


Bu case study’de basitçe bir DevOps süreci yürütölmüş, 2 günde tamamlanmıştır. Nice to have işlemlerle birlikte detaylara bakılacak olursa:

1. Terraform kullanarak Google Kubernetes Engine kullanılarak bir Kubernetes cluster oluşturulmalıdır. Cluster ın logging ve monitoring özellikleri disable edilmelidir. Aynı operasyon sırasında 2 adet node pool da açılmalıdır. 1. node pool un adı main-pool 2. node pool un adı application-pool olmalıdır. Bu node pool’lar açılırken n2d makina type kullanılmalı ve cluster europe-west-1 region da açılmalıdır. main-pool da auto scaling kapalı olurken application-pool un 1 - 3 node arasında auto scaling yapması sağlanmalıdır.
2. Cluster available olduktan sonra, Cluster üzerine YAML manifest kullanarak örnek bir Kubernetes uygulaması deploy edilmeli ve bu uygulamanın sadece application-pool üzerinde çalışması sağlanmalıdır. Bu uygulama, spesifik olarak sonraki adımlarda da istendiğı için auto-scaling’i destekleyen bir uygulama deploy edilecektir. Aynı zamandaö deploy edilen bu uygulamada HPA aktifleştirilmelidir. CPU kullanımı %25’in üzerinde olursa deployment 1 - 3 pod arası scale olması sağlanmalıdır. Ek olarak, bunun testi sağlanıp gösterimi yapılacaktır.
3. Cluster üzerinde Prometheus ve Grafana kurulumu yapılmalı ve Kubernetes metriklerinin Prometheus’a aktarılması sağlanmalıdır.
4. Grafana üzerinden pod restart alarmı kurulmalıdır.
5. Cluster’a keda kurulumu yapılmalı ve 2. madde keda kullanılarak da yapılabilmelidir.
6. Cluster’a istio kurulumu yapılmalıdır. (istiod, istio-ingress, istio-egress) Bu aşamada sadece kurulum istendiğı için, ilgili bileşenler aktif edilip available oldukları gösterilecek ve kullanım amaçları bu raporda açıklanacaktır.

Bu aşamaları gerçekleştirmeden önce, bazı gereksinimler olduğunu belirtmek gerek. İlk olarak, deneme sürümünü kullanarak bir Google Cloud hesabı aktif edilmeli. Bunun için Google Cloud Provider sitesine gidip halihazırda var olan bir Gmail hesabı ile hızlıca giriş yaptım. Sonrasında, sınırlı süreli bakiyeyi de aktif ettikten sonra artık platformu kullanmaya başlayabilir hale geldim.


Daha sonrasında, bu case’de gerekli olduğu için öncelikle kendi local’ime Terraform ve istenilen ekstra teknolojileri de aktif etmek için Helm yükledim. Bunları “apt install” komutları ile hızlıca yükleyebildim.

Bunlardan sonra, Terraform’un düzgün çalışması için GCP console’unda bazı ayarlamalar yapmak gerekiyor. Öncelikle bir proje oluşturdum çünkü bütün kaynakları bu projeye konumlandırmak gerekiyor:



Search (/) for resour


New Project

 You have 10 projects remaining in your quota. Request an increase or delete projects. [Learn more](#)

[Manage Quotas](#)

Project name *
GKE Terraform

Project ID: gke-terraform-460420. It cannot be changed later. [Edit](#)

Location *
 No organization [Browse](#)

Parent organization or folder

[Create](#) [Cancel](#)

Sonrasında, Terraform'un yetkilendirmeye sahip olması için Service Account oluşturdum ve kullandığı servisler için ilgili yetkilerini ekledim ve onayladım:

[←](#) Create service account

1 Service account details

Service account name
Terraform SA

Display name for this service account

Service account ID *
terraform-sa

Email address: terraform-sa@gke-terraform-460420.iam.gserviceaccount.com

Service account description
Describe what this service account will do

[Create and continue](#)

2 Grant this service account access to project (optional)

3 Grant users access to this service account (optional)

[Done](#) [Cancel](#)

[←](#) Create service account

✓ Service account details

2 Grant this service account access to project (optional)

3 Grant users access to this service account (optional)

Grant this service account access to GKE Terraform so that it has permission to complete specific actions on the resources in your project. [Learn more](#)

Role

Kubernetes Engine Admin

Full management of Kubernetes Clusters and their Kubernetes API objects.

IAM condition (optional) ⓘ

+ Add IAM condition

🗑

Role

Service Account User

Run operations as the service account.

IAM condition (optional) ⓘ

+ Add IAM condition

🗑

Role

Compute Admin

Full control of all Compute Engine resources.

IAM condition (optional) ⓘ

+ Add IAM condition

🗑

[+ Add another role](#)

Continue

Done

Cancel

[←](#) Create service account

✓ Service account details

✓ Grant this service account access to project (optional)

3 Grant users access to this service account (optional)

Grant access to users or groups that need to perform actions as this service account. [Learn more](#)

Service account users role ⓘ

Grant users the permissions to deploy jobs and VMs with this service account

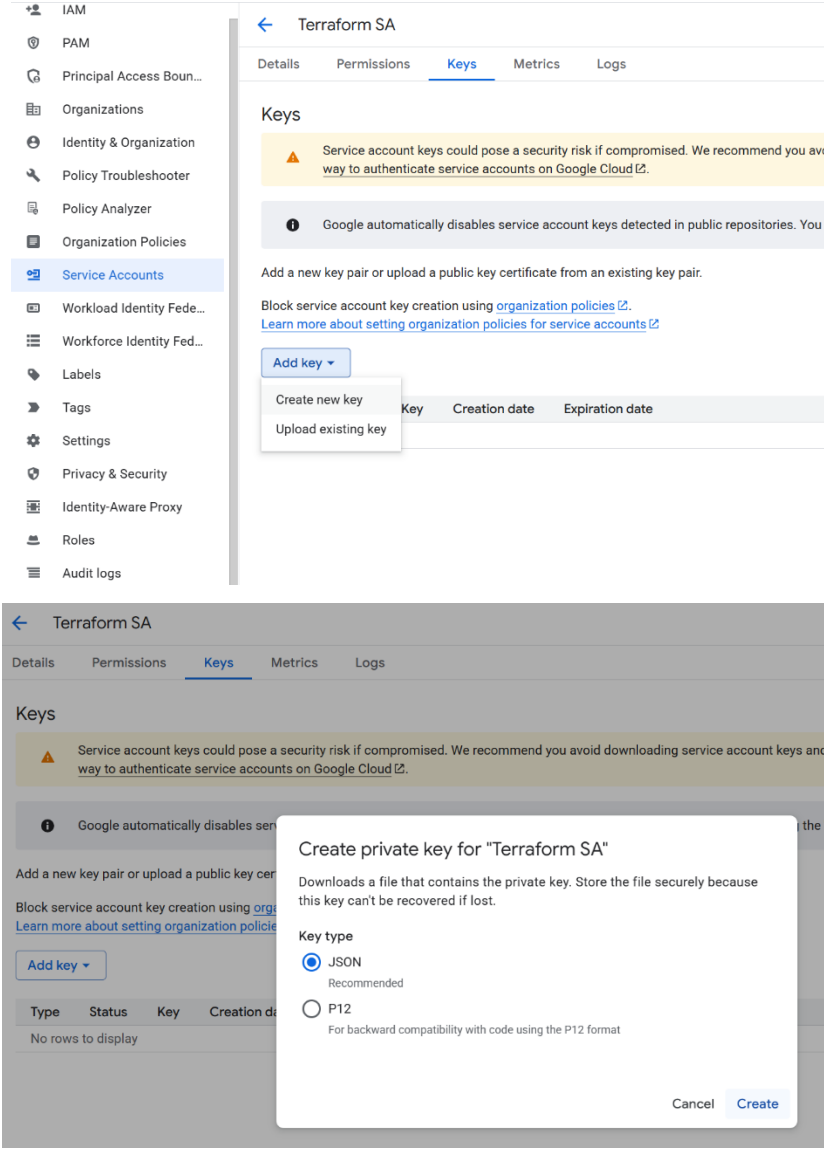
Service account admins role ⓘ

Grant users the permission to administer this service account

Done

Cancel

Daha sonrasında, Terraform'un ihtiyaç duyduğu key'i, oluşturduğum service account içerisinde temin ettim:



Bu şekilde artık adımları gerçekleştirmeye başlayabilir hale geldim.

