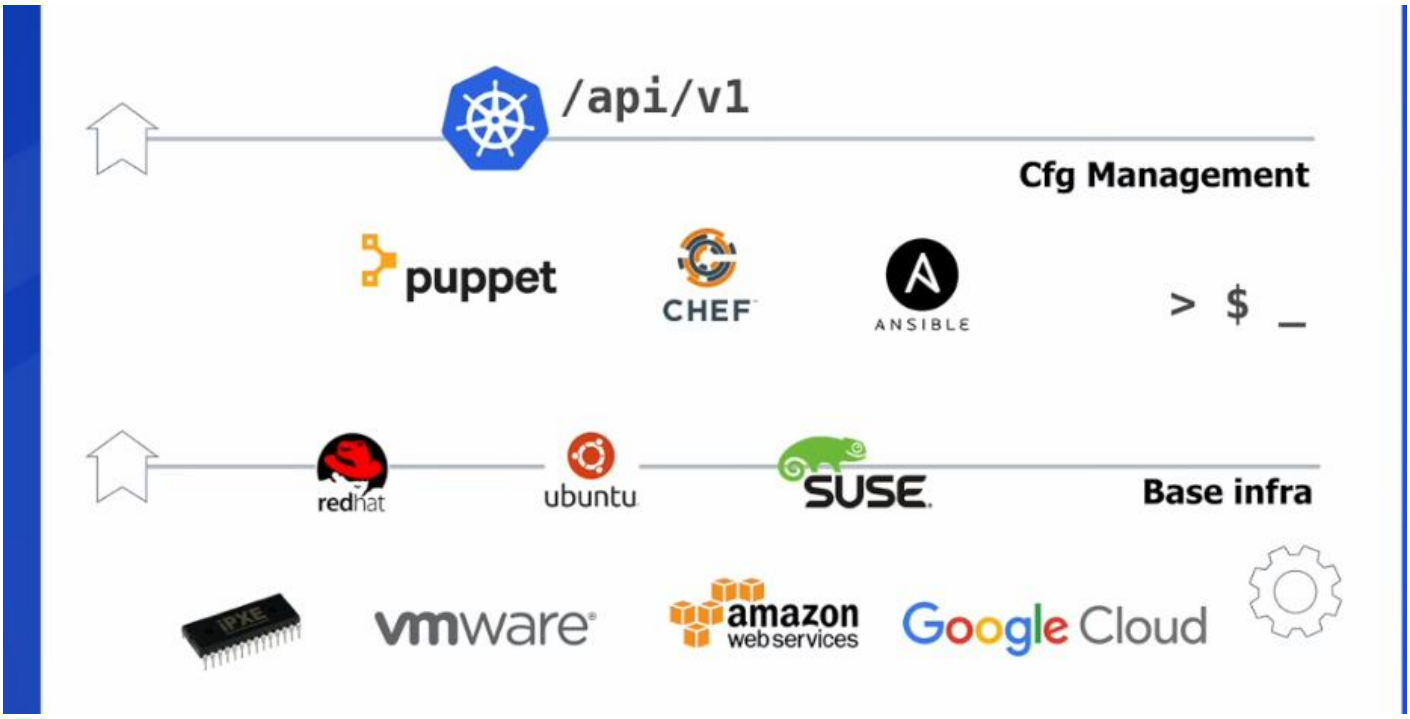
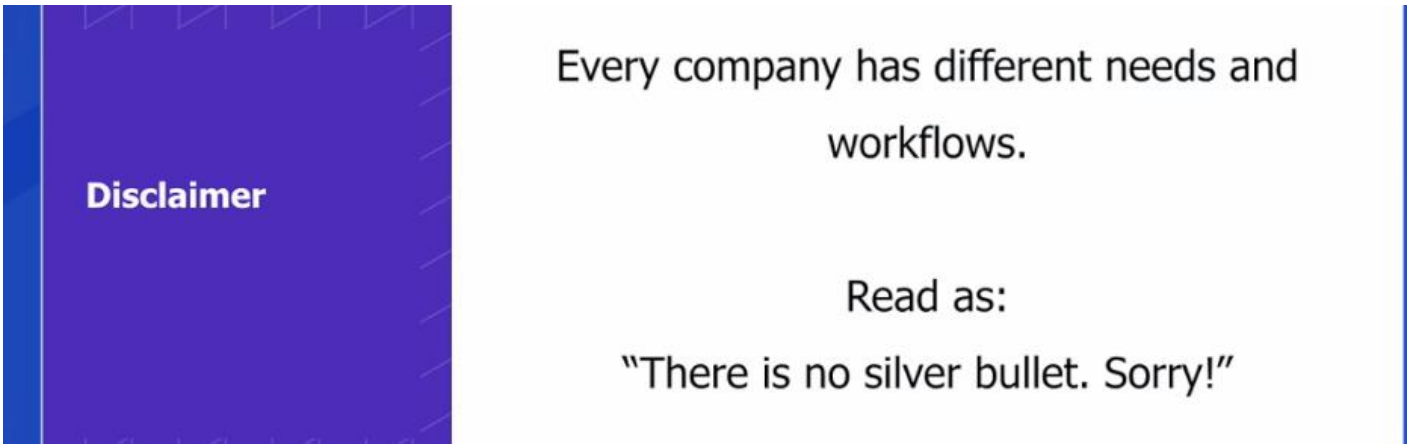


Configuration management uses tools like Ansible and Puppet and in k8s, it involves things like etcd, setting up the API Server and installing K8s, setting up CI certs, etc. The last layer is the K8s API where you can configure things like Config Maps, pods, replication controllers, general k8s resources, etc. you can use terraform to manage the lower layers.



Outline

1. HCL
2. Nature of Workload
3. Synchronicity
4. Full Lifecycle Coverage
5. Responsibility Overlap
6. DEMO !!!

1. HCL

HashiCorp Config Language

If you are not using Terraform to manage your infra, then you will be building your own automation scripts using things like Python, Go as below

high-level language

DSL

puppet

JSON
{}

language for data

YAML
-- :

HCL

- github.com/hashicorp/hcl
- Used in various HashiCorp projects
  - Consul
  - Vault
  - Nomad
  - Terraform
- JSON compatible
  - Useful for generated code

alphagov / govuk-dns

Code | Issues 0 | Pull requests 0

No description or website provided.

dns dyn gce terraform route53

`github.com/alphagov/govuk-dns`

2. Workload Nature

What kind of k8s workload is best managed by terraform?

- Resources
  - kubernetes_config_map
  - kubernetes_horizontal_pod_autoscaler
  - kubernetes_limit_range
  - kubernetes_namespace
  - kubernetes_persistent_volume
  - kubernetes_persistent_volume_claim
  - kubernetes_pod
  - kubernetes_replication_controller
  - kubernetes_resource_quota
  - kubernetes_secret
  - kubernetes_service
  - kubernetes_service_account
  - kubernetes_storage_class
- Data Sources
  - kubernetes_service
  - kubernetes_storage_class

These resources are supported at the moment in terraform

Ops-focused resources

- Config Map
- Limit Range
- Resource Quota
- Secret
- Namespace
- Service Account

Ops Workload

- Pod
- Replication Controller
- Horizontal Pod Autoscaler
- Service

3. Synchronicity

The k8s API is designed to be asynchronous which enables scalability, k8s always expects you to be proactive and ask when an operation completes

Pod

in Terraform

main.tf

```
resource "kubernetes_pod" "example" {
  metadata {
    name = "terraform-example"
  }

  spec {
    container {
      image = "nginx:1.7."
      name  = "example"
    }
  }
}
```

Failed Pod Creation

kubectl

Terminal

```
$ kubectl create -f pod.yaml
service "my-pod" created

$ kubectl get pod my-pod
NAME                READY     STATUS             RESTARTS   AGE
my-pod              0/1       ImagePullBackOff   0          31s

$ kubectl get events | head -3
...
4m          15m          7          my-pod          Pod
spec.containers{example}   Warning   Failed             kubelet, minikube
Failed to pull image "nginx:1.7.": rpc error: code = 2 desc = Tag 1.7. not
found in repository docker.io/library/nginx
```

