1. Terraform ile Google Cloud Platform'da Kubernetes Kurulumu

Terraform dosyalarınızı oluşturacağınız dizine geçelim. Bu dizinde değişkenleri tanımlamak için **variables.tf** ana işlemler için ise **main.tf** dosyalarını oluşturuyoruz.

```
variable "project_id" {
  type = string
  default = "gproject-360616"
}
```

```
variable "region" {
  type = string
default = "europe-west1"
}
```

project_id ve **region** isimlerinde iki adet değişkenin tanımlanmasını gerçekleştiriliyor

type kısmında veri tipi, **default** kısmında değişkenin varsayılan değeri belirtiliyor

main.tf

```
provider "google" {
 credentials = file("gproject-360616-1decd194ddc5.json")
 project     = var.project_id
}
```

Google Cloud, **provider** olarak ekleniyor. Kimlik bilgileri dosyadan okutuluyor. **project** değişkeninin değeri, daha önce hazırladığımız dosyadan değişken olarak getiriliyor.

```
resource "google_compute_network" "vpc_network" {
  name         = "my-vpc"
  project      = var.project_id
}
```

Google Cloud üzerinde **Virtual Private Cloud** oluşturmak için gerekli resource yazılarak, vpc için belirtilmek istenen bilgiler kolonun içine giriliyor.

```
resource "google_container_cluster" "gke_cluster" {
  name     = "my-cluster"
  project = var.project_id
  location = var.region
  network = google_compute_network.vpc_network.name
  remove_default_node_pool = true
  initial_node_count       = 2
}
```

Google Kubernetes Engine üzerinde **Cluster** oluşturmak için gerekli resource yazılarak, yukarıda oluşturulan VPC Network'üne ekleniyor. Cluster için belirtilmek istenen diğer parametreler yine kolonun içine giriliyor.

```
resource "google_container_node_pool" "nodepool" {
  name       = "my-node-pool"
  project    = var.project_id
  location   = var.region
  cluster    = google_container_cluster.gke_cluster.name
  node_count = 2
  node_config {
    preemptible  = true
    machine_type = "e2-micro"
  }
  autoscaling {
    min_node_count = 1
    max_node_count = 3
  }
  depends_on = [
    google_container_cluster.gke_cluster
  ]
}
```

Google Kubernetes Engine üzerinde Node Pool oluşturmak için gerekli resource yazılıyor. Kolon içinde node sayısı, sanal makine cinsi, autoscaling gibi belirtilmek istenen parametreler giriliyor.

```
resource "null_resource" "configure_kubectl" {
  provisioner "local-exec" {
    command = "gcloud container clusters get-credentials
${google_container_cluster.gke_cluster.name} --region
${google_container_cluster.gke_cluster.location} --project
${google_container_cluster.gke_cluster.project}"
  }
  depends_on = [
    google_container_cluster.gke_cluster
  ]
}
```

