February 6th-7th

nic

20/20 VISION

Oslo Spektrum


Allianz

Managing your Azure infrastructure using Terraform

Jan Egil Ring
Microsoft MVP | Lead Architect Infrastructure, Crayon

@JanEgilRing

Agenda

- Getting started with Infrastructure as Code
- Introduction to Terraform
- Terraform vs Azure Resource Manager (ARM) templates
- Demos
  - Basic resource creation
  - Complex resource creation
  - Organizing configurations
  - Deployment pipeline integration

Getting Started with Infrastructure as Code

The journey of an Azure User

Crawl

Manually create and manage resources in **Azure Portal** or **QuickStart**

Walk

Automate deployment of Azure resources using **Infrastructure as Code**.

E.g. **ARM templates**, PowerShell or Terraform.

Run

Orchestrate deployment of Azure resources using CI/CD tools.

E.g. **Azure DevOps** or Jenkins.

“A good Admin is a lazy Admin”

What is Infrastructure as Code (IaC)

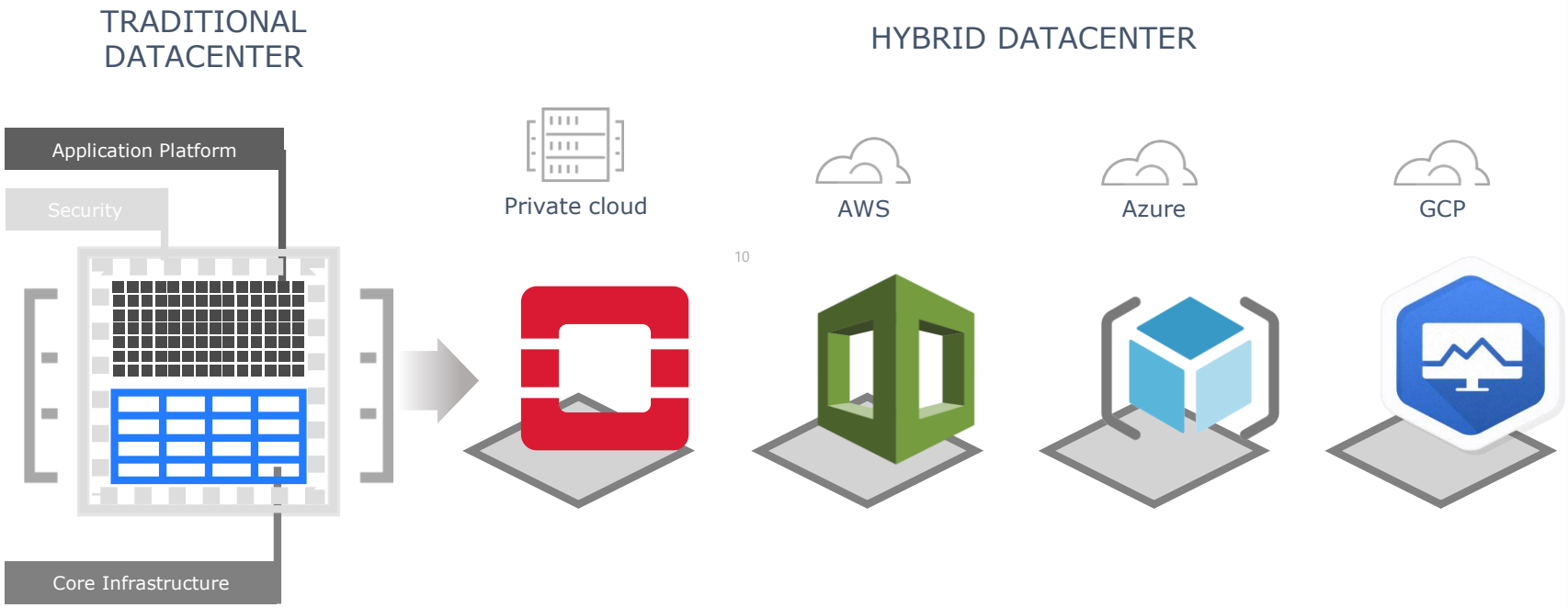
- Build the infrastructure for an App all at once through automation
- Not just for Cloud, Software Defined Data Center
- Embedded Documentation
- Source Control
- Flexible Build Process

Infrastructure as Code

- ☑ Reproducible Environments
- ☑ Automation – CI/ CD
- ☑ Trackable – Git
- ☑ Language - HCL
- ☑ Workflow
- ☑ Providers

- ☒ Apply same config across clouds

Multi-cloud infrastructure transition



Methods for deploying resources 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

@nepeters

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"
}
```

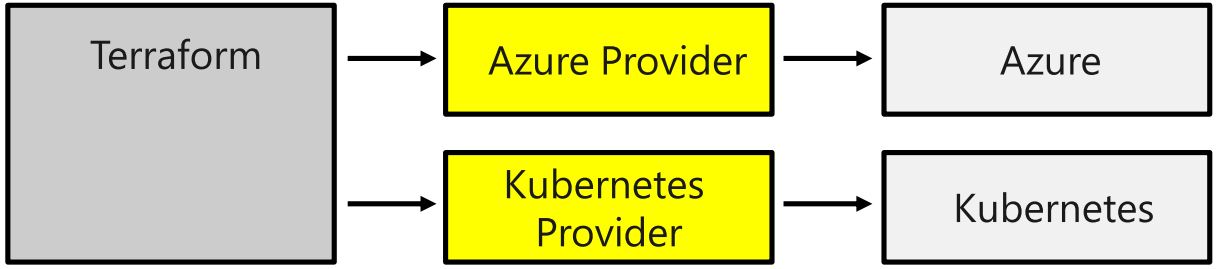
Resource Group

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)



Providers

Defines how Terraform will interact with:

Cloud	Infrastructure	Network	Version Control	Monitoring	Database	Etc.
Azure	Kubernetes	DNS	GitHub	DataDog	InfluxDB	
AWS	Docker	Cloudflare	GitLab	Grafana	MySQL	
Google	Rancher	HTTP	Bitbucket	PagerDuty	PostgreSQL	
AliCloud						
OpenStack						

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

Demo 1 – Basic Resource Creation

Terraform 0.12

Announcing Terraform 0.12

MAY 22 2019 | THE TERRAFORM TEAM

We are very proud to announce the release of Terraform 0.12.

Terraform 0.12 is a major update that includes dozens of improvements and features spanning the breadth and depth of Terraform's functionality.

Some highlights of this release include:

- **First-class expression syntax**: express references and expressions directly rather than using string interpolation syntax.
- **Generalized type system**: use lists and maps more freely, and use resources as object values.
- **Iteration constructs**: transform and filter one collection into another collection, and generate nested configuration blocks from collections.
- **Structural rendering of plans**: plan output now looks more like configuration making it easier to understand.
- **Context-rich error messages**: error messages now include a highlighted snippet of configuration and often suggest exactly what needs to be changed to resolve them.

The full release changelog can be found here.

