Deploy Kubernetes Load Balancer Service with Terraform

Kubernetes services

A service is a grouping of pods that are running on the cluster. Services are "cheap" and you can have many services within the cluster. Kubernetes services can efficiently power a microservice architecture.

Services provide important features that are standardized across the cluster: load-balancing, service discovery between applications, and features to support zero-downtime application deployments.

Each service has a pod label query which defines the pods which will process data for the service. This label query frequently matches pods created by one or more replication controllers. Powerful routing scenarios are possible by updating a service's label query via the Kubernetes API with deployment software.

Why Terraform?

While you could use `kubectl` or similar CLI-based tools mapped to API calls to manage all Kubernetes resources described in YAML files, orchestration with Terraform presents a few benefits:

- **One language** - You can use the same configuration language to provision the Kubernetes infrastructure and to deploy applications into it.
- **Drift detection** - `terraform plan` will always present you the difference between reality at a given time and the config you intend to apply.
- **Full lifecycle management** - Terraform doesn't just initially create resources, but offers a single command to create, update, and delete tracked resources without needing to inspect the API to identify those resources.
- **Synchronous feedback** - While asynchronous behavior is often useful, sometimes it's counter-productive as the job of identifying operation results (failures or details of created resource) is left to the user. e.g. you don't have the IP/hostname of the load balancer until it has finished provisioning, hence you can't create any DNS record pointing to it.
- **Graph of relationships** - Terraform understands relationships between resources which may help in scheduling - e.g. Terraform won't try to create a service in a Kubernetes cluster until the cluster exists.

Task 1. Clone the sample code

1. In Cloud Shell, start by cloning the sample code:

```
gsutil -m cp -r gs://spls/gsp233/* .
```

Copied!
content_copy

2. Navigate to the `tf-gke-k8s-service-lb` directory:

```
cd tf-gke-k8s-service-lb
```

Copied!
content_copy

Task 2. Understand the code

1. Review the contents of the `main.tf` file:

```
cat main.tf
```

Copied!
content_copy
Example output:

```
  ...

variable "region" {
  type        = string
  description = "Region for the resource."
}

variable "location" {
  type        = string
  description = "Location represents region/zone for the resource."
}

variable "network_name" {
  default = "tf-gke-k8s"
}

provider "google" {
  region = var.region
}

resource "google_compute_network" "default" {
  name                    = var.network_name
  auto_create_subnetworks = false
}

resource "google_compute_subnetwork" "default" {
  name                     = var.network_name
  ip_cidr_range            = "10.127.0.0/20"
  network                  = google_compute_network.default.self_link
  region                   = var.region
  private_ip_google_access = true
}
  ...
```

- Variables are defined for `region`, `zone`, and `network_name`. These will be used to create the Kubernetes cluster.
- The Google Cloud provider will let us create resources in this project.
- There are several resources defined to create the appropriate network and cluster.
- At the end, there are some outputs which you'll see after running `terraform apply`.

2. Review the contents of the `k8s.tf` file:

```
cat k8s.tf
```

Copied!
content_copy
Example output:

```
provider "kubernetes" {
  version = "~> 1.10.0"
  host    = google_container_cluster.default.endpoint
  token   = data.google_client_config.current.access_token
  client_certificate = base64decode(
    google_container_cluster.default.master_auth[0].client_certificate,
  )
  client_key =
base64decode(google_container_cluster.default.master_auth[0].client_key)
  cluster_ca_certificate = base64decode(

google_container_cluster.default.master_auth[0].cluster_ca_certificate,
  )
}

resource "kubernetes_namespace" "staging" {
  metadata {
    name = "staging"
  }
}

resource "google_compute_address" "default" {
  name   = var.network_name
  region = var.region
}

resource "kubernetes_service" "nginx" {
  metadata {
    namespace = kubernetes_namespace.staging.metadata[0].name
    name      = "nginx"
  }

  spec {
    selector = {
      run = "nginx"
    }

    session_affinity = "ClientIP"

    port {
```

```
      protocol    = "TCP"
      port        = 80
      target_port = 80
    }

    type             = "LoadBalancer"
    load_balancer_ip = google_compute_address.default.address
  }
}

resource "kubernetes_replication_controller" "nginx" {
  metadata {
    name      = "nginx"
    namespace = kubernetes_namespace.staging.metadata[0].name

    labels = {
      run = "nginx"
    }
  }

  spec {
    selector = {
      run = "nginx"
    }

    template {
      container {
        image = "nginx:latest"
        name  = "nginx"

        resources {
          limits {
            cpu    = "0.5"
            memory = "512Mi"
          }

          requests {
            cpu    = "250m"
            memory = "50Mi"
          }
        }
      }
    }
  }
}
```

```
output "load-balancer-ip" {
  value = google_compute_address.default.address
}
```

- The script configures a Kubernetes provider with Terraform and creates the service, namespace and a replication_controller resource.
- The script returns an `nginx` service IP as an output.

Task 3. Initialize and install dependencies

The `terraform init` command is used to initialize a working directory containing the Terraform configuration files.

This command performs several different initialization steps in order to prepare a working directory for use and is always safe to run multiple times, to bring the working directory up to date with changes in the configuration:

1. Run `terraform init`:

`terraform init`
Copied!
content_copy
Example output:

```
...
* provider.google: version = "~> 3.8.0"
* provider.kubernetes: version = "~> 1.10.0"

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.
```

2. Run the `terraform apply` command, which is used to apply the changes required to reach the desired state of the configuration:

```
terraform apply -var="region=us-east4" -var="location=us-east4-c"
```

Copied!
content_copy

3. Review Terraform's actions and inspect the resources which will be created.

4. When ready, type **yes** to begin Terraform actions.

On completion, you should see similar output:

Example output:

```
Apply complete! Resources: 7 added, 0 changed, 0 destroyed.

Outputs:

cluster_name = tf-gke-k8s
cluster_region = "us-east4"
cluster_zone = "us-east4"
load-balancer-ip = 35.233.177.223
network = https://www.googleapis.com/compute/beta/projects/qwiklabs-gcp-
5438ad3a5e852e4a/global/networks/tf-gke-k8s
subnetwork_name = tf-gke-k8s
```

Verify resources created by Terraform

1. In the console, navigate to **Navigation menu** > **Kubernetes Engine**.
2. Click on `tf-gke-k8s` cluster and check its configuration.
3. In the left panel, click **Gateways, Services & Ingress** and check the `nginx` service status.
4. Click the **Endpoints** IP address to open the `Welcome to nginx!` page in a new browser tab.

Click **Check my progress** to verify your performed task. If you have successfully deployed infrastructure with Terraform, you will see an assessment score.

Deploy infrastructure with Terraform

Check my progress

Congratulations!

In this lab, you used Terraform to initialize, plan, and deploy a Kubernetes cluster along with a service.

Finish your quest

This self-paced lab is part of the Managing Cloud Infrastructure with Terraform and DevOps Essentials quests. A quest is a series of related labs that form a learning path. Completing a quest earns you a badge to recognize your achievement. You can make your badge or badges public and link to them in your online resume or social media account. Enroll in any quest that contains this lab and get immediate completion credit. See the Google Cloud Skills Boost catalog to see all available quests.

Take your next lab

Continue your Quest with HTTPS Content-Based Load balancer with Terraform, or check out these suggestions:

- Modular Load Balancing with Terraform - Regional Load Balancer