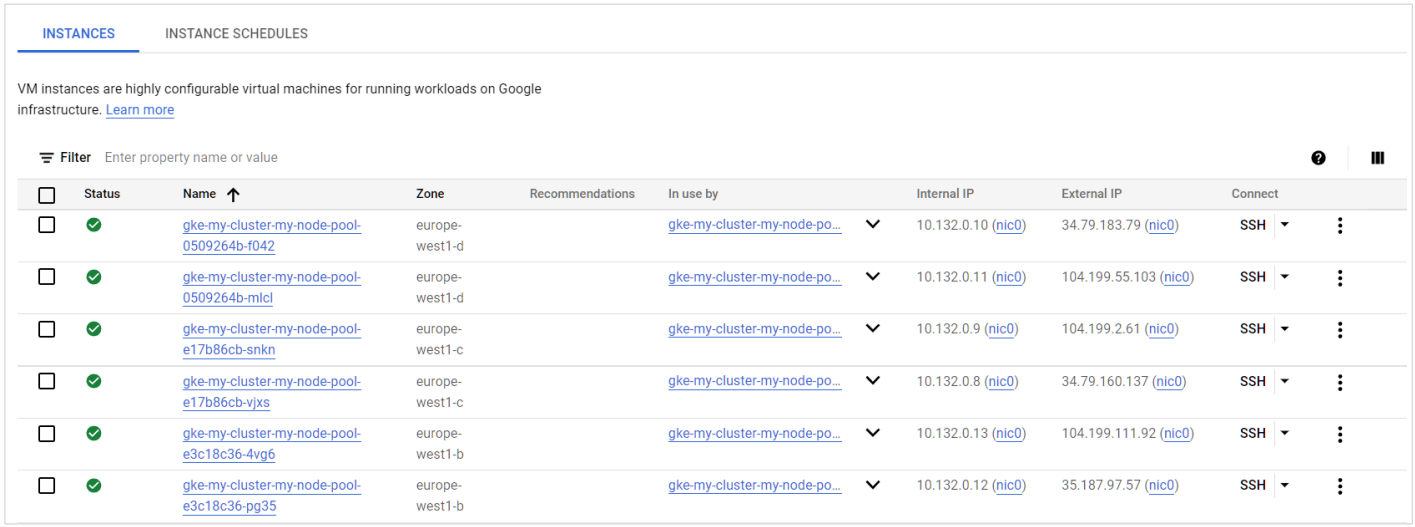
Yerel gcloud - kubectl ayarlaması yapılıyor.

Terraform Uygulama

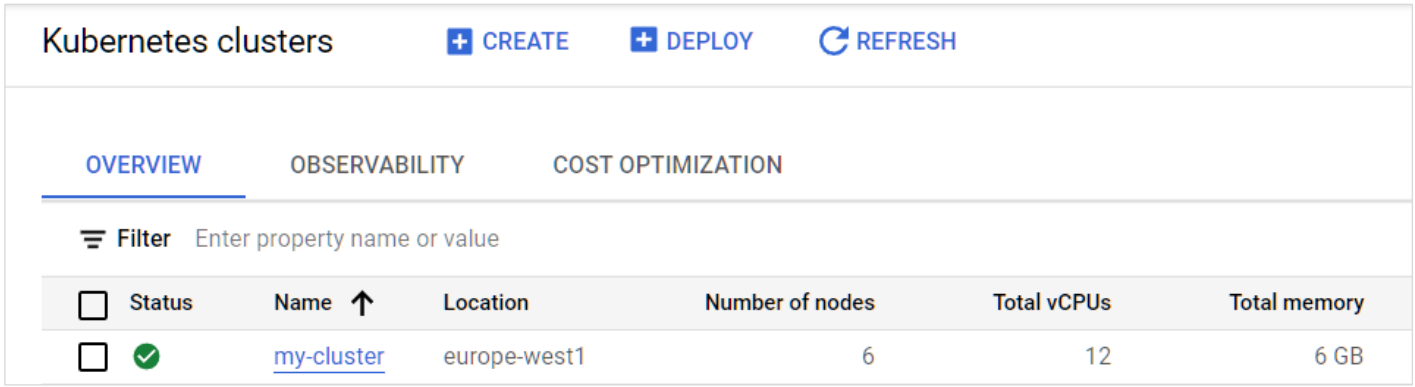
- Komut satırında Terraform dosyalarınızın bulunduğu dizine geçin.
- **‘terraform init’** komutu ile Terraform yapılandırma dosyalarını hazırlayın. Bu komut, yeni bir Terraform konfigürasyonu sonra çalıştırılması gereken ilk komuttur.
- **‘terraform validate’** komutu ile oluşturduğunuz dosyaların sözdizimsel olarak doğruluğunu ve gerçeliliğini kontrol edin.
- Son olarak **‘terraform apply’** komutu ile konfigürasyonunuzdan bir Terraform planı oluşturup, yapılacaklardan emin iseniz **‘yes’** yazarak gerekli işlemleri başlatmasına onay verin.

