Kubectl outocomlete i kurmayı unutma !!!!

İnit docker diye bir kavram var ona bak.

Donet te polly library si bi bak ne işe yarıyor.

Harbour = on premise docker image registerer

<https://www.digitalocean.com/community/tutorials/how-to-set-up-a-private-docker-registry-on-top-of-digitalocean-spaces-and-use-it-with-digitalocean-kubernetes>

<https://www.nearform.com/blog/how-to-run-a-public-docker-registry-in-kubernetes/>

ingrees : http header a bakarak (api gateway gibi) load balancer lık yapıyor.

Uygulamayı dışatı açmak için podport kullanmak lazım. Pod üzerinde açmak isiyorsak. Bir diğeri service ip aşağılarda var bilgiler.

https://github.com/jpetazzo/container.training

<http://container.training/>

bizim kullandığımız version lar. Farklı container teknolojilerini de kullanabiliyor Kubernetes

* kubernetes version : 1.15.3
* docker engine: 18.06.1.-ce
* docker compose : 1.21.1

Docker compose da service kısmında verilen service ismi için eğer kendimiz yazdıysak service ismiyle aynı olan bir klasör adında dockerfile var oradan çalıştırıyor.

Eğer docker hub da bir projenin repository sinde github adresi varsa ozman bu docker image inin direk olarak docker hub tarafından build alındığını ve içinde de repositery deki docker file da olanlar var.

**Architecture (PDF 69)**

silde başlığı : Overlay network (flannel / openvswithc / weave)

Kubernetes da clustera erişmek için api server kullanılır, kubernetes verilerini de ETCD tutar. Yani her ğer için bir api endpoint vardır.

Conroller manager en önemli componentitidir. Buna bol bol değineceğiz. Burada mevzu şu. Diyelimki desired state i git e koyduk ve api iel ayağa kaldırdık. U durumda actual state desired state ile aynı oldu. Git de eğer desired state i değiştirisek ve api yede emri veririsek actual state git deki desired state e gelecek. Controller manager ın işidir.

Workerlar üzerinde ise kubelet ve kubeproxy componentleri var. Bunlar üzerinden worker lar çalışıyor. Kubeproxy kubernetes içindeki reverse proxy görevi görür. Kube proxy ip table gibi çalışıyor. Alternatifleri var.

Kubelet ise kubernetes in yerel yönetmi gibi çalışıyor. Örneğin container ın stop olması gerekiyorsa kubelet durduyor. Yani master tek tek container lar la lgilenmiyor.

Düşen bir contaner ı diyelimki önce 1 saniyede bir ayağa kaldırmaya çalışıyor. Daha sonra 2sn, 5 sn 1 dk 5dk gibi arayı açarak ayağa kaldırmaya çalışır.

Overlay network aslında ip over ip demek. Burada hareket eden paketleri büyüterek daha fazla network bilgisi taşıyoruz.

**Nodes (PDF 71)**

Kubernetes architecture: the nodes

Kubernetes docker engine istiyor ama default da Docker geliyor. Ama diğerleride var (PDF 76)

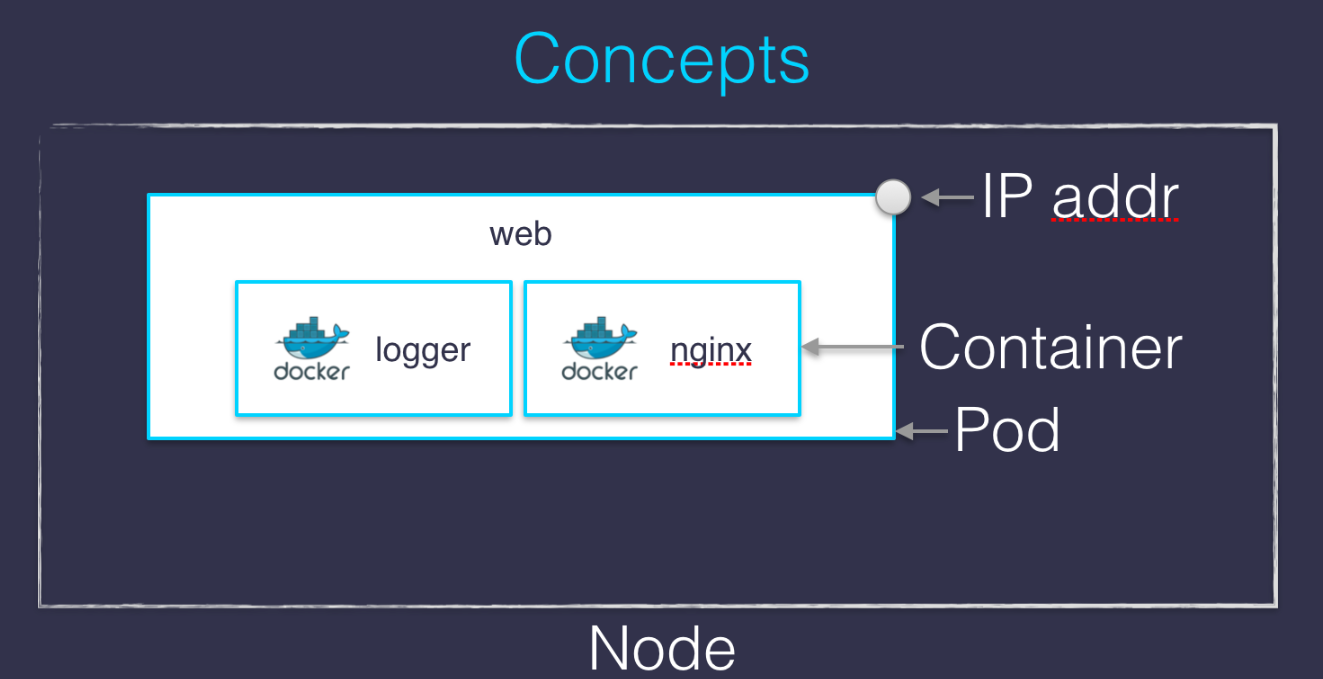
Kubernetes in Container Tuntime Interface i var burada hangi engine takılırsa onunla çalışabiliyor.

Kubernetes için herşey bir resource dur.

