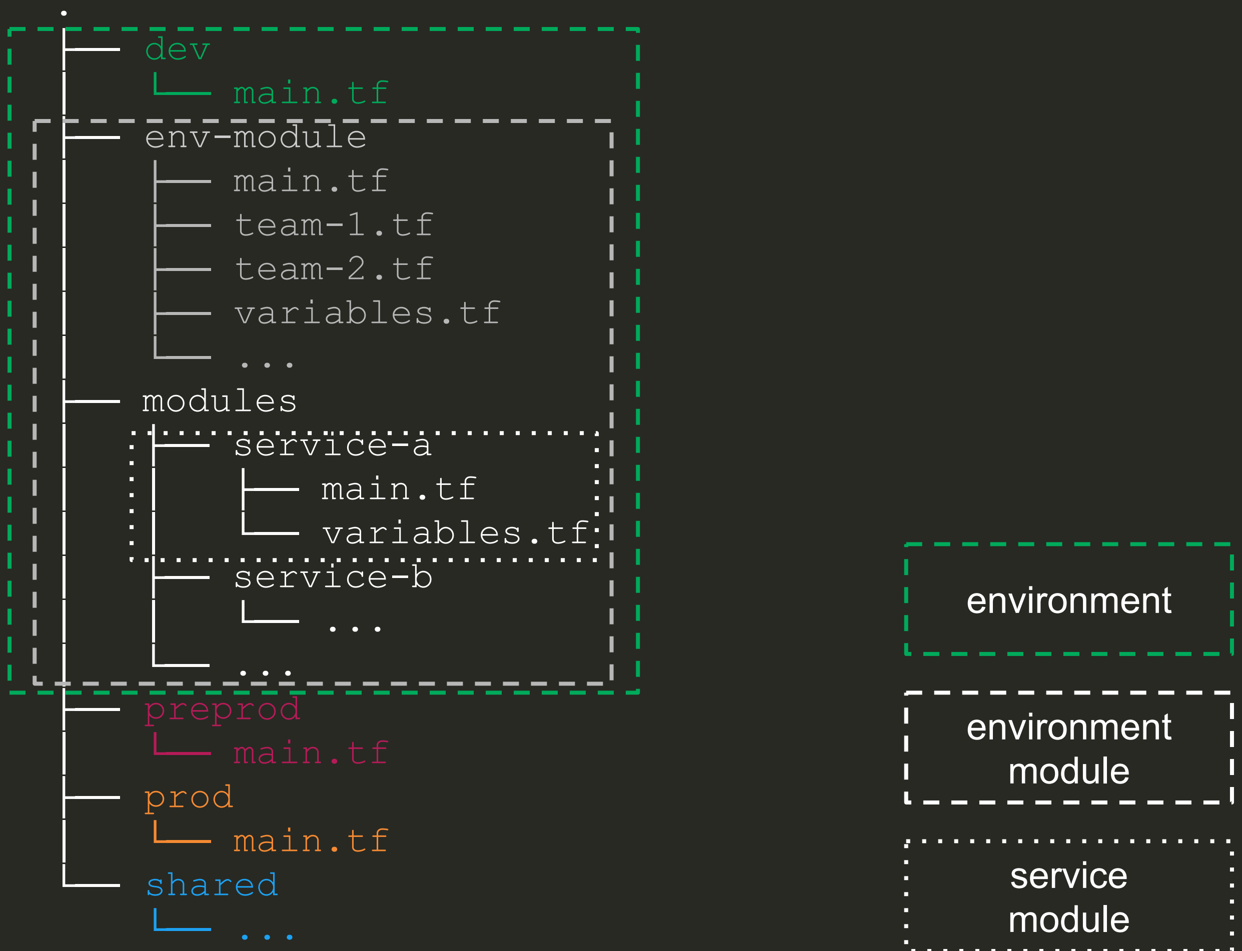
Least Privileges for Pipeline(s)

Staging (Environments)



Modularize Code




```
.
├── dev
│   └── main.tf
├── env-module
│   ├── main.tf
│   ├── team-1.tf
│   ├── team-2.tf
│   ├── variables.tf
│   └── ...
├── modules
│   ├── service-a
│   │   ├── main.tf
│   │   └── variables.tf
│   ├── service-b
│   │   └── ...
│   └── ...
├── preprod
│   └── main.tf
├── prod
│   └── main.tf
└── shared
    └── ...
```

```hcl
# modules/service-a/main.tf

data azurerm_resource_group "default" {
   name = var.resource_group_name
}

resource "azurerm_container_registry" "build_support" {
   name                = "buildsupport"
   resource_group_name = data.azurerm_resource_group.default.name
   location            = data.azurerm_resource_group.default.location
   admin_enabled       = true
   sku                 = "Standard"
}
```

```
# env-module/team-1.tf

locals {
   tag_maintainer_team_1 = {
     maintainer = "Team 1"
   }
}

module "service-a" {
   source = "../modules/service-a"

   stage                   = var.stage
   default_tags            = merge(local.module_tags, local.tag_maintainer_team_1)
   resource_group_name     = azurerm_resource_group.project.name
}

module "service-a" {
   source = "../modules/service-b"
   # ...
}
```

```
# dev/main.tf

module "environment" {
   source = "../env-module"

   stage  = "dev"

   dev_user_group_id = "xxxx-xxxx-xxxx-xxxx"
   pipline_sp_id     = "xxxx-xxxx-xxxx-xxxx"

   k8s_public_ip = "10.20.30.40"
}
```