```
# Example for older versions of Terraform; not valid for v0.12
example = ["${var.any_list}"]
```

```
example = var.any_list
```

https://www.hashicorp.com/blog/announcing-terraform-0-12

Variables and output

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

```
$ TF_VAR_azureclientid = “00000000-0000-0000-0000-00000000000”

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”]
}
```

Demo 2 – Complex Resource Creation

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>
}
```

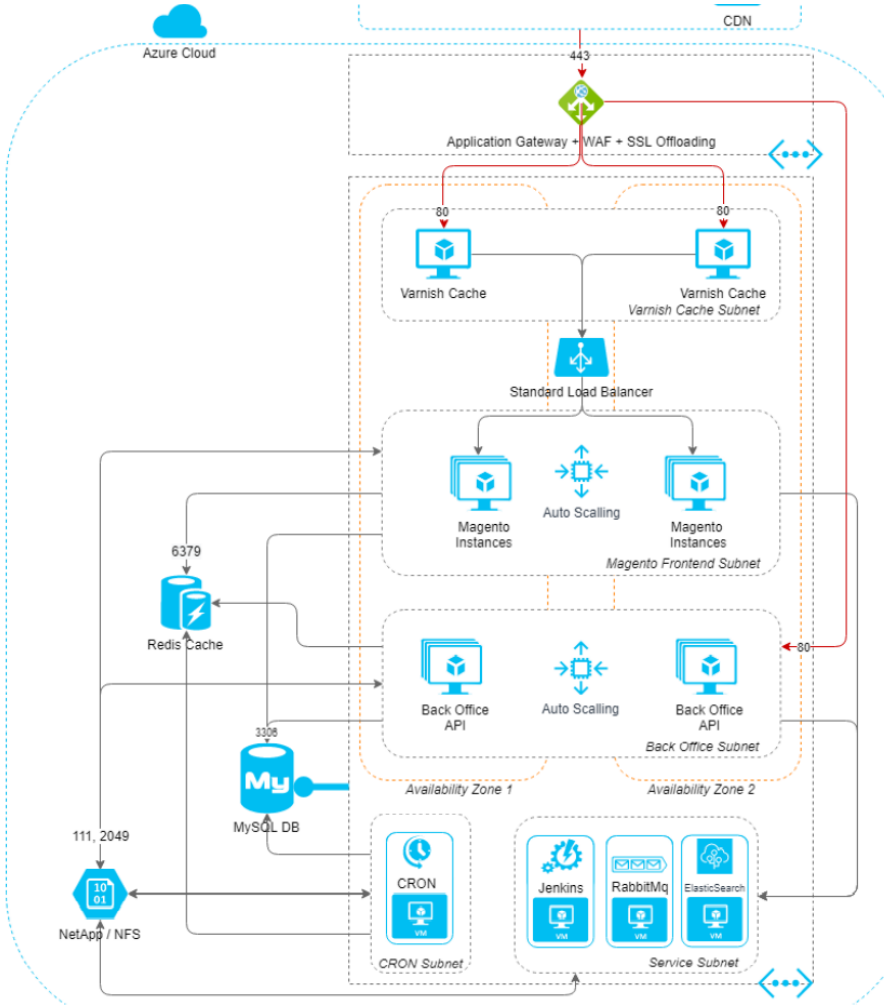
```
"${data.terraform_remote_state.azurerm.resource-group}"
```

Demo 3 – Remote State and Data Source

Organizing your configurations

- Real world example
- Azure infrastructure for an e-commerce platform
- 3 environments
  - Production
  - Pre-production
  - Integration
- Shared infrastructure

https://www.terraform.io/docs/cloud/guides/recommended-practices/part1.html


CDN
Azure Cloud
443
Application Gateway + WAF + SSL Offloading
80
80
Varnish Cache
Varnish Cache
Varnish Cache Subnet
Standard Load Balancer
Magento Instances
Auto Scalling
Magento Instances
Magento Frontend Subnet
6379
Redis Cache
80
Back Office API
Auto Scalling
Back Office API
Back Office Subnet
3306
MySQL DB
Availability Zone 1
Availability Zone 2
111, 2049
NetApp / NFS
CRON
CRON Subnet
Jenkins
RabbitMq
ElasticSearch
Service Subnet