1- Terraform ile GKE Cluster, Node Pool ve Node Pool Autoscaling Oluşturulması

Case'e istenilen duruma göre, Terraform kullanımı ile kaynaklarımı ve ilgili ayarlarını yapılandırmam gerekti. Bunun için basit bir Terraform projesi başlatarak, 2 adet dosyasını oluşturdum. Bunlardan ilki, provider.tf dosyası idi:

```
provider "google" {
  project      = "gke-terraform-460420"
  region      = "europe-west1"
  credentials = file("~/gcp/service-account.json")
}
```

Bu dosyaya ve Terraform'un kendi dokümantasyonuna göre, project-id, region ve service

account key'inin tanımlanması gerekiyor. Bu şekilde Terraform GCP'ye erişip ilgili proje içerisinde kaynakları konumlandırabiliyor. Buna bağlı olarak, asıl operasyonların gerçekleştiği main.tf dosyasını da aşağıdaki gibi oluşturdum:

1. VPC Network

```
#resource "google_compute_network" "main_vpc" {
#  name                = "case-vpc"
#  auto_create_subnetworks = false
#}
```

2. Subnet

```
#resource "google_compute_subnetwork" "main_subnet" {
#  name            = "case-subnet"
#  ip_cidr_range = "10.10.0.0/16"
#  region         = "europe-west1"
#  network        = google_compute_network.main_vpc.id
#}
```

3. GKE Cluster

```
resource "google_container_cluster" "primary" {
  name          = "case-cluster"
  location      = "europe-west1"
  node_locations = ["europe-west1-b"]
  remove_default_node_pool = true
  initial_node_count      = 1

  network    = "default"
  subnetwork = "default"

  logging_service    = "none"
  monitoring_service = "none"
}
```

Kubernetes API erişimini, kolaylık olması açısından public internete açabiliriz.

```
master_authorized_networks_config {
  cidr_blocks {
    cidr_block    = "0.0.0.0/0"
    display_name = "All networks"
  }
}
}
```

4. main-pool (autoscaling kapalı)

```
resource "google_container_node_pool" "main_pool" {
  name = "main-pool"
}
```

```

cluster    = google_container_cluster.primary.name
location   = google_container_cluster.primary.location

node_count = 1
node_config {
  machine_type = "n2d-standard-2"
  oauth_scopes = [
    "https://www.googleapis.com/auth/cloud-platform"
  ]
  labels = {
    pool = "main"
  }
}
}

# 5. application-pool (autoscaling açık)
resource "google_container_node_pool" "application_pool" {
  name          = "application-pool"
  cluster       = google_container_cluster.primary.name
  location      = google_container_cluster.primary.location

  initial_node_count = 1
  node_config {
    machine_type = "n2d-standard-2"
    oauth_scopes = [
      "https://www.googleapis.com/auth/cloud-platform"
    ]
    labels = {
      pool = "application"
    }
  }
}

autoscaling {
  min_node_count = 1
  max_node_count = 3
}
}

```

Buna göre, normalde network tanımlarının yani VPC ve subnet'in de olduğu, sonrasında ilk olarak cluster, sonrasında nodepool'ları içeren dosyayı oluşturdum. Region olarak istenen "europe-west-1" kullanıldı. node_locations olarak bir ibare koymak gerekti, çünkü ilk denemede default olan d zone'unda istenen n2d tipinde kaynak kalmadığı için, cluster oluşmadı ve hata verdi. Bunun haricinde 2 tane nodepool tanımı yapıldı, main olan için autoscaling tanımı yokken, application için olanda node autoscaling'i istenilen şekilde 1-3 arası değer

atandı. Buna göre Terraform tanımları hazırlanmış oldu.

! case-cluster

Google Compute Engine: Not all instances running in IGM after 35m19.65604411s. Expected 1, running 0, transitioning 1. Current errors: [GCE_STOCKOUT]: Instance 'gke-case-cluster-default-pool-889963de-blcz' creation failed: The zone 'projects/gke-terraform-460420/zones/europe-west1-d' does not have enough resources available to fulfill the request. Try a different zone, or try again later.

Bu hatadan sonra yukarıdaki node_location tanımı yapıp işlemler yeniden başlatıldı ve aşağıdaki gibi ilerleyip başarılı şekilde tamamlandı:

```
google_container_node_pool.application_pool: Still creating... [19s elapsed]
google_container_node_pool.application_pool: Still creating... [29s elapsed]
google_container_node_pool.main_pool: Still creating... [29s elapsed]
google_container_node_pool.main_pool: Still creating... [39s elapsed]
google_container_node_pool.application_pool: Still creating... [39s elapsed]
google_container_node_pool.application_pool: Still creating... [48s elapsed]
google_container_node_pool.main_pool: Still creating... [48s elapsed]
google_container_node_pool.main_pool: Still creating... [58s elapsed]
google_container_node_pool.application_pool: Still creating... [58s elapsed]
google_container_node_pool.application_pool: Creation complete after 1m1s [id=projects/gke-terraform-460420/locations/europe-west1/clusters/case-cluster/nodePools/application-pool]
google_container_node_pool.main_pool: Creation complete after 1m1s [id=projects/gke-terraform-460420/locations/europe-west1/clusters/case-cluster/nodePools/main-pool]
Apply complete! Resources: 3 added, 0 changed, 0 destroyed.
```

← Clusters Edit Delete + Add node pool + Deploy Connect Duplicate

✓ case-cluster

Details Nodes Storage Observability Logs App Errors (0)

Node Pools

Name ↑	Status	Version	Number of nodes	Machine type	Image type	Autoscaling	Default IP
application-pool	✓ Ok	1.32.3-gke.1785003	1	n2d-standard-2	Container-Optimized OS with containerd (cos_containerd)	1 - 3 nodes per zone	10.20.0.0
main-pool	✓ Ok	1.32.3-gke.1785003	1	n2d-standard-2	Container-Optimized OS with containerd (cos_containerd)	Off	10.20.0.0

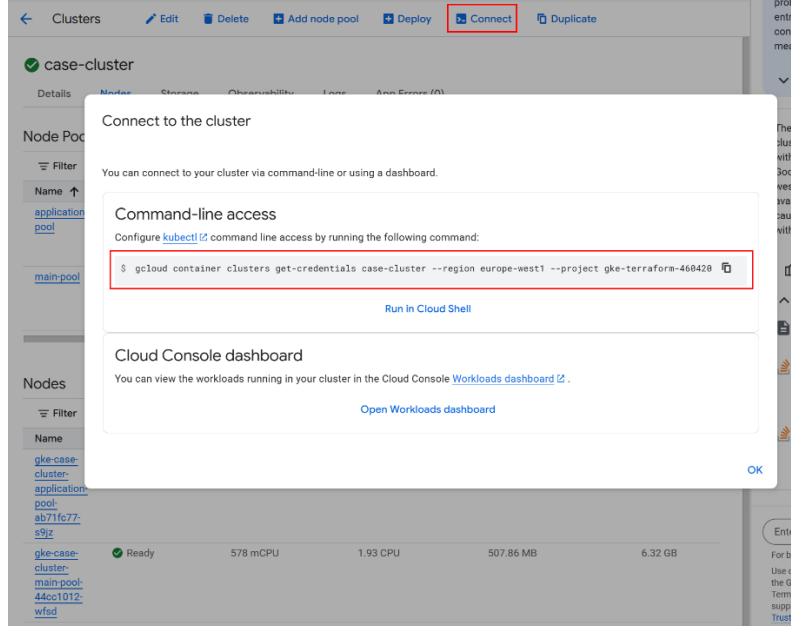
Nodes

Name	Status	CPU requested	CPU allocatable	Memory requested	Memory allocatable
gke-case-cluster-application-pool-ab71fc77-s9jz	✓ Ready	411 mCPU	1.93 CPU	308.42 MB	6.32 GB
gke-case-cluster-main-pool-44cc1012-wfsd	✓ Ready	578 mCPU	1.93 CPU	507.86 MB	6.32 GB

Bu şekilde ilk adım tamamlanmış oldu.