Tebrikler! Google Cloud Platorm üzerinde **VPC**, **GKE Cluster** ve **GKE Cluster Nodepool**, yerelde ise kubectl ile gcloud’u yönetecek şekilde gerekli ayarlamaları yaptınız.

Google Cloud Üzerinde Çalışan Kubernetes Node'larının Ekran Görüntüsü

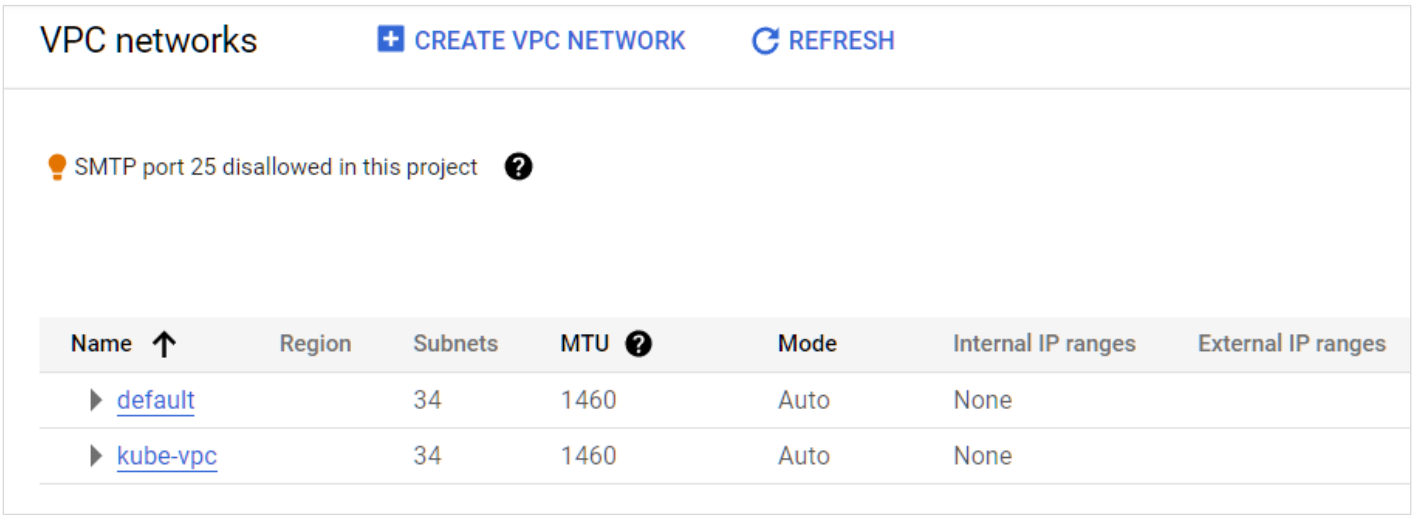


Google Cloud Kubernetes Engine Ekran Görüntüsü



Europe West1 Region'ındaki Cluster'ımızda, Terraform'da belirttiğimiz şekilde tüm Zone'lar için 2 adet node çalıştığı görülüyor.

Google Cloud VPN Networks Ekran Görüntüsü



Oluşturduğumuz kube-vpc ağı görülüyor.

2. Kubernetes Üzerine MySQL Kurulumu Hazırlığı

Kubernetes üzerine MySQL kurmak için **Secret, Deployment**, **Service** ve **Persistent Volume Claim** olmak üzere dört adet Kubernetes nesnesine ihtiyacınız var. Bu nesneleri oluşturmak ayrı ayrı, veya hepsini içeren tek bir **.yaml** dosyası hazırlayabilirsiniz.

secrets.yaml

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: mysql
type: Opaque
data:
  password: bXlwYXNzMTIz
```

apiVersion; nesneyi oluşturmak için hangi Kubernetes API sürümünü kullanılacağı belirtiliyor

kind; hangi Kubernetes nesnesi olduğu belirtiliyor

type; Secret tipi belirtiliyor

data; anahtar-değer çiftleri belirtiliyor

mysql-persistentvolumeclaim.yaml

```yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: mysql-volumeclaim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 4Gi
```

kind; hangi Kubernetes nesnesi olduğu belirtiliyor

apiVersion; nesneyi oluşturmak için hangi Kubernetes API sürümünü kullanılacağı belirtiliyor

accessModes; Volume'e erişimin kapsamı belirtiliyor

Storage; istenen depolama miktarı belirtiliyor

mysql-service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: mysql
  labels:
    app: mysql
spec:
  type: ClusterIP
  ports:
    - port: 3306
      targetPort: 3306
  selector:
    app: mysql
```

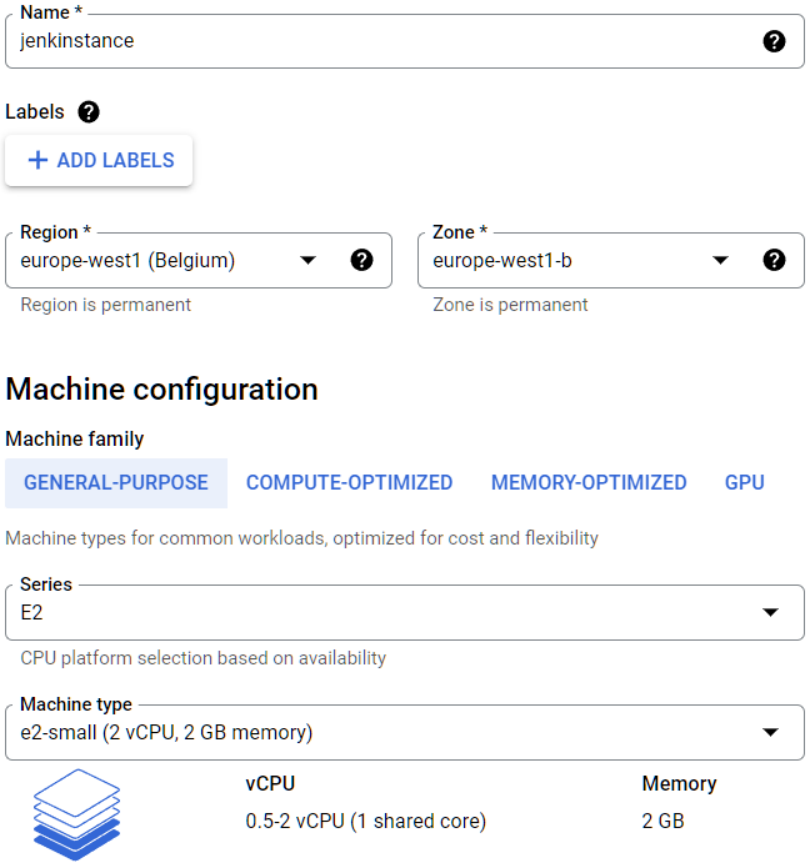
kind; hangi Kubernetes nesnesi olduğu belirtiliyor

apiVersion; nesneyi oluşturmak için hangi Kubernetes API sürümünü kullanılacağı belirtiliyor

Type: hangi tip Service olduğu belirtiliyor

3. Jenkins Kurulumu

Öncelikle Jenkins kullanmak için bir sanal makineye yüklemeli, veya Container'da kullanmalısınız. Biz burada yeni bir **sanal makineye Jenkins kuracağız**.



Görüntüde Google Compute Engine hizmetinden Jenkins için kullanacağımız sanal makinenin oluşturulması örenği görülüyor.

Buradan sanal makinenin ismi, çalışacağı Region ve Zone, makinenin donanımsal özellikleri, işletim sistemi ve network gibi bir çok detayı kurulum aşamasında ayarlanabiliyor.

Kurulum tamamlandıktan sonra Compute Engine arayüzünden SSH tuşuna tıklayarak makineye SSH bağlantımızı gerçekleştirebiliriz.
Not: Firewall tanımlamalarınızdan 22 portuna erişim izni vermeniz gerekebilir.

