Kubernetes, today's saviour

a.k.a. greenfield devops in 18 minutes

whoami

- Extrovert, skeptical, social geek
- Now Infrastructure Lead @ Connatix
  Before: CTO @ SmartUp
- Industry experience: IoT, Payments, EdTech, Automotive, Advertising
- Interests: Architecture, OSS, Linux
  Hobbies: drumming, CrossFit

Def. Kubernetes - aka k8s

*"**Kubernetes is a** portable, extensible, open-source **platform for managing containerized workloads and services**, that facilitates both **declarative configuration** and **automation**. [...] **Google** open-sourced the Kubernetes project in 2014. Kubernetes builds upon a **decade and a half of experience** [...] running production workloads at scale, combined with best-of-breed ideas and practices from the community."*

https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

Logical reasoning 101

1. k8s is from Google, which is good
2. We start using k8s
3. We are now Google, which is good

http://www.allthingspresentations.com/mirror-mirror

Lions versus cats

Big Tech

- Huge infrastructure spread in multiple geographical regions
- Millions of customers and high utilization
- Hundreds of engineering teams working in choreography deploying hundreds of changes to production daily
- Failing hardware is everyday, BAU and not affecting customers
- Years of investment into operation automation and tooling

Our company

- Tens of servers usually in a single cloud datacenter
- Couple hundred customers
- Handful of engineering teams, high level of orchestration, deploying a couple times a month into production
- Failing hardware is catastrophical, causing P1 incidents and downtime
- Some investment in documenting operation or some scripts

Lions and Cats

Lions are cats, but bigger

What do we want?

- Ability for scaling computing power and utilize computers as cluster
- Automated, easy to understand standard workflows and primitives for handling deployments
- Tooling for packaging and shipping software in an immutable way (no more works on my machine)
- Consistency across environments both in features, tooling and behavior
- Maximize resource utilization maintaining appropriate level of isolation
- Platform features like service discovery and load balancing