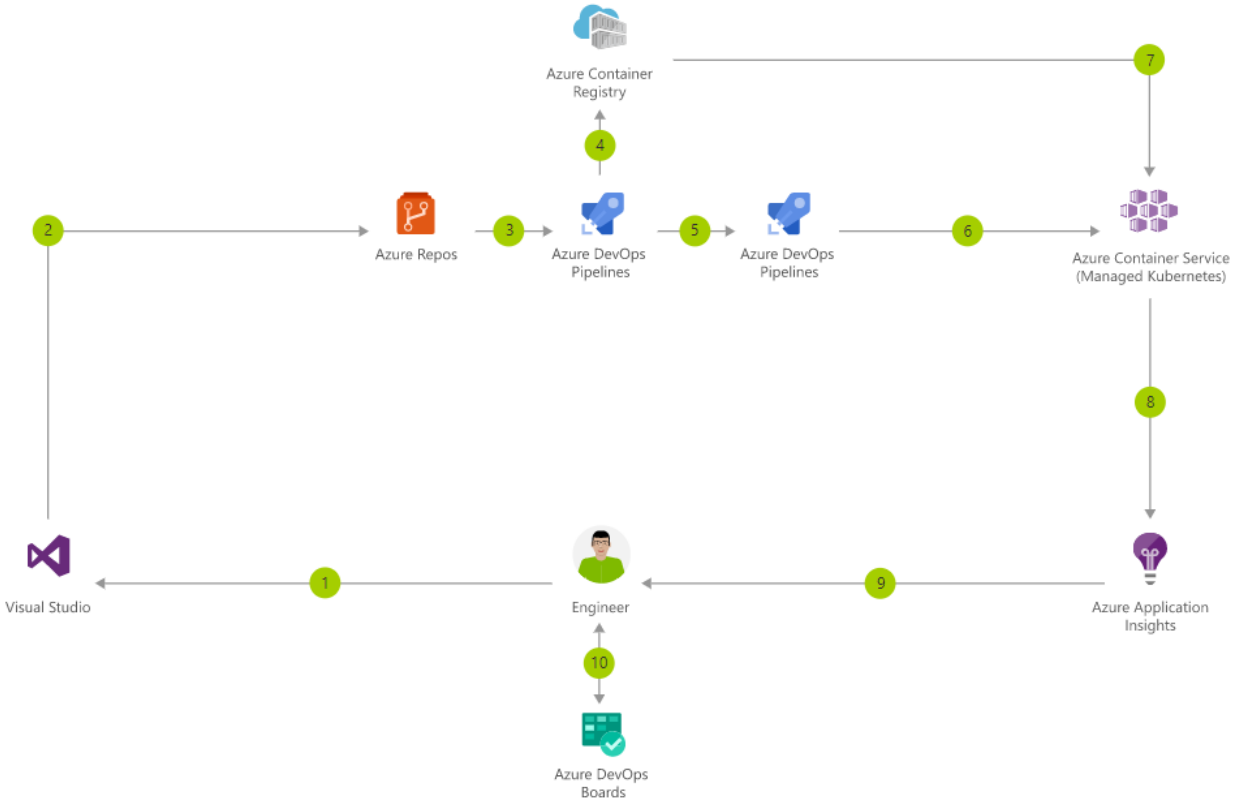
Demo 4 – Organizing configurations

Running Terraform in deployment pipelines

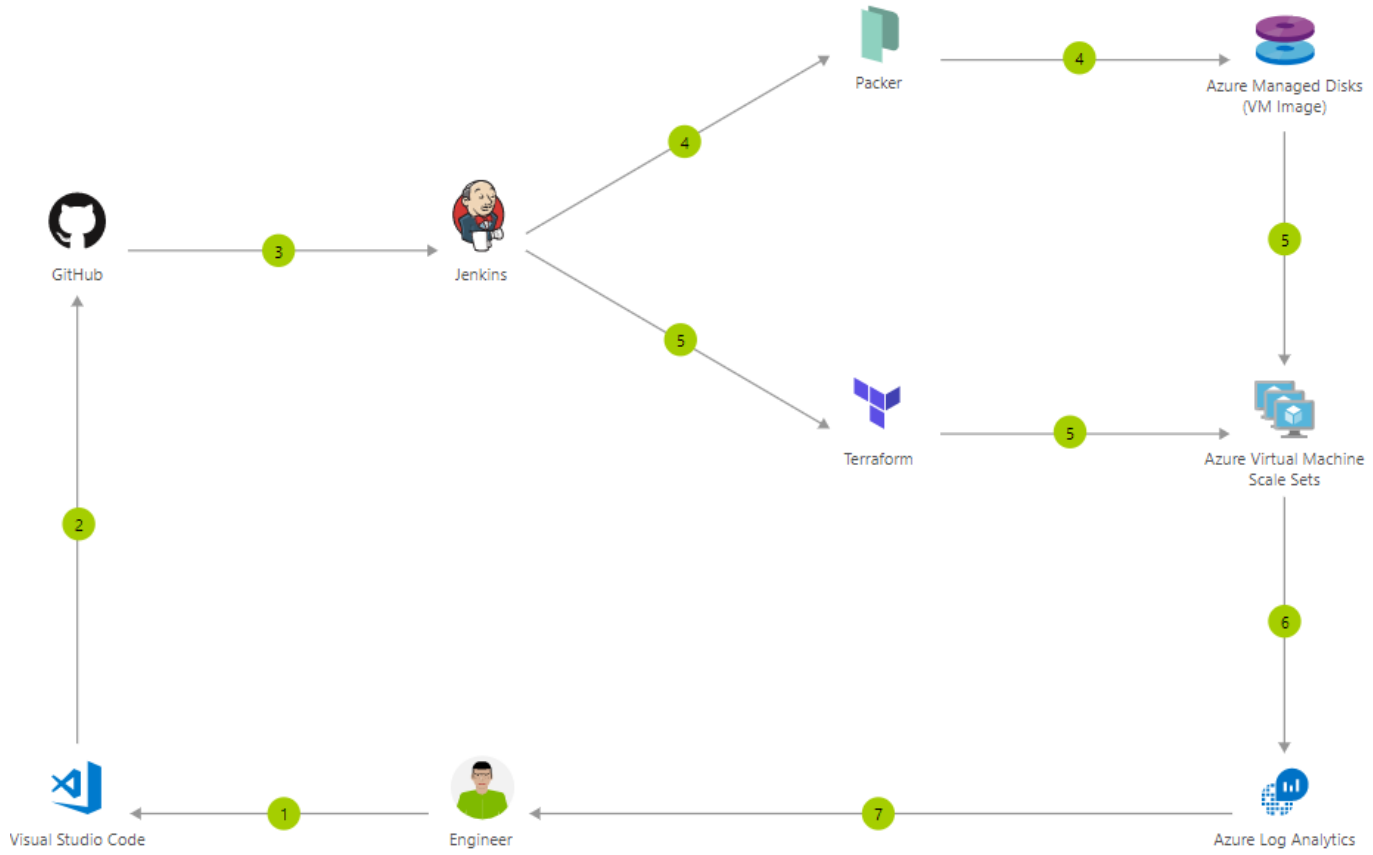
1. Initialize the Terraform working directory.
2. Produce a plan for changing resources to match the current configuration.
3. Have a human operator review that plan, to ensure it is acceptable.
4. Apply the changes described by the plan.