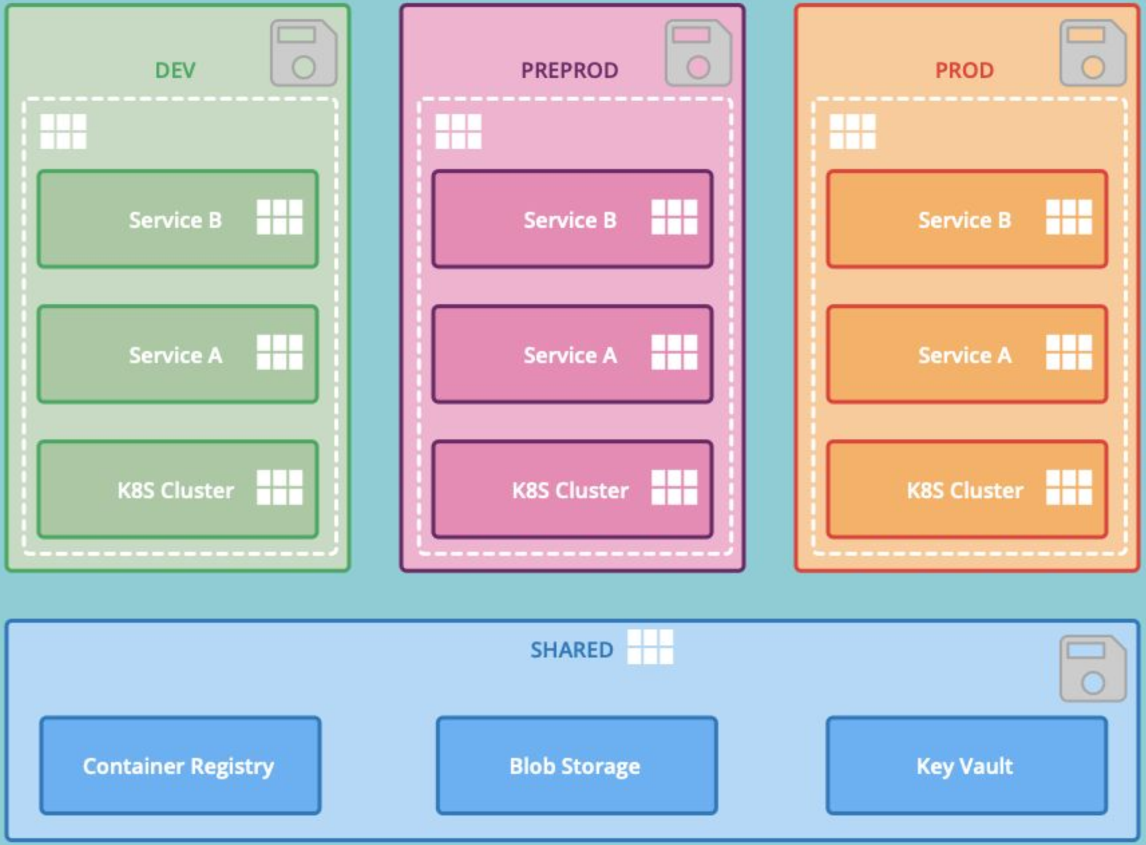
HashiCorp
Terraform

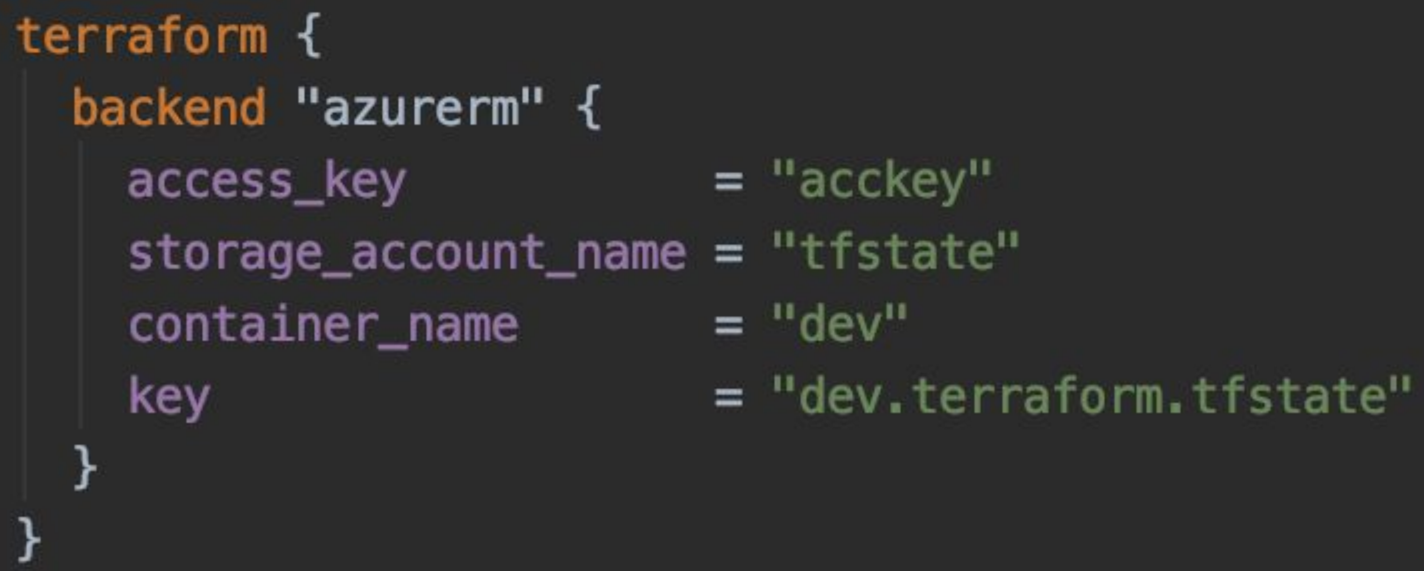
Structuring HashiCorp Terraform Configuration for Production