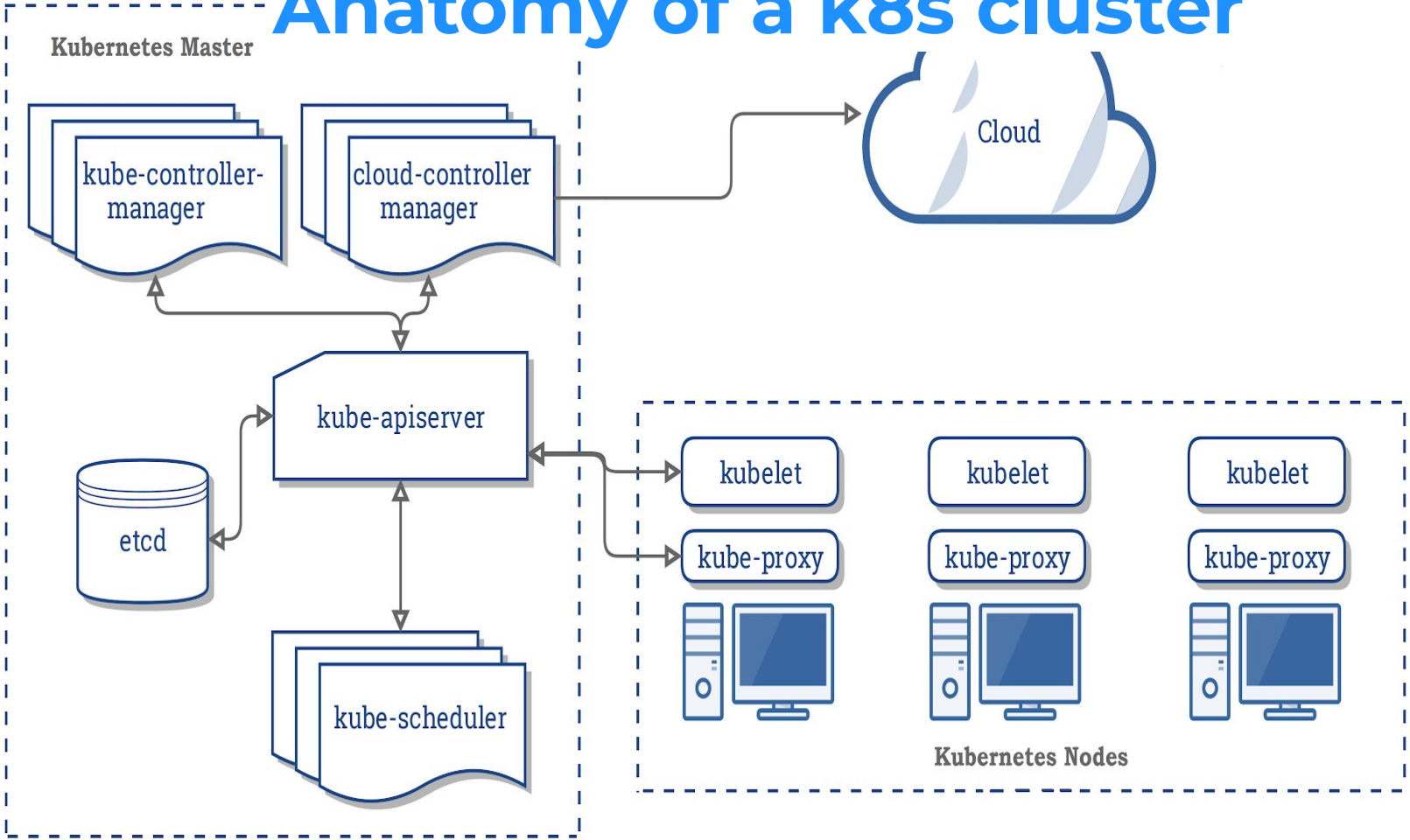
Lions are cats, but bigger

What do we want?

- Ability for scaling computing power a… …as cluster
- Automated, easy to understand s… …d primitives for handling deployments
- Tooling for packaging a… …n an immutable way (no more works on my machine)
- Consistency ac… …th in features, tooling and behavior
- Maximize r… …aintaining appropriate level of isolation
- Platform… …iscovery and load balancing

These are what cats dream about and lions can't live without.

Anatomy of a k8s cluster



https://kubernetes.io/docs/concepts/overview/components/

The atom of k8s - the Pod

- Basic execution unit of a Kubernetes application [...]
  represents processes running on your Cluster
- Single or multiple containers scheduled together that share resources
  - Containers are usually Docker but can use other runtimes (e.g. CRI-O, containerd)
  - Containers from a Pod share network namespace, can communicate via localhost
  - Containers from a Pod share volumes, can communicate via filesystem
- In practice we almost never launch them directly, but use higher level concepts
- Are ephemeral, do not self-heal, they go away with a failing node

Controllers - managing pods

- Continuously “work” towards achieving the desired state of pods, scheduling, replacing, relocating and destroying them as necessary
- Different kinds
  - ReplicaSet - makes sure a given number of pods are running
  - StatefulSet - like RS, but also takes care of maintaining state (e.g. mysql-1, mysql-2)
  - DaemonSet - run pod on all nodes (e.g. log shipper, Consul agent)
  - Deployment
- Most commonly we use Deployment, which can
  - create ReplicaSets (e.g. my-awesome-api:0.0.1 should run 3 pods)
  - update ReplicaSets (e.g. my-awesome-api should be updated from 0.0.1 to 0.0.2)
  - scale a deployment (e.g. from 3 pods to 30 pods)
  - clean up ReplicaSets

Service - your pod is not alone

- *“An abstract way to expose an application running on a set of Pods as a network service.”* (kubernetes.io)
- Each pod has its own IP address, but service provides a unified way of accessing pods of a kind
- Types
  - ClusterIP - service reachable from within the cluster
  - NodePort - open port on each node (you take care of collisions)
  - LoadBalancer - cloud provider dependant
  - ExternalName - just use DNS without any proxying
- Almost all services use kube-proxy to route traffic, except ExternalName

Volumes - things are worth holding on to

- Containers are by default ephemeral, and their data is removed on deletion
- Volumes add support for maintaining state of pod and sharing data between containers
- If we need persistence, we need Persistent Volumes (PV) and Persistent Volume Claims (PVC)
- There are lots of volume providers, a couple examples:
  - emptyDir
  - hostPath
  - awsElasticBlockStore
  - azureDisk
  - gcePersistentDisk