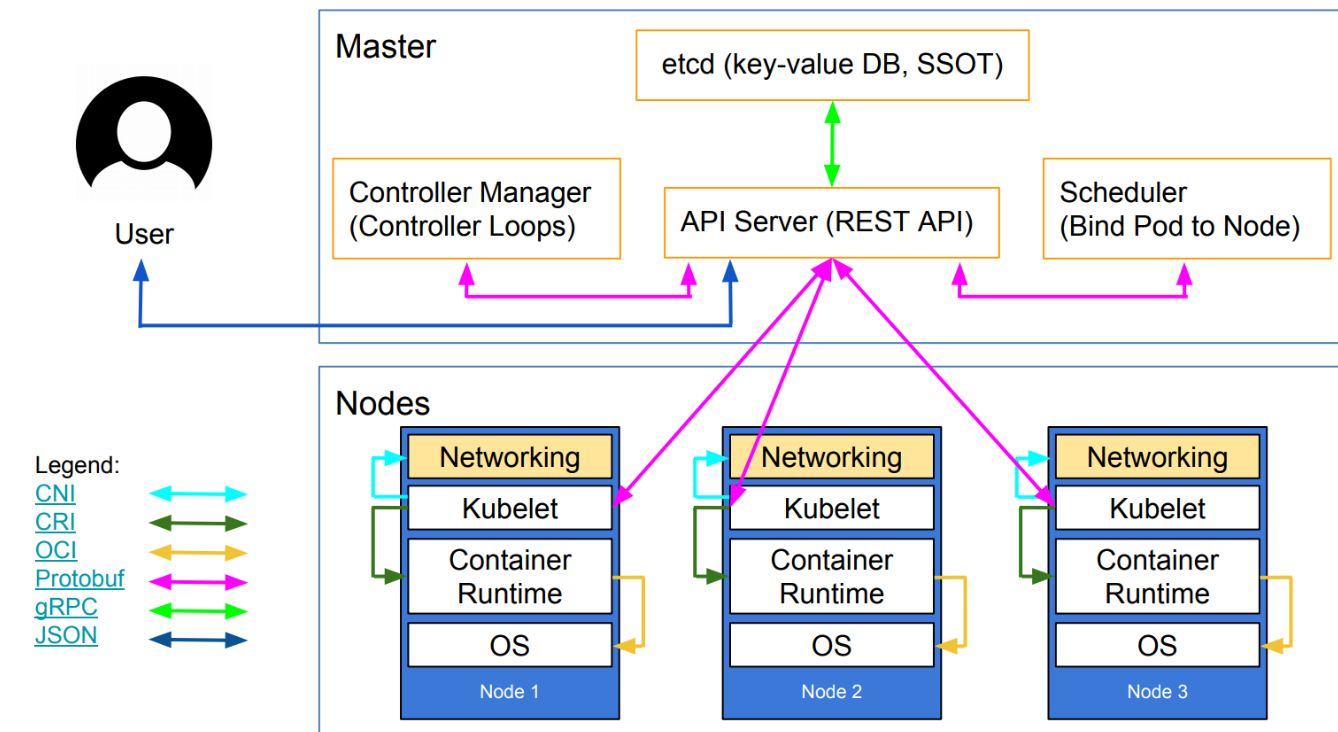
**resources (PDF 86)**

* node (a machine — physical or virtual — in our cluster)
* pod (group of containers running together on a node)
* service (stable network endpoint to connect to one or multiple containers)
* namespace (more-or-less isolated group of things)
* secret (bundle of sensitive data to be passed to a container)

[Kubernetes in en küçük yapı taşı container değil Pod dur. Nadiren bir pod da birden fazla contaşner olur. Örneğin nginx bir container ve logger bir container bunların ikisi bir pod içine konuariblir. Bunlar aynı networkspace içinde aynı ip yi alarak çalışır. Daha oncede docker eğitiminde aynı namespace içinde çalıştırmaıştık bu da aynı şey.](https://training.gokhansengun.com/kube-intro.html" \l "81) (PDF 81)



(PDF 82)



[tabi bu şekil sadece mimari yapıyı anlamak için. Biz sisteme bakarken tek bir yapıymış gibi görüyoruz.](https://training.gokhansengun.com/kube-intro.html" \l "82)

Görüldüğü üzere herkes etcd ye api üzerinden erişiyor.

Örnek case: diyelimki job atılmış ama görünmüyor service.

Önce API ye bak Pod var mı? Pod varsa diğer problemlere bakılır. Diğelimki Kubelet durmuş oaman onu ayağa kaldırıyoruz.

**Declerative vs imprative (PDF 94)**

deklerative da bir iş çalışırken adımladan birinde hata olursa kaldığı yerden devam edilebilir. Ancak imperative de işler birbirine girmiş durumda fark anlamak imkasızlaşıyor. (SQL ve c#)

**Network Model**

Kubernetes multilayer netwotk modeli öldürdü. Artık one big flat IP network var diyor. Ancak sistemciler için internet öünde DMZ arada firewall, daha sonra web layer, app layer, data layer ve bunların arasında da firewall var. Kubernetes bunu kullanmıyor. (PDF 99)

Kubernetes da tüm node lar NAT olmadan konuşabilirler (imkan olmalı- eğer istenirse –gerekliyse – yani bir engel olmamalı).

Her pod kendi ip si ile NAT olmadan cevap verebilmeli.

Bütün pod lar NAT olmadan birbirlerine erişebilmeli

**Kubectl (PDF 105)**

Docker cli in kubernetes daki karşılığı

Bu komut api den verileri çekip gösteriyor. Komut sonucu json ve içinde hakikaten agelen verileri api den çekmek için gereken api adresini bile veriyor.

#kubectl get nodes -o wide

(PDF 109 ) mesela alttaki describe ile herhangi bir type in daha detaylı bilgilerde alınabilir.

kubectl describe type/name (yada kubectl describe type name) kod template i

#kubectl describe node master-0

sadece type hakkında bilgi almak için

kod template i (kubectl explain type)

#kubectl explain node

**Services (PDF 112)**

#kubectl get services

Aslında kubernes in kendi componentleri de aslında contaner peki burada bir ksır döngü yokmu kendi kendininasıl ayağa kaldıyor ?

Bu durumda

#kubectl get pods

Dediğimizde no source ceva bı alıyor halbüki kendi komponentleri de pod içinde. Sebebi namepsace isimini vermemiz lazım. Namespace vermezsek default namespace i gelir o yüzden boş geldi. Şöyle yazarsak

#kubectl get pods –n kube-system

Bu sayede system pod larını da görebiliriz. Burda her node a bir dns olduğunu görebiliriz.

istio service mesh aracı bütün bu istemindeki podlara kendini kuruyor ve bizim bütün sistemimizdeki yükü manitor debiliyor. Kubernetes in bu extend edilebilir durumu çok farklı tool ların yazılmasını sağlıyor. Yani bir node a başka bir componenet olarak kendini kurup çalışabiliyor.

Diyelimki amacımız master node içindeki abc nodunun içini tanımak

#kubectl describe node master-0 abc

List the secrets in the kube-public namespace:

#kubectl -n kube-public get secrets

İlk Pod umuzu ayağa kaldırıyoruz

#kubectl run pingpong --image alpine ping 1.1.1.1

Sensible default için hangi node da çalışacağıyla ilgilenmiyoruz (aksi belirtilmedikçe).

Bu komut bir pod create edip içine cotainer ayağa kaldırdı. Anca kubernetes bakış açısıyla bir ygulama deployment yapmış oladuk. Yani bol bol abstraction yapıyoruz.

Şu komutla detayları görebiliriz.

#Kubectl get all

Replicaset kaçtane replikası çalışacak onu belirtiyor. Deployment objesi ise versiyonlamayı kontrol ediyor.

Deployment replicasetleri, replicasetler de posları yönetiyor.

Replicaset lerde desired/current/ready kavramı var. Yeni versione geçişlerde buradaki sayılar değişecek.

#kubectl logs deploy/pngpongs –follow (sayfa sayfa gösterir)

kubectl scale deploy/pingpong --replicas 8

#kubectl get all

Bu komutla artık desired/current/ready de rakamlar değişmiş oluyor. Deployment ın replicaset ini 8 yapmış oluyoruz. Bu arada aslında biz imperative gidiyoruz. Desired state bakış açıız yok aslında c# gibi davranıyoruz yani SQL gibi değil.

#kubectl delete <podid>

Bu durumda signal bekliyor kill için kıbernetes da 30 sn. Docker da 10 sn di. Bu udurmda hemen yeni birtane create ediyor. Çünki deploy replica sayısı 8. (resilience)

**Çok öenmli deploy tipinde küçük bir iş yapıp kapanan bir iş verirsek kubernetes tekrar ayağa kaldırmaya çalışacak. Ancak daha önce çalışan bir deployment a yeni versiyon geçtik ve yeni gelen deployment da böyle değişliklik yapmıştık yani eskisi uzun çalışıyorken yenisinde çok daha kısa bir iş yapan bir deployment geçtik bu durumda eskisi düşüp duruyorsa onu daha fazla create etmeyip eskisinden ayakta olanlalrı ayakta tutar (crashloopbackoff olur ). Buda kubernetes in depleyment modeli. Bu durumda replicaset de bu deopleyment için iki kayıt olur. Yeni replicaset tin ready sayısı tamamlanana kadar eskisi kapanmaz.**

**Deploy tipi için bütün objeler eşit ve aynı işi yapıyor demektir. Peki bir tanesi master diğerleri worker olan bir uygulama olsa yada stateful bir uygulama yapıyor isek deploy haricinde başka objelerde var? !!!!**

**Eviction**

CPU ve Network geri alınabilen kaynaktır ancak Memory ve Storage geri alınamaz.