2- YAML ile Örnek Bir Uygulama Deploy Edilmesi ve Pod Autoscaling Aktifleştirilmesi

Ortamları ayarladıktan sonra, örnek bir uygulamayı case’de istendiği şekilde YAML ile deploy edip, uygulamaya erişim için ekstradan bir service deploy edip, sonrasında da bu uygulama için bir pod autoscaling yani HPA (Horizontal Pod Autoscaling) policy’sini aktif hale getirdim. Bunun için, öncelikle GKE’ye local’den erişebilmek için console’den bunu aktifleştirip, kubeconfig dosyasını almak gerekti.



Buradan alabileceğimi keşfettikten sonra, görüldüğü üzere, gcloud-cli tool’unun yüklenmesi de gerekiyordu. Bunu da öncesinde hızlıca local’ime yükleyerek ve sonrasında bu komutu kullanarak, config dosyasını kendi local’imde konumlandırımdım. Tabii ki bunun öncesinde, kendi local’imde hali hazırda kubectl’in yüklü olduğunu ve başka operasyonlar için kullandığımı belirtiyim. Dolayısıyla sıfırdan bir işlem yapılacaksa tabii ki kubectl’in de yüklenmesi gerekmekte.

```
erenf@LAPTOP-N2DH6VR7:~/g-dep$ gcloud container clusters get-credentials case-cluster --region europe-west1 --project gke-terraform-460420
Fetching cluster endpoint and auth data.
kubeconfig entry generated for case-cluster.
erenf@LAPTOP-N2DH6VR7:~/g-dep$ kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
gke-case-cluster-application-pool-ab71fc77-s9jz	Ready	<none>	44m	v1.32.3-gke.1785003
gke-case-cluster-main-pool-44cc1012-wfsd	Ready	<none>	44m	v1.32.3-gke.1785003

Cluster erişimi sağladıktan sonra, YAML dosyalarını oluşturup uygulamaya geçtim. Bunun için ilgili dosya içerikleri aşağıdaki gibidir:

```
# deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: php-apache
spec:
  replicas: 1
```



```
selector:
  matchLabels:
    run: php-apache
template:
  metadata:
    labels:
      run: php-apache
  spec:
    nodeSelector:
      pool: application    # Sadece application-pool'da çalışması için bu eklendi.
    containers:
      - name: php-apache
        image: k8s.gcr.io/hpa-example
        resources:
          requests:
            cpu: 200m
          limits:
            cpu: 500m
        ports:
          - containerPort: 80

# service.yaml
apiVersion: v1
kind: Service
metadata:
  name: php-apache
spec:
  selector:
    run: php-apache
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
  type: ClusterIP

# hpa.yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: php-apache
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
```

```

name: php-apache
minReplicas: 1
maxReplicas: 3
metrics:
- type: Resource
  resource:
    name: cpu
    target:
      type: Utilization

```

Bu dosyaları oluşturduktan sonra uygulayıp, console'dan da kontrolünü sağladım.

```

erenf@LAPTOP-N2DH6VR7:~/g-dep$ kubectl apply -f deployment.yaml
deployment.apps/php-apache created
erenf@LAPTOP-N2DH6VR7:~/g-dep$ kubectl apply -f service.yaml
service/php-apache created
erenf@LAPTOP-N2DH6VR7:~/g-dep$ kubectl apply -f hpa.yaml
horizontalpodautoscaler.autoscaling/php-apache created

```

php-apache

Overview Details Observability Revision history Events Logs App Errors (0) YAML

Select the Cloud Monitoring account to see charts.

Cluster	case-cluster
Namespace	default
Labels	No labels set
Replicas	1 updated, 1 ready, 1 available, 0 unavailable
Pod specification	Revision 1, containers: php-apache
Horizontal Pod Autoscaler	OK
Vertical Pod Autoscaler	Not configured

Active revisions

Revision	Name	Status	Summary	Created on	Pods running/Pods total
1	php-apache-69c758b6db	OK	php-apache: k8s.gcr.io/hpa-example	May 22, 2025, 3:10:49 AM	1/1

Managed pods

Revision	Name	Status	Restarts	Created on
1	php-apache-69c758b6db-t9jd6	Running	0	May 22, 2025, 3:10:50 AM

Exposing services

Name	Type	Endpoints
php-apache	Cluster IP	34.118.232.76

php-apache

Overview Details Observability Revision history Events Logs App Errors (0) YAML

Select the Cloud Monitoring account to see charts.

Cluster	case-cluster
Namespace	default
Labels	No labels set
Replicas	1 updated, 1 ready
Pod specification	Revision 1, containers: php-apache
Horizontal Pod Autoscaler	OK
Vertical Pod Autoscaler	Not configured

Active revisions

Revision	Name	Status	Restarts	Created on
1	php-apache-69c758b6db-t9jd6	Running	0	May 22, 2025, 3:10:50 AM

Managed pods

Revision	Name	Status	Restarts	Created on
1	php-apache-69c758b6db-t9jd6	Running	0	May 22, 2025, 3:10:50 AM

Exposing services

Name	Type	Endpoints
php-apache	Cluster IP	34.118.232.76

Configure Horizontal Pod Autoscaler

Horizontal Pod autoscaling increases and decreases the number of replicated pods to maintain performance and minimize cost. [Horizontal Pod Autoscaling](#)

Minimum number of replicas: 1 Maximum number of replicas: 3

Autoscaling metrics

Use metrics to determine when to autoscale the deployment

✓ CPU (25%)

Add a metric

* Indicates required field

Cancel Delete Save

Bunlardan sonra, yapılan işlemi test etmek için bir load-generator pod'u çalıştırıp, deploy edilen

uygulamaya sürekli istek yollayarak istek oluşturmasını sağladım. Local’imde “kubectl run -i --tty load-generator --rm --image=busybox -- /bin/sh” komutunu çalıştırarak pod’un oluşmasını sağlayıp içerisine girip, “while true; do wget -q -O- http://php-apache; done” komutu ile de sürekli istek yollama işlemini gerçekleştirdim.

```
erenf@LAPTOP-N2DH6VR7:~$ kubectl run -i --tty load-generator --rm --image=busybox /bin/sh
If you don't see a command prompt, try pressing enter.
/ # while true; do wget -q -O- http://php-apache; done
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
```

Sonrasında oluşan değişikliği gözlemlediğimde, HPA'nın başarılı şekilde çalıştığını gözlemledim.

```

erenf@LAPTOP-N2DH6VR7:~$ kubectl get hpa
NAME                REFERENCE                TARGETS          MINPODS   MAXPODS   REPLICAS   AGE
php-apache          Deployment/php-apache     cpu: 1%/25%     1         3         1          21h
erenf@LAPTOP-N2DH6VR7:~$ kubectl get hpa
NAME                REFERENCE                TARGETS          MINPODS   MAXPODS   REPLICAS   AGE
php-apache          Deployment/php-apache     cpu: 219%/25%   1         3         1          21h
erenf@LAPTOP-N2DH6VR7:~$ kubectl get hpa
NAME                REFERENCE                TARGETS          MINPODS   MAXPODS   REPLICAS   AGE
php-apache          Deployment/php-apache     cpu: 219%/25%   1         3         1          21h
erenf@LAPTOP-N2DH6VR7:~$ kubectl get hpa
NAME                REFERENCE                TARGETS          MINPODS   MAXPODS   REPLICAS   AGE
php-apache          Deployment/php-apache     cpu: 219%/25%   1         3         1          21h
erenf@LAPTOP-N2DH6VR7:~$ kubectl get pods
NAME                READY   STATUS    RESTARTS   AGE
load-generator      1/1     Running   0          66s
php-apache-69c758bfd-92rf2  1/1     Running   0          15s
php-apache-69c758bfd-bcnf8  1/1     Running   0          15s
php-apache-69c758bfd-t9jd6  1/1     Running   0          21h
erenf@LAPTOP-N2DH6VR7:~$ |

```

İşlem sonlandıktan sonra cpu kullanımı yavaş yavaş düşerek, uygulama eski haline geri döndü.

