Published in Level Up Coding

Pascal Zwikirsch Follow

Nov 28, 2022 · 5 min read · ✦ · Listen

Set up Helm on your GKE cluster with Terraform

In this brief article, I will show you how you can use Terraform to set up your Kubernetes cluster on GCP using GKE and apply your Helm chart using Terraform without any manual intervention.

GitHub Repository

If you want to see all th ater, I prepared a small repository for you

GitHub - mr-pascal/medium-tf-gke-helm

When this error happens: Error: Attempted to load application default credentials since neither credentials nor...

github.com

Why?

Normally you would log in to your cluster after setting it up via Terraform or ClickOps and apply the Helm chart there directly.

But besides you needing additional access and permissions, it might not be that nice for a big enterprise company. Also if you are using Terraform, you usually don't want to log in to your created infrastructure and do some manual imperative changes, even if it's just applying a Helm chart.

So in this article, you will see how you can apply your Helm Chart on your Kubernetes cluster (I chose GCP here), but actually, the Terraform defining the Helm chart resource doesn't care where your cluster is running.

The Code

But let's get down to the actual code. Even though I won't go through all the code, just the essential parts. Check out the public repository if you want to see the whole setup.

GKE Cluster Setup

You can skip this section if you are not running on GCP and want to use GKE.

```hcl
# google_client_config and kubernetes provider must be explicitly specified like the fo
# Retrieve an acces
data "google_client_
# GKE cluster
resource "google_container_cluster" "primary" {
  name     = "my-gke"
  project  = var.project
  location = var.region

  # We can't create a cluster with no node pool defined, but we want to only use
  # separately managed node pools. So we create the smallest possible default
  # node pool and immediately delete it.
  remove_default_node_pool = true
  initial_node_count       = 1

  networking_mode = "VPC_NATIVE"
  ip_allocation_policy {}
}

# Separately Managed Node Pool
resource "google_container_node_pool" "primary_nodes" {
  project    = var.project
  name       = "${google_container_cluster.primary.name}-node-pool"
  location   = var.region
  cluster    = google_container_cluster.primary.name
  node_count = 1


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

    labels = {
      env = var.project
    }

    preemptible  = true
    machine_type = "e2-small"
    tags         = ["gke-node"]
    metadata = {
      disable-legacy-endpoints = "true"
    }
  }
}
```

cluster.tf hosted with ❤ by GitHub view raw

terraform documentati default node pool is removed (“remove_def ely managed node pool (“google_container_node_pool” resource) is created based on the provider’s recommendation.

The data resource called “google_client_config” is essential since the Helm provider will need this one to get a token to connect to the GKE cluster.

Helm Setup

The Helm chart that will be applied can be found in the helm folder of the mentioned repository. I won’t detail those files since they are irrelevant and could be some arbitrary Helm chart you want to apply.

In the following snippet, you can find the Helm provider and Terraform resource initialization.

```
provider "helm" {
  kubernetes {
    host                   = "https://${google_container_cluster.primary.endpoint}"
    token                  = data.google_client_config.default.access_token
  }
}

resource "helm_release" "example" {
  name  = "my-local-chart"
  chart = "./helm"

  depends_on = [
    google_container_cluster.primary
  ]
}
```

helm.tf hosted with ❤ by GitHub view raw

To know where to apply the Helm chart and how to connect to the underlying Kubernetes cluster, the helm provider has to be initialized with an access token (“token”) for the cluster and, of course, where the cluster can be reached (“host”). There are other ways to initialize it, but that is the easiest one with the current setup.

If you are using a Terraform-managed GKE cluster, you can get those values by referencing the cluster uration (see "GKE Cluster Setup" section).

The "helm_release" resource is the resource that represents the Helm chart you want to apply. You have to reference the local file path to your Helm chart, e.g., "./helm" to reference the "helm" folder on the same level as this Terraform code.

If you maintain your GKE cluster (or any other cloud) via Terraform, it's essential to add the "depends_on" meta-argument.

The reason is that Helm needs to know when the cluster is ready before accessing it. If the "depends_on" wouldn't be there and the cluster would be created together with the Helm resource, then it would fail since Helm would try to access a cluster that is not ready yet, since it's just being created. That is why the dependency between those resources has to be set explicitly.

Run It

But now, let's verify that everything is working as expected. For this, it's best to clone the repository and have your local gcloud SDK (if you use GCP) configured before running it.