This is how you can create a pod using kubectl. You need to be proactive by asking if an operation passed or failed

Failed Pod Creation

kubectl

```
Terminal
$ terraform apply

...

* kubernetes_pod.test: timeout while waiting for state to
become 'Running' (last state: 'Pending', timeout: 5m0s)
   * FailedSync: Error syncing pod
   * Failed: Failed to pull image "nginx:blablah": rpc
error: code = 2 desc = Tag blablah not found in repository
docker.io/library/nginx
```

In case of Terraform, you get to see the error message that we get to pull out of the log as above

Failed Service Creation

kubectl

```
Terminal
$ kubectl create -f service.yaml
service "my-service" created

$ kubectl get service my-service
NAME         CLUSTER-IP       EXTERNAL-IP    PORT(S)        AGE
my-service   10.103.242.199   <pending>      80:30310/TCP   31s

$ kubectl get events | head -3
...
30s        3m          6         my-service                Service
Warning   CreatingLoadBalancerFailed   service-controller
Error creating load balancer (will retry): Failed to create load balancer for service
default/my-service: failed to ensure static IP : error creating gce static IP address:
googleapi: Error 403: Quota 'STATIC_ADDRESSES' exceeded. Limit: 7.0, quotaExceeded
```

Successful Service Creation

kubectl

```
Terminal
$ kubectl get service my-service
NAME         CLUSTER-IP       EXTERNAL-IP      PORT(S)        AGE
my-service   10.103.242.199   35.197.115.52    80:30310/TCP   1m
```

Service

in Terraform

main.tf

```
resource "kubernetes_service" "svc" {
  metadata { name = "terraform-example" }
  spec {
    selector {
      app = "${kubernetes_pod.example.metadata.0.labels.app}"
    }
    port {
      port = 8080
      target_port = 80
    }
    type = "LoadBalancer"
  }
}
```

A service in terraform can be defined as above,

Route 53 DNS record

in Terraform

main.tf

```
resource "aws_route53_record" "example" {
  zone_id = "${data.aws_route53_zone.k8.zone_id}"
  name    = "my-service"
  type    = "CNAME"
  ttl     = "300"
  records = ["${kubernetes_service.svc.load_balancer_ingress.0.hostname}"]
}
```

You can then reference the created LB as above

Failed Service Creation

in Terraform

Terminal

```
$ terraform apply

...

* kubernetes_service.test: Waiting for service "default/my-service" to
assign IP/hostname for a load balancer
   * CreatingLoadBalancerFailed: Error creating load balancer (will retry):
Failed to create load balancer for service default/my-service: requested ip
10.0.0.1 is neither static nor assigned to LB
a049988bc615511e781c642010a8a005(default/my-service)
```

Successful Service Creation

in Terraform

```
Terminal

$ terraform apply
kubernetes_service.svc: Creating...
  load_balancer_ingress.#:     "" => "<computed>"
...
  spec.0.type:                 "" => "LoadBalancer"
kubernetes_service.svc: Still creating... (10s elapsed)
kubernetes_service.svc: Still creating... (20s elapsed)
...
kubernetes_service.svc: Still creating... (1m10s elapsed)
kubernetes_service.svc: Creation complete (ID: default/terraform-example)
aws_route53_record.example: Creating...
...
  records.3894320035: "" => "35.188.181.251"
...
aws_route53_record.example: Still creating... (10s elapsed)
...
aws_route53_record.example: Still creating... (40s elapsed)
aws_route53_record.example: Creation complete (ID: Z1H3..95_my-service_A)
```

4. Full Lifecycle Coverage

Full Lifecycle

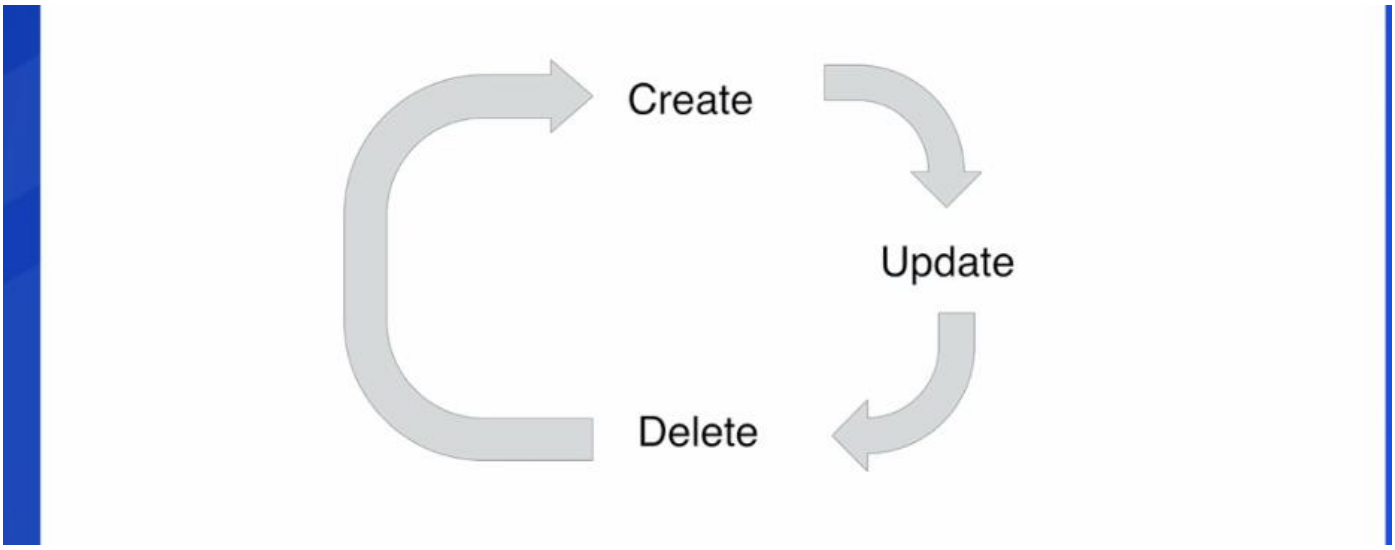
kubectl

```
Terminal

$ kubectl apply -f my-fancy-app.yaml

...

$ kubectl apply -f my-fancy-app.yaml

...

$ kubectl delete -f my-fancy-app.yaml
```

Full Lifecycle

in Terraform

```
Terminal

$ terraform apply

...

$ terraform apply

...

$ terraform apply
```



Service

in Terraform

main.tf

```
resource "kubernetes_service" "example" {
  metadata { name = "terraform-test" }
  spec {
    selector { app = "MyApp" }
    cluster_ip = "10.0.0.2"
    port {
      port = 8080
      target_port = 80
    }
    type = "ClusterIP"
  }
}
```

Not every field is mutable, you can check the plan to see which field is editable

Updating Mutable Field

in Terraform

Terminal

```
$ terraform plan

  ~ kubernetes_service.example
      spec.0.selector.app: "MyApp" => "Different"

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

Updating Immutable Field

in Terraform