Bir kaynak gauranteed ise requşred ve limit kaynak aynıdır. Burstable da reqired limitden daha düşüktür, son olarak best efford ise iksininde ucu açıktır. Bu durumda node a ağır bi abi gelirde guaranteed olarak işaretli ise kaynak bustable da değiştirelemeceği için gerekirse onlarda evicted olurlar ve öldürülürse. Daha düşük limit kaynağı ile tekrar ayğa kalkabilir. Bu durum linux de de aynıdır.

**Pause Container**

Kubernetes her bir pod için pause adında çok küçük bir container crate ediyor. Daha sonra pod a koyduğu her conaşner ı bu container a network namespace ine bağlıyor. Bu aslında kubernetes daki bir hack (yani bir kurnazlık).

**Exposing Containers (PDF 156)**

**Service types**

**-ClusterIP Fix ip :** bütün objeler ölümlü ozaman kubernetes ölümsüz olan componentler create ediyor mesela service ve service ip si create ederek pod lara erişimi bu service ismi ip siüzerinden yapıyor. Arkada ölecek bir pod varsa ona giden paketleri göndermeyip diğer yeni yaratılacak poda yada diğer podlara yönelendiriyor buna ClusterIp deniyor.

**-Node Port** . **Fix port:** kişi A poduna bağlanmak isiyor ve diyelimki 31000 portunu kullanıyor. Ancak A portu aslında abc servisini hizmet veriyor diyelim. **NodePort** aralığında yer alan bu 31000 portunu A podu dipğer podlarda dinliyor daha doğrusu istek bu porta geldiğinde diğer podlarda ilgili porta yönşendireceğini biliyor ve onlara yönlendiriyor isteği. Dışarı açarken bu önemli oluyor. Ancak bu yolla sistem ipleri çözüyor ve artık ip üzerinde devam ediyor.

**-LoadBalancer** : içerdeki servicleri dış load başancer lara veriyor.

**-External Name: kubernetes da çlıştımayacağımız kaynaklara ismiyle erişebilmek için bir isim tanımlamsı yapıp kubernetes içinden dışarıya dns lik gibi birşey yapmayı sağlıyor.**

**Service postu ayrı pod ların portu ayrı.**

**Running containers with open ports**

kubectl run httpenv --image=jpetazzo/httpenv --replicas=10

burada kullandığımız httpenv uygulama adımız.

kubectl expose deploy/httpenv --port 8888

bu deopleymentı 8888 den expose et diyoruz.

Oluşturduğumuz bu uygulamanın bir service ip si var yani cluster ip si onun fixed olduğu garanti biz bu serivce i silmediğimiz sürece bizim uygulamamızın ip si cluster ip sidir.

Oluşturulan her uygulamanın bir DNS adı olur. Nslookup da sadece httpenv diye sorulamak foğru sonuç vermez. Doğru node un içinde /etc/.... adresinde aranacak dns fdqn adı görülebilir.

**Headless Services**

Aradan clusterip (service ip) yi çıkartırsak appname i çağırdığımızda dns pod aiplerini döner buna headless service denir. (PDF 165)

Neden ihtiyaç olabilir

Sometimes, we want to access our scaled services directly:

* if we want to save a tiny little bit of latency (typically less than 1ms)
* if we need to connect over arbitrary ports (instead of a few fixed ones)
* if we need to communicate over another protocol than UDP or TCP
* if we want to decide how to balance the requests client-side

**Services and endpoints (PDF 167)**

Aslında her pod ayağa kalkıp hazır olduğunda pod ile service arasında bir endpoint create edilir ve kullanıcı A podundan service i talep ettiğinde service arkadaki end poinlere isteği gönderir. Service label ları kullanarak endpoindleri çağırır.

Peki yaml dosyamızda selector ile yazan endponti elimizde manuel crate ettiğimiz bir service in endpoint adını yazıp onuda dış bir kaynağı hedeflersek aslında sistemin bu özelliğini kendi mellerimiz için kullanmış oluyoruz.

Herşeyi silmek için

Kubectl delte all --all

**Our app on Kube (PDF 170)**

In this part, we will:

-build images for our app,

-ship these images with a registry,

-run deployments using these images,

expose these deployments so they can communicate with each other,

expose the web UI so we can access it from outside.

Ancak burada build için local makinamız yok cluster da her node un ulaşabilceği bir local regestry e ihtiyacımız olacak.

The plan

Build on our control node (node1)

* Tag images so that they are named $REGISTRY/servicename
* Upload them to a registry
* Create deployments using the images
* Expose (with a ClusterIP) the services that need to communicate
* Expose (with a NodePort) the WebUI

Which registry do we want to use?

* We could use the Docker Hub
* Or a service offered by our cloud provider (ACR, GCR, ECR...)
* Or we could just self-host that registry

We'll self-host the registry because it's the most generic solution for this workshop.

Using the open source registry

* We need to run a registry container
* It will store images and layers to the local filesystem

(but you can add a config file to use S3, Swift, etc.)

* Docker requires TLS when communicating with the registry
  + unless for registries on 127.0.0.0/8 (i.e. localhost)
  + or with the Engine flag --insecure-registry
* Our strategy: publish the registry container on a NodePort, so that it's available through 127.0.0.1:xxxxx on each node

Create the registry service:

kubectl run registry --image=registry

Expose it on a NodePort:

kubectl expose deploy/registry --port=5000 --type=NodePort

Connecting to our registry

View the service details:

kubectl describe svc/registry

Get the port number programmatically:

NODEPORT=$(kubectl get svc/registry -o json | jq .spec.ports[0].nodePort)

REGISTRY=127.0.0.1:$NODEPORT

Testing our registry

View the repositories currently held in our registry:

curl $REGISTRY/v2/\_catalog

Testing our local registry

Make sure we have the busybox image, and retag it:

docker pull busybox

docker tag busybox $REGISTRY/busybox

Push it:

docker push $REGISTRY/busybox

Checking again what's on our local registry

Ensure that our busybox image is now in the local registry:

curl $REGISTRY/v2/\_catalog

The curl command should now output:

{"repositories":["busybox"]}

Building and pushing our images

We are going to use a convenient feature of Docker Compose

Go to the stacks directory:

cd ~/container.training/stacks

Build and push the images:

export REGISTRY

export TAG=v0.1

docker-compose -f dockercoins.yml build

docker-compose -f dockercoins.yml push

Let's have a look at the dockercoins.yml file while this is building and pushing.

|  |
| --- |
| version: "3"  services:  rng:  build: dockercoins/rng  image: ${REGISTRY-127.0.0.1:5000}/rng:${TAG-latest}  deploy:  mode: global  ...  redis:  image: redis  ...  worker:  build: dockercoins/worker  image: ${REGISTRY-127.0.0.1:5000}/worker:${TAG-latest}  ...  deploy:  replicas: 10 |

İmage taglarine bakarsak REGISTERY adını bulazsa 127.0.0.1 e bak TAG i bulamzsan lastest ı al demektir bu.

image: ${REGISTRY-127.0.0.1:5000}/rng:${TAG-latest}

artık image crate oldu ve locak registery ye konuldu

**şimdi deploy ediyoruz**

Deploy redis:

kubectl run redis --image=redis

Deploy everything else: (kırmızı yazan yerlere dikkat)

for SERVICE in hasher rng webui worker; do

kubectl run $SERVICE --image=$REGISTRY/$SERVICE:$TAG

done

**Is this working?**

After waiting for the deployment to complete, let's look at the logs!

(Hint: use kubectl get deploy -w to watch deployment events)

Look at some logs:

kubectl logs deploy/rng

kubectl logs deploy/worker

ancak şuan da çlıaşmaz. Öünki redis service ini expose etmeyi unuttuk.

**Exposing services internally**

* Three deployments need to be reachable by others: hasher, redis, rng
* worker doesn't need to be exposed
* webui will be dealt with later

Expose each deployment, specifying the right port:

kubectl expose deployment redis --port 6379

kubectl expose deployment rng --port 80

kubectl expose deployment hasher --port 80

**Is this working yet?**

The worker has an infinite loop, that retries 10 seconds after an error

Stream the worker's logs:

kubectl logs deploy/worker --follow

(Give it about 10 seconds to recover)

We should now see the worker, well, working happily.