For the upcoming shell commands, make sure that you replace "YOUR_PROJECT" with your GCP project.

First, let's cross-check the Terraform plan via the following:

```
terraform plan -var "project=YOUR_PROJECT"
```

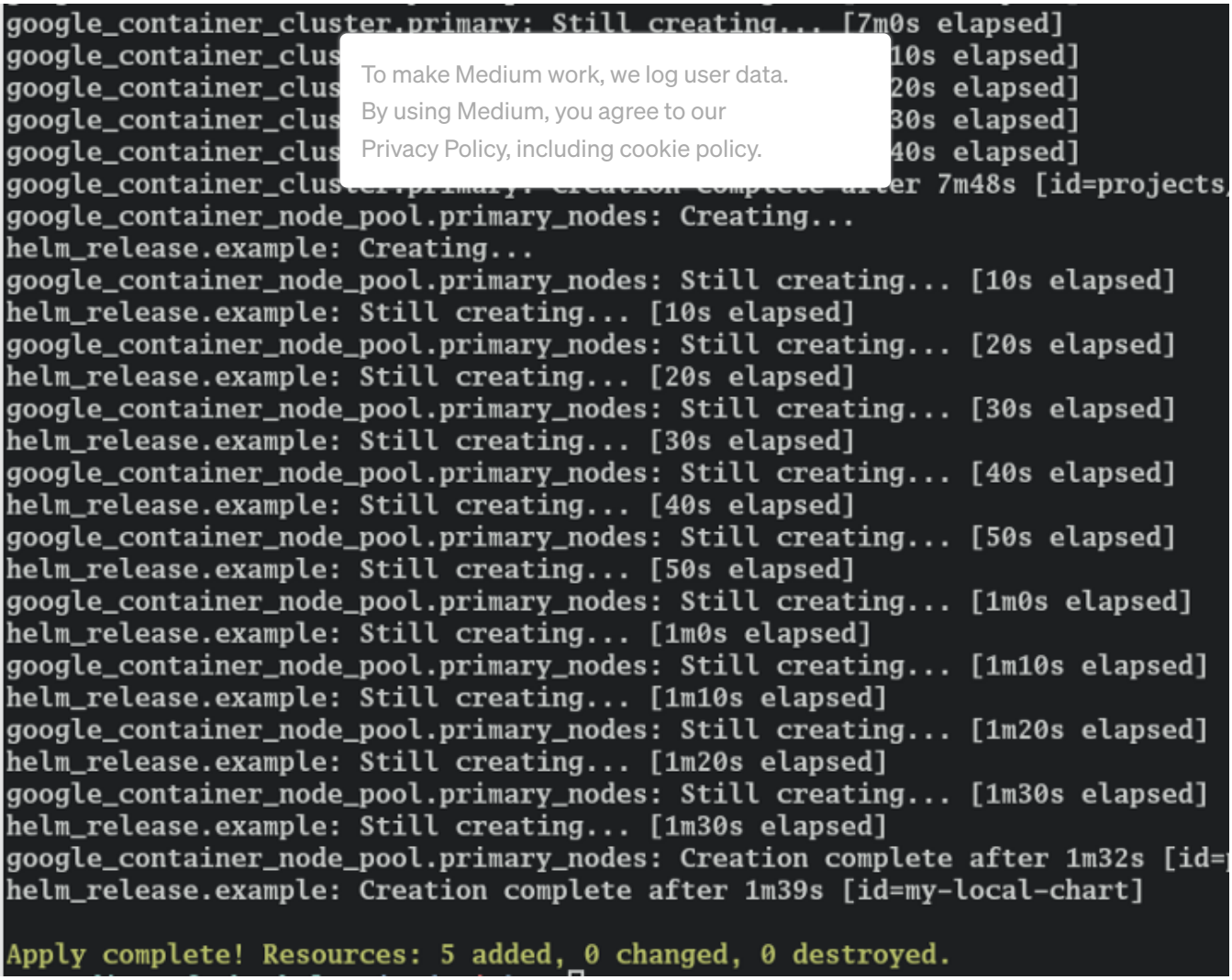
Terraform plan

The whole plan is too long to post here, but if you run it on your machine, feel free to inspect it further.

Overall, five resources look good since we have the three resources mentioned above and two API enablement in the “main.tf” file you don’t have to worry about.