```
Terminal
$ terraform plan
-/+ kubernetes_service.example (new resource required)
...
      spec.0.cluster_ip:       "10.0.0.2" => "10.0.0.3" (forces new resource)
...
Plan: 1 to add, 0 to change, 1 to destroy.
```

5. Responsibility Overlap

Responsibilities

- Annotations
- Container workload
- Storage
- Services

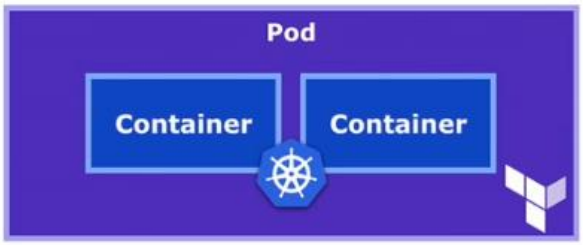
Annotations

Kubernetes/Terraform

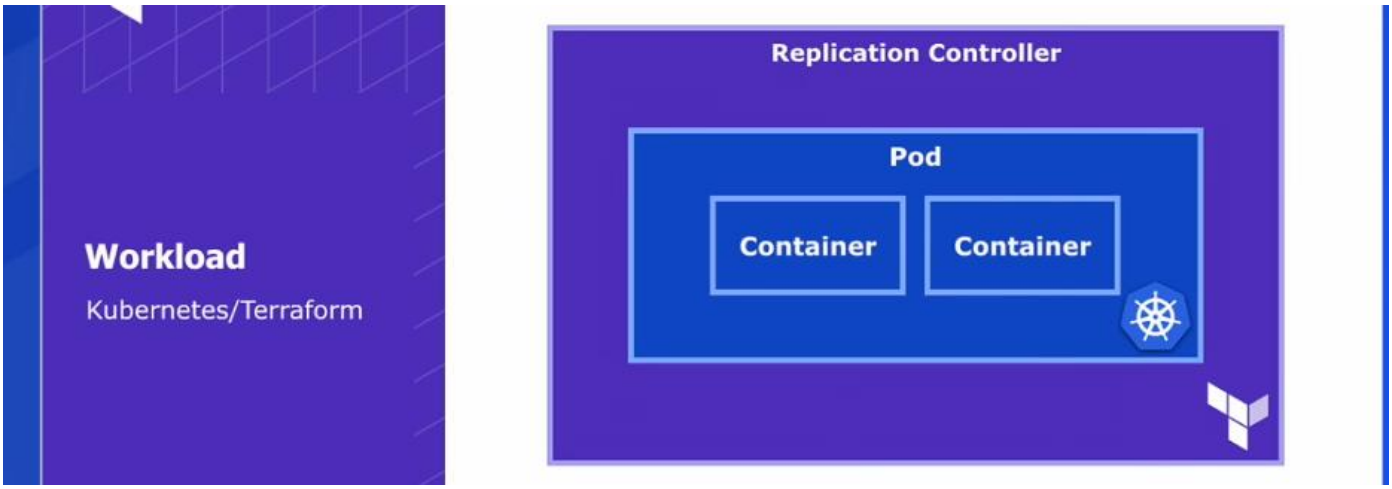
- `kubernetes.io/*`
- Not clear line between user-defined VS K8S-managed
- Ignored by Terraform, managed by K8S
- Needs revisiting

Workload

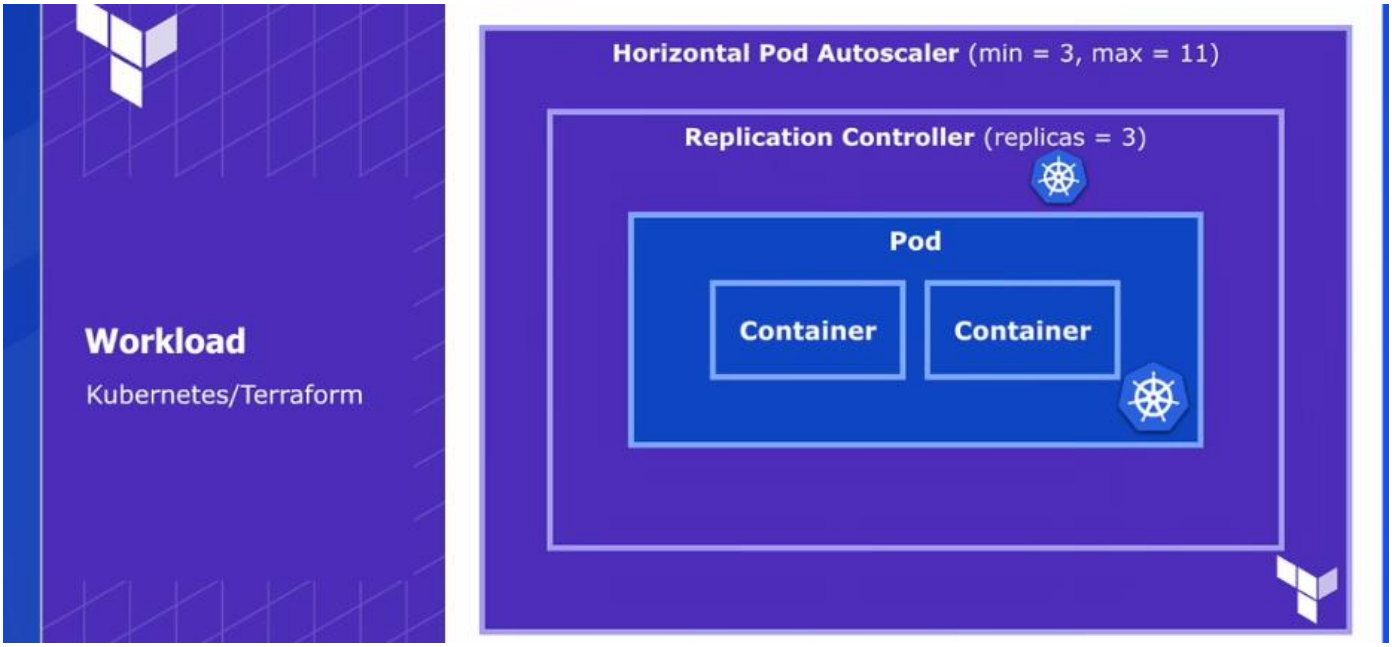
Kubernetes/Terraform



You can use terraform to schedule a pod, then k8s is responsible for keeping the containers within the pod running. Terraform can give you the initial state result like ‘container has failed to start because you made a typo’, etc



Terraform can schedule a RC for you, then k8s manages the pods and containers within the RC. Terraform does not see inside the Pod.



Terraform can create the Horizontal pod autoscaler and the RC, k8s will then have to manage the pods and the containers in the RC

you can easily deal with the overlap between terraform and k8s by specifying the lifecycle block by telling terraform to ignore things it can’t control as above,

Volumes like EBS can be managed by Terraform when using k8s,

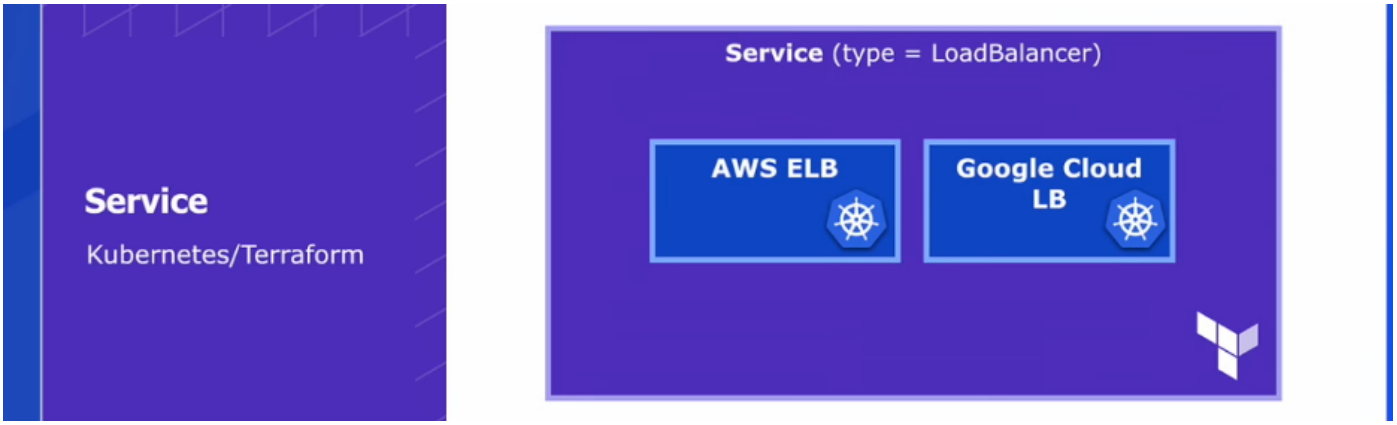
We can also instead let terraform manage the Storage Class and let k8s manage the provisioning of the volumes for you

The same with PV

The same with PVC

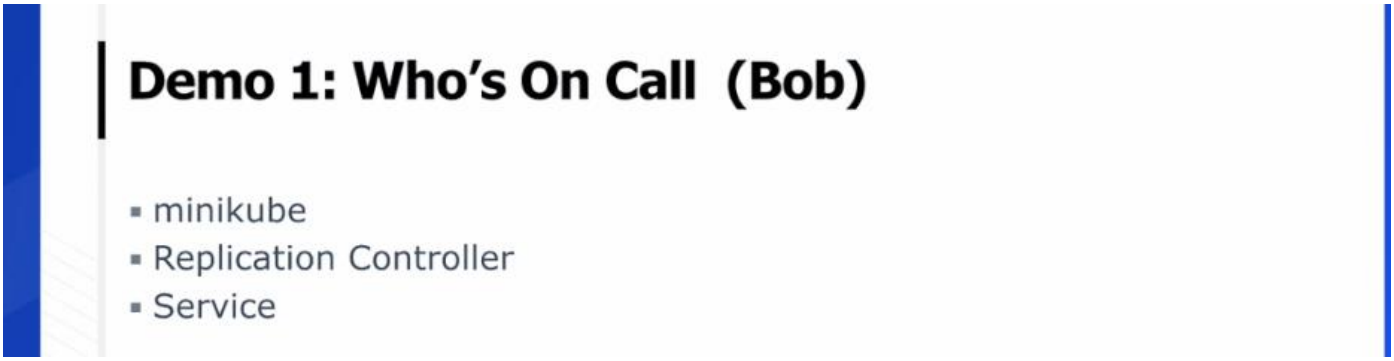
The same with a standalone pod also

Terraform can also help manage your LBs directly, but this will not be convenient if you have a more dynamic workload



It is better to let Terraform manage the service and let k8s provision the LBs

We are going to have 2 personas in this demo,



In Demo 1, we are going to deploy a 'who is on call' app which is going to involve using minikube, a RC and a service.