Running Terraform in Automation
https://learn.hashicorp.com/terraform/development/running-terraform-in-automation

Example 1: CI/CD for Containers

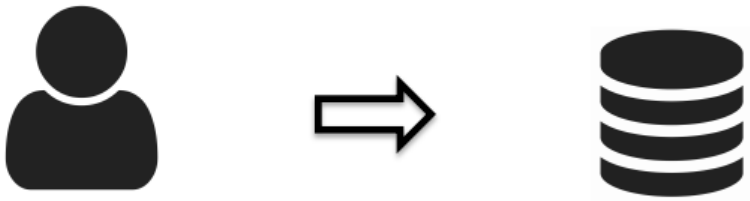


Example 2: Immutable Infrastructure CI/CD using Jenkins and Terraform on Azure Virtual Machines

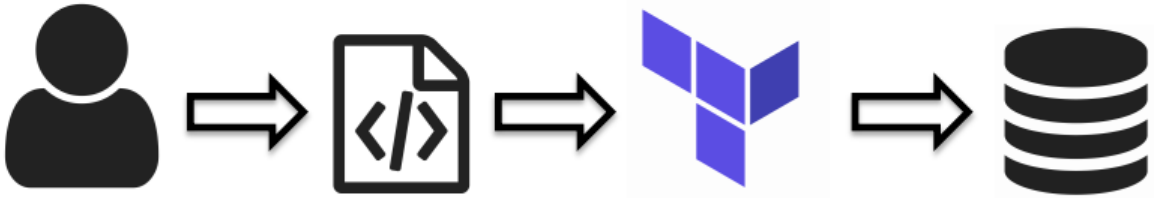


Demo 5 – Azure DevOps Integration

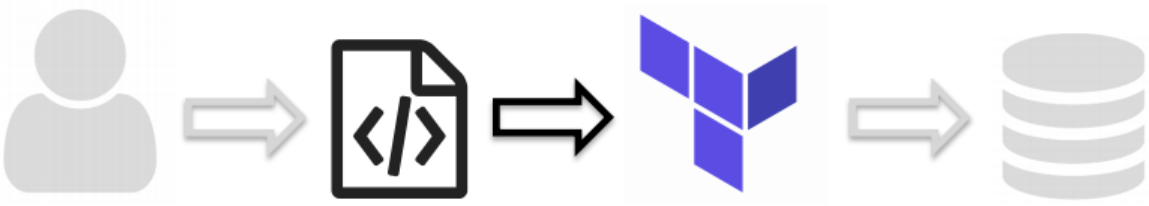
https://www.youtube.com/watch?v=RTEgE2lcyk4



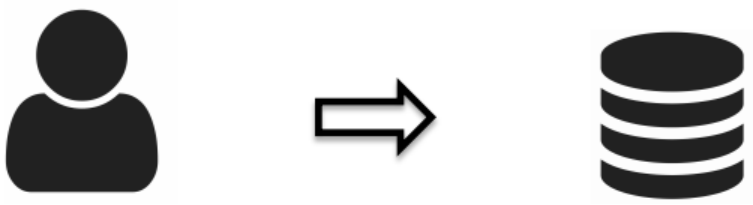
Old way: make changes directly and manually



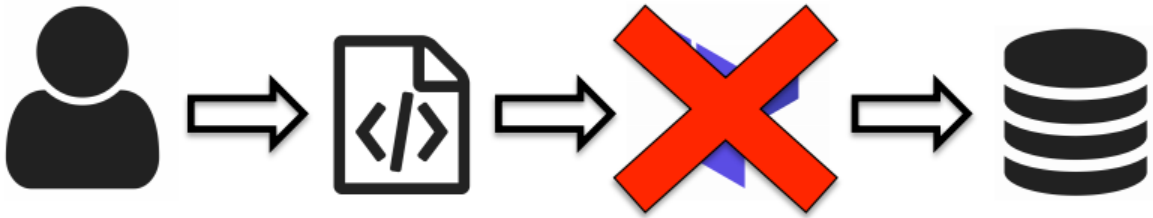
New way: make changes indirectly and automatically