Now let’s apply it:

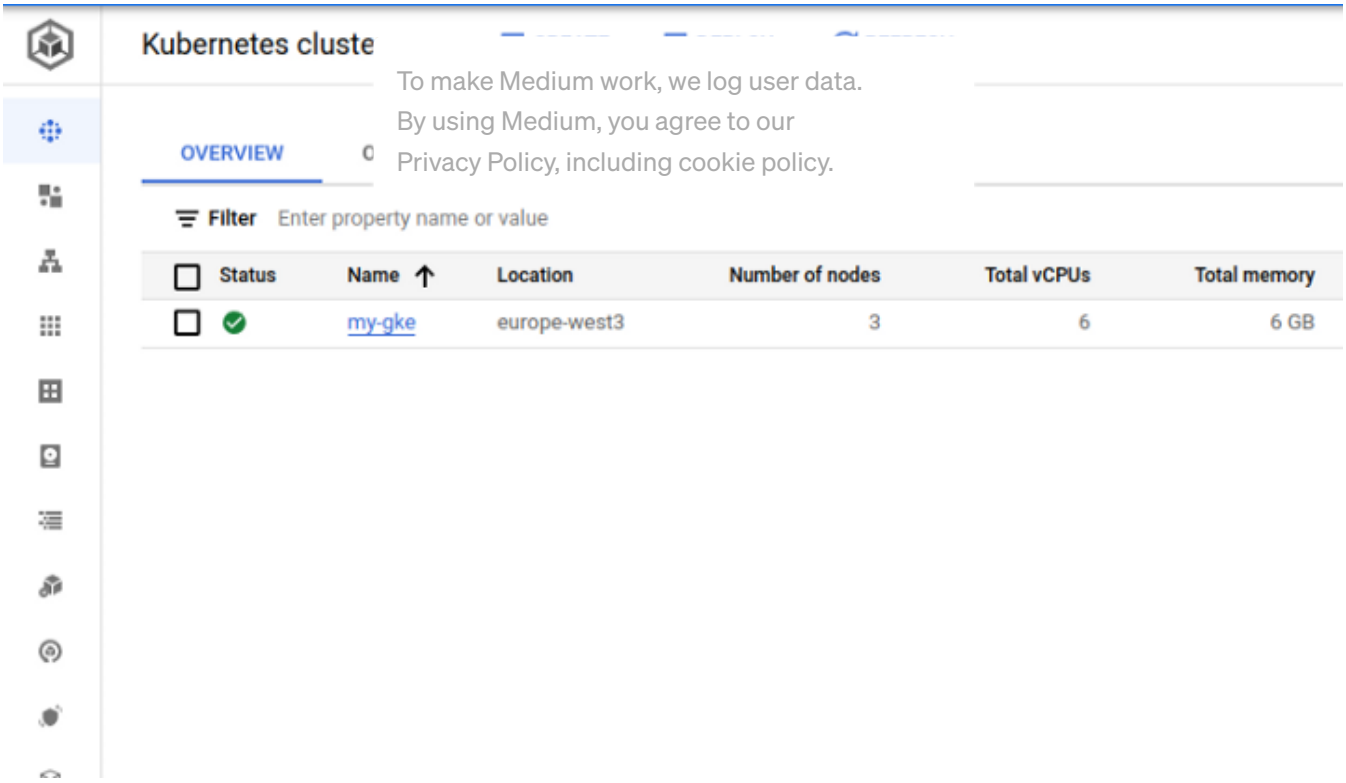
```
terraform apply -var “project=YOUR_PROJECT”
```