Java ve Jenkins Kurulumu
SSH bağlantımızı tamamladıktan sonra Jenkins kurulumu için önce Java'yı kuracağız.

```
sudo apt update
sudo apt install openjdk-11-jre
```

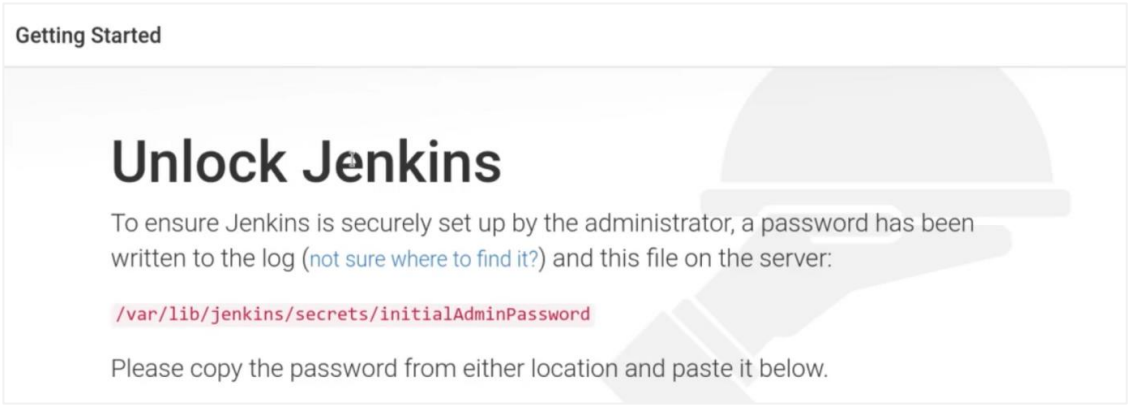
Jenkins Kurulumu:

```
curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo tee \
  /usr/share/keyrings/jenkins-keyring.asc > /dev/null
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
 https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
 /etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update
sudo apt-get install Jenkins
```

```
sudo systemctl enable jenkins
sudo systemctl start jenkins
sudo systemctl status jenkins
```

Eğer süreci başarıyla tamamladıysak Jenkins 8080 portunda çalışmaya başlamış demektir. Eğer bu adrese erişemiyorsanız yine firewall ayarlarından 8080 portunu eklemeniz gerekebilir.

Karşımıza çıkan ilk sayfaya Jenkins admin şifresini girmemiz isteniyor.



Bilgilendirmede görülen adrese cat komutu ile okuma yaparak admin şifremizi öğrenebiliriz.

Şifreyi girdikten sonra Plugin yüklemeleri ile alakalı sayfa gelecektir. Buradan **Install Suggested Plugins**'i seçerek default bir kurulum yapabiliriz.

Son kurulum aşamasında ise bir kullanıcı oluşturmamız isteniyor. Gerekli alanları doldurarak kurulumu bitirebiliriz. **Tebrikler Jenkins hazır!**

4. Kubernetes Üzerine MySQL Kurulumu

Önceki adımlardan **Kubernetes Üzerine MySQL Kurulumu Hazırlığı** adımında gösterdiğim .yaml dosyalarını kubectl ile kullanmak için sanal makineye veya direkt olarak kendi bilgisayarımıza kubectl yükleyebiliriz. Yüklü kubectl'i Google Cloud Kubernetes Cluster'ı ile aşağıdaki komutu kullanarak eşleyebilir kubectl ile yönetime başlayabiliriz.

```
gcloud container clusters get-credentials --region=europe-west1 my-cluster
```

Github'dan dosyaları da kullandığımız makineye indirdiysek sırada kubectl komutlarının kullanımı bulunuyor. **https://github.com/ahmettsezis/enuygunbootcampproject**

Wordpress ve MySQL'i çalışır hale getirmek için indirdiğimiz .yaml dosyalarının bulunduğu dizine geçip aşağıdaki komutları sırasıyla çalıştırıyoruz

```
kubectl apply -f mysql-all.yaml
kubectl apply -f wordpress-volumeclaim.yaml
kubectl apply -f wordpress-deployment.yaml
kubectl apply -f wordpress-service.yaml
```

Tüm komutları çalıştırdıktan kısa bir süre sonra her şeyin hazır hale gelmesi gerekiyor. Bir sorun olması halinde `kubectl describe pod/service/deployment` komutlarıyla sorunu tespit etmeye çalışabiliriz.