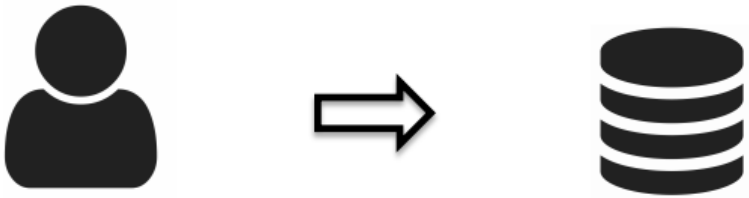
Learning these takes time



More time than making a change directly…

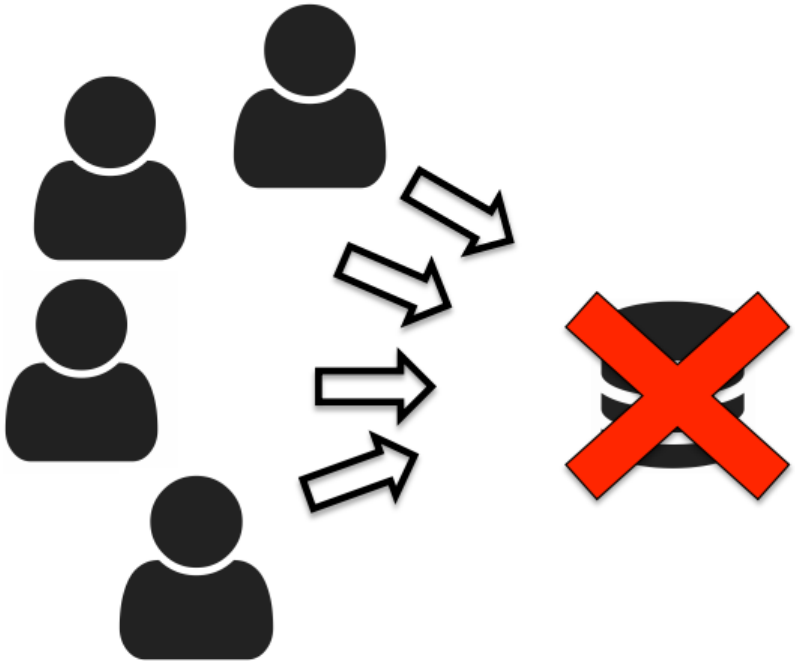


And the next person to try to use it will get errors

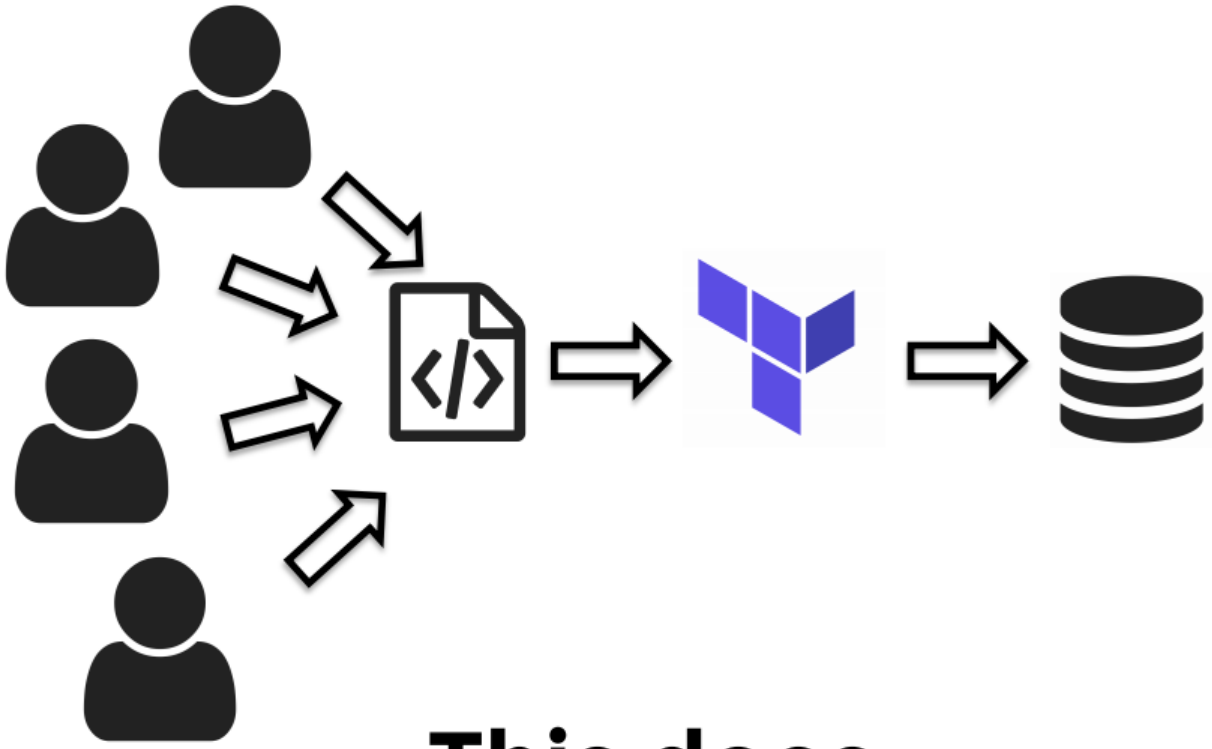