```

erent@LAPTOP-N2DH6VR7:~$ kubectl get hpa
NAME                                REFERENCE                               TARGETS          MINPODS   MAXPODS   REPLICAS   AGE
php-apache                         Deployment/php-apache                   cpu: 165%/25%    1          3          3           21h
erent@LAPTOP-N2DH6VR7:~$ kubectl describe hpa
Name:                               php-apache
Namespace:                          default
Labels:                             <none>
Annotations:                         <none>
CreationTimestamp:                  Thu, 22 May 2025 03:11:06 +0300
Reference:                          Deployment/php-apache
Metrics:                             ( current / target )
  resource cpu on pods  (as a percentage of request): 1% (1m) / 25%
Min replicas:                      1
Max replicas:                      3
Deployment pods:                    1 current / 1 desired
Conditions:
  Type      Status      Reason
  ----      -
  AbleToScale    True      ReadyForNewScale
  ScalingActive  True      ValidMetricFound
  ScalingLimited False     DesiredWithinRange
erent@LAPTOP-N2DH6VR7:~$ kubectl get events
Type      Reason      Age      From      Message
----      -
Normal    SuccessfulRescale    15s     horizontal-pod-autoscaler    New size: 3; reason: cpu resource utilization (percentage of request) above target
Normal    SuccessfulRescale    47s     horizontal-pod-autoscaler    New size: 2; reason: cpu resource utilization (percentage of request) below target
Normal    SuccessfulRescale    17s     horizontal-pod-autoscaler    New size: 1; reason: cpu resource utilization (percentage of request) below target
erent@LAPTOP-N2DH6VR7:~$ kubectl get hpa
NAME                                REFERENCE                               TARGETS          MINPODS   MAXPODS   REPLICAS   AGE
php-apache                         Deployment/php-apache                   cpu: 1%/25%      1          3          1           22h

```

Bu şekilde bu adım da tamamlanmış oldu.

3- Cluster Üzerine Prometheus-Grafana Kurulumu

Sonraki aşamada ise, cluster üzerine kullanılan en yaygın monitoring tool'larından biri olan Prometheus-Grafana ikilisini kurdum. Bunun için, raporun en başında bahsettiğim Helm tool'u ile bunu gerçekleştirdim. Helm kısaca bir paket yönetimi olup, bu tarz tool'ların kurulumunu hızlıca ve efektif bir şekilde kurulmasına olanak sağlıyor. İstersek, yüklemek istediğimiz paketleri indirip, içerisindeki values.yaml dosyasını güncelleyerek özelleştirebiliriz. Bu case

özelinde default olan yeterli olduğu için bu operasyona ihtiyaç duymadığımı belirteyim.

Buna göre, operasyona ilgili Helm komutlarını kullanarak yükleme işlemini gerçekleştirdim. Öncelikle Prometheus-Grafana reposunu “helm repo add prometheus-community <https://prometheus-community.github.io/helm-charts>” ile ekleyip, sonrasında “helm repo update” komutu ile erişilen versiyonların metadata’sını güncelleyip, yüklenecek kaynakların izolasyonu için “kubectl create namespace monitoring” komutu ile ayrı namespace oluşturup, “helm install monitoring prometheus-community/kube-prometheus-stack -n monitoring” komutu ile de son olarak monitoring tool ikilimizin cluster’a yüklenmesini sağlamış oldum.

```

erenf@LAPTOP-N2DH6VR7:~$ helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
"prometheus-community" has been added to your repositories
erenf@LAPTOP-N2DH6VR7:~$ helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "keda" chart repository
...Successfully got an update from the "keda-core" chart repository
...Successfully got an update from the "keda-operator" chart repository
...Successfully got an update from the "keda-webhook" chart repository
...Successfully got an update from the "keda-webhook" chart repository
...Successfully got an update from the "keda-webhook" chart repository
...Successfully got an update from the "keda-webhook" chart repository
...Successfully got an update from the "keda-webhook" chart repository
Update Complete. Happy Helming!
erenf@LAPTOP-N2DH6VR7:~$ kubectl create namespace monitoring
namespace/monitoring created
erenf@LAPTOP-N2DH6VR7:~$ helm install monitoring prometheus-community/kube-prometheus-stack -n monitoring
NAME: monitoring
LAST DEPLOYED: Thu May 22 23:19:08 2025
NAMESPACE: monitoring
STATUS: deployed
REVISION: 1
NOTES:
kube-prometheus-stack has been installed. Check its status by running:
  kubectl --namespace monitoring get pods -l "release=monitoring"

Get Grafana 'admin' user password by running:

  kubectl --namespace monitoring get secrets monitoring-grafana -o jsonpath="{.data.admin-password}" | base64 -d ; echo

Access Grafana local instance:

  export POD_NAME=$(kubectl --namespace monitoring get pod -l "app.kubernetes.io/name=grafana,app.kubernetes.io/instance=monitoring" -o name)
  kubectl --namespace monitoring port-forward $POD_NAME 3000

Visit https://github.com/prometheus-operator/kube-prometheus for instructions on how to create & configure Alertmanager and Prometheus instances using the Operator.
erenf@LAPTOP-N2DH6VR7:~$

```

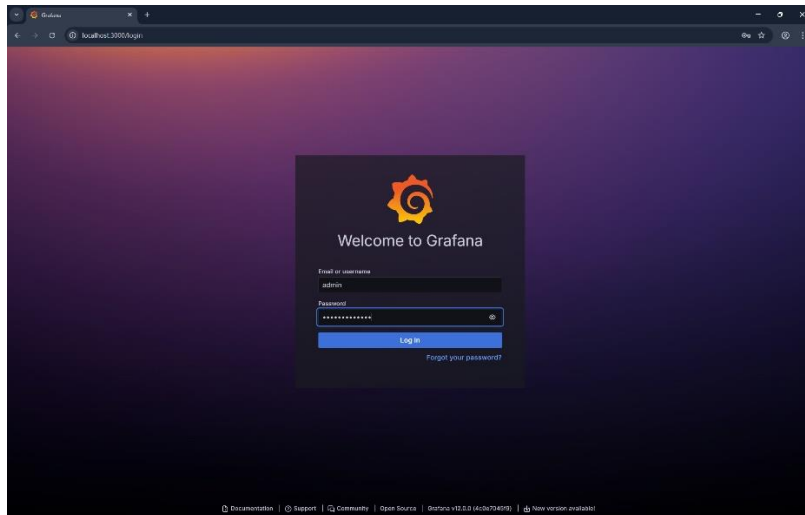
Yükleme gerçekleştikten sonra, yukarıdaki görselde de belirtildiği gibi Grafana’ya girerken sorduğu şifreyi elde etmek için “kubectl --namespace monitoring get secrets monitoring-grafana -o jsonpath="{.data.admin-password}" | base64 -d ; echo” komutunu çalıştırarak elde ettim.

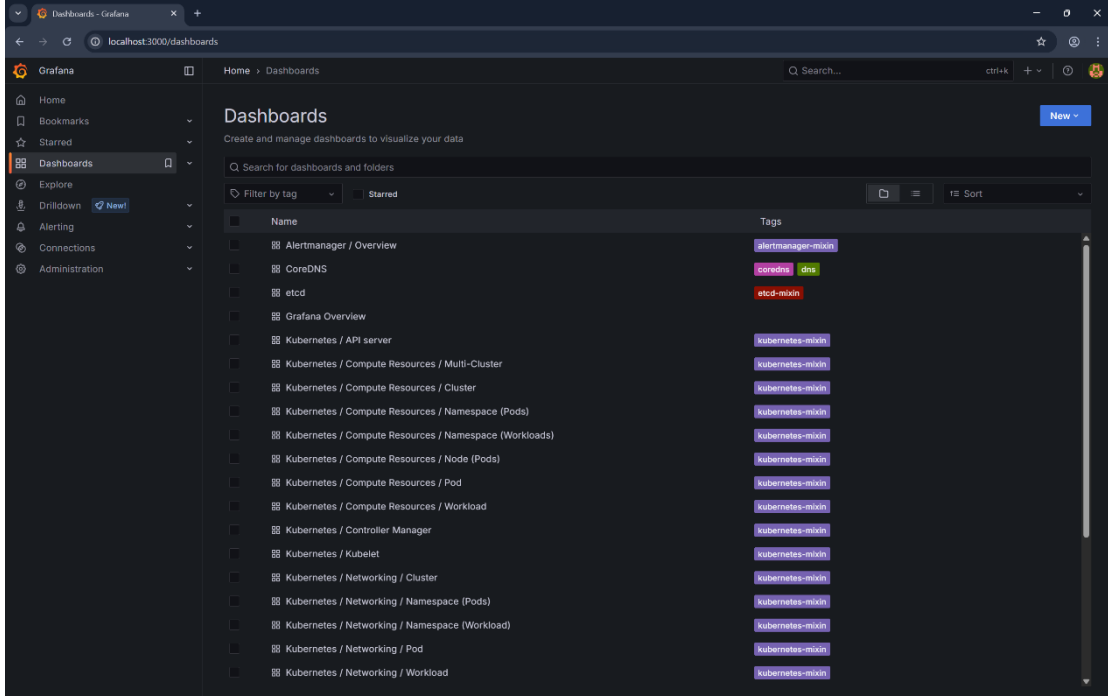
Grafana’ya erişim için de, yine görselde verilen port-forward yöntemi ile localhost:3000 olarak çalıştırmak için “export POD_NAME=\$(kubectl --namespace monitoring get pod -l "app.kubernetes.io/name=grafana,app.kubernetes.io/instance=monitoring" -o name)” ile “kubectl --namespace monitoring port-forward \$POD_NAME 3000” komutları ile bunu gerçekleştirdim.