Some more advanced topics

- Contexts
- Namespaces
- Ingress and Ingress Controllers
- Secrets
- Custom Resources and Definitions (CR, CRD)
- Operators
- Role based access control (RBAC)

kubectl

Syntax: *$ kubectl [command] [TYPE] [NAME] [flags]*

Examples:

```
# List all pods in plain-text output format.
kubectl get pods
# Create a service using the definition in example.yaml
kubectl apply -f example.yaml
# Get output from running 'date' from pod <pod-name>
kubectl exec <pod-name> date
# Start streaming the logs from pod <pod-name>. This is similar to the 'tail -f' Linux command.
kubectl logs -f <pod-name>
# Create port forward from local environment to a pod
kubectl port-forward <pod-name> <local_port>:<pod_port>
```

kubectl

Syntax: `$ kubc`

Examples:

```
# List all pods in pl
kubectl get pods
# Create a service usi
kubectl apply -f exampl
# Get output from runni
kubectl exec <pod-name>
# Start streaming the lo                                    Linux command.
kubectl logs -f <pod-name
# Create port forward fro
kubectl port-forward <pod-
```

YAML. YAML everywhere.

Under the hood they are objects with:

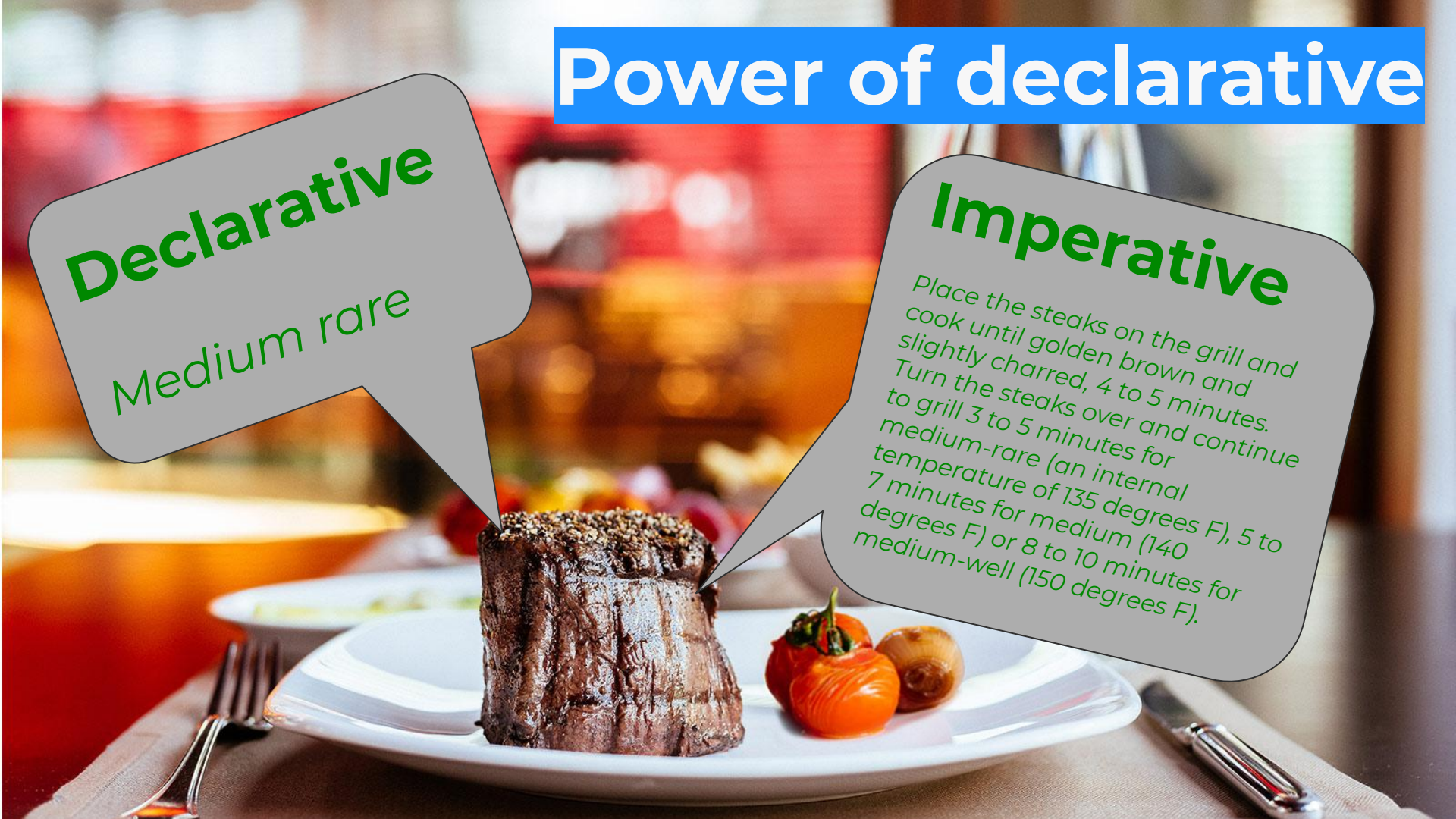
- apiVersion
- kind
- metadata
- spec

And status, which is managed by the control plane.

```yaml
apiVersion: apps/v1