MAR 27 2020 | XANDER GRZYWINSKI

When you start learning to use HashiCorp Terraform, you might start with one configuration file containing all of your infrastructure as code. As you learn more, you start to share and collaborate on those configuration files with peers or teams. Eventually, multiple team

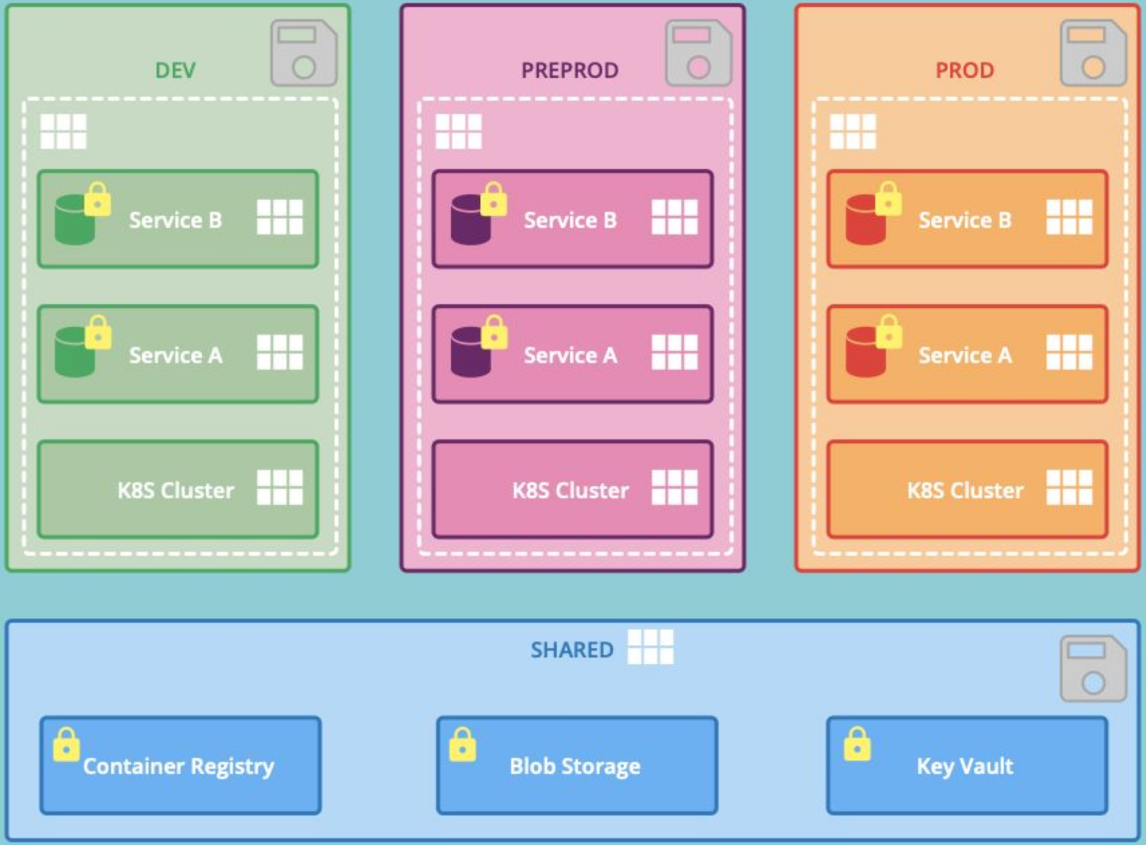
Split Terraform State





```
terraform {
  backend "azurerm" {
    access_key           = "acckey"
    storage_account_name = "tfstate"
    container_name       = "dev"
    key                  = "dev.terraform.tfstate"
  }
}
```