```

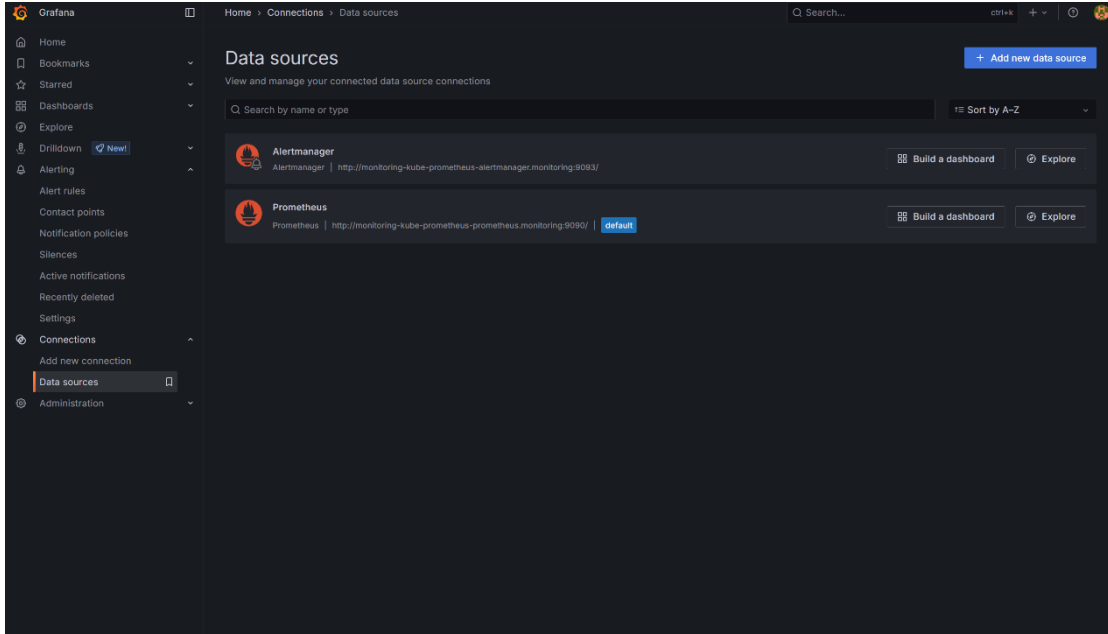
erenf@LAPTOP-N2DH6VR7:~$ export POD_NAME=$(kubectl --namespace monitoring get pod -l "app.kubernetes.io/name=grafana,app.kubernetes.io/instance=monitoring" -o name)
erenf@LAPTOP-N2DH6VR7:~$ kubectl --namespace monitoring port-forward $POD_NAME 3000
Forwarding from 127.0.0.1:3000 -> 3000
Forwarding from [::1]:3000 -> 3000