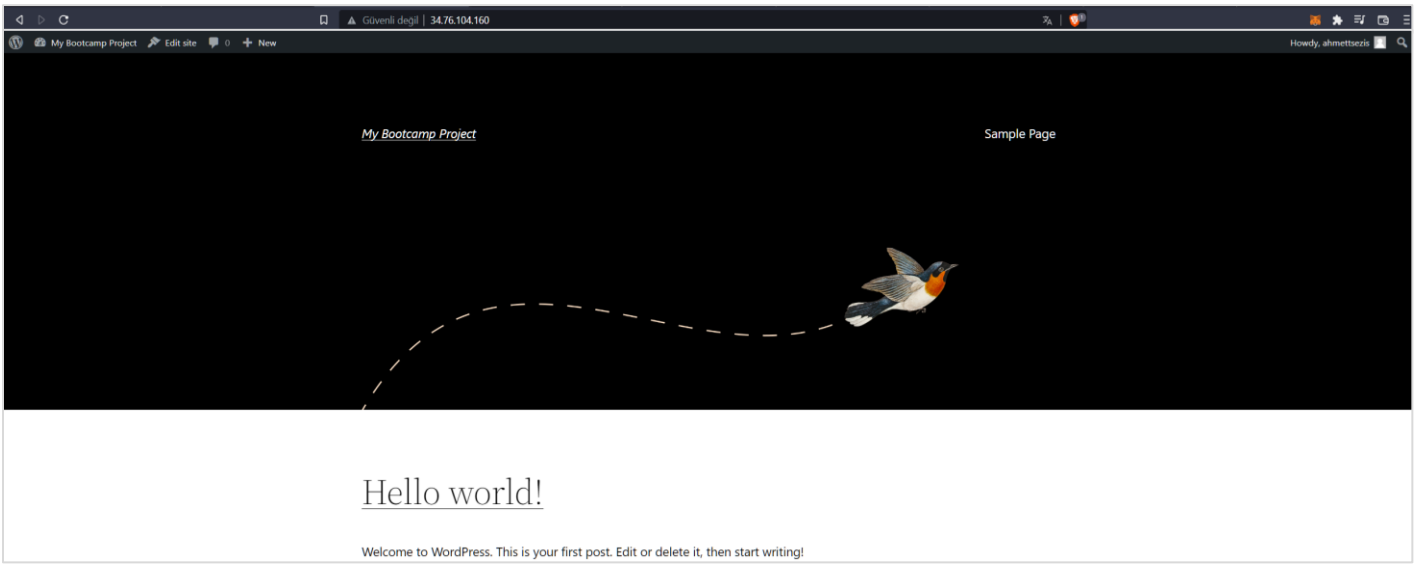
İşlemler sorunsuz şekilde tamamlandıysa aşağıdaki komutu çalıştırıp Wordpress ip adresimizi öğrenelim.

```
kubectl get service
```

Çıktıda gördüğünüz EXTERNAL-IP dışarıdan erişime açık olan IP'dir. Port Firewall'dan da erişime açıksa web sayfasına bağlanabilirsiniz.

```
NAME         TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)        AGE
kubernetes   ClusterIP      10.23.240.1     <none>          443/TCP        4h39m
mysql        ClusterIP      10.23.251.230   <none>          3306/TCP       177m
wordpress    LoadBalancer   10.23.249.248   34.76.104.160   80:31069/TCP   175m
```

IP açıldığında gözüken Wordpress web sayfası:



5. Ingress Tanımı

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: myingress
  labels:
    app: wordpress
spec:
  rules:
  - http:
      paths:
      - path: /test
        pathType: Prefix
        backend:
          service:
            name: wordpress
            port:
              number: 80
```

kind; hangi Kubernetes nesnesi olduğu belirtiliyor

apiVersion; nesneyi oluşturmak için hangi Kubernetes API sürümünü kullanılacağı belirtiliyor

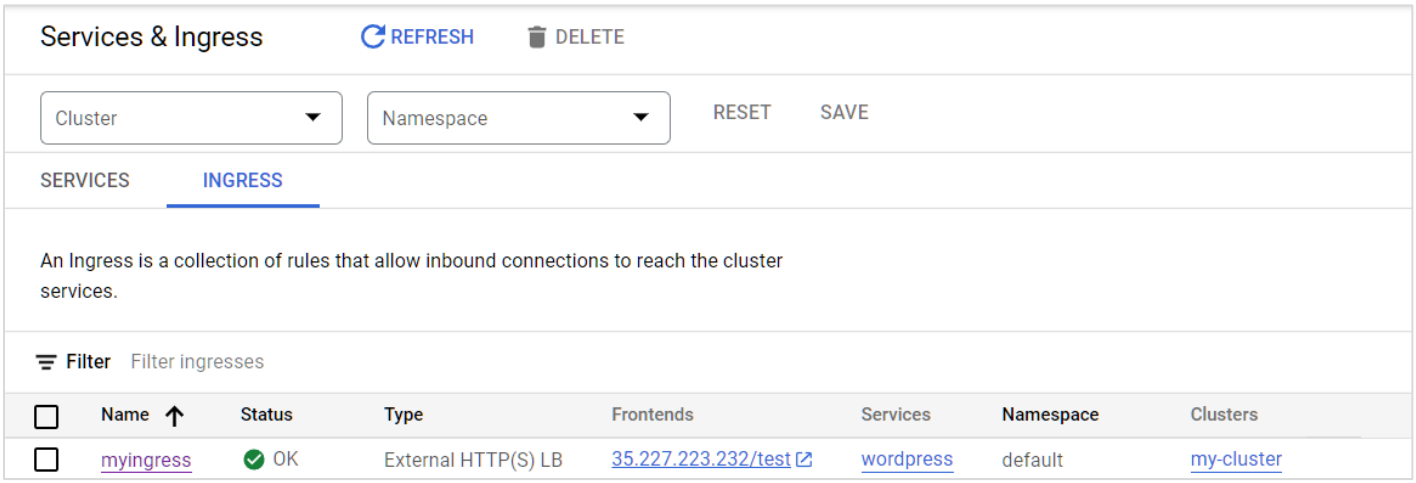
path; url yolu tanımlanıyor

backend; yönlendirilecek service tanımlanıyor

Artık aşağıdaki komut ile ingress'i devreye alabiliriz.

```
kubectl apply -f ingress.yaml
```

Ingress başarılı bir şekilde tanımlanırsa bir süre sonra Google Cloud Kubernetes Engine, Serive & Ingress alanında aşağıdaki gibi görünecektir.



Bu ingressin IP adresine gidildiğince bir şey çıkmayacak fakat girdiğimiz kural gereği IP'nin /test yoluna gitiğimizde yine bizi **Wordpress Service'i karşılayacaktır.**

5. Jenkins İle Yeni Wordpress'ler Çalıştırma

Jenkins üzerinde bir iş oluşturmalıyız. Ben burada **Pipeline**'ı tercih ettim. Bu Pipeline çalıştırıldığında bir parametre isteyecek. Bu parametre sizin yeni Wordpress-MySQL-Service-VolumeClaim setinizi parametreye verdiğiniz değer ile çalıştıracak ve diğerlerinden bir set halinde ayırt edilebilmesini ve sorunsuz çalışmasını sağlayacak.

Parametreye verdiğiniz isim (değer), tüm Pipeline sürecinde tüm .yaml dosyalarındaki gerekli alanlara yazılacak. Bunun için **Pipeline Utility Steps** Jenkins pluginininden faydalandım. ReadYaml ve WriteYaml metodları bu işi gerçekleştirmeye yardımcı olacaklar.

Pipeline Akış Şeması (kod hali Github'da bulunuyor)