So then they'll fall back and make manual changes

nic

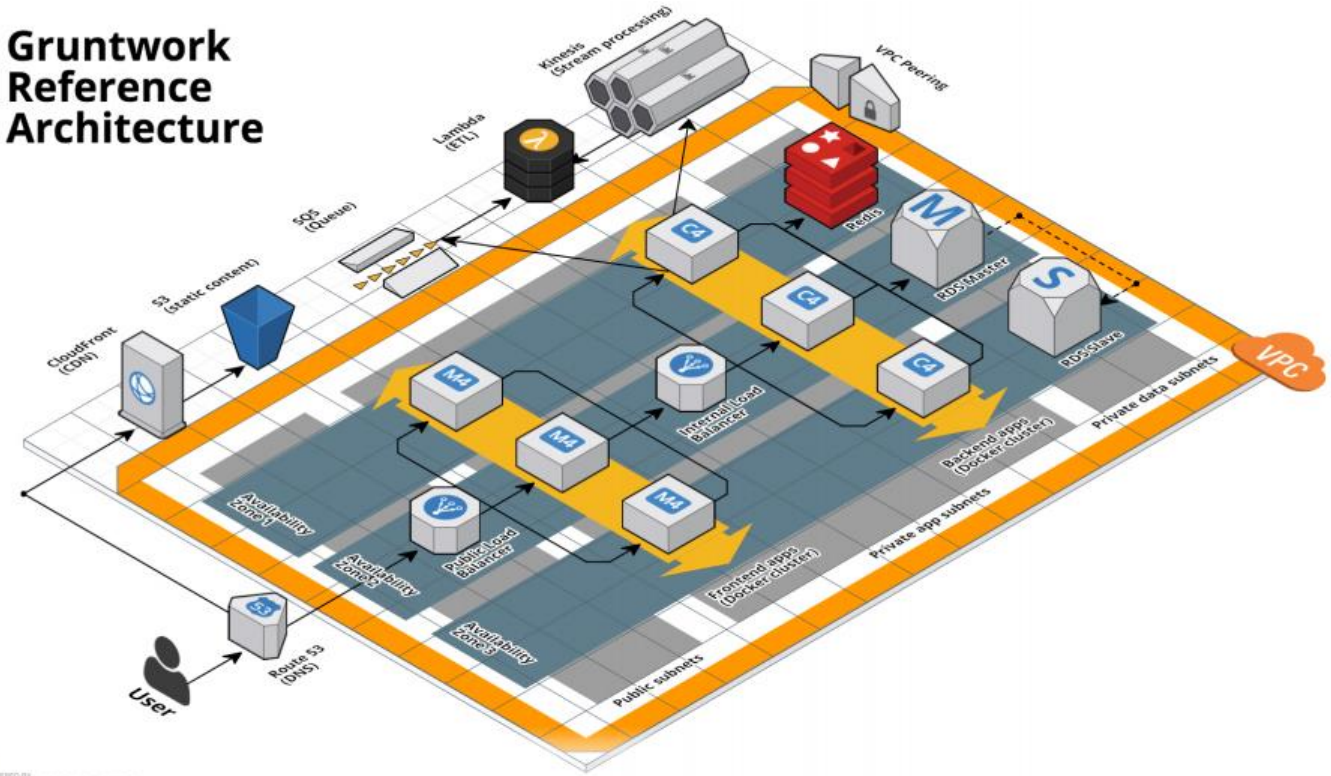


But making manual changes does not scale



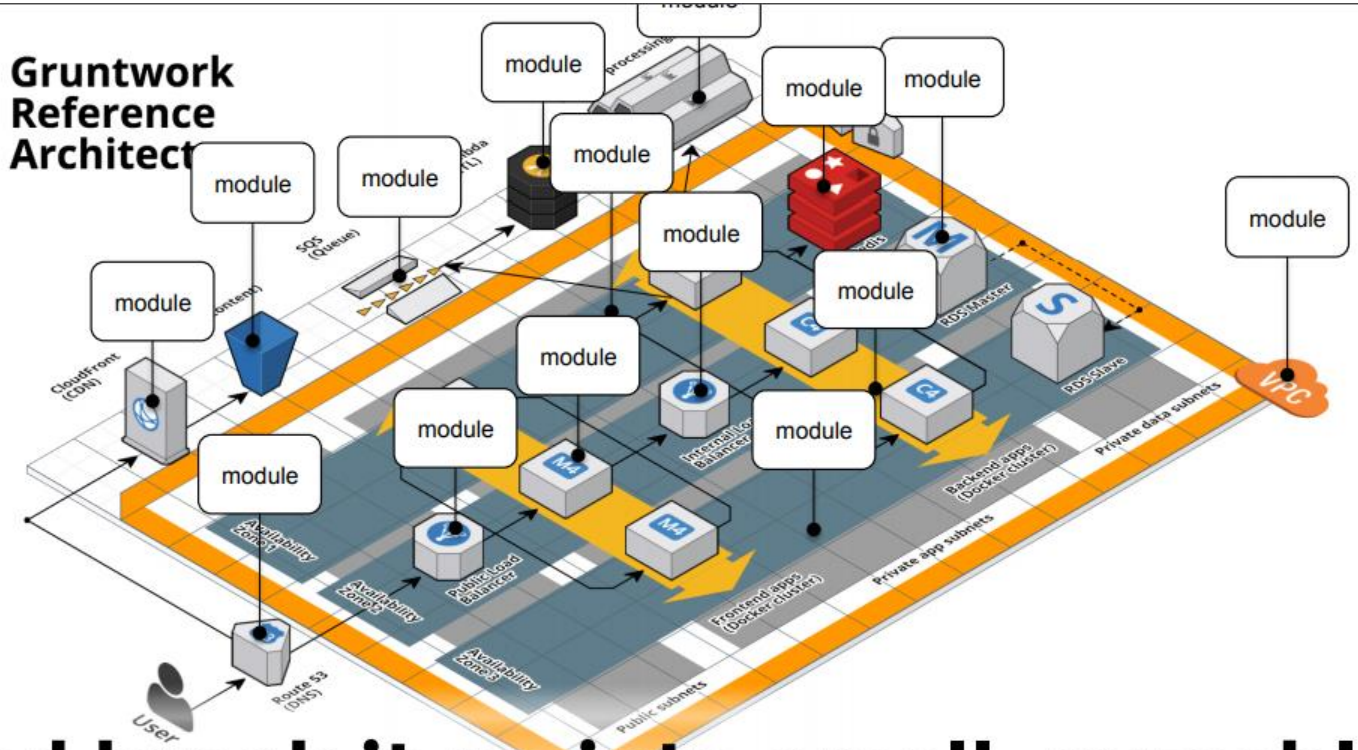
This does

Key takeaway: tools are not enough. You also need to change behavior.