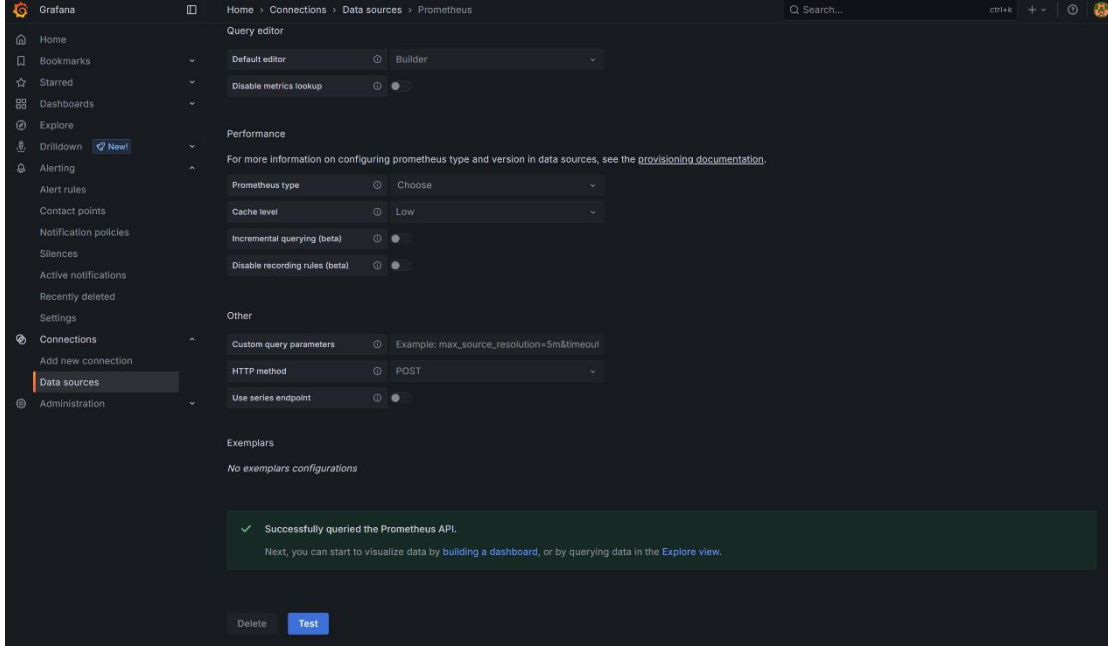
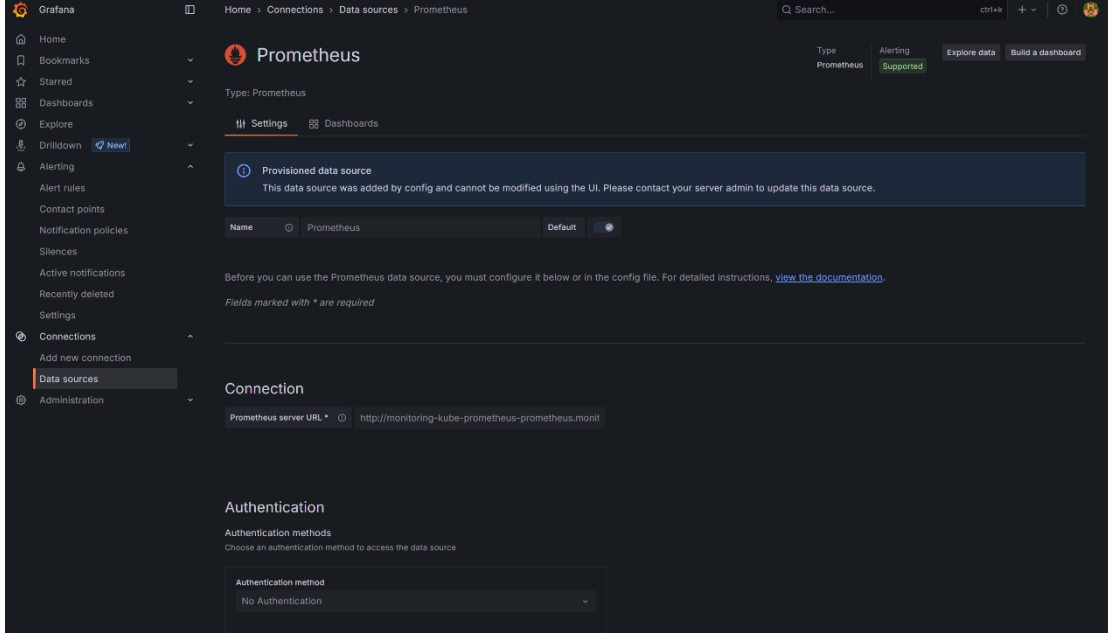
```





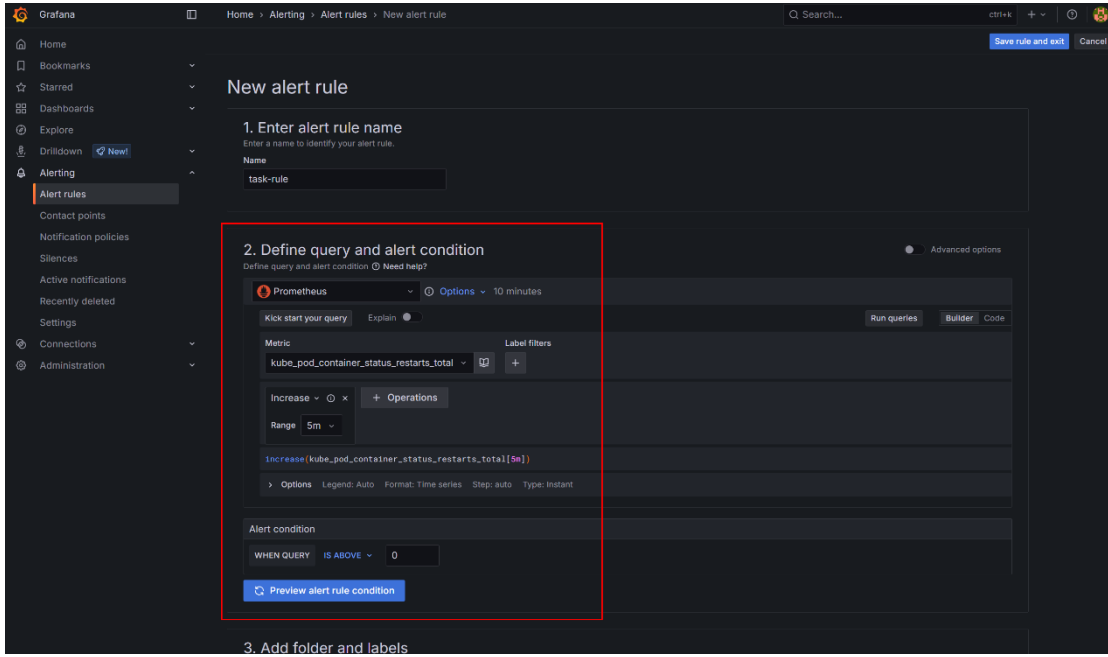
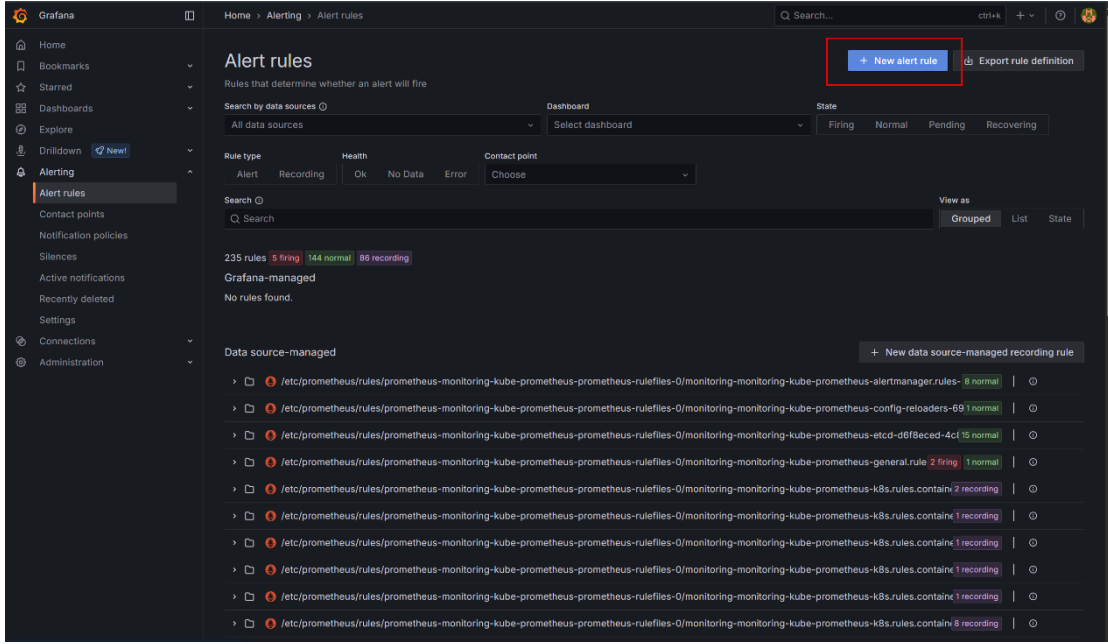
Görüldüğü üzere, Prometheus-Grafana ikilisi başarılı şekilde yüklenmiş olup, kendiliğinden konfigürasyonları da yapılmış olarak geldiğinden, halihazırda çalışan cluster'ımıza entegrasyonu direkt sağlanmış oldu. Eğer ki entegre olmamış şekilde gelmiş olsaydı, yapacağımız işlem, data source olarak Prometheus'u eklemek ve Prometheus'un cluster'da çalışan service name'ini alarak Connection kısmına konumlandırmak olurdu. Örneği aşağıdaki gibidir:





4- Grafana Üzerinde Alarm Oluşturulması

Case’de istenen detaylara göre, Grafana üzerinde pod restart ile alakalı bir rule oluşturulması isteniyordu. Bununla alakalı Grafana’da Alerting sekmesine gidip, New alert rule’a tıklayarak oluşturdum.



Alarm oluşturmadan önce, bunların konumlandırılacağı bir dosya ve eval ismi istedi, onları hızlıca oluşturdum. Daha sonrasında Alarm rule oluşturma kısmına geçtiğimde, öncelikle bir isim verdim. Sonrasında asıl önemli olan aşağıdaki kısımları doldurmuş oldum. Buna göre, 5 dakikalık zaman aralığında, “kube_pod_container_status_restarts_total” değeri, yani podlarla alakalı bir restart işleminin total sayısı 0’ı geçerse, bize bunu bildirecek şekilde ayarlamış oldum. Böylece Alarm rule tanımı gerçekleştirilmiş oldu. Testi için de pod’ları elle restart edip doğruluğunu test etmek mümkün.

5- Cluster’da Keda’nın Aktifleştirilmesi ve Autoscaling’in Buradan Gerçekleştirilmesi

Bu adımda ise, çok farklı senaryoları destekleyerek bir workload üzerinde HPA işlemini

sağlayan Keda'nın yüklenip aktifleştirilmesi ile birlikte, önceki adımlardaki işlemin buradan tekrarlanması isteniyor. HPA için en kapsamlı tool olan Keda ile, birçok farklı tool'u entegre ederek bu işlemi yapabiliyoruz. Bunun için öncelikle cluster'a yüklenmesini Helm ile sağladım. Operasyon ise aşağıdaki gibi gerçekleşti:

```
erenf@LAPTOP-N2DH6VR7:~$ helm repo add kedacore https://kedacore.github.io/charts
"kedacore" already exists with the same configuration, skipping
erenf@LAPTOP-N2DH6VR7:~$ helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "kedacore" chart repository
...Successfully got an update from the "commvault" chart repository
...Successfully got an update from the "secrets-store-csi-driver" chart repository
...Successfully got an update from the "elastic" chart repository
...Successfully got an update from the "external-secrets" chart repository
...Successfully got an update from the "grafana" chart repository
...Successfully got an update from the "prometheus-community" chart repository
Update Complete. 🎉Happy Helming!🎉
```

```
erenf@LAPTOP-N2DH6VR7:~$ helm install keda kedacore/keda --namespace keda --create-namespace
NAME: keda
LAST DEPLOYED: Fri May 23 01:15:16 2025
NAMESPACE: keda
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
#+#          .+#+          .+#+          .^
7??~ ^?7???~ 7????????????? 7?????????771^    .7?7
7??~ ^?7??~ ~!!!!!!            :???!!!!7??~      .7???.
7??~ ^?7??~ :??? :~?7???:   :7???:       :7???.
7??~?7???: :~?7???:   :7???:   :7???:   :7???:
7??~?7???: 7???:   :7???:   :7???:   :7???:
7???:~?7??^ 17777777777^ :??? :??? :^??7?#P7???.
7??~ ^?7??~ :??? :7???: ^??7J#0J7???.
7??~ :???: :??? :~?7?1 ~?77V&0H77777777.
7??~ :?7???: :??? 17777777^ ~?77500CJ7???.
7??~ :?7??^ :??? 7???: 17777777^ 1777G00000057???.
+#+          +#+          +#+          +#+          +#+
              :00#~
              P0B^
              :&G
              15.
              .Kubernetes Event-driven Autoscaling (KEDA) - Application autoscaling made simple.
```