```
radeksimko@local:hashiconf-talk $ cd ../deve
-bash: cd: ../deve: No such file or directory
radeksimko@local:hashiconf-talk $ ls
radeksimko@local:hashiconf-talk $ cd developer-bob/
radeksimko@local:developer-bob $ ls -la
total 16
drwxr-xr-x  4 radeksimko  wheel   136 Sep 20 15:50 .
drwxr-xr-x  5 radeksimko  wheel   170 Sep  6 15:04 ..
-rw-r--r--  1 radeksimko  wheel  1026 Sep 19 06:31 main.tf
-rw-r--r--  1 radeksimko  wheel    49 Sep 19 06:30 variables.tf
radeksimko@local:developer-bob $ subl main.tf
radeksimko@local:developer-bob $
```

```
provider "kubernetes" {
  config_context = "minikube"
}

resource "kubernetes_replication_controller" "example" {
  metadata {
    name = "terraform-example"
    labels {
      app = "MyExampleApp"
    }
  }

  spec {
    selector {
      app = "MyExampleApp"
    }
    template {
      container {
        image = "hashicorp/http-echo:0.2.3"
        name  = "example"
        args = ["-text='${var.text}'"]

        resources{
          limits{
```

```
    }
  }

  spec {
    selector {
      app = "MyExampleApp"
    }
    template {
      container {
        image = "hashicorp/http-echo:0.2.3"
        name  = "example"
        args = ["-text='${var.text}'"]

        resources{
          limits{
            cpu    = "500m"
            memory = "512Mi"
          }
          requests{
```

```
13  spec {
14    selector {
15      app = "MyExampleApp"
16    }
17    template {
18      container {
19        image = "hashicorp/http-echo:0.2.3"
20        name  = "example"
21        args = ["-text='${var.text}'"]
22
23        resources{
24          limits{
25            cpu    = "500m"
26            memory = "512Mi"
27          }
28          requests{
29            cpu    = "250m"
30            memory = "50Mi"
31          }
```

```
31          }
32        }
33      }
34    }
35  }
36 }
37
38 resource "kubernetes_service" "example" {
39   metadata {
40     name = "terraform-example"
41   }
42   spec {
43     selector {
44       app = "${kubernetes_replication_controller.example.metadata.0.labe
45     }
46     port {
47       name = "http"
48       port = 80
```

```
resource "kubernetes_service" "example" {
  metadata {
    name = "terraform-example"
  }
  spec {
    selector {
      app = "${kubernetes_replication_controller.example.metadata.0.labe
    }
    port {
      name = "http"
      port = 80
      target_port = 5678
    }
    type = "NodePort"
  }
}
```

```
kubernetes_service" "example" {
 {
 "terraform-example"

or {
= "${kubernetes_replication_controller.example.metadata.0.labels.app}"

 = "http"
 = 80
et_port = 5678

 "NodePort"
```

```
  }
  spec {
    selector {
      app = "${kubernetes_replication_controller.example.metadata.0.labe
    }
    port {
      name = "http"
      port = 80
      target_port = 5678
    }
    type = "NodePort"
  }
}

output "service_name" {
  value = "${kubernetes_service.example.metadata.0.name}"
}
```

```
radeksimko@local:hashiconf-talk $ cd ../deve
-bash: cd: ../deve: No such file or directory
radeksimko@local:hashiconf-talk $ ls
radeksimko@local:hashiconf-talk $ cd developer-bob/
radeksimko@local:developer-bob $ ls -la
total 16
drwxr-xr-x  4 radeksimko  wheel   136 Sep 20 15:50 .
drwxr-xr-x  5 radeksimko  wheel   170 Sep  6 15:04 ..
-rw-r--r--  1 radeksimko  wheel  1026 Sep 19 06:31 main.tf
-rw-r--r--  1 radeksimko  wheel    49 Sep 19 06:30 variables.tf
radeksimko@local:developer-bob $ subl main.tf
radeksimko@local:developer-bob $
radeksimko@local:developer-bob $ terrafo
radeksimko@local:developer-bob $ minikube status
minikube: Running
localkube: Running
kubectl: Correctly Configured: pointing to minikube-vm at 192.168.99.100
radeksimko@local:developer-bob $
radeksimko@local:developer-bob $ min
radeksimko@local:developer-bob $ kubectl get nodes
NAME       STATUS    AGE       VERSION
minikube   Ready     54m       v1.7.0
radeksimko@local:developer-bob $
```

We verify that minikube is actually running on our laptop using the command ***$ minikube status***, we also verified that kubectl can reach the running minikube using the command ***$ kubectl get nodes*** as above

```
radeksimko@local:developer-bob $ terraform init

Initializing provider plugins...