Locking Stateful Resources

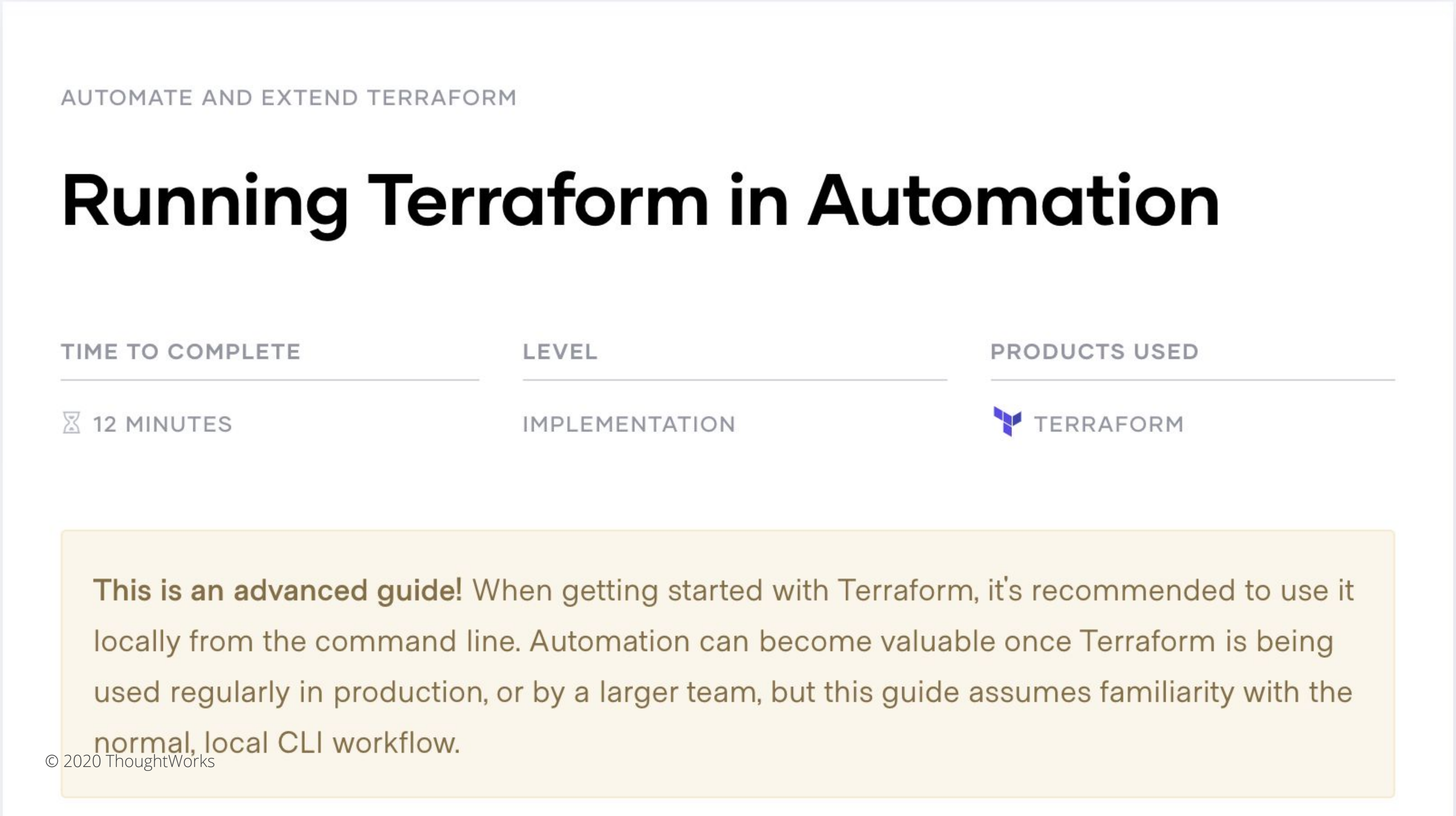


```
resource "azurerm_key_vault" "vault" {
   name = "team-vault"
   //...
}

resource "azurerm_management_lock" "vault" {
   name = "team-vault-lock"
   scope = azurerm_key_vault.vault.id
   lock_level = "CanNotDelete"
}
```

Running Terraform in Pipelines

- Provide familiar workflow for developers
  - code ⇢ push ⇢ build ⇢ test ⇢ deploy
- Enforce staging
  - no **`terraform apply`**'s to PROD without testing on DEV
- No more forgotten commits
- Traceability of changes (audits)