**Exposing services for external access**

* Now we would like to access the Web UI
* We will expose it with a NodePort

(just like we did for the registry)

* Create a NodePort service for the Web UI:

kubectl expose deploy/webui --type=NodePort --port=80

Check the port that was allocated:

kubectl get svc

**Accessing the web UI**

We can now connect to any node, on the allocated node port, to view the web UI

Open the web UI in your browser (<http://node-ip-address:3xxxx/>)

**Scale Edelim**

Rng servisin replicaset ini arttırmak istiyoruz nasıl yaparız.

Kubectl edit deploy/rng

Burada YAML dosyasını editliyorus replica bölümünü 2 yapıyruz.

Ancak bu yetmez birde worker ı rıda arttrıyoruz 5 yapıyrpuz

Kubectl edit depley/worker

**Artık replicamız arttı**

**Accessing the API with kubectl proxy (PDF 204)**

TLS ile kişiye özel sertifika üretilip girş yaprıyor. Bütün CI araçları aslında bu sistem kullanılıyor. Sadece kişiye özel private key üretildiği için o kişi dünyanın her yerinden açabiliyor. Private kişide public serverda yani.

**Accessing the API directly**

Retrieve the ClusterIP allocated to the kubernetes service:

kubectl get svc kubernetes

Replace the IP below and try to connect with curl:

curl -k https://10.96.0.1/

The API will tell us that user **system:anonymous** cannot access this path. (PDF 206)

[https://kubernetes.io/docs/reference/access-authn-authz/authentication/#authentication-strategies](https://kubernetes.io/docs/reference/access-authn-authz/authentication/" \l "authentication-strategies)

oluşturulan client pem dosyası ~/.kube/config dosyası içini localde oluşturdugumuz ~/.kube/config içine kaydediyoruz.

Ve daha sonra alttaki komutla local bilgisayardan servera bağlanabiliriz.

Bu arada serverdan config doyasını kopyalamakda mümkün yolu (PDF 219)

scp USER@X.X.X.X:.kube/config ~/.kube/config

bu arada bu kanal açıldığı için bu yolla kubernetes içindeki bir database a bile güvenli yolla erişmiş oluyoruz.

Sunucudan indirdiğimiz config doyasının örneği bu word dosyasının bulunduğu klasöre koydum.

Daha sonra aşağıdaki kodla test ediyoruz

kubectl get pods -A

ve böylece server daki kubernetes daki pod ların listesi gelmiş oldu.

Bu durumda aslında localde sistem bu key i kullarak sunucunun üzrindeki sistemin api sini local de 127.0.0.1:8001 de yayın yaptırır. Ve bu güvenş yolla biz local deki bu API ile uzak sisteme bağlanıyruz.

Proxy nin adresini öğrenmek için

Kubectl proxy

Bu yolla birden fazla cluster yönetiyorsak farklı kube config doyasını değiştimek update etmek gerekebilir (PDF 220)

ancak bunun için bir tool var

yazarı türk Ahmet Alp Balkan kubectx bunu yapabiliyor daha kolay

şuana kadar biz API ye (yani web) ulaşmış olduk ama diğer servisleri de dışarı aşmak istersek (PDF 225)

**kubectl proxy in theory**

daha ilginç bir yolla aslında API üzerinden içerideki bir servisi örneğin bir web sitesini uzaktan çalıştırma şasnımız var çok ütopik bir durum (PDF 228)

/api/v1/namespaces/<namespace>/services/<service>/proxy

baya adres satırında kurduğumuz bu güvenli yol ile yani proxy üzerinden api ile içerideki bir servisi çağırmış oluyoruz.

Örnek (ngin x ayağa kaldırıp bakıyoruz)

kubectl run web --image=nginx

kubectl expose deploy/web -port:8000

bu haliyle aslında biz sunucudaki cluster da servisi 8000 portundan açmış olduk ama bu hale dışarıya açık değil.

Bu nedenle kendi güvenli proxy kanalımızda üstteki adres gibi bir adres çağırarak siteyi çağırmış oluyoruz. Tabi siteyi çağırmadan once kubectl proxy çalıştırıp test etmek lazım ulaşılabiliyor mu diye.

Ancak bunu debug için kullanmak mantıklı

Tabi DB gibi birşeyi TCP (yani http olmayan) nasıl görebiliriz. Bunun için de

kubectl port-forward in theory (PDF 230)

#kubectl port-forward service/name\_of\_service local\_port:remote\_port

Böylece aslında önceki versiyonu yani web i api üzerinden açma işini iptal edip sadece bu komut da öğrenilebilir.

Böylece örneğin redis i içeriden kendi lokalimize bağmaış oluyoruz.

Kubectl port-forward svc/redis 10000:6379

Redis local de 10000 portuna bağlamış olduk

Eğer yukarıda API üzerinden değilde bu şekilde web i bağlarsak. Diyelimki localde 9000 i çağırdığımızda sunucudaki 80 portundan nginx çalışmış olacak.

**Daemon sets (diyelimki bütün node lara kurmamız gereken birşey olsun ozaman bunu kullanıyoruz.)**

Buraya kadar deploy görmüştük. Deamonset ise her node da bir tane olmasını isteğimizi servisler için kullanılır. Sadece bir tane olması öenmli. Diyelimki loglama yapacak bir araç.

Hiyerarşik olarak deamonset ile deploy aynı

Sadece deploy atında replica set onun altında da pod var. Deamonset ise direkt olarak pod ları yönetiyor.

Deamon set i uodate etmek zor çünki silip tekrar yapmak gerekir.

**Expose edilirse bu da service ile dışarı açılabilir. Ancak spresifik bir node un içindeki deamon set poduna erişmek istiyorsak ozmaan service loadbalancer gibi çalıştığı için işimize yaramaz hangisine bağlanacağnı bilemeyiz nodelarda. Bu nedenle pod a özel bir port atayıp nokta atışı bağlanabiliriz.**

**Creating the YAML file for our daemon set (PDF 245)**

deployment oluştur tipini deaomanset e çevir. Ama bu yeterli değil hatayı görmek için **(PDF **251**)**

kubectl apply -f rng.yml --validate=false **(validate false u asla kullanma) (PDF **254**)**

burada oluşturulan pd lara bakarsak birçok görürüz sebebi her node için bir pod create eder **(PDF **259**)**

çalışan tüm podlara bak

kubectl logs -l run=rng --tail 1

**(PDF **261**)**

**Updating a service through labels and selectors**

Şuan hem deploy servisi hem de deamonset ayakta (pdf örneğine göre)

Teknikler : **(PDF **275**)**

Burada demonsetlere bir label ekledi rahat bir şekilde bulmak için (Adding our label başlığı)

**(PDF **279**)**

silme işlemi (Cleaning up stale pods)

**(PDF **282**)**

devamında ise depşoymen ın silinmesini anlatıyor.

**Rolling updates**

Diğelimki nıde lar ağzına kadar dolu ve biz update yapmak istiyoruz. Bu durumda nasıl yapacağız?

Diyelinki a servisinde 2 pod var bu durumda eskiden birtanesini ykapatmalıyız.

Bu durumda bizim maxunevaible değeri 1 yani en az bir tane ayakta kalmalı.

Maxsurge ise yüzdesel bir değer

At any given time ...

* there will always be at least replicas-maxUnavailable pods available
* there will never be more than replicas+maxSurge pods in total
* there will therefore be up to maxUnavailable+maxSurge pods being updated

örneğin maxunevaible 1 maxsutge 0 ise tek tek update eder

Checking current rollout parameters

Recall how we build custom reports with kubectl and jq:

Show the rollout plan for our deployments:

kubectl get deploy -o json | jq ".items[] | {name:.metadata.name} + .spec.strategy.rollingUpdate"

**Rolling updates in practice**

As of Kubernetes 1.8, we can do rolling updates with:

deployments, daemonsets, statefulsets (henüz statefulset görmedik örneğin DB gibi)

statefulset lerde önce bir pod ayağa kalkar mecburen mesela potgre için master kalkar sonra slave ler. Silerkende önce slave ler silinir.