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.
radeksimko@local:developer-bob $ terraform plan
```

We run the ***$ terraform init*** command that will download the Kubernetes plugin for terraform. We then run the ***$ terraform plan*** command to see what resources will be created for us

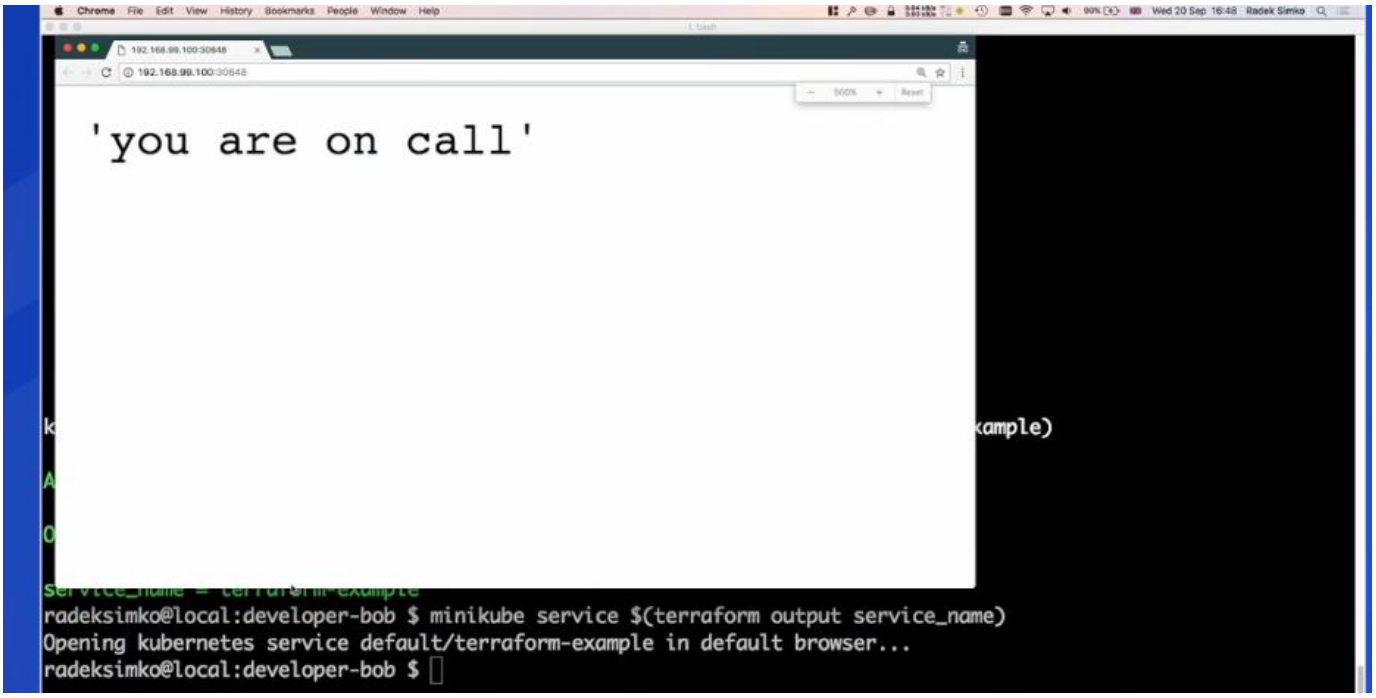
```
"apply" is called, Terraform can't guarantee this is what will execute.

  + kubernetes_replication_controller.example
      metadata.#:                                          "1"
      metadata.0.generation:                               "<computed>"
      metadata.0.labels.%:                                 "1"
      metadata.0.labels.app:                               "MyExampleApp"
      metadata.0.name:                                     "terraform-example"
      metadata.0.namespace:                                "default"
      metadata.0.resource_version:                         "<computed>"
      metadata.0.self_link:                                "<computed>"
      metadata.0.uid:                                      "<computed>"
      spec.#:                                              "1"
      spec.0.min_ready_seconds:                            "0"
      spec.0.replicas:                                     "1"
      spec.0.selector.%:                                   "1"
      spec.0.selector.app:                                 "MyExampleApp"
      spec.0.template.#:                                   "1"
      spec.0.template.0.container.#:                       "1"
      spec.0.template.0.container.0.args.#:                "1"
      spec.0.template.0.container.0.args.0:                "-text='you are on call'"
      spec.0.template.0.container.0.image:                 "hashicorp/http-echo:0.2.3"
      spec.0.template.0.container.0.image_pull_policy:     "<computed>"
      spec.0.template.0.container.0.name:                  "example"
```

```
      metadata.0.generation:       "<computed>"
      metadata.0.name:             "terraform-example"
      metadata.0.namespace:        "default"
      metadata.0.resource_version: "<computed>"
      metadata.0.self_link:        "<computed>"
      metadata.0.uid:              "<computed>"
      spec.#:                      "1"
      spec.0.cluster_ip:           "<computed>"
      spec.0.port.#:               "1"
      spec.0.port.0.name:          "http"
      spec.0.port.0.node_port:     "<computed>"
      spec.0.port.0.port:          "80"
      spec.0.port.0.protocol:      "TCP"
      spec.0.port.0.target_port:   "5678"
      spec.0.selector.%:           "1"
      spec.0.selector.app:         "MyExampleApp"
      spec.0.session_affinity:     "None"
      spec.0.type:                 "NodePort"


Plan: 2 to add, 0 to change, 0 to destroy.
radeksimko@local:developer-bob $
radeksimko@local:developer-bob $ terraform apply
```

We then run the ***$ terraform apply*** command

```
  metadata.0.namespace:        "" => "default"
  metadata.0.resource_version: "" => "<computed>"
  metadata.0.self_link:        "" => "<computed>"
  metadata.0.uid:              "" => "<computed>"
  spec.#:                      "" => "1"
  spec.0.cluster_ip:           "" => "<computed>"
  spec.0.port.#:               "" => "1"
  spec.0.port.0.name:          "" => "http"
  spec.0.port.0.node_port:     "" => "<computed>"
  spec.0.port.0.port:          "" => "80"
  spec.0.port.0.protocol:      "" => "TCP"
  spec.0.port.0.target_port:   "" => "5678"
  spec.0.selector.%:           "" => "1"
  spec.0.selector.app:         "" => "MyExampleApp"
  spec.0.session_affinity:     "" => "None"
  spec.0.type:                 "" => "NodePort"
kubernetes_service.example: Creation complete after 0s (ID: default/terraform-example)

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

Outputs:

service_name = terraform-example
radeksimko@local:developer-bob $
```

We can see the result of the ***$ terraform apply*** command above.



We can them use the terraform output within the ***$ minikube service $(terraform output service_name)*** command to see if the service is now running as above

Alice is an operator, she is going to deploy a GKE cluster and then deploy a namespace in K8s for scoping our resources.

```
  spec.0.port.0.port:          "" => "80"
  spec.0.port.0.protocol:      "" => "TCP"
  spec.0.port.0.target_port:   "" => "5678"
  spec.0.selector.%:           "" => "1"
  spec.0.selector.app:         "" => "MyExampleApp"
  spec.0.session_affinity:     "" => "None"
  spec.0.type:                 "" => "NodePort"
kubernetes_service.example: Creation complete after 0s (ID: default/terraform-example)

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

Outputs:

service_name = terraform-example
radeksimko@local:developer-bob $ minikube service $(terraform output service_name)
Opening kubernetes service default/terraform-example in default browser...
radeksimko@local:developer-bob $ cd ../operator-alice/
radeksimko@local:operator-alice $ ls -la
total 8
drwxr-xr-x  4 radeksimko  wheel  136 Sep 20 15:50 .
drwxr-xr-x  5 radeksimko  wheel  170 Sep  6 15:04 ..
drwxr-xr-x  7 radeksimko  wheel  238 Sep 18 06:48 infra
-rw-r--r--  1 radeksimko  wheel  547 Sep 17 18:25 main.tf
radeksimko@local:operator-alice $
```

```
radeksimko@local:operator-alice $ ls -la
total 8
drwxr-xr-x  4 radeksimko  wheel  136 Sep 20 15:50 .
drwxr-xr-x  5 radeksimko  wheel  170 Sep  6 15:04 ..
drwxr-xr-x  7 radeksimko  wheel  238 Sep 18 06:48 infra
-rw-r--r--  1 radeksimko  wheel  547 Sep 17 18:25 main.tf
radeksimko@local:operator-alice $ cd infra/
radeksimko@local:infra $ ls -la
total 56
drwxr-xr-x  7 radeksimko  wheel    238 Sep 18 06:48 .
drwxr-xr-x  4 radeksimko  wheel    136 Sep 20 15:50 ..
drwxr-xr-x  3 radeksimko  wheel    102 Sep 18 06:45 .terraform
-rw-r--r--  1 radeksimko  wheel    850 Sep 17 19:03 main.tf
-rw-r--r--  1 radeksimko  wheel    132 Sep  6 14:28 provider.tf
-rw-r--r--@ 1 radeksimko  wheel    140 Sep 17 19:03 random.tf
-rw-r--r--  1 radeksimko  wheel  12680 Sep 18 06:48 terraform.tfstate
radeksimko@local:infra $ subl main.tf
```