AUTOMATE AND EXTEND TERRAFORM

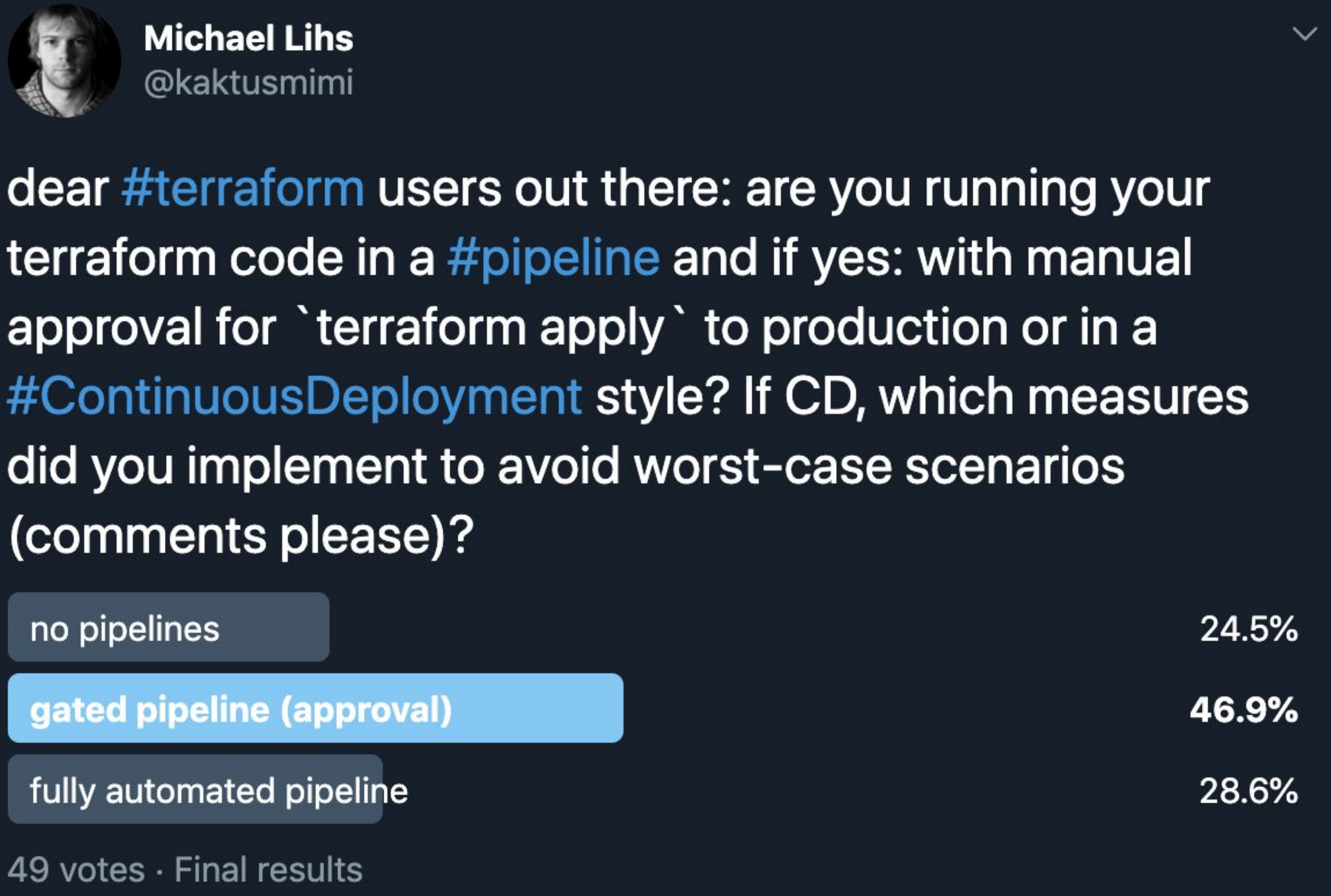
Running Terraform in Automation

TIME TO COMPLETE	LEVEL	PRODUCTS USED
12 MINUTES	IMPLEMENTATION	TERRAFORM

This is an advanced guide! When getting started with Terraform, it's recommended to use it locally from the command line. Automation can become valuable once Terraform is being used regularly in production, or by a larger team, but this guide assumes familiarity with the normal, local CLI workflow.

Considerations when running Terraform in Pipelines

- Disable all manual input
  - `terraform init|plan|apply` **`-input=false`**
- Archive Terraform plan
  - `terraform plan` **`-out=tfplan`**
  - `terraform apply` **`tfplan`**
- Package `.terraform` folder and restore at the same absolute path
  - Pre-install plugins `-plugin-dir=...`
- Turn on auto-approval
  - `terraform apply -auto-approve tfplan`



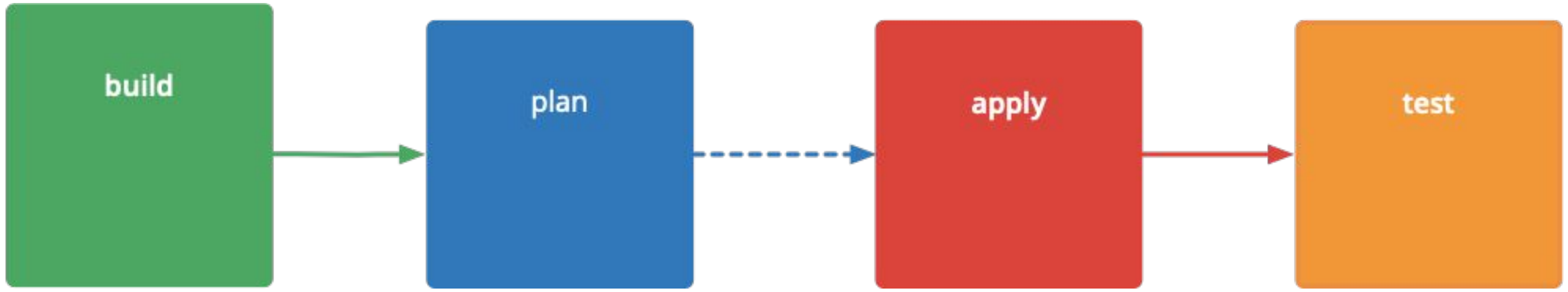
Michael Lihs
@kaktusmimi

dear #terraform users out there: are you running your terraform code in a #pipeline and if yes: with manual approval for `terraform apply` to production or in a #ContinuousDeployment style? If CD, which measures did you implement to avoid worst-case scenarios (comments please)?