kind: Deployment
Metadata:
  name: nginx-deployment
  Labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  Template:
    Metadata:
      Labels:
        app: nginx
    Spec:
      Containers:
      - name: nginx
        image: nginx:1.7.9
        Ports:
        - containerPort: 80
```

Power of declarative

Power of declarative



That’s cool. I want it.

- For local development *kubernetes/minikube* can be used
- For production you can:
  - Download k8s binaries and run k8s as systemd services
  - Run kubelet in systemd and other k8s components as containers
  - Use kubeadm
  - “The hard way” documented by Kelsey Hightower in *kelseyhightower/kubernetes-the-hard-way*

If you are in the cloud you should probably use a k8s as a service version(managed).
They are called: AWS EKS, GCP GKE, AZ AKS.

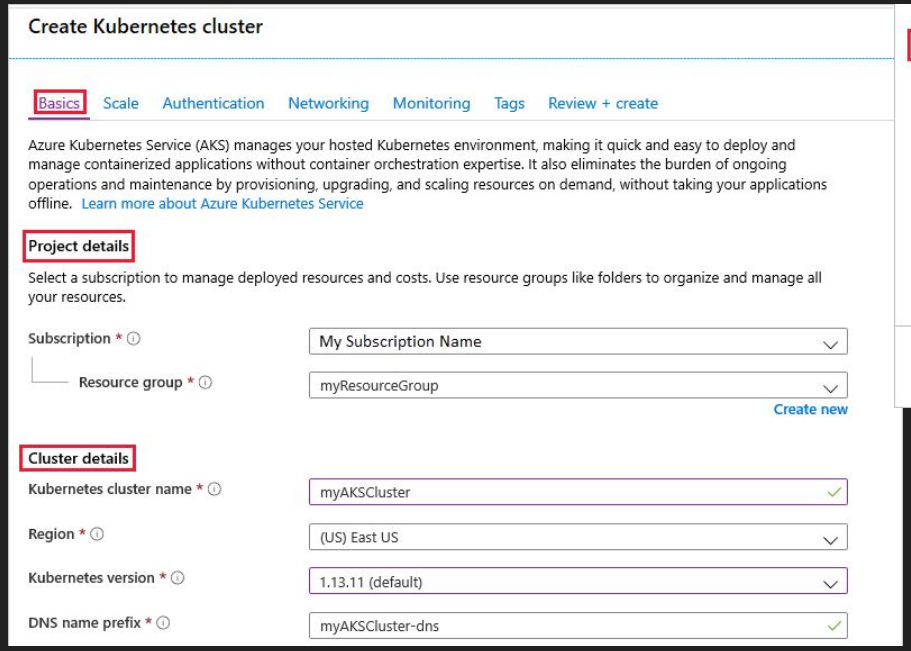
That's cool. I want it.

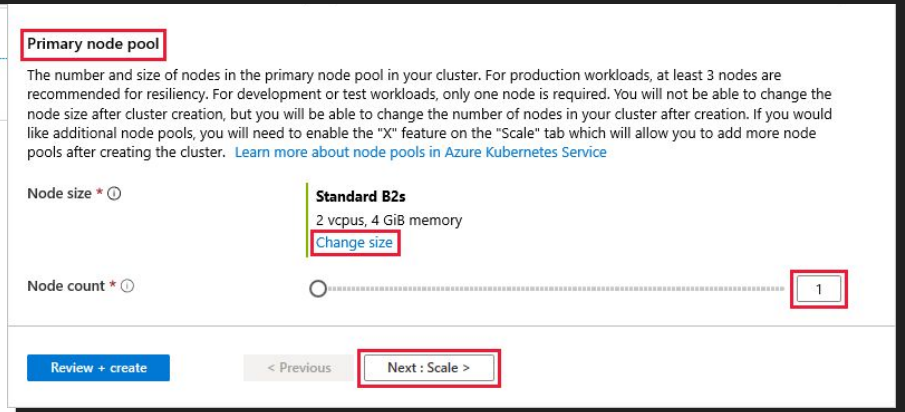
- For local development *kubernet* *be* can be used
- For production you can:
  - Download k8s binaries a systemd services
  - Run kubelet in syste k8s components
  - Use kubeadm
  - "The hard w nted by Kelsey Hig
    kelseyhight bernetes-the-hard

If you are in the cloud you shou use a k8s as a service version(managed).
They ar S EKS, GCP GKE, AZ AKS.