```
resource "google_container_cluster" "primary" {
  name               = "marcellus-wallace"
  zone               = "us-central1-a"
  initial_node_count = 3

  additional_zones = [
    "us-central1-b"
  ]

  master_auth {
    username = "${random_string.user.result}"
    password = "${random_string.password.result}"
  }

  node_config {
    oauth_scopes = [
      "https://www.googleapis.com/auth/compute",
      "https://www.googleapis.com/auth/devstorage.read_only",
```

We have the container cluster and we are leveraging Google cloud provider

```
  }

  node_config {
    oauth_scopes = [
      "https://www.googleapis.com/auth/compute",
      "https://www.googleapis.com/auth/devstorage.read_only",
      "https://www.googleapis.com/auth/logging.write",
      "https://www.googleapis.com/auth/monitoring",
    ]
  }
}

output "zone" {
  value = "${google_container_cluster.primary.zone}"
}

output "cluster_name" {
  value = "${google_container_cluster.primary.name}"
}
```

```
radeksimko@local:infra $ subl main.tf
radeksimko@local:infra $ terraform output
cluster_name = marcellus-wallace
zone = us-central1-a
radeksimko@local:infra $
```

We also have our created cluster shown above, now we can create the namespace within it

```
radeksimko@local:infra $ terraform output
cluster_name = marcellus-wallace
zone = us-central1-a
radeksimko@local:infra $
radeksimko@local:infra $ cd ../
radeksimko@local:operator-alice $ ls -la
total 8
drwxr-xr-x  4 radeksimko  wheel  136 Sep 20 15:50 .
drwxr-xr-x  5 radeksimko  wheel  170 Sep  6 15:04 ..
drwxr-xr-x  7 radeksimko  wheel  238 Sep 18 06:48 infra
-rw-r--r--  1 radeksimko  wheel  547 Sep 17 18:25 main.tf
radeksimko@local:operator-alice $ subl main.tf
radeksimko@local:operator-alice $ cd infr
radeksimko@local:operator-alice $ cd infra/
radeksimko@local:infra $ ls -la
total 56
drwxr-xr-x  7 radeksimko  wheel    238 Sep 18 06:48 .
drwxr-xr-x  4 radeksimko  wheel    136 Sep 20 15:50 ..
drwxr-xr-x  3 radeksimko  wheel    102 Sep 18 06:45 .terraform
-rw-r--r--  1 radeksimko  wheel    850 Sep 17 19:03 main.tf
-rw-r--r--  1 radeksimko  wheel    132 Sep  6 14:28 provider.tf
-rw-r--r--@ 1 radeksimko  wheel    140 Sep 17 19:03 random.tf
-rw-r--r--  1 radeksimko  wheel  12680 Sep 18 06:48 terraform.tfstate
radeksimko@local:infra $ subl main.tf
```

```
radeksimko@local:infra $
radeksimko@local:infra $ cd ../
radeksimko@local:operator-alice $ ls -la
total 8
drwxr-xr-x  4 radeksimko  wheel  136 Sep 20 15:50 .
drwxr-xr-x  5 radeksimko  wheel  170 Sep  6 15:04 ..
drwxr-xr-x  7 radeksimko  wheel  238 Sep 18 06:48 infra
-rw-r--r--  1 radeksimko  wheel  547 Sep 17 18:25 main.tf
radeksimko@local:operator-alice $ subl main.tf
radeksimko@local:operator-alice $ cd infr
radeksimko@local:operator-alice $ cd infra/
radeksimko@local:infra $ ls -la
total 56
drwxr-xr-x  7 radeksimko  wheel    238 Sep 18 06:48 .
drwxr-xr-x  4 radeksimko  wheel    136 Sep 20 15:50 ..
drwxr-xr-x  3 radeksimko  wheel    102 Sep 18 06:45 .terraform
-rw-r--r--  1 radeksimko  wheel    850 Sep 17 19:03 main.tf
-rw-r--r--  1 radeksimko  wheel    132 Sep  6 14:28 provider.tf
-rw-r--r--@ 1 radeksimko  wheel    140 Sep 17 19:03 random.tf
-rw-r--r--  1 radeksimko  wheel  12680 Sep 18 06:48 terraform.tfstate
radeksimko@local:infra $ subl main.tf
radeksimko@local:infra $
radeksimko@local:infra $ gcloud container clusters get-credentials --zone=$(terraform output zone) $(terraform
output cluster_name)
```

We use the command above to get credentials for the cluster

```
radeksimko@local:infra $ kubectl get nodes
NAME                                          STATUS    AGE       VERSION
gke-marcellus-wallace-default-pool-9824aee2-2889   Ready     2d        v1.6.9
gke-marcellus-wallace-default-pool-9824aee2-6rvm   Ready     2d        v1.6.9
gke-marcellus-wallace-default-pool-9824aee2-hv3m   Ready     2d        v1.6.9
gke-marcellus-wallace-default-pool-eadd556e-1td2   Ready     2d        v1.6.9
gke-marcellus-wallace-default-pool-eadd556e-4j87   Ready     2d        v1.6.9
gke-marcellus-wallace-default-pool-eadd556e-hflx   Ready     2d        v1.6.9
radeksimko@local:infra $
```

```
radeksimko@local:operator-alice $ ls -la
total 8
drwxr-xr-x  4 radeksimko  wheel  136 Sep 20 15:50 .
drwxr-xr-x  5 radeksimko  wheel  170 Sep  6 15:04 ..
drwxr-xr-x  7 radeksimko  wheel  238 Sep 18 06:48 infra
-rw-r--r--  1 radeksimko  wheel  547 Sep 17 18:25 main.tf
radeksimko@local:operator-alice $
```

```
1 provider "kubernetes" {
2   config_context = "gke_hc-terraform-testing_us-central1-a_marcellus-wall
3 }
4
5 resource "kubernetes_namespace" "example" {
6   metadata {
7     name = "terraform-example-namespace"
8   }
9 }
10
11 resource "kubernetes_limit_range" "example" {
12     metadata {
13         name = "terraform-example"
14         namespace = "${kubernetes_namespace.example.metadata.0.name}"
15     }
16     spec {
17         limit {
18             type = "Pod"
```

```
8   }
9 }
10
11 resource "kubernetes_limit_range" "example" {
12     metadata {
13         name = "terraform-example"
14         namespace = "${kubernetes_namespace.example.metadata.0.name}"
15     }
16     spec {
17         limit {
18             type = "Pod"
19             max {
20                 cpu = "1000m"
21                 memory = "1024M"
22             }
23         }
24     }
25 }
```

Alice can now create the namespace using this file

```
radeksimko@local:operator-alice $ ls -la
total 8
drwxr-xr-x  4 radeksimko  wheel  136 Sep 20 15:50 .
drwxr-xr-x  5 radeksimko  wheel  170 Sep  6 15:04 ..
drwxr-xr-x  7 radeksimko  wheel  238 Sep 18 06:48 infra
-rw-r--r--  1 radeksimko  wheel  547 Sep 17 18:25 main.tf
radeksimko@local:operator-alice $ subl main.tf
radeksimko@local:operator-alice $ terraform init

Initializing provider plugins...