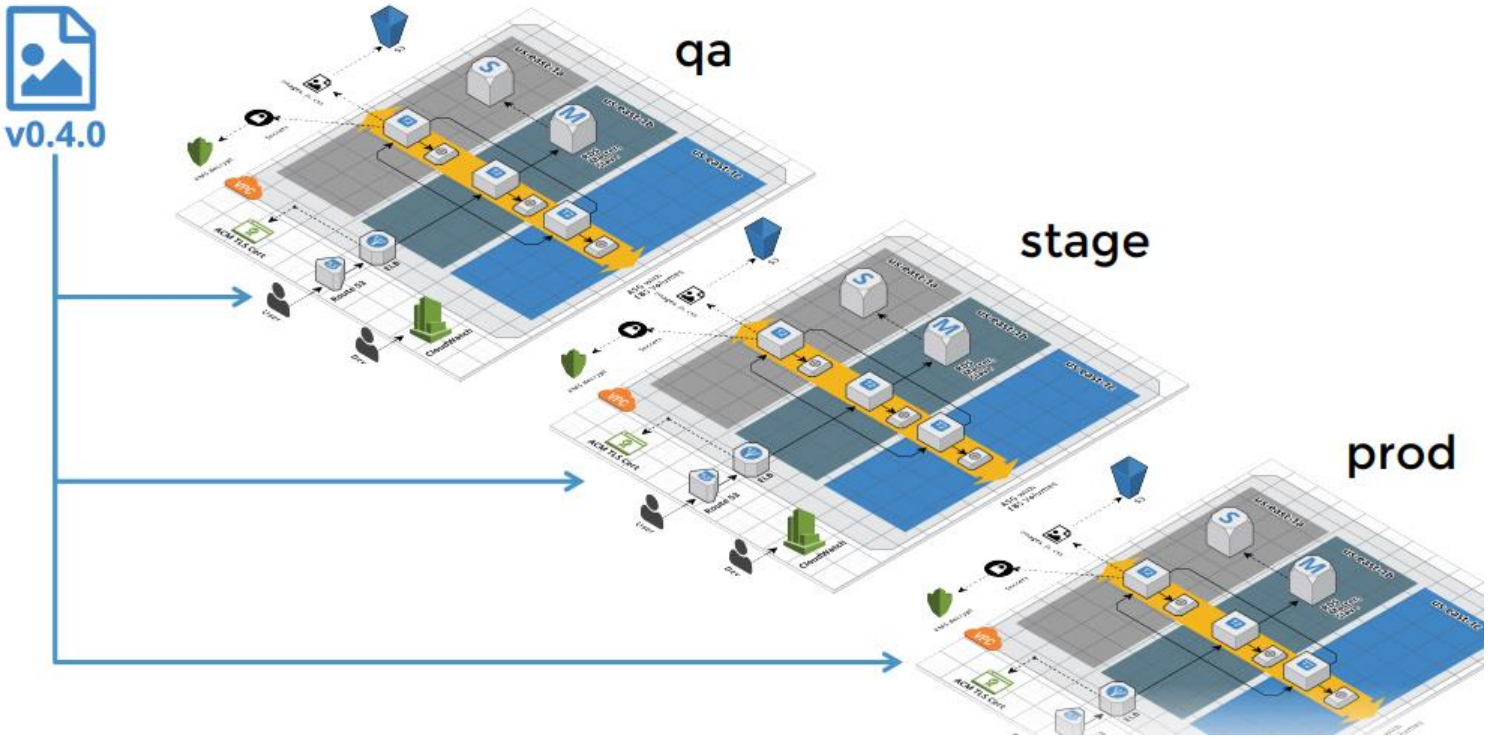


POWERED BY CLOUDCRAFT.CO

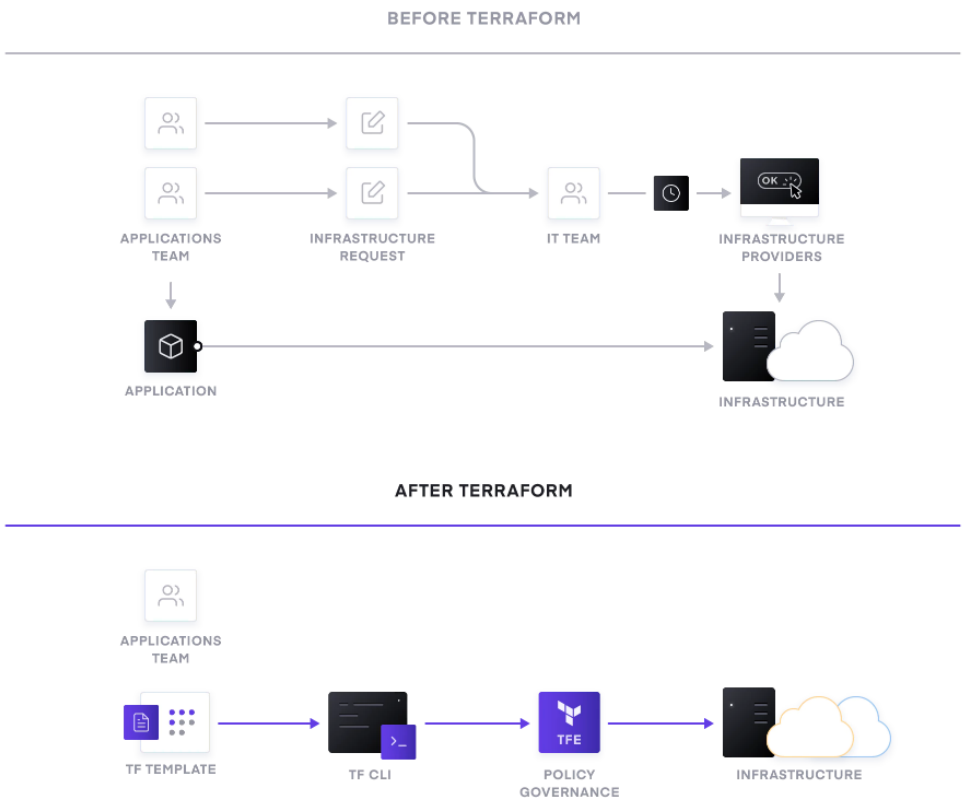
Take your architecture...