No control plane
=
no control plane ache.

Creating one is easy as 1-2-3



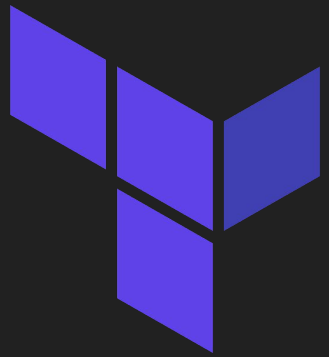


Complexity equal to ordering pizza.
But we hate picking up the phone over and over again.

Source: Domino’s

Terraform

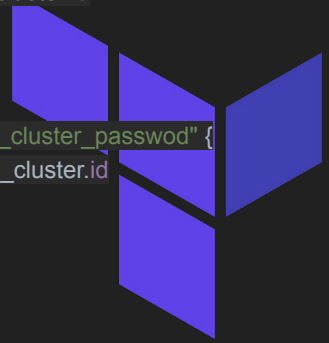
- *“Terraform is a tool for **building, changing, and versioning infrastructure safely and efficiently**. (terraform.io)*
- Human readable DSL, infrastructure as code
- Declarative (like a good steak)
- Validate, Plan, Apply
- Manages several types of resources using different Providers
- State can be remote and locked(S3, Azure Blob, git, etc)
- Modules for reusability



Terraform - module example

```
resource "azurerm_resource_group" "aks" {
 name = "${local.prefix}-${var.identifier}"
 location = var.location
}

resource "azurerm_kubernetes_cluster" "cluster" {
 name = var.identifier
 location = azurerm_resource_group.aks.location
 resource_group_name = azurerm_resource_group.aks.name
 kubernetes_version = “1.15.1”
 dns_prefix = "${var.identifier}"
agent_pool_profile {
   name = "default"
   count = 5
   vm_size = var.vm_size
   os_type = "Linux"
   os_disk_size_gb = var.default_agent_pool_node_os_disk_size
 }
 service_principal {
   client_id = azuread_application.aks_cluster.application_id
   client_secret = random_string.aks_cluster_password.result
 }
}
```

```
resource "azuread_application" "aks_cluster" {
 name = "aks-${var.identifier}"
}

resource "azuread_service_principal" "aks_cluster" {
 application_id = azuread_application.aks_cluster.application_id
}

resource "random_string" "aks_cluster_password" {
 length = 16
 special = false
keepers = {
   service_principal = azuread_service_principal.aks_cluster.id
 }
}

resource "azuread_service_principal_password" "aks_cluster_passwod" {
 service_principal_id = azuread_service_principal.aks_cluster.id
 value = random_string.aks_cluster_password.result
 end_date = timeadd(timestamp(), "87600h")
}
```



Terraform - main

Main module

```
terraform {
 backend "azurerm" {
   storage_account_name = "hwsw2019"
   container_name       = "tfstate"
   key                  = "terraform.tfstate"
 }
}

module "aks" {
source = "./modules/aks/"
 identifier              = "poc"
 location                = "westeurope"
 default_agent_pool_size = "5"
}
```

