kubectl Nedir?

Kubernetes'in kumandası, debug aracı ve günlük hayatın vazgeçilmezi

kubectl Nedir?

Kubernetes'in kumandası

Cluster'ınızı kontrol etmenin ana aracı

Debug aracı

Sorunları tespit etmek ve çözmek için

Günlük hayatın vazgeçilmezi

Her gün kullanacağınız temel araç

kubectl Derinlemesine İnceleme

kubectl'in Temel Rolü

kubectl, Kubernetes API sunucusu ile iletişim kurarak cluster'daki kaynakları yönetmenizi sağlayan bir komut satırı aracıdır. Kullanıcıların girdiği komutları, Kubernetes API'sinin anlayacağı HTTP isteklerine dönüştürür. Bu istekler, API sunucusu tarafından işlenir ve cluster'ın mevcut durumu güncellenir veya istenilen bilgiler geri döndürülür.

Neden kubectl Önemli?

kubectl, Kubernetes ekosisteminin kalbinde yer alır. Geliştiriciler ve operasyon ekipleri için vazgeçilmezdir çünkü:

- Cluster bileşenlerine doğrudan erişime ihtiyaç duymadan uygulama dağıtma, yönetme ve izleme imkanı sunar.
- Hata ayıklama (debug) süreçlerinde logları görüntüleme, container'lara bağlanma ve sistem kaynaklarını inceleme gibi kritik işlevleri yerine getirir.
- Tüm Kubernetes ortamlarında standart bir arayüz sağlayarak tutarlı bir yönetim deneyimi sunar.

kubectl'in Yetenekleri

kubectl sadece temel kaynak oluşturma, okuma, güncelleme ve silme (CRUD) işlemlerinin ötesine geçer. İşte bazı gelişmiş yetenekleri:

- **Uygulama Yönetimi:** `kubectl apply` ile deklaratif olarak kaynakları yönetebilir, `kubectl rollout` ile dağıtımlarınızı (deployment) güvenli bir şekilde güncelleyebilir veya geri alabilirsiniz.
- **Hata Ayıklama ve Gözlemleme:** `kubectl logs` ile container loglarını takip edebilir, `kubectl exec` ile çalışan container'lar içinde komut çalıştırabilir ve `kubectl describe` ile herhangi bir Kubernetes kaynağının ayrıntılı durumunu görebilirsiniz.
- **Ağ Yönetimi:** `kubectl port-forward` ile cluster içindeki servislerinize yerel makinenizden erişim sağlayabilir, böylece uygulama geliştirme ve test süreçlerini kolaylaştırabilirsiniz.
- **Kaynak İzleme:** `kubectl top nodes` veya `kubectl top pods` gibi komutlarla cluster kaynaklarının (CPU, bellek) anlık kullanımını izleyebilirsiniz.

Kubernetes biliyorum demek, **kubectl'i okuyabiliyorum** demek.

İlk Komutlar

kubectl get nodes

Cluster'daki node'ları listele

kubectl get pods

Çalışan pod'ları görüntüle

kubectl describe pod

Pod detaylarını incele

kubectl logs

Pod loglarını oku

Komutların Anatomisi

`kubectl` komutları genellikle belirli bir yapıya sahiptir: `VERB` (eylem), `RESOURCE` (kaynak tipi) ve isteğe bağlı `NAME` (kaynak adı) veya `FLAG` (parametreler). Bu yapıyı anlamak, Kubernetes kaynaklarını etkili bir şekilde yönetmek için temeldir.

- **VERB (Eylem):** Yapmak istediğiniz işlemi belirtir. Örneğin, bir kaynağı almak (`get`), tanımlamak (`describe`), uygulamak (`apply`) veya silmek (`delete`).
- **RESOURCE (Kaynak Tipi):** Üzerinde işlem yapacağınız Kubernetes kaynağının tipini belirtir. Örneğin, `pods`, `nodes`, `deployments`, `services` veya `configmaps`. Bu kısım tekil veya çoğul olabilir (`pod` veya `pods`).
- **NAME (Kaynak Adı):** İsteğe bağlıdır. Üzerinde işlem yapılacak belirli bir kaynağın adını belirtir. Eğer bir isim belirtmezseniz, komut genellikle o tipteki tüm kaynaklar üzerinde işlem yapar.
- **FLAG (Parametreler):** Komutun davranışını değiştiren ek seçeneklerdir. Örneğin, çıktı formatını belirlemek (`-o json`), bir namespace belirtmek (`-n my-namespace`) veya daha fazla detay görmek (`--show-labels`).

Örnek: `kubectl get pods my-app-pod -n default`

- **VERB:** `get`
- **RESOURCE:** `pods`
- **NAME:** `my-app-pod`
- **FLAG:** `-n default` (default namespace'i belirtir)

Sık Kullanılan Komutlar

İlk öğrendiğiniz temel komutlara ek olarak, Kubernetes ile günlük etkileşimde sıklıkla kullanacağınız başka güçlü `kubectl` komutları da bulunmaktadır:

- `kubectl apply -f [FILE_OR_URL]`**:** Belirtilen YAML/JSON dosyasındaki konfigürasyonu uygulayarak kaynakları oluşturur veya günceller. Declarative (bildirimsel) yönetim için vazgeçilmezdir.
- `kubectl delete [RESOURCE_TYPE] [NAME]`**:** Belirtilen tipteki kaynağı siler. Örneğin, `kubectl delete pod my-pod` veya `kubectl delete -f my-deployment.yaml`.
- `kubectl edit [RESOURCE_TYPE] [NAME]`**:** Belirtilen kaynağın YAML konfigürasyonunu doğrudan düzenlemenizi sağlar. Genellikle hızlı değişiklikler için kullanılır.
- `kubectl scale deployment [DEPLOYMENT_NAME] --replicas=[NUMBER]`**:** Belirtilen deployment'ın pod replika sayısını ayarlar. Uygulamanızın ölçeğini anında değiştirmek için kullanılır.
- `kubectl rollout status deployment/[DEPLOYMENT_NAME]`**:** Bir deployment'ın güncelleme (rollout) durumunu izler.
- `kubectl rollout undo deployment/[DEPLOYMENT_NAME]`**:** Bir deployment'ın önceki versiyonuna geri dönmesini (rollback) sağlar.