```
Get started by deploying Scaled Objects to your cluster:
- Information about Scaled Objects : https://keda.sh/docs/latest/concepts/
- Samples: https://github.com/kedacore/samples
```

```
Get information about the deployed ScaledObjects:
kubectl get scaledobject [--namespace <namespace>]
```

```
Get details about a deployed ScaledObject:
kubectl describe scaledobject <scaled-object-name> [--namespace <namespace>]
```

```
Get information about the deployed ScaledObjects:
kubectl get triggerauthentication [--namespace <namespace>]
```

```
Get details about a deployed ScaledObject:
kubectl describe triggerauthentication <trigger-authentication-name> [--namespace <namespace>]
```

```
Get an overview of the Horizontal Pod Autoscalers (HPA) that KEDA is using behind the scenes:
kubectl get hpa [--all-namespaces] [--namespace <namespace>]
```

```
Learn more about KEDA:
- Documentation: https://keda.sh/
- Support: https://keda.sh/support/
- File an issue: https://github.com/kedacore/keda/issues/new/choose
```

```
erenf@LAPTOP-N2DH6VR7:~$ kubectl get pods -n keda
```

NAME	READY	STATUS	RESTARTS	AGE
keda-admission-webhooks-8bf696987-4959r	1/1	Running	0	48s
keda-operator-7cb85c7c47-svnbm	1/1	Running	1 (41s ago)	48s
keda-operator-metrics-apiserwer-7f569bd9c5-5ff6q	1/1	Running	0	48s

Daha öncesinde kendim de deneyimlediğim için, “helm repo add kedacore <https://kedacore.github.io/charts>” komutunu kullandıktan sonra, Helm deposu çoktan eklenmiş olarak gözüküyor. “helm repo update”, “helm install keda kedacore/keda –namespace keda –create-namespace” komutları ile birlikte yüklemeyi sağlamış oldum. Bununla birlikte, yükleme adımları Prometheus-Grafana ile benzer olduğu için işlemi kolaylıkla tamamlamış ve en son da yüklendiğine dair kontrolleri “kubectl get pods -n keda” komutu ile yapmış olduk.

Bu aşamadan sonra ise, Keda'nın kendi tanımı olan ScaledObject için ilgili YAML'ı oluşturup, önceki HPA'yı temizleyip, daha sonrasında oluşturulmuş olan ScaledObject'i uyguladım.

ScaledObject tanımı aşağıdaki gibi olup, takibinde ise uygulamış oldum:

```
# scalingobject.yaml
```

apiVersion: keda.sh/v1alpha1


```
erenf@LAPTOP-N2DH6VR7:~/g-dep$ vim scalingobject.yaml
erenf@LAPTOP-N2DH6VR7:~/g-dep$ ls
deployment.yaml hpa.yaml scalingobject.yaml service.yaml
erenf@LAPTOP-N2DH6VR7:~/g-dep$ kubectl apply -f scalingobject.yaml
scaledobject.keda.sh/php-apache-scaledobject created
erenf@LAPTOP-N2DH6VR7:~/g-dep$ kubectl get hpa
```

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
keda-hpa-php-apache-scaledobject	Deployment/php-apache	cpu: 1%/25%	1	3	1	27s

[illegible]

```

erenf@LAPTOP-N2DH6VR7:~/g-dep$ kubectl get pods
NAME                                READY    STATUS    RESTARTS   AGE
load-generator                      1/1      Running   0           28s
php-apache-69c758bfdb-t9jd6        1/1      Running   0           22h
erenf@LAPTOP-N2DH6VR7:~/g-dep$ kubectl get hpa
NAME                                REFERENCE                TARGETS      MINPODS   MAXPODS   REPLICAS   AGE
keda-hpa-php-apache-scaledobject    Deployment/php-apache     cpu: 1%/25%  1          3          1           2m2s

```

```

erenf@LAPTOP-N2DH6VR7:~/g-dep$ kubectl get hpa
NAME                                REFERENCE                                TARGETS                                MINPODS    MAXPODS    REPLICAS    AGE
keda-hpa-php-apache-scaledobject    Deployment/php-apache                    cpu: 144%/25%                          1           3           1           2m27s
erenf@LAPTOP-N2DH6VR7:~/g-dep$ kubectl get pods
NAME                                READY    STATUS    RESTARTS    AGE
load-generator                      1/1     Running   0            64s
php-apache-69c758bfdb-2nxpr         1/1     Running   0            14s
php-apache-69c758bfdb-5nc45         1/1     Running   0            14s
php-apache-69c758bfdb-t9jd6         1/1     Running   0            22h

```

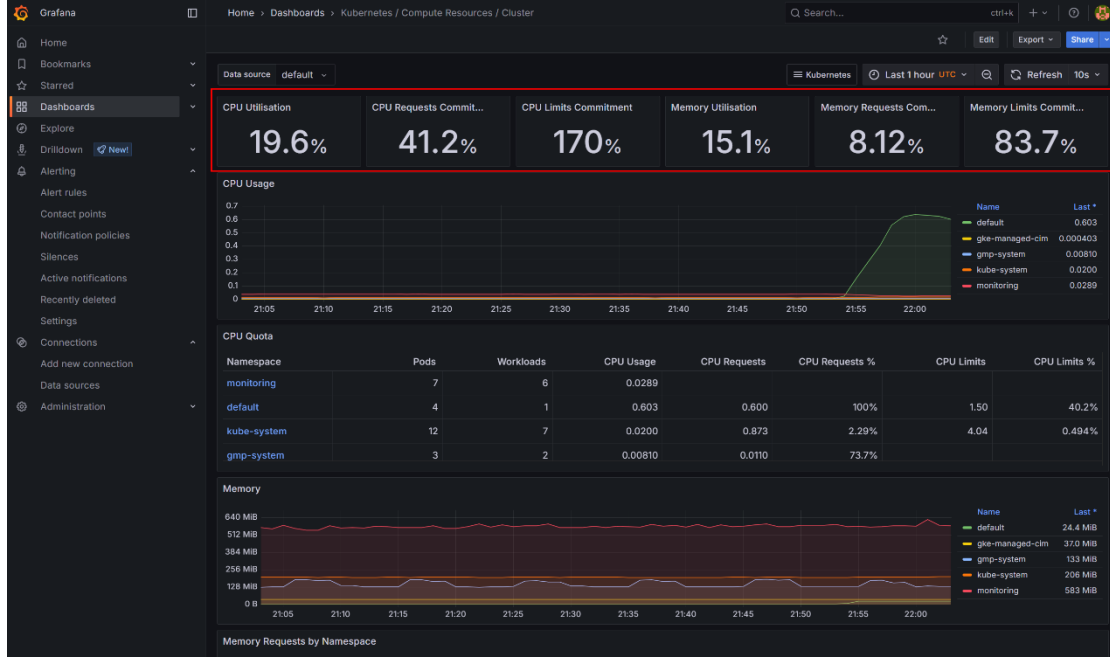
```
k!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!^C
/ #
Session ended, resume using 'kubectl attach load-generator -c load-generator -i -t' command when the pod is running
pod "load-generator" deleted
```