no pipelines	24.5%
gated pipeline (approval)	**46.9%**
fully automated pipeline	28.6%

49 votes · Final results

Basic Terraform Pipeline



Build Stage

Checkout	
Install Terraform	*version exists*
`terraform init`(all environments)	*providers exist, remote backend works*
`terraform validate`(all environments)	*syntactical correctness*
Publish Terraform package	*promotion of infrastructure code*

terraform validate

```
Regarding validate step (took 47s to catch this)
Error: Missing required argument


 on main.tf line 20, in module "environment-module":

 20: module "environment-module" {



The argument "core_developers_group_id" is required, but no definition was found.
```

Plan Stage

Download Terraform package	*use validated Terraform package*
`terraform init -get-plugins=false`	*re-initialize Terraform environment*
`terraform plan -out=plan.tfplan`	*create plan for manual approval*
Publish Terraform package + plan	*add Terraform plan to package*

```
function task_tf_plan {
    terraform plan -lock=false -out="plan.tfplan" \
        -detailed-exitcode > /dev/null
    OUT=$?

    if [ $OUT -eq 0 ];then
        echo "No changes. Infrastructure is up-to-date!"
    elif [ $OUT -eq 1 ];then
        echo "Terraform planned has failed!"
        exit 1
    elif [ $OUT -eq 2 ];then
        echo "Changes have been noticed!"
        terraform show "plan.tfplan"
    else
        echo "Terraform planned has failed!"
        exit 1
    fi
}
```

Apply Stage

Gated stage

Download Terraform package	*download validated package & plan*
`terraform init -get-plugins=false`	*re-initialize Terraform environment*
`terraform apply -auto-approve tfplan`	*apply planned changes*
tag version as applied to ENV	*assure traceability*

Test Stage

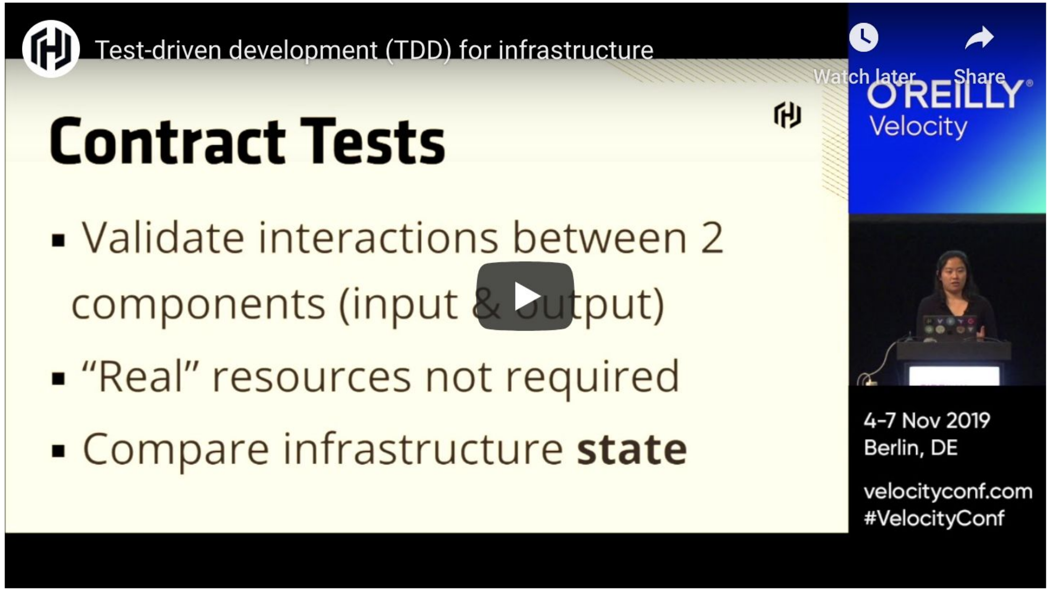
Download Terraform package	*use validated Terraform package*
Run smoke tests	*validate system works end 2 end*

```
function test {
   az aks get-credentials --resource-group my-resource-group \
        --name swf-nonprod --output yaml --file kubeconfig.yaml \
        --overwrite-existing
   export KUBECONFIG=$PWD/kubeconfig.yaml
   kubectl cluster-info
}
```

Test-Driven Development (TDD) for Infrastructure

DEC 10, 2019

Learn how to adapt TDD to deploying and configuring infrastructure.