Editing one of these resources will automatically result in a rolling update

Rolling updates can be monitored with the kubectl rollout subcommand

Örnek çalışma için **(PDF **296**)**

Böümün sonuna kar bu örnek devam ediyor. Kesin yapmalısın bu örneği

Diğer bir yöntenmde YAML üzerinden patch lemek

**Applying changes through a YAML patch (güzel bir yöntem. Burda tüm herşeyi uodate etmiş olmuyoruz)**

**Dışatrıdan almış oldupumuz hai hazırda çalışan bir uygulama için bir değişiklik gerektiğinde ilgili yerleri değiştirmek yeterli.**

**(PDF **307**)**

örnek kod

kubectl patch deployment worker -p "

spec:

template:

spec:

containers:

- name: worker

image: $REGISTRY/worker:v0.1

strategy:

rollingUpdate:

maxUnavailable: 0

maxSurge: 1

minReadySeconds: 10

"

**Healthchecks**

Each container can have two (optional) probes: (iki tür check var)

* liveness = is this container dead or alive? (pod var mı)
* readiness = is this container ready to serve traffic? (pod var ama hazırmı)

servis ready diyene kadar kubernetes veri göndermiyor.

Ready olduktan sonra da belirli aralıklarda ayaktamı diye servisi kontrol eder.

Ancak yine belli aralıklarla da ready tekrar kontrol ediyor.

Bundan dolayı aslında 2 intance ile bile çalışabilen bir uygulamayı bu dinamiklere uygun çalıştırabimek için örneğin 10 instance ile çalıştırmak mantıklı olacaktır.

Liveness ın çok hılzı dönmesi lazım o yüzen örneğin donet projesi için donet runtşme ayakta mı diye bakmak yeterli ve daha sık yapılabilir. Ancak readiness daha seyrek ama zor bi chek yapmak lazım. Mesela uyguamaya bir iş yaptırılabilir.

(**Liveness probe**) **(PDF **311**)**

genelde alway demek lazım. Çünki uyguma belki doğr şekilde hataya düşemden resturn 0 olarak çıkış yapıyor o yüzden devaml ayağa kaldırmak mantıklı.

**Readiness probe**

* Indicates if the container is ready to serve traffic
* If a container becomes "unready" (let's say busy!) it might be ready again soon
* If the readiness probe fails:
  + the container is not killed
  + if the pod is a member of a service, it is temporarily removed
  + it is re-added as soon as the readiness probe passes again

When to use a readiness probe

* To indicate temporary failures
  + the application can only service N parallel connections
  + the runtime is busy doing garbage collection or initial data load
* The container is marked as "not ready" after failureThreshold failed attempts (3 by default)
* It is marked again as "ready" after successThreshold successful attempt (1 by default)

Different types of probes

* HTTP request
  + specify URL of the request (and optional headers)
  + any status code between 200 and 399 indicates success
* TCP connection
  + the probe succeeds if the TCP port is open-
* arbitrary exec
  + a command is executed in the container
  + exit status of zero indicates success

(Example: HTTP probe) **(PDF **317**)**

(Example: exec probe)**(PDF **318**)**

kubectl edit deploy/abcservisi

deyip yaml i editme yapıyoruz ve örneklerdeki http veya tcp prob züerinde readliness veya liveliness healthcheck yapabiliriz.

**Accessing logs from the CLI**

Her node dan ve pod lardan ayrı ayrı log almak zor iyileştirmek mümkün

Araçlardan biri Stern **(PDF **326**)**

stern rng (rng service adı)

bu kod ile toplu log okuma şansımız olmuyor.

**Centralized logging**

Bu çok öenmli logları merkezileştirmek lazım.

**Tabi bu örnekteki gibi dışarıdan buldupumuz bir yaml bu şekilde çaıştırlmamalı. HELM kullanmak lazım. İleride göreceğiz.**

**Ayrıca kullandğımız servicelerin uygulamaların healthcheck lerinin doğru laıştığından emin olmak lazım. Doğru çalışmıyorsa hazır olmayan bir uygulama yayına çıkmış olur.**

What is EFK? (örnek var**(PDF **335**)**

* ElasticSearch (to store and index log entries)
* Fluentd (to get container logs, process them, and put them in ElasticSearch)
* Kibana (to view/search log entries with a nice UI)

kubectl apply -f ~/container.training/k8s/efk.yaml

daha sonra kurulumu

watch ‘kubectl get pods’

ile izliyoruz kurulup kurulmadıpını

kubectl get pods -o wide

oluşan podlara bakıyoruz kibana adresi için. Yalnız bu örnek bizim workerlardan birinin işlemcisini doldurdu. Bunun için alttaki komutla worker da çalışan servisin replicasını 0 yaprak kaldırıyoruz.

Kubectl edit deoply/kibana

Değişiklikten sonra adresten loglar görülür ama bu iyi bir yöntem değil. Docker ların ekrana yazdığı logları topladı.

Şimdi bu uygulamayı bu şekilde çalıştırmamız bilmediğimiz bir yaml çalıştırmak demekti. Örneğin kibananın upgrade i gelse update yapamayız. Daha stabil daha kubernetes kafasıyla bir sistem lazım. Bunun için HELM kullnıyoruz

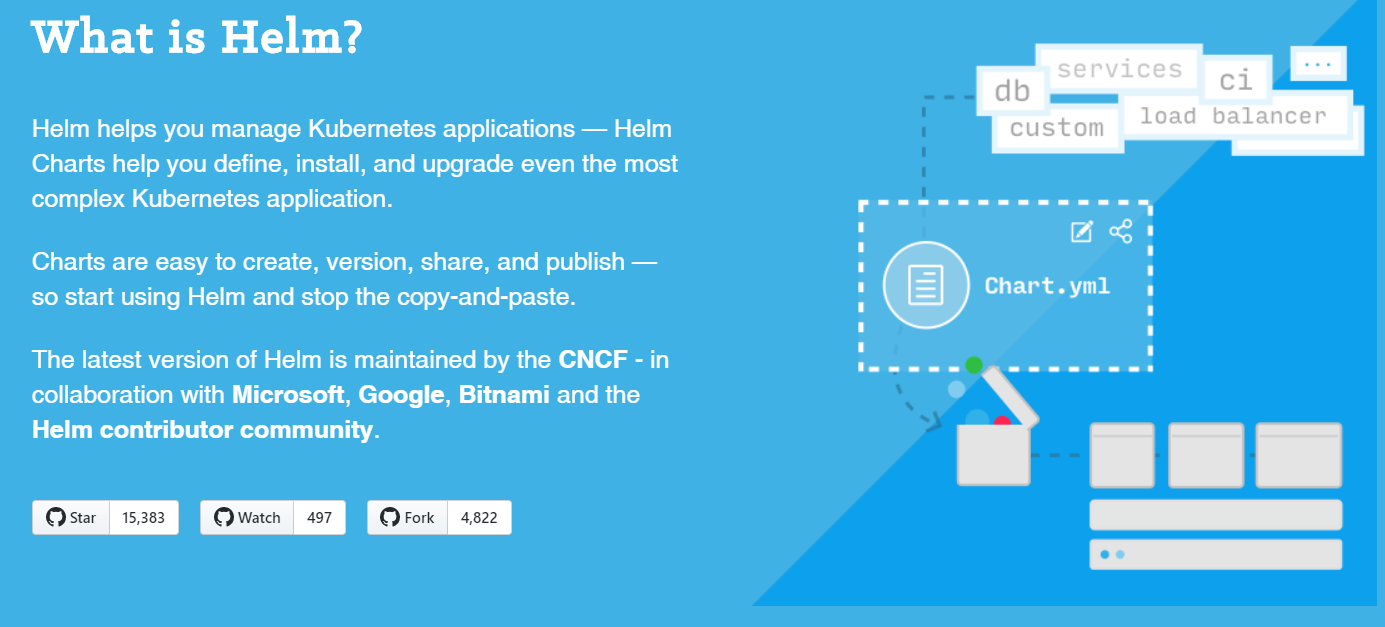
**Managing stacks with Helm (PDF **342**)**

Helm bütün bu gördüğümüz kubernetes objelerinin üstünde bir çatı.