Çıktı Formatları

`kubectl` komutlarının çıktısını farklı formatlarda görüntülemek, bilgiyi daha okunabilir hale getirmek veya otomasyon betiklerinde kullanmak için çok önemlidir. Çıktı formatını `-o` veya `--output` bayrağı ile belirtebilirsiniz:

- `-o json`**:** Kaynak detaylarını JSON formatında gösterir. Programatik olarak işlemek için idealdir.
- `-o yaml`**:** Kaynak detaylarını YAML formatında gösterir. Kubernetes objelerini okumak ve anlamak için en yaygın formatlardan biridir.
- `-o wide`**:** Varsayılan tablo çıktısına ek olarak daha fazla bilgi (örneğin Pod'ların IP adresleri, Node isimleri) gösterir.
- `-o custom-columns=[HEADERS:]FIELD_PATH[,HEADERS:]FIELD_PATH,...`**:** Kendi özel sütunlarınızı belirleyerek çıktıyı istediğiniz gibi özelleştirmenizi sağlar.
- `-o jsonpath=[JSONPATH_EXPRESSION]`**:** JSON çıktısı üzerinden belirli alanları seçmek için gelişmiş bir sorgulama dili kullanır.

"Bu pod neden çalışmıyor?" sorusunun cevabı buradadır.

YAML ile Çalışmak

YAML = Sözleşme

Ben ne istiyorum

Desired state'i tanımlarsınız. Bu, Kubernetes'ten beklediğiniz son durumu açıkça belirtmeniz anlamına gelir.

Kubernetes ne yapacak

Sistemi istenen duruma getirir. Kubernetes, sizin tanımladığınız "desired state" ile sistemin "actual state" arasındaki farkı kapatmaya çalışır.

Status kısmını **biz yazmayız**, Kubernetes yazar. Bu kısım, objenin mevcut gerçek durumunu gösterir ve Kubernetes tarafından sürekli güncellenir.

Temel YAML Yapısı

Kubernetes objelerini tanımlarken kullanılan YAML dosyaları belirli temel alanlara sahiptir. Bu alanlar, Kubernetes'in objeyi tanıması ve nasıl yönetmesi gerektiğini anlaması için kritik öneme sahiptir:

apiVersion

Bu alan, objenin kullandığı Kubernetes API sürümünü belirtir. Örneğin, `apps/v1` Deployment'lar için kullanılırken, `v1` Pod'lar veya Service'ler için geçerlidir. Sürüm, objenin yeteneklerini ve yapısını tanımlar.

kind

Oluşturulacak Kubernetes objesinin tipini tanımlar. `Pod`, `Deployment`, `Service`, `ConfigMap` gibi değerler alabilir. Kubernetes bu alana bakarak objeyi nasıl işleyeceğini anlar.

metadata

Objenin kimliğini ve tanımlayıcı bilgilerini içeren bir haritadır. İçerisinde objenin `name` (adı), `labels` (etiketleri) ve `annotations` (ek açıklamalar) gibi önemli bilgileri barındırır. Bu bilgiler, objeleri bulmak ve gruplamak için kullanılır.

spec

Objenin "istenen durumunu" (desired state) detaylı bir şekilde tanımlayan ana kısımdır. Bir Pod için hangi container imajının kullanılacağı, hangi portların açılacağı veya bir Deployment için kaç replica olacağı gibi tüm yapılandırma detayları bu bölümde yer alır.

İstenen Durum (Desired State) ve Gerçek Durum (Actual State)

Kubernetes'in temel prensiplerinden biri, her objeyi tanımladığınız "istenen duruma" getirme çabasıdır. Bu, bizim YAML dosyalarında `spec` alanı altında tanımladığımız durumdur. Cluster'ın kontrol düzlemi (Control Plane), sürekli olarak bu istenen durumu gözlemler ve mevcut "gerçek durumu" (actual state) ile karşılaştırır.

İstenen Durum (Desired State)

Bir uygulamanın nasıl çalışmasını istediğinizi anlatan, sizin tarafınızdan tanımlanmış ideal durumdur. Örneğin, bir Deployment için 3 adet Pod'un çalışmasını istersiniz. YAML dosyanızda `replicas: 3` olarak belirtirsiniz.

Gerçek Durum (Actual State)

Cluster'daki objelerin anlık, mevcut durumudur. Örneğin, bir Deployment'ı oluşturduğunuzda, Kubernetes hemen 3 Pod oluşturmaya başlar. Bu Pod'lardan birisi başlangıçta hata verip kapanırsa, gerçek durum 2 Pod'un çalıştığını gösterir.

Kubernetes'in akıllıca yaptığı şey, gerçek durum ile istenen durum arasındaki herhangi bir tutarsızlığı otomatik olarak gidermektir. Önceki örnekte, eğer istenen durum 3 Pod ise ve gerçek durum 2 Pod ise, Kubernetes eksik olan 1 Pod'u otomatik olarak yeniden başlatır veya oluşturur. Bu, sistemin kendi kendini iyileştiren (self-healing) yapısının temelini oluşturur ve altyapı yönetim yükünü büyük ölçüde azaltır.