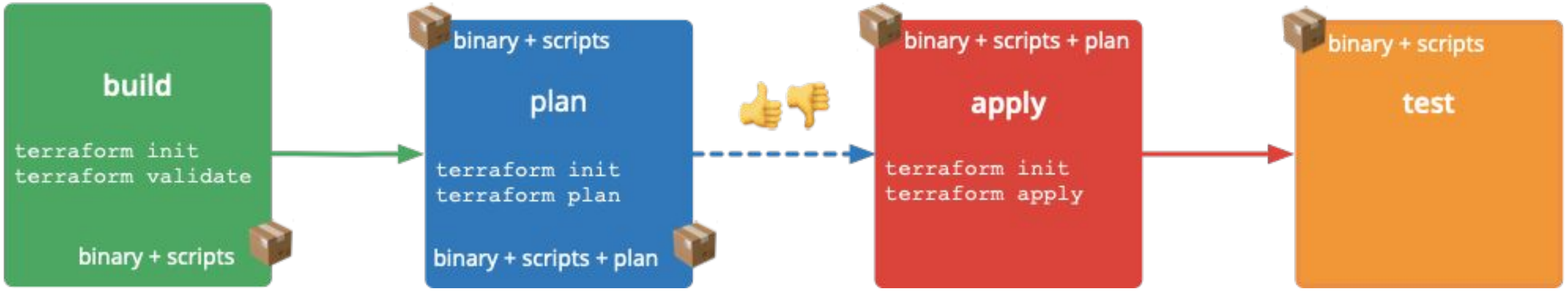


Test-driven development (TDD) for infrastructure

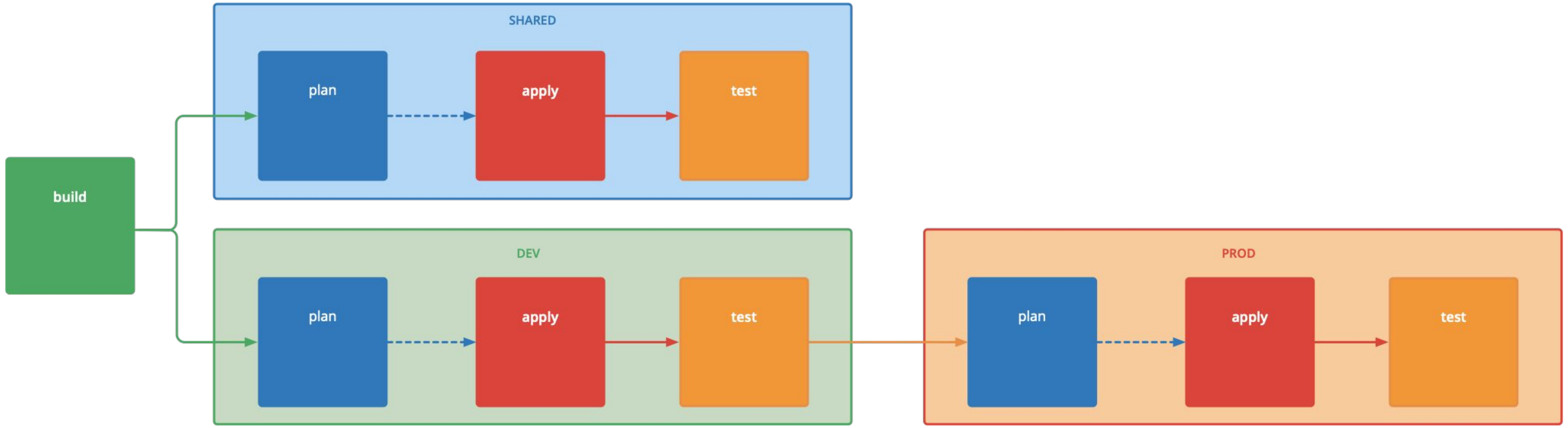
Contract Tests

- Validate interactions between 2 components (input & output)
- "Real" resources not required
- Compare infrastructure **state**

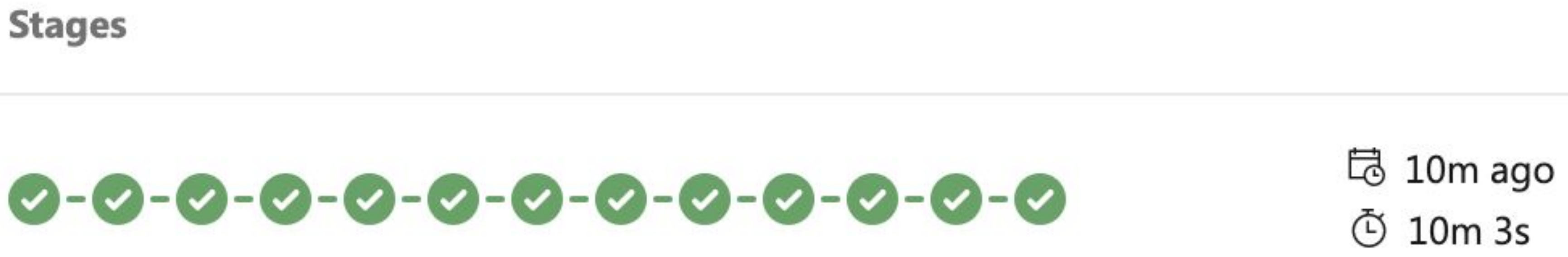
Basic Terraform Pipeline



Adding Environments (Staging)