Helm biraz compose gibi ama yum a daha çok benziyor. Yani ben istediğimi söylüyorum helm gerekli herşeyi toplayıp çalışrıyor.

Helm (The package manager for Kubernetes)

<https://helm.sh/>



Şuan asla helm 2 kullanılmamalı

Prometues u yüklüyoruz mesela

<https://hub.helm.sh/> bu adresten arıyoruz.

Bu örneği yarın yapacağız. Stable tag ile başlayanlar dockerhub daki güvenilir repository si demektir.

Bitnabi de güvenilir repository lerden biri

<https://hub.helm.sh/charts/stable/prometheus>

<https://hub.helm.sh/charts/stable/rabbitmq-ha>

bu helm repositoriylerine helm chart denilir.

Kendi chartımızı nasıl yaparız.

En önemli dosya values.yaml dosyası. Herşey bunda yazıyor.

Helm create

Bir chart oluşturuyor. Values.yaml den configure ediyoruz.

Templates altında başka yaml lar buralarda service vs ne varsa ayarlayabiliyoruz.

Buradaki values.yaml bir yere kopyalaıyoruz. Ve kopya ile devam ediyoruz.

İmage repositery yi değiştiriyoruz.

Pullpolicy siliyoruz. İstersek değiştiredebiliriz tabiiki.

Nameoverride : buradan oluşturduğumuz char tın adını değiştirmek isiyorsak adını değiştiriyoruz.

İngress siliyoruz. http header a bakarak loadbalancing yapyor bizde gerek yok.

Service kalıyor, resources kalıyor, limits kalıyor,

Geri kalanı silip kaydediyorur.

Bu yeni values.yaml dosyasını kaydediyoruz.

Helm install abc\_app –f yeni-vules.yaml hangi templates i kullancaksak onun klasörünü veriyoruz.

Template dosyaları içindeki dosyalardaki value lar için values.yaml doyamızda tanımlamaları yapıyoruz.

Her uygulamamız için bir values.yaml dosyamızaolmalı ve bunları aynı template dosası ile deploy edebiliriz.

Aslında matık şu. Bu ana kadar ellimizel yaptıpımız bütün kodları helm yaml dosyalari ile yazarak çalıştırıyoruz. Yani kubernetes işlerimizi de configuasyon doyaları şeklinde tutmuş oluyoruruz.

**Helm e çok iyi çalış (!!!!!!) helm siz yönetmek imkansız.**

**Helm values verileri ci dan da verilebilir. Diyelimki username password var onu helm i çalıştırıken vermek isteyebiliriz.**

**Namespaces**

Kaynakları namespace e gore belirleyebiliriz.

Kaynağımız default namespace altında değilse ozaman namespace adını yazarak komutları çalıştırmak lazım. Kaynkalraı hep namespace ile çağırmak gerekiyor.

**Network Policies**

One big flat network de iki pod birbirine ulaşamasın demek için kullanılcak yöntem network policies dir.

**(PDF **379**)** (örnek için)

**(PDF **380**)** (Adding a very restrictive network policy)

**(PDF **381**)** (Looking at the network policy) (böylece helm de bu alanı template e ekleyip kullanıyoruz)

**(PDF **382**)**(Allowing connections only from specific pods)

**(PDF **383**)**(Looking at the network policy)

dökümann devamında da birçok uygulama var

Remove all network policies:

kubectl delete networkpolicies --all

peki soru şu bütün bu firewall işlerini kim yapıyor. (flunnal, weawe, calibu .... bunlar overlay netwotk plug inleri, podlar arası trafiği de aslında bunlar yapıyor.) ip over ip ipsec demek. Belki 15 tane network plugin i var.

network policy yazabiliriz yani defautl (vanillia) olarak geliyor ancak enforece etmek için network pluginlere ihtiyaç oluyor. DNS de çalışmamış oluyor haliyle. Zaten bunar olmazsa node lardan node lara service lere erişim gitmiş oluyor. Default da bunlar kurulu gelmşyor.

Standar iptables ile her node un iptables ına kural koyarak pdolar konuşturulabilir ancak firewall yapamayız. Çünki arada router, switch vs yok. Bu nedenle network component kullanıyoruz.

Bu ndenle direwall yapcaksak bir network plugini kullanmak lazım.

**Authentication and authorization**

**(PDF **399**)**

birde service lere account verilebilir. Kullanıcıların sistme girişi başka birşey bir de sistemin çalışırken kullanacağı service hesapları create edilebilir. **(PDF **408**)**

service hesabı uygulama **(PDF **421**)**

Binding a role to the service account **(PDF **422**)**

Authorization in Kubernetes **(PDF **415**)**

Role-based access control **(PDF **416**)**

Güvenlik check için güzel komut

**Testing directly with kubectl**

We can also check for permission with kubectl auth can-i:

kubectl auth can-i list nodes

kubectl auth can-i create pods

kubectl auth can-i get pod/name-of-pod

kubectl auth can-i get /url-fragment-of-api-request/

kubectl auth can-i '\*' services

And we can check permissions on behalf of other users:

kubectl auth can-i list nodes \

--as some-user

kubectl auth can-i list nodes \

--as system:serviceaccount:<namespace>:<name-of-service-account>

**Exposing HTTP services with Ingress resources**

Kubernetes de şuan serivce componentin sadce doğrudan podlara gönderiyor ancak daha akıllı bazı çözümlere ihtiyaç var. Örneğin http paketine bakarak karar verebilirse daha iyi olur. Kubernetes da bunu plugin le yapıyor.

Ingrees interface e bir plugin takiyoruz alttakiler ornek. Ssl i ingrees te terminate edip devamını http devam edebiliyoruz.

Mesela nginx ingrees (ama iki tane var birini kubernetes comunitee geliştiriyor onu tercih etmeliyiz. parasız)

HaProxy Traefik var mesela

Counter ent bişey var

Vs vs

Daha sonra kurallar yazılabiliyor.

(nip.io, dyndns vs)

Örnek

What does an ingress resource look like?

**(PDF **452**)**

Here is a minimal host-based ingress resource:

apiVersion: extensions/v1beta1

kind: Ingress

metadata:

name: cheddar

spec:

rules:

- host: cheddar.A.B.C.D.nip.io

http:

paths:

- path: /

backend:

serviceName: cheddar

servicePort: 80

(It is in k8s/ingress.yaml.)

Burada ingrees isteği yine bir service yönendiiriyor. Bu da latency demek aradan service i de çıkartmak mümkün.

**Kubernesin scale olup olmayacağını prometheous dan bir kpi aya bakarak karar verdirebiliriz.**

**Creating our first ingress resources**

Edit the file ~/container.training/k8s/ingress.yaml

Replace A.B.C.D with the IP address of node1

Apply the file

Open http://cheddar.A.B.C.D.nip.io

**Collecting metrics with Prometheus (PDF **459**)**

Collecting metrics with Prometheus

* Prometheus is an open-source monitoring system including:
  + multiple service discovery backends to figure out which metrics to collect
  + a scraper to collect these metrics
  + an efficient time series database to store these metrics
  + a specific query language (PromQL) to query these time series
  + an alert manager to notify us according to metrics values or trends

We are going to deploy it on our Kubernetes cluster and see how to query it

Why Prometheus?