```

erenf@LAPTOP-N2DH6VR7:~/g-dep$ kubectl get hpa
NAME                                REFERENCE                                TARGETS      MINPODS    MAXPODS    REPLICAS    AGE
keda-hpa-php-apache-scaledobject    Deployment/php-apache                    cpu: 1%/25%    1           3           3           9m53s
erenf@LAPTOP-N2DH6VR7:~/g-dep$ kubectl get hpa
NAME                                REFERENCE                                TARGETS      MINPODS    MAXPODS    REPLICAS    AGE
keda-hpa-php-apache-scaledobject    Deployment/php-apache                    cpu: 1%/25%    1           3           1           10m
erenf@LAPTOP-N2DH6VR7:~/g-dep$ kubectl get pods
NAME                                READY    STATUS    RESTARTS    AGE
php-apache-69c758bfdb-t9id6         1/1      Running    0           22h

```

Yukarıdan da anlaşılabileceği üzere, yük testini başlattığımda önceki HPA denemesi ile aynı tepkiyi gösterdi ve denemeyi sonlandırdığımda ise uygulama eski haline geri dönmüş oldu. Burada direkt aklıma bir detay geldi ve bunu da sorguladım. Biz node auto-scaling aktif ettiğimiz halde, neden node sayısında bir artış yaşanmadı? Aslında bunun cevabını aşağıdaki görselden anlayabiliriz:



Buna göre, node genelinde kullanım yani utilization çok düşük kalıyor. Her ne kadar pod'daki CPU kullanımı %161'e çıkıyor olsa bile, node seviyesinde yine de düşük kalıyor. Bu yüzden pod'un ve node'un konfigürasyonlarını düzgün ayarlayıp, autoscaling değerlerini de yine düzgünce vermek gerekiyor. Bu şekilde Keda ile HPA işlemini de tamamlamış oldum.

6- Cluster Üzerine Istio (istiod, istio-ingress, istio-egress) Kurulumu

Son adımda ise, cluster üzerine service mesh tool'u olan Istio'nun ve bu kapsamda istiod, istio-ingress ve istio-egress kurulması isteniyordu. Ama, bunlarla alakalı herhangi bir kullanım senaryosu belirtilmediği için sadece kurulumları yapıp, doğru kurulduğunu test etmek üzere bir operasyon yapılmış oldu.

Istio, cluster'daki service yönetimi için gelişmiş özellikler sunup, bunları tek bir noktadan yönetmeye imkan vermekte. Bunun için kurulumu ve kullanımı gayet yaygın bir tool'dur. Bu yüzden ben de cluster'a yüklemesini yapmış oldum. Bunu sağlamak için yine Helm'den faydalandım. Normalde birden farklı yükleme yöntemi bulunuyor. Bunlardan en yaygın olanları, istioctl ve Helm'dir. istioctl, Istio'nun kendi yönetim CLI'ı olup tercih edilmesi durumunda yönetimi daha da çeşitlendirerek burada hakimiyeti artırır. Ama apayrı bir tool olduğu ve yükleme adımları da Helm kullanılarak yapılan önceki yüklemelerden farklı olduğu için, standart olmaya pek yakın sayılmaz. Bu sebeple Helm ile kurmak, hem case'deki adımlar arasında bütünlük sağlayacak, hem de canlı bir ortama geçirildiğinde ise aktarılması daha kolay olacak. Bu yüzden Helm ile kurulumu yaparak, sırası ile "helm repo add istio https://istio-

release.storage.googleapis.com/charts”, “helm repo update”, “kubectl create namespace istio-system” ve “helm install istiod istio/istiod -n istio-system” komutları ile yüklemesini gerçekleştirmiş oldum. Buradaki istiod, Istio servis mesh’in control plane ana bileşenidir ve Istio ekosisteminde merkezi yönetim, konfigürasyon ve mesh içi iletişimin “beyni” gibi çalışır. Bu yüzden Istio yüklemek demek aslında bunu yüklemek demektir.

```
erenf@LAPTOP-N2DH6VR7:~/g-dep$ helm repo add istio https://istio-release.storage.googleapis.com/charts
"istio" has been added to your repositories
erenf@LAPTOP-N2DH6VR7:~/g-dep$ helm update
Error: unknown command "update" for "helm"
Run 'helm --help' for usage.
erenf@LAPTOP-N2DH6VR7:~/g-dep$ helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "istio" chart repository
...Successfully got an update from the "kedacore" chart repository
...Successfully got an update from the "commvault" chart repository
...Successfully got an update from the "secrets-store-csi-driver" chart repository
...Successfully got an update from the "elastic" chart repository
...Successfully got an update from the "prometheus-community" chart repository
...Successfully got an update from the "external-secrets" chart repository
...Successfully got an update from the "grafana" chart repository
Update Complete. ✨Happy Helming!✨
```

```
erenf@LAPTOP-N2DH6VR7:~/g-dep$ kubectl create namespace istio-system
namespace/istio-system created
erenf@LAPTOP-N2DH6VR7:~/g-dep$ helm install istiod istio/istiod -n istio-system
NAME: istiod
LAST DEPLOYED: Fri May 23 01:54:55 2025
NAMESPACE: istio-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
"istiod" successfully installed!

To learn more about the release, try:
  $ helm status istiod -n istio-system
  $ helm get all istiod -n istio-system

Next steps:
* Deploy a Gateway: https://istio.io/latest/docs/setup/additional-setup/gateway/
* Try out our tasks to get started on common configurations:
  * https://istio.io/latest/docs/tasks/traffic-management
  * https://istio.io/latest/docs/tasks/security/
  * https://istio.io/latest/docs/tasks/policy-enforcement/
* Review the list of actively supported releases, CVE publications and our hardening guide:
  * https://istio.io/latest/docs/releases/supported-releases/
  * https://istio.io/latest/news/security/
  * https://istio.io/latest/docs/ops/best-practices/security/

For further documentation see https://istio.io website
```

Bu işlemlerden sonra, ingress ve egress’i de aktifleştirip, case’deki istenen detayı gerçekleştirmiş olacağım. Bunun için yine Helm’den yararlanıp, “helm install istio-ingress istio/gateway -n istio-system” ve “helm install istio-egress istio/gateway -n istio-system” komutları ile ilgili kaynakları aktif etmiş oldum. Normalde, bunların tanımlarının düzgünce yapılıp, uygulamaların dışarıdan gelen trafiğe açılmasını ingress üstünden, dışarıya giden kontrolü de egress üzerinden kısaca sağlayabiliriz. Tanımlar tarafında ise, Istio’nun Gateway ve VirtualService gibi bazı custom resource’larını cluster içinde tanımlamak gerek. Sonrasında aynı standart servis veya ingress deploy’u gibi özel bir dosya içeriği oluşturarak aktive edebiliriz. Bu case kapsamında bu istenmediği için, en sonda bu yüklenen resource’ların sorgusunu yaparak case study’i tamamlamış oluyorum.

```
erenf@LAPTOP-N2DH6VR7:~/g-dep$ kubectl get pods -n istio-system
NAME                                READY   STATUS    RESTARTS   AGE
istio-egressgateway-689bd65db7-zxxjf 1/1     Running   0           30s
istio-ingressgateway-5cf8f4cc4f-lr57b 1/1     Running   0           67s
istiod-7d8db9dd-q9zql                 1/1     Running   0          2m48s
```

```
erenf@LAPTOP-N2DH6VR7:~/g-dep$ kubectl get svc -n istio-system
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
istio-egressgateway	LoadBalancer			15021:32306/TCP,80:31817/TCP,443:32087/TCP	51s
istio-ingressgateway	LoadBalancer			15021:30868/TCP,80:31564/TCP,443:32195/TCP	87s
istioid	ClusterIP		<none>	15010/TCP,15012/TCP,443/TCP,15014/TCP	3m8s

Sonuç olarak, istenen tüm adımları sorunsuz şekilde gerçekleştirip, bu çalışmayı tamamladım. Kullandığım tüm dosya içerikleri ve komutlarını da bu rapor içinde konumlandırımdım. Özet olarak ise, çok basit bir şekilde bir Kubernetes ortamı aktif ederek ve buna ilgili eklemelerle temel düzeyde DevOps süreci gerçekleştirmiş olduk. Buradaki çıkarımlar ise, GCP/GKE standart bir Kubernetes ortamı sağlıyor ve kullandığım open-source tool'larla entegrasyonu rahat. Öte yandan, arayüz olarak çok da gelişmiş özellikleri bulunmadığından, CLI üzerinden ilerlemek çok da rahat bir kullanım sağlıyor. Diğer tool'larla alakalı da, yine temel düzeyde entegrasyon yapıldığı için bir aksilik olmadan adımları gerçekleştirebildim. Daha karışık ve zorlu senaryolarda karşılaçağım zorlukların daha fazla olması ve bunlara daha derin çözümler getirmeyi beklediğimi söyleyebilirim.

Kullanılan dosyalarının bulunduğu Github reposu linki:

<https://github.com/SpeedMentor/WEG-GC-TF.git>