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.
radeksimko@local:operator-alice $ terraform plan
```

We then run ***$ terraform init*** and ***$ terraform plan*** as above

```
      metadata.0.name:                  "terraform-example"
      metadata.0.namespace:             "terraform-example-namespace"
      metadata.0.resource_version:      "<computed>"
      metadata.0.self_link:             "<computed>"
      metadata.0.uid:                   "<computed>"
      spec.#:                           "1"
      spec.0.limit.#:                   "1"
      spec.0.limit.0.default_request.%: "<computed>"
      spec.0.limit.0.max.%:             "2"
      spec.0.limit.0.max.cpu:           "1000m"
      spec.0.limit.0.max.memory:        "1024M"
      spec.0.limit.0.type:              "Pod"

  + kubernetes_namespace.example
      metadata.#:                  "1"
      metadata.0.generation:       "<computed>"
      metadata.0.name:             "terraform-example-namespace"
      metadata.0.resource_version: "<computed>"
      metadata.0.self_link:        "<computed>"
      metadata.0.uid:              "<computed>"


Plan: 2 to add, 0 to change, 0 to destroy.
radeksimko@local:operator-alice $ terraform apply
```

We then run the ***$ terraform apply*** command

```
  metadata.0.name:              "" => "terraform-example-namespace"
  metadata.0.resource_version:  "" => "<computed>"
  metadata.0.self_link:         "" => "<computed>"
  metadata.0.uid:               "" => "<computed>"
kubernetes_namespace.example: Creation complete after 1s (ID: terraform-example-namespace)
kubernetes_limit_range.example: Creating...
  metadata.#:                       "" => "1"
  metadata.0.generation:            "" => "<computed>"
  metadata.0.name:                  "" => "terraform-example"
  metadata.0.namespace:             "" => "terraform-example-namespace"
  metadata.0.resource_version:      "" => "<computed>"
  metadata.0.self_link:             "" => "<computed>"
  metadata.0.uid:                   "" => "<computed>"
  spec.#:                           "" => "1"
  spec.0.limit.#:                   "" => "1"
  spec.0.limit.0.default_request.%: "" => "<computed>"
  spec.0.limit.0.max.%:             "" => "2"
  spec.0.limit.0.max.cpu:           "" => "1000m"
  spec.0.limit.0.max.memory:        "" => "1024M"
  spec.0.limit.0.type:              "" => "Pod"
kubernetes_limit_range.example: Creation complete after 0s (ID: terraform-example-namespace/terraform-example)

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
radeksimko@local:operator-alice $
```

```
  metadata.0.generation:            "" => "<computed>"
  metadata.0.name:                  "" => "terraform-example"
  metadata.0.namespace:             "" => "terraform-example-namespace"
  metadata.0.resource_version:      "" => "<computed>"
  metadata.0.self_link:             "" => "<computed>"
  metadata.0.uid:                   "" => "<computed>"
  spec.#:                           "" => "1"
  spec.0.limit.#:                   "" => "1"
  spec.0.limit.0.default_request.%: "" => "<computed>"
  spec.0.limit.0.max.%:             "" => "2"
  spec.0.limit.0.max.cpu:           "" => "1000m"
  spec.0.limit.0.max.memory:        "" => "1024M"
  spec.0.limit.0.type:              "" => "Pod"
kubernetes_limit_range.example: Creation complete after 0s (ID: terraform-example-namespace/terraform-example)

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
radeksimko@local:operator-alice $ cd ../production/
radeksimko@local:production $ ls -la
total 16
drwxr-xr-x  4 radeksimko  wheel   136 Sep 20 15:51 .
drwxr-xr-x  5 radeksimko  wheel   170 Sep  6 15:04 ..
-rw-r--r--  1 radeksimko  wheel  1263 Sep 19 06:41 main.tf
-rw-r--r--@ 1 radeksimko  wheel    48 Sep 19 06:41 variables.tf
radeksimko@local:production $
```

```
radeksimko@local:production $ subl main.tf
```

```
provider "kubernetes" {
  config_context = "${terraform.workspace == "default" ? "minikube" : "gk
}

resource "kubernetes_replication_controller" "example" {
  metadata {
    name = "real-example"
    labels {
      app = "MyExampleApp"
    }
  }

  spec {
    selector {
      app = "MyExampleApp"
    }
    template {
      container {
```

You can use ${terraform.workspace} to have the same config file talk to multiple clusters as above

```
ubernetes" {
ntext = "${terraform.workspace == "default" ? "minikube" : "gke_hc-terra

ubernetes_replication_controller" "example" {
{
"real-example"
{
 "MyExampleApp"

r {
 "MyExampleApp"

e {
iner {
```

```
radeksimko@local:production $ subl main.tf
radeksimko@local:production $ terraform workspace list
* default

radeksimko@local:production $
```

Terraform provides a default workspace as seen above

```
  spec {
    selector {
      app = "MyExampleApp"
    }
    template {
      container {
        image = "hashicorp/http-echo:0.2.3"
        name  = "example"
        args = ["-text='${var.text}'"]

        resources{
          limits{
            cpu    = "500m"
            memory = "512Mi"
          }
          requests{
            cpu    = "250m"
            memory = "50Mi"
```

```
        }
      }
    }
  }
}

resource "kubernetes_service" "example" {
  metadata {
    name = "real-example"
  }
  spec {
    selector {
      app = "${kubernetes_replication_controller.example.metadata.0.label
    }
    port {
      name = "http"
      port = 80
      target_port = 5678
```

```
  spec {
    selector {
      app = "${kubernetes_replication_controller.example.metadata.0.label
    }
    port {
      name = "http"
      port = 80
      target_port = 5678
    }
    type = "${terraform.workspace == "default" ? "NodePort" : "LoadBalanc
  }
}

output "service_name" {
  value = "${kubernetes_service.example.metadata.0.name}"
}

output "lb_ingress" {
```

```
radeksimko@local:production $ subl main.tf
radeksimko@local:production $ terraform workspace list
* default

radeksimko@local:production $ terraform init

Initializing provider plugins...

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.
radeksimko@local:production $ terraform plan
```

```
      load_balancer_ingress.#:     "<computed>"
      metadata.#:                  "1"
      metadata.0.generation:       "<computed>"
      metadata.0.name:             "real-example"
      metadata.0.namespace:        "default"
      metadata.0.resource_version: "<computed>"
      metadata.0.self_link:        "<computed>"
      metadata.0.uid:              "<computed>"
      spec.#:                      "1"
      spec.0.cluster_ip:           "<computed>"
      spec.0.port.#:               "1"
      spec.0.port.0.name:          "http"
      spec.0.port.0.node_port:     "<computed>"
      spec.0.port.0.port:          "80"
      spec.0.port.0.protocol:      "TCP"
      spec.0.port.0.target_port:   "5678"
      spec.0.selector.%:           "1"
      spec.0.selector.app:         "MyExampleApp"
      spec.0.session_affinity:     "None"
      spec.0.type:                 "NodePort"


Plan: 2 to add, 0 to change, 0 to destroy.
radeksimko@local:production $
```


```
      load_balancer_ingress.#:     "<computed>"
      metadata.#:                  "1"
      metadata.0.generation:       "<computed>"
      metadata.0.name:             "real-example"
      metadata.0.namespace:        "default"
      metadata.0.resource_version: "<computed>"
      metadata.0.self_link:        "<computed>"
      metadata.0.uid:              "<computed>"
      spec.#:                      "1"
      spec.0.cluster_ip:           "<computed>"
      spec.0.port.#:               "1"
      spec.0.port.0.name:          "http"
      spec.0.port.0.node_port:     "<computed>"
      spec.0.port.0.port:          "80"
      spec.0.port.0.protocol:      "TCP"
      spec.0.port.0.target_port:   "5678"
      spec.0.selector.%:           "1"
      spec.0.selector.app:         "MyExampleApp"
      spec.0.session_affinity:     "None"
      spec.0.type:                 "NodePort"