Feedback Cycles



Stages

10m ago
10m 3s

Declarative Pipeline & ./do Script

```yaml
- deployment: ApplyShared
  dependsOn: CheckSharedChanges
  strategy:
    runOnce:
      deploy:
        steps:
          - task: AzureCLI@2
            inputs:
              azureSubscription: $(serviceConnection)
              scriptType: 'bash'
              workingDirectory: automated/shared
              scriptLocation: 'inlineScript'
              inlineScript: |
                ./do tf-init
                ./do tf-apply
```

```bash
function task_tf_init {
  access_key=$(az storage account keys list \
                 --resource-group "${resource_group}" \
                 --account-name ${storage_account} \
                 --subscription ${subscription_id} \
                 --query '[0].value' -o tsv)
  ../terraform init -get-plugins=false \
                 -backend-config="access_key=$access_key"
}

function task_tf_apply {
    rm -f service-principal.json
    az keyvault secret download --name "${pipeline_service_principal}" \
        --vault-name "${azure_vault}" --file service-principal.json
    export ARM_CLIENT_ID="$(cat service-principal.json | jq '.appId' -r)"
    export ARM_CLIENT_SECRET="$(cat service-principal.json | jq '.password' -r)"
    export ARM_SUBSCRIPTION_ID="${subscription_id}"
    export ARM_TENANT_ID="$(cat service-principal.json | jq '.tenant' -r)"
    ../terraform apply -auto-approve ${BUILD_BUILDNUMBER}.tfplan
}
```

Summary

- Once you have Infrastructure-as-Code, pipelines are a good next step
- Provide familiar workflow to your developers for infrastructure code
- Prepare your Terraform code to run in automation
- Reduce blast radius as far as possible
- Go one step at a time, check & iterate

Acknowledgements

A big **thank you** to all the people who helped me on this journey

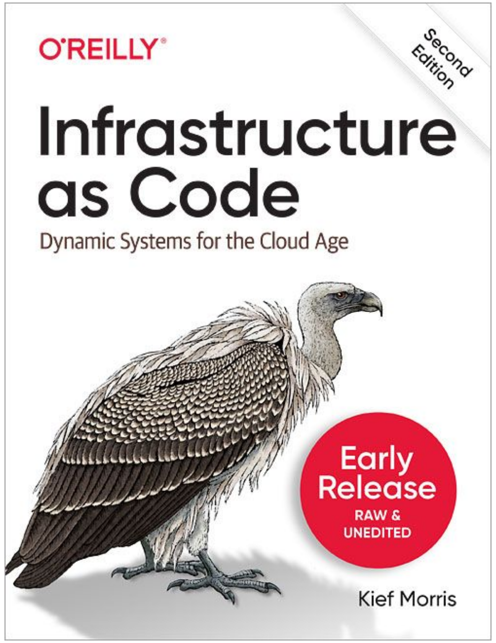
- **Jonathan Nowak** who was a great mentor for Terraform and Azure
- **Tim Fletcher** for being a great onboarding buddy and the father of many ideas in this presentation
- **Alaa Mansour** for reviewing the presentation and improving it a lot
- The ThoughtWorks infrastructure community for all the good discussions

Questions?!?

Ping me on Twitter any time:

@kaktusmimi

Recommended Books



O'REILLY

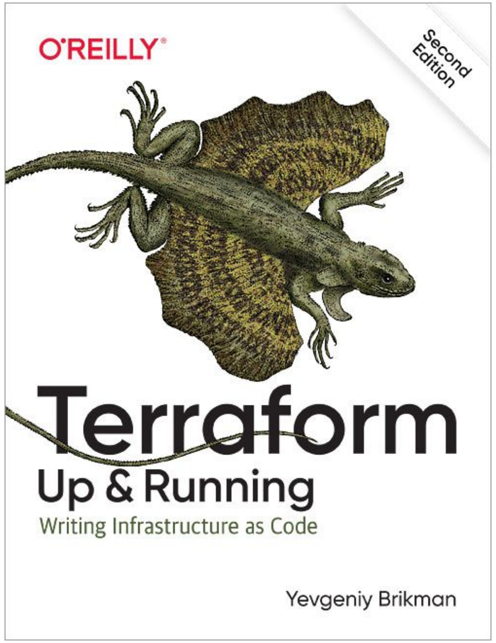
Second Edition

Infrastructure as Code

Dynamic Systems for the Cloud Age

Early Release RAW & UNEDITED

Kief Morris



O'REILLY

Second Edition

Terraform

Up & Running

Writing Infrastructure as Code

Yevgeniy Brikman

Further Resources

- https://www.hashicorp.com/blog/structuring-hashicorp-terraform-configuration-for-production/
- https://learn.hashicorp.com/terraform/development/running-terraform-in-automation
- https://www.hashicorp.com/resources/test-driven-development-tdd-for-infrastructure/
- https://medium.com/faun/a-ci-cd-journey-with-azure-devops-and-terraform-part-2-524144511294

THANK YOU

Michael Lihs
Infrastructure Consultant
michael.lihs@thoughtworks.com

ThoughtWorks®