And break it up into small, reusable, standalone, tested modules

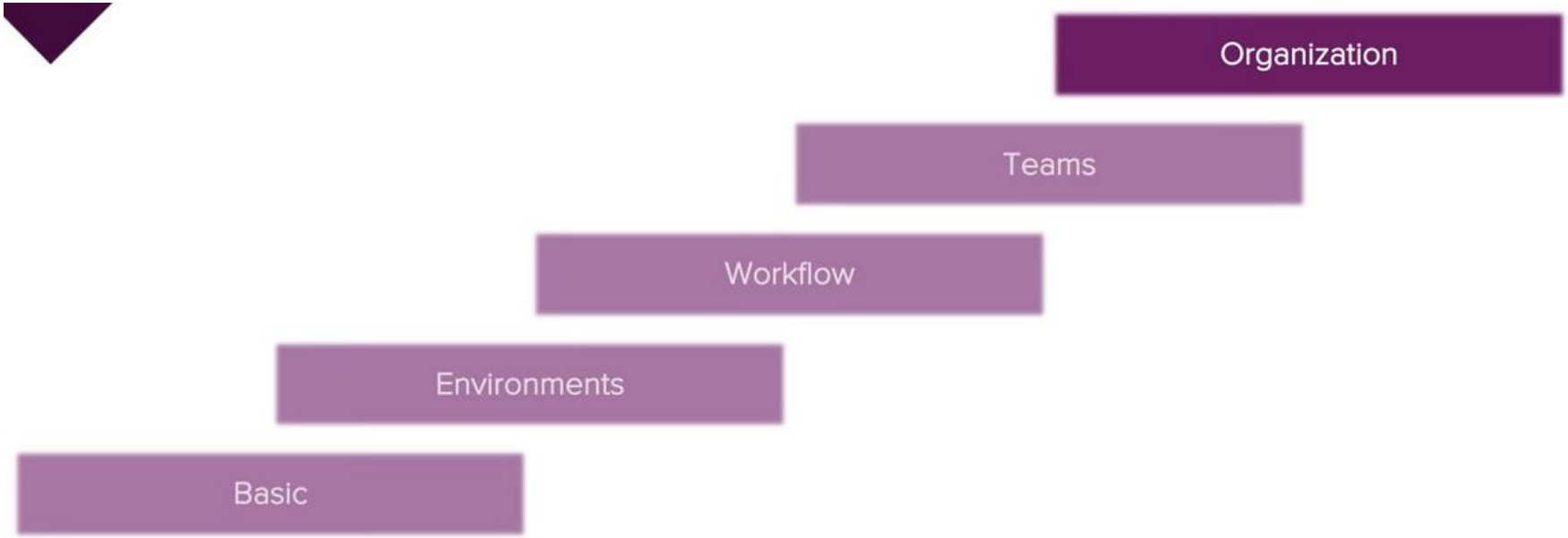

v0.4.0
qa
stage
prod

Cloud Operating Model



https://www.hashicorp.com/cloud-operating-model

Adopting Infrastructure as Code

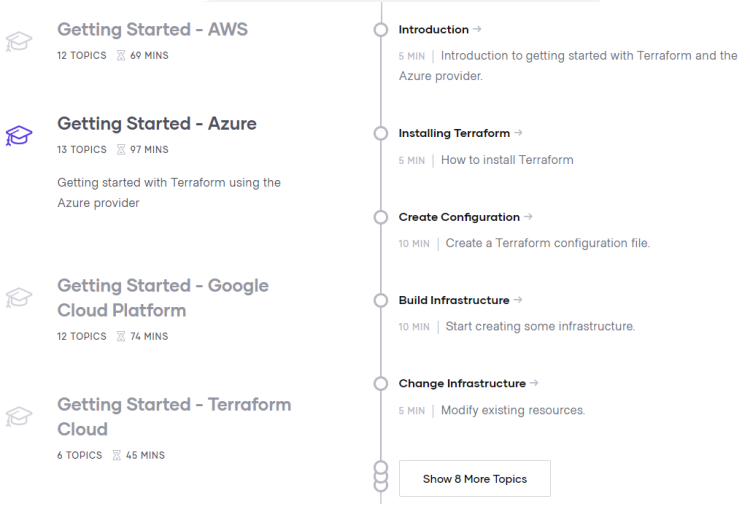


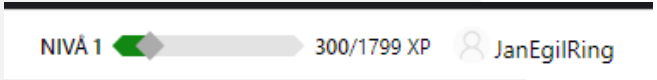
https://www.youtube.com/watch?v=G06j6HLWyYo

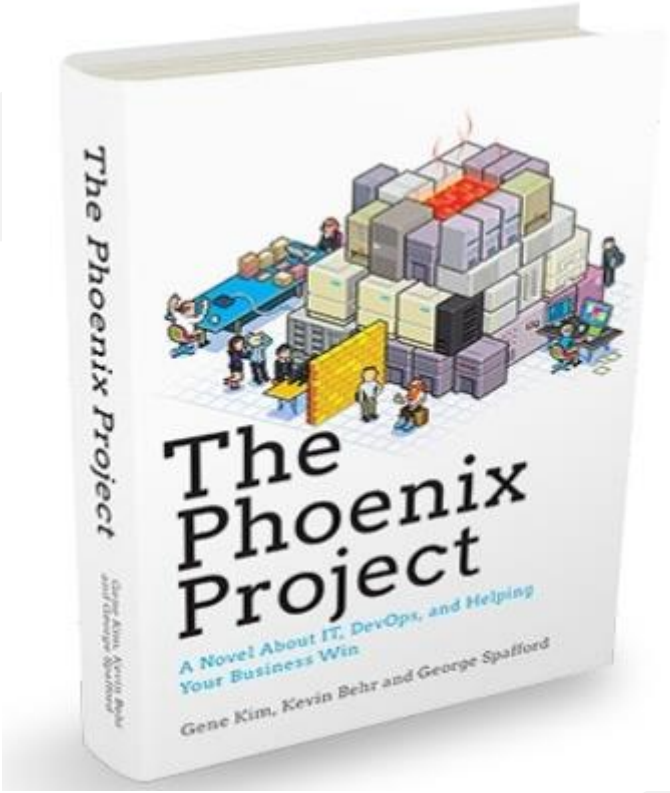
HashiCorp Learn
https://learn.hashicorp.com

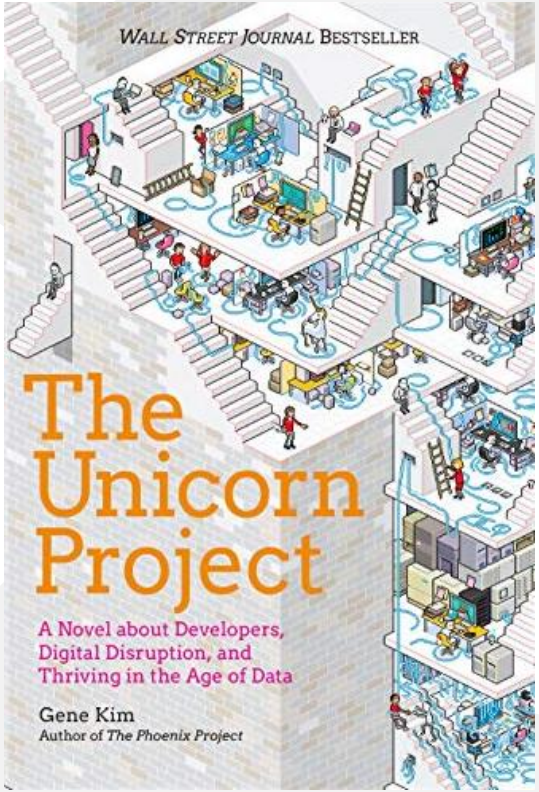
Microsoft Learn
https://docs.microsoft.com/nb-no/learn

Azure DevOps Labs
https://www.azuredevopslabs.com/labs/vstsextend/terraform






The Phoenix Project
The Phoenix Project
A Novel About IT, DevOps, and Helping Your Business Win
Gene Kim, Kevin Behr and George Spafford


WALL STREET JOURNAL BESTSELLER
The Unicorn Project
A Novel about Developers, Digital Disruption, and Thriving in the Age of Data
Gene Kim
Author of The Phoenix Project

Slides and demos from the conference will be available at

https://github.com/nordicinfrastructureconference/2020

Key takeaway

```
C:\Program Files\PowerShell\7-preview\pwsh.exe
PS C:\> Get-ContactInfo

Name    :  Jan Egil Ring
E-mail  :  jan.egil.ring@crayon.com
Twitter :  @JanEgilRing
Website : {www.crayon.no, www.powershell.no,  www.powershellmagazine.com}


PS C:\>
```