* We don't endorse Prometheus more or less than any other system
* It's relatively well integrated within the Cloud Native ecosystem
* It can be self-hosted (this is useful for tutorials like this)
* It can be used for deployments of varying complexity:
  + one binary and 10 lines of configuration to get started
  + all the way to thousands of nodes and millions of metrics

**Volumes**

Kubernetes daki volume podların içindeki containerların kullanabileceği ortak disk alanı gibi düşünebiliriz.

Simle örnek

apiVersion: v1

kind: Pod

metadata:

name: nginx-with-volume

spec:

volumes:

- name: www

containers:

- name: nginx

image: nginx

volumeMounts:

- name: www

mountPath: /usr/share/nginx/html/

diğer örnek ama bunda sıralama sorunu var. Readenes prob koyarak kontrol edebiliriz. Yani volume hazır olunca yaz gibi. Bu arada kubernetes da daha iyi yöntem var initcontainer kullanılabilir. Çünki readeness da da tehlike var çünki volume un hazır odupunu nasıl anlayacak. Ama işme hem karışık hemde kurgu yavaş çalışabilir.

A volume shared between two containers

apiVersion: v1

kind: Pod

metadata:

name: nginx-with-volume

spec:

volumes:

- name: www

containers:

- name: nginx

image: nginx

volumeMounts:

- name: www

mountPath: /usr/share/nginx/html/

- name: git

image: alpine

command: [ "sh", "-c", "apk add --no-cache git && git clone https://github.com/octocat/Spoon-Knife /www" ]

volumeMounts:

- name: www

mountPath: /www/

restartPolicy: OnFailure

ancak bu pod da olduğu için pod uçarsa bu da uçar

ileride Volumes and Persistent Volumes bölümü var.

**Managing configuration**

Injecting configuration files **(PDF **506**)**

Configmaps storing entire files **(PDF **507**)**

Configmaps storing individual parameters

* In this case, each key/value pair corresponds to a parameter
* Key = name of the parameter
* Value = value of the parameter
* Examples:

# Create a configmap with two keys

kubectl create cm my-app-config \

--from-literal=foreground=red \

--from-literal=background=blue

# Create a configmap from a file containing key=val pairs

kubectl create cm my-app-config \

--from-env-file=app.conf

Exposing configmaps to containers **(PDF **509**)**

**Creating the configmap**

* Go to the k8s directory in the repository:

cd ~/container.training/k8s

* Create a configmap named haproxy and holding the configuration file:

kubectl create configmap haproxy --from-file=haproxy.cfg

* Check what our configmap looks like:

kubectl get configmap haproxy -o yaml

devamına bak **(PDF **511**)**

Go to the k8s directory in the repository:

cd ~/container.training/k8s

Create a configmap named haproxy and holding the configuration file:

kubectl create configmap haproxy --from-file=haproxy.cfg

Check what our configmap looks like:

kubectl get configmap haproxy -o yaml

tabi bunkarın hepsini helm olarak yazmamız lazım

**Passwords, tokens, sensitive information**

**(PDF **519**)**

Differences between configmaps and secrets **(PDF **520**)**

**Stateful sets**

Deploy, deamon set, ingrees, ve son olarak stateful set

**(PDF **523**)**

Stateful sets unique features

* Pods in a stateful set are numbered (from 0 to R-1) and ordered
* They are started and updated in order (from 0 to R-1)
* A pod is started (or updated) only when the previous one is **ready (yani master ayağa kalkmış olmalı)**
* They are stopped in reverse order (from R-1 to 0)
* Each pod know its identity (i.e. which number it is in the set) (herkez kendi numarasından kendini bilmebilmeli)
* Each pod can discover the IP address of the others easily (öncemli çünki birbilerini bulabilmeleri lazım. Mesela postgresql gibi)
* The pods can have persistent volumes attached to them

Wait a minute ... Can't we already attach volumes to pods and deployments?

**Volumes and Persistent Volumes**

* Volumes are used for many purposes:
  + sharing data between containers in a pod
  + exposing configuration information and secrets to containers
  + accessing storage systems

The last type of volumes is known as a "Persistent Volume"

**Persistent Volumes types**

* There are many types of Persistent Volumes available:
  + public cloud storage (GCEPersistentDisk, AWSElasticBlockStore, AzureDisk...)
  + private cloud storage (Cinder, VsphereVolume...)
  + traditional storage systems (NFS, iSCSI, FC...)
  + distributed storage (Ceph, Glusterfs, Portworx...)
* Using a persistent volume requires:
  + creating the volume out-of-band (outside of the Kubernetes API)
  + referencing the volume in the pod description, with all its parameters

cluod da tabi bu işi clud a vermek daha mantıklı.

Direkt olarak disk bağlıyorsak bu docker contaşner daki bind a denk geliyor. Eğer örneğin cloud dan bir path bağlıyorsak buda mıunt oluyor.

Buraya bi bak (Shortcomings of Persistent Volumes) **(PDF **528**)**

**Persistent Volume Claims**

To abstract the different types of storage, a pod can use a special volume type. Aslında örneğin azure storage ı direct olarak pod dakullanmıyoruz onu abstract etmiş oluyoruz. Aynı zamanda limit koyma, disk hızı belirleme işlerimizi de yapabiliyoruz.

What's in a Persistent Volume Claim? **(PDF **530**)**

Örnek

kind: PersistentVolumeClaim

apiVersion: v1

metadata:

name: my-claim

spec:

accessModes:

- ReadWriteOnce **(!!!!!)**

resources:

requests:

storage: 1Gi

yine tabii ki bunu da helme yazmak lazım. PersistentValumeClaim temolate oluşturup values.yaml e yazıyoruz.

**Using a Persistent Volume Claim**

Here is the same definition as earlier, but using a PVC:

apiVersion: v1

kind: Pod

metadata:

name: pod-using-a-claim

spec:

containers:

- image: ...

name: container-using-a-claim

volumeMounts:

- mountPath: /my-ebs

name: my-volume

volumes:

- name: my-volume

persistentVolumeClaim:

claimName: my-claim

Çok önemli

**Stateful sets in action**

Discovering the addresses of other pods **(PDF **538**)**

* consul-0.consul.default.svc.cluster.local
* consul-1.consul.default.svc.cluster.local
* consul-2.consul.default.svc.cluster.local

burada dikkat podlar 0,1,2 diye çağrılıyor ve arkasından consule dstatefulset in adı geliyor namespace olarak.

**Highly available Persistent Volumes**

Cheph gibi gluster gibi sistemler kullanılıyor. Ama şuan için hafif yazma ve okumlarda iyi ama çok çok hızlarda yavaş.

**Uygulama Notları**

Şunu takip ediyoruz.

https://github.com/muratcabuk/troubleshooting-k8s/tree/master/install

troubleshooting-k8s-master.zip zip de var dosyalar

**Kubernetes Installation Steps**

Assumptions

3 VMs pre-installed with Ubuntu 18.04.3 LTS, accessing internet (**ama aslında bizim örnekte 16.04 dü onun paket adresi xenial olduğu için kubernetes yüklendi 18.04 bionic olduğu için paroblem çıkabiliyor**)

All VMs should have at least 1 core, 2GB RAM and 10GB disk space

Steps

Install docker on the all the nodes (master-0, node-0, node-1). Use below set of commands.

sudo apt-get install -y \

apt-transport-https \

ca-certificates \

curl \

software-properties-common

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

sudo add-apt-repository \

"deb [arch=amd64] https://download.docker.com/linux/ubuntu \

$(lsb\_release -cs) \

stable edge"

sudo apt-get -y update

Install very specific version (18.06.1) of docker

sudo apt-get install -y docker-ce=18.06.1~ce~3-0~ubuntu

Install Kubernetes packages on all the nodes (master-0, node-0, node-1). Use below set of commands.

curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -

sudo apt-add-repository "deb http://apt.kubernetes.io/ kubernetes-$(lsb\_release -cs) main"

sudo apt-get -y update

Now install the packages.

sudo apt-get install -y kubernetes-cni=0.7.5-00

sudo apt-get install -y kubelet=1.15.3-00

sudo apt-get install -y kubectl=1.15.3-00

sudo apt-get install -y kubeadm=1.15.3-00

Make sure that the swap is disabled, disable it if enabled using below command. Note that this only sets the swap off until next reboot. Make sure that there are no swap entry in /etc/fstab file.

sudo swapoff -a

Run below command only on master-0 to initialize the Kubernetes cluster. (**development ortamında swap mode on olabilir. Çünki performans önemli değil. Developer ortamlarında swap alanı bol bol mesela ram in 2 katı ayarlanıp swapmode on yapılabilir.**)

Swapoff u kubelet zorunlu kılar.