Plan: 2 to add, 0 to change, 0 to destroy.
```

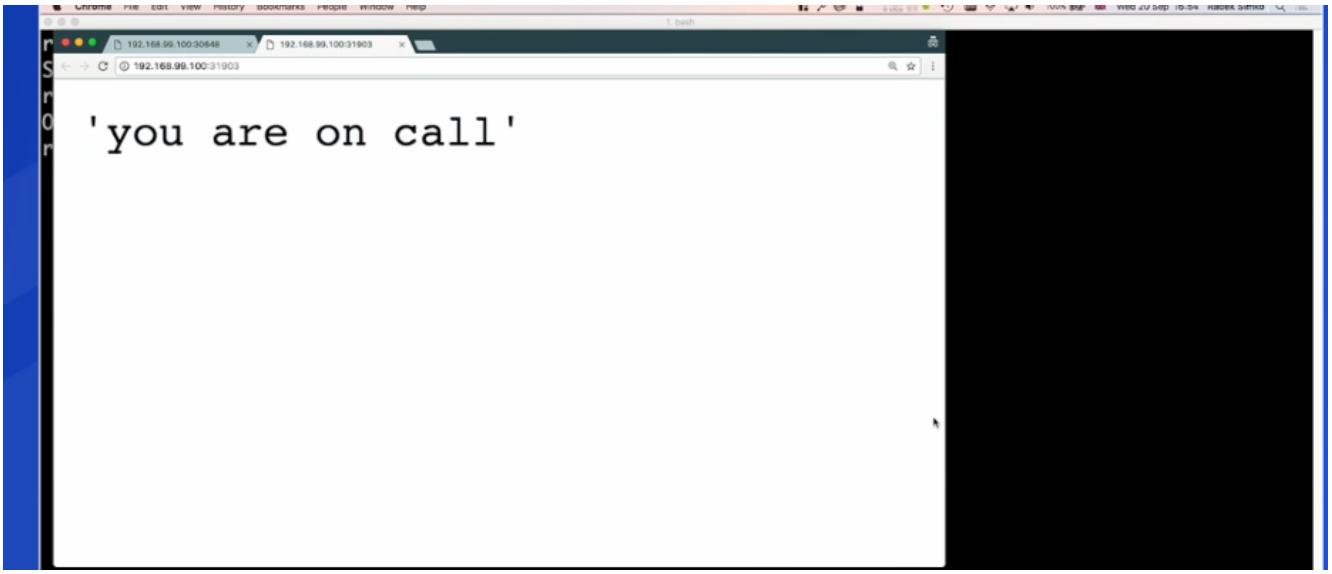

```
radeksimko@local:production $ terraform apply
```

```
  metadata.0.resource_version: "" => "<computed>"
  metadata.0.self_link:        "" => "<computed>"
  metadata.0.uid:              "" => "<computed>"
  spec.#:                      "" => "1"
  spec.0.cluster_ip:           "" => "<computed>"
  spec.0.port.#:               "" => "1"
  spec.0.port.0.name:          "" => "http"
  spec.0.port.0.node_port:     "" => "<computed>"
  spec.0.port.0.port:          "" => "80"
  spec.0.port.0.protocol:      "" => "TCP"
  spec.0.port.0.target_port:   "" => "5678"
  spec.0.selector.%:           "" => "1"
  spec.0.selector.app:         "" => "MyExampleApp"
  spec.0.session_affinity:     "" => "None"
  spec.0.type:                 "" => "NodePort"
kubernetes_service.example: Creation complete after 0s (ID: default/real-example)

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

Outputs:

lb_ingress = []
service_name = real-example
radeksimko@local:production $
```

```
  spec.#:                      "" => "1"
  spec.0.cluster_ip:           "" => "<computed>"
  spec.0.port.#:               "" => "1"
  spec.0.port.0.name:          "" => "http"
  spec.0.port.0.node_port:     "" => "<computed>"
  spec.0.port.0.port:          "" => "80"
  spec.0.port.0.protocol:      "" => "TCP"
  spec.0.port.0.target_port:   "" => "5678"
  spec.0.selector.%:           "" => "1"
  spec.0.selector.app:         "" => "MyExampleApp"
  spec.0.session_affinity:     "" => "None"
  spec.0.type:                 "" => "NodePort"
kubernetes_service.example: Creation complete after 0s (ID: default/real-example)

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

Outputs:

lb_ingress = []
service_name = real-example
radeksimko@local:production $ minikube service $(terraform output service_name)
Error opening service: Could not find finalized endpoint being pointed to by real-example: Error getting kubern
etes client: Error creating new client from kubeConfig.ClientConfig(): No Auth Provider found for name "gcp"
radeksimko@local:production $ kubectl
```

```
radeksimko@local:production $ kubectl config use-context minikube
Switched to context "minikube".
radeksimko@local:production $ minikube service $(terraform output service_name)
```

We can change the context and try again to see the app




```
radeksimko@local:production $ terraform workspace new prod
Created and switched to workspace "prod"!

You're now on a new, empty workspace. Workspaces isolate their state,
so if you run "terraform plan" Terraform will not see any existing state
for this configuration.
radeksimko@local:production $ terraform workspace list
  default
* prod

radeksimko@local:production $ terraform init

Initializing provider plugins...

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.
radeksimko@local:production $
```

We create a new workspace called prod and switch to that prod workspace. Then we run ***$ terraform init*** and then run ***$ terraform plan*** as above

```
      metadata.#:                   "1"
      metadata.0.generation:        "<computed>"
      metadata.0.name:              "real-example"
      metadata.0.namespace:         "default"
      metadata.0.resource_version:  "<computed>"
      metadata.0.self_link:         "<computed>"
      metadata.0.uid:               "<computed>"
      spec.#:                       "1"
      spec.0.cluster_ip:            "<computed>"
      spec.0.port.#:                "1"
      spec.0.port.0.name:           "http"
      spec.0.port.0.node_port:      "<computed>"
      spec.0.port.0.port:           "80"
      spec.0.port.0.protocol:       "TCP"
      spec.0.port.0.target_port:    "5678"
      spec.0.selector.%:            "1"
      spec.0.selector.app:          "MyExampleApp"
      spec.0.session_affinity:      "None"
      spec.0.type:                  "LoadBalancer"


Plan: 2 to add, 0 to change, 0 to destroy.
radeksimko@local:production $ terraform apply
```

We can see that we have 2 new resources to be created since we are now in the prod workspace and we need to create the GCP resources now

```
  spec.0.template.0.termination_grace_period_seconds:          "" => "30"
kubernetes_replication_controller.example: Creation complete after 1s (ID: default/real-example)
kubernetes_service.example: Creating...
  load_balancer_ingress.#:      "" => "<computed>"
  metadata.#:                   "" => "1"
  metadata.0.generation:        "" => "<computed>"
  metadata.0.name:              "" => "real-example"
  metadata.0.namespace:         "" => "default"
  metadata.0.resource_version:  "" => "<computed>"
  metadata.0.self_link:         "" => "<computed>"
  metadata.0.uid:               "" => "<computed>"
  spec.#:                       "" => "1"
  spec.0.cluster_ip:            "" => "<computed>"
  spec.0.port.#:                "" => "1"
  spec.0.port.0.name:           "" => "http"
  spec.0.port.0.node_port:      "" => "<computed>"
  spec.0.port.0.port:           "" => "80"
  spec.0.port.0.protocol:       "" => "TCP"
  spec.0.port.0.target_port:    "" => "5678"
  spec.0.selector.%:            "" => "1"
  spec.0.selector.app:          "" => "MyExampleApp"
  spec.0.session_affinity:      "" => "None"
  spec.0.type:                  "" => "LoadBalancer"
```

We are now letting k8s provision the LB

```
  spec.0.port.0.target_port:   "" => "5678"
  spec.0.selector.%:           "" => "1"
  spec.0.selector.app:         "" => "MyExampleApp"
  spec.0.session_affinity:     "" => "None"
  spec.0.type:                 "" => "LoadBalancer"
kubernetes_service.example: Still creating... (10s elapsed)
kubernetes_service.example: Still creating... (20s elapsed)
kubernetes_service.example: Still creating... (30s elapsed)
kubernetes_service.example: Still creating... (40s elapsed)
kubernetes_service.example: Still creating... (50s elapsed)
kubernetes_service.example: Creation complete after 56s (ID: default/real-example)

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

Outputs:

lb_ingress = [
    {
        hostname = ,
        ip = 104.198.132.67
    }
]
service_name = real-example
radeksimko@local:production $
```

We now have the app running in production also

Alpha/Beta

- alpha/beta
  - Deployment
  - Ingress
  - ...

Keep in mind

- DSL
- Ops-focused provider
- Culture & workflows come first
- Be aware of resource ownership

Thank you.

@radeksimko

github.com/terraform-providers/
terraform-provider-kubernetes