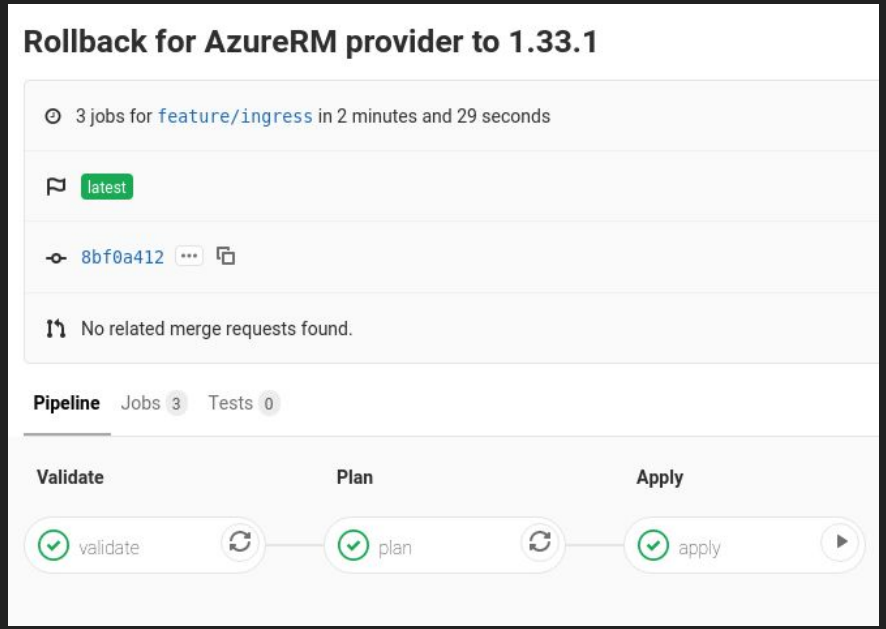
Outputs from module

```
output "client_certificate" {
 value = azurerm_kubernetes_cluster.cluster.kube_config[0].client_certificate
}

output "kube_config" {
 value = azurerm_kubernetes_cluster.cluster.kube_config
 sensitive = true
}

output "kube_admin_config" {
 value = azurerm_kubernetes_cluster.cluster.kube_admin_config
 sensitive = true
}
```

GitLab

- *is a web-based DevOps lifecycle tool that provides a Git-repository manager providing wiki, issue-tracking and CI/CD pipeline [..]*

```
# gitlab-ci.yml


image:
 name:
hashicorp/terraform:light

stages:
 - validate
 - plan
 - apply
```




```
# Apply stage

apply:
 stage: apply
 script:
  - terraform apply -input=false "planfile"
 dependencies:
  - plan
 when: manual
```

GitLab - some exciting features

Support for variables (protected & masked)

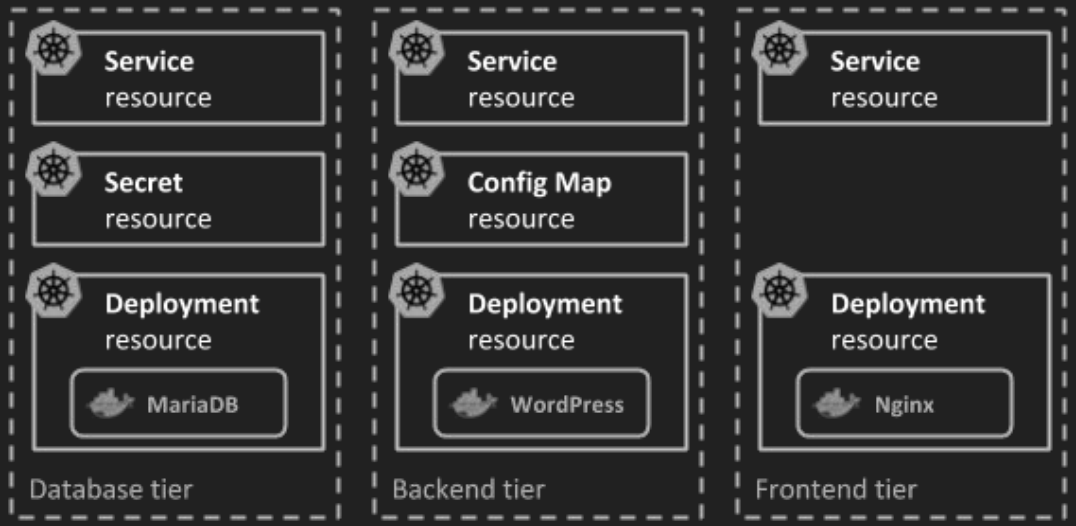
Environment Spec	`production`	`staging`	`review/feature-1`	`review/feature-2`
*	Matched	Matched	Matched	Matched
production	Matched			
staging		Matched		
review/*			Matched	Matched
review/feature-1			Matched	

Support for environments

Helm

helps you manage Kubernetes applications — Helm Charts help you define, install, and upgrade even the most complex Kubernetes application. (helm.sh)

Typical kubernetes deployment



Final thoughts

Thanks for your kind attention!

Láng Máté
mate @ matelang.dev

Twitter: @langmate
GitHub: matelang
Medium: @matelang
LinkedIn: in/matelang