Çalışan sistemede swapoff u kapatmak için bu komut kullanılır.

Peki swap açıkmı kapalımı nasıl anlarız komut satırına free yazarsak off mu değilmi görülür.

Sebebi ise performansın ölçülememesi

kubeadm init \

--pod-network-cidr=10.240.0.0/16 \

--apiserver-advertise-address=<internal\_ip\_of\_your\_master\_node> \

--ignore-preflight-errors=NumCPU

Yani

kubeadm init \

--pod-network-cidr=10.240.0.0/16 \

--apiserver-advertise-address= 10.240.0.4 \

--ignore-preflight-errors=NumCPU

Wait for initialization to complete, then run below commands

**Pod-network-cidr ise flunnal kullancaksak zorunlu kılıyor. Burada biz flunnal kullanmıyoruz ama defaultda bunu herhalükarda vermek lazım.**

**Apiserver-advertise-address buraya external yazarsak internetten erişilebilir hale geliyor.**

Master ip adresimi zi yazıyoruz. Ve bu master ip li versiyonunu sadece master da çalışıtırıyoruz.

Biraz vakit alıyor sonuçta bir key üretecek. Başta birçok warning çıkıyor takılma bekle.

Kubeadm bizim yerimize kubernetes için gerekli diğer komponentleri indiriyor. Kubelet i biz zaten kurmuşturk. Konktrol manager, etcd, ... vs

Komuttan sonra ekranda yazılanlar abakarsak bütün komponentler için conf doyalarının ve sertifiakalrının oluşturulduğunu görebiliriz.

[control-plane] Using manifest folder "/etc/kubernetes/manifests"

[control-plane] Creating static Pod manifest for "kube-apiserver"

[control-plane] Creating static Pod manifest for "kube-controller-manager"

[control-plane] Creating static Pod manifest for "kube-scheduler"

Bu satırlar daha kubernetes başlamadan gerekli olan komponentlerin pod larını buraya koyuyor. Biz herşeyi

Kubectl delete all –all komutuyla sisel bile kubelet buradan hangi node a ise oradan bu podları okuyorarak ayağa kaldırıyor.

pod manifestlerini kubelet yönetiyor. Manifest altına biz de kendi yaml dosyamızı koyarsak pod u silsek bile bu path dan kubelet hemen bu pod u tekrar ayağa kaldırır.

[mark-control-plane] Marking the node master-0 as control-plane by adding the label "node-role.kubernetes.io/master=''"

[mark-control-plane] Marking the node master-0 as control-plane by adding the taints [node-role.kubernetes.io/master:NoSchedule]

Buradaki kodla node u **tained** lemiş oluyor (lekelemek). Master noschedule olarak işaretledi. Ancak **tolerant** larda var yani noschedule a tolerant konulabilir. Mesela master komponentleri için böyle yapıldığı için master jobleri master üzerinde çaışabilir.

[addons] Applied essential addon: CoreDNS

[addons] Applied essential addon: kube-proxy (node lar in içinde servislerin pod lara erişimi için gerekli)

Bu satırla kubernetes a bir dns ve proxy yi addon ekliyor.

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

bu sayede kubectl ile bağlanma şansımız oluyor api ye. Mesela localden master sunucuya erişebilmek için bu doyayı local imize almamız gerekiyor üste sayfalarda örneği var.

Peki ekranı clear yaparsak ekrandaki key nasıl alacağız.

Kubeadm token create –print-join-command

Ekran da kubeadm join diye basşyan ve token bilgiside olan bu satırı worker larda çalıştırıyoruz.

Aşağıdaki kubeadm join diye başlayan örnek bir satır.

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

IMPORTANT

Make a note of the last line, should be like below. This command will be executed on worker nodes to join them to the cluster. Below is just an example.

kubeadm join 128.199.44.30:6443 --token l356h2.hywubufcxmtglx7s \

--discovery-token-ca-cert-hash sha256:633fbfab4015427e30327f3c5410fa698a1db7bf9d13f396b841b2de1e4e2987

Run the command you have taken note of from the previous step on nodes (node-0, node-1).

kubeadm join <internal\_ip\_of\_your\_master\_node> --token <token> \

--discovery-token-ca-cert-hash <hash>

After commands finish from previous step, go to the master node and issue below commands to observe the status.

You will see that the coredns pods are in Pending state while everything else is OK. Can you think of why?

Kubectl get nodes –A –o wide

Komutuu masterda çlıştırdığımızda coredns ler pending de görülüyor. Aslında bunlar notready oladuğunu göteriyor. Bunun bir sebebide weave veya flunnal gibibir networking plugin inin kurulu olmaması gibi düşüebiliriz ancak bu pod ları describe edersek node ların hazır olmadığını görebiliriz. Ayrıca nodeların ready tain inine takıldığını görebiliriz.

Yani node lar notready olduğu için coredns çalışmıyor.

Bu arada biz weave kullanıyoruz network plugini olarak.

Peki node lerin neden notready olup olmadığını nasıl anlarız

Kubectl describe node master-0

kubectl get nodes

kubectl get pods

kubectl get pods -A

kubectl get pods -A -o wide

We are missing a networking plugin, let's install Weave using below command.

Son olarak alttaki komutu çalıştırıyoruz. masterda

kubectl apply -f https://cloud.weave.works/k8s/net?k8s-version=1.15

After weave pods are created, situation should be better and everything should be in Running state.

Konrol için

kubectl get pods –A

pod manifestlerini kubelet yönetiyor.

Kurlumlar bittikten sonra helm i kur release den /var/usr/bin galiba exract et

Dah sonra helm den ingress kuruyorum.

Nginx-ingrees veya haproxy-engress kurabiliriz.

Bunu urdukran sonra locakdeki diyelimki we servismizi dışarı ingress ile açıyoruz. Belirli kurallara göre istekleri load balancer yapmış oluyoruz.

Yalnız nginx-ingress kurulduğunda –o wide ile bakarsak loadbalncer tipinde kurulduğunu görürüz. Bu işimize yaramaz. Bütün nginx-ingress helml lerini helm delete komutuyla silip kendi values.yaml doyamızı oluşturuyoruz. Tipini deamonset olarak kuruyoruz ve network olarakda host olarak aylarıruz. Böylece bu pod bütün node lara bir kez kuruluyor ve tüm netwokü host üzerinden görebiliyor.

Daha sonra bir ingress template buluyoruz. Bizde deploy/web servisini dışarı açmak itiyorduk onun için bir yaml bu aslında helm le alakası oyk. Yaml üzerinden ingress uyguluyoruz. Web için host ismi post rule lar (ingres aslında load balancer gibi olduğu için)

Nginx-ingress asında bir controller bu arada.

~~https olarak http-ingress i off load yapıp içeride http kullanabiliriz. Bunun için nginx-ingress e CA sertifika, private key, vs koymak lazım. Bunları kubernetes da secret a (configmap a) koyuyoruz.~~

bunun için cert manager plugin kuruyoruz. Çünki nginx-ingress bir controller ve belki yüzlerce site için kullanılacak. Tek sitemiz yokki configmap işimize yaramaz.

Bütün kubernetes in kubeadm siz kurulumunun yapılmasını anlatan git

<https://github.com/kelseyhightower/kubernetes-the-hard-way>

bunu bu arada en az bir kez yapmak lazım.

Rancher diye bir tool var. Kubernetes i de kuruyor.

Ve kubernetes yönetimini bu arayüz üzerinden kurabiliyoruz.