Terraform apply

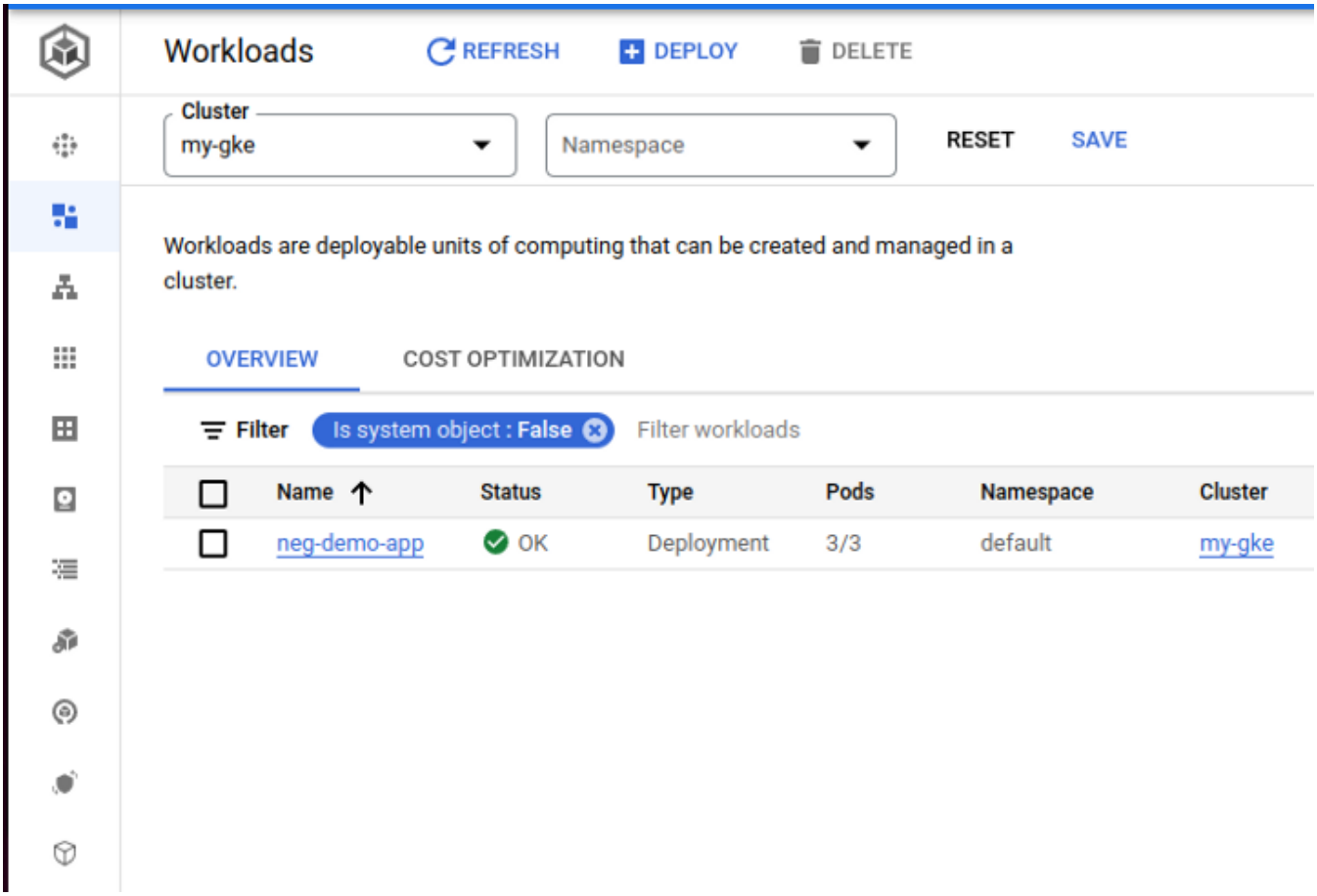
As you can see in the above image, it took approximately 8 minutes (but I also saw times up to 15 minutes) to create the cluster and another 1.5 minutes to set up the node pool and apply the Helm chart. So be prepared for quite some waiting time when working with clusters.

A quick look into the GCP console also verifies that the cluster is running



Kubernetes Clusters in GCP

and the deployments that were defined in the Helm chart are also up and healthy:



Kubernetes Workloads in GCP

Cleanup

Now, if you want to clean up your cluster and Helm chart again, just run the following, but be prepa ıtes till the cluster is shut down:


```
terraform destroy -var "project=YOUR_PROJECT"
```


Also, feel free to check out my book recommendation 📚.

My Book Recommendations

In the following sections, you can find my book recommendations on all my daily life topics that greatly helped me.

mr-pascal.medium.com

Join Medium with my referral link — Pascal Zwikirsch

As a Medium member, a portion of your membership fee goes to writers you read, and you get full access to every story...

mr-pascal.medium.com

Kubernetes Terraform Helm Google Cloud Platform Infrastructure

Enjoy the read? Rewa

Your tip will go to Pascal Zwikirs(ng them know you appreciate their story.

Give a tip

Sign up for Top Stories

By Level Up Coding

A monthly summary of the best stories shared in Level Up Coding Take a look.

By signing up, you will create a Medium account if you don't already have one. Review our Privacy Policy for more information about our privacy practices.

Get this newsletter