Değişkenlerimizi hassas ise seDeğişkenlerimizi hassas ise secret ile değilse configmap ile tutuyoruz.

Hassas bilgileri secret objesi tutuyor, bu sayede güvenli oluyor ve cluster içinde dolaştırabiliyoruz. Veri tuttuğu için volum type olarak geçiyor, kubernetes in temel objelerindendir.

Built-in Type 
Opaque is the default Secret type 
kubernetes. io/seNice-account -token 
if omitted from a Secret 
configuration file. When you 
ku 10/ dockercfg 
create a Secret using kubectl, you 
will use the generic subcommand 
-auth 
to indicate an Opaque Secret 
kubernetes. io/ssh-aueh 
type. 
io/toke 
arbitrary user-defined data 
ServiceAccount to ken 
serialized -dockercfg file 
serialized —%docker/config. "son file 
credentials for basic authentication 
credentials for SSH authentication 
data for a TLS client or sewer 
bootstrap token data 

Default secret tipi opawue tır. Diğerleri de secret tipleridir. Dockerconfigjson u docker crederntials için kullanıyoruz, cpastone da kullanacağız.

Ssh bağlantı bilgilerini ssh-auth ta kullanabiliyoruz. generic in kullanımını handsonda göreceğiz.

komut satırından da, yaml dosyasından da secret oluşturabiliyoruz.

K8s Master 
Secrets can be mounted as 
data volumes or exposed as 
environment variables to be 
used by a container in a Pod. 

secret ı pod a env olarak ya da volume olarak iki türlü bağlayabiliriz.

Hassas olmayan ve env şeklindeki bilgileri (örn db ismi) configmap ile key-value şeklinde tutuyoruz.

ConfigMaps 
A ConfigMap is an API object used to 
store non-confidential data in 
key-value pairs. Pods can consume 
DEV 
TEST 
PROD 
ConfigMaps 
variables, 
environment 
as 
command-line 
arguments, or asl configuration files 
in a volume. 

farklı ns için configmap ler oluşturulmuş, daha sonra clusterda bu configmap kullanılabiliyor.

o EV-NAMESPACE : 
A ConfigMap allows you to 
decouple environment-specific 
my-app 
configuration from your container 
images, so that your applications 
are easily portable. 
CONFIGMAP 
TEST*' 
. Environn•.ent Specific Cor@rauonsL• 
my-a A) 
no-con 

configmap te tutulan örneğin db ismini farklı ns ler için farklı isimlerle kullanabilmek için değişikliği sadece configmapte yapmamız yetecektir

secret bilgi tutarken base64 le encode ederek saklar. base64 aslında ekran koruyucusu gibidir, ekranda okunmamasını sağlar. encode için çok daha farklı yöntemler kullanılıyor.

"Secret" 
1385guD+qpvKOjLt 
zrucscMk8qqüQk 
*YINy2E04ttgRT 
yblRn4u7QiUDyPQP 
xw3LhTaHzYJsn/ p 
Bcw34m9KnHykN 
kRMCn8GOHvcaj8 
vars 
"Config map" 
Confor.up 
ran-e: srcial-config 
defauR 
SPECIAL LEVEL: S 

Confi Ma s Sa ples 
User can create configMap via Literal or from Files. 
1 
Via File: A path to a directory containing one or more configuration 
files, indicated using the —from-file flag. 
kubectl create configmap th0ÄMÉj;--from-file [/PATH/TO/FILE.PROPERTIES] 
--from-file [/PATH/TO/FlLE2.PROPERTIES] 
User can also put complete directory, containing multiple files. 
kubectl create configmap [NAME] --from-file [/PATH/TO/DIRECTORY] 

configmap dosyadan klasörden de oluşturuluyor.

Confi Ma s Sa ples 
Via Literal Values: To create a ConfigMap from literal values 
—from-literal. 
kubectl create configmap literal-data rfrom-literalkeyl=valuel 
from-literal key2=va/ue2 
kubectl create configmap special config 
- -from- literal—special . type—charm 
Get ConfigMap via CLI. 
kubectl get configmaps <confifig-map_Name> -o yaml/json 

form literal diyerek komut satırında da env tanımayabiliyoruz.

imperative 
(cli) 
declarative 
(yaml) 
key-value 
file/directory 
key-value 
file/directory 
env. var. 
volume 
env. var. 
volume 

önce user ve password bilgilerini içeren iki farklı dosya oluşturup bunları secret ta tutacaz.

handsona gçeiyoruz, dosyaları oluşturalım:

bash 
# Create 
echo -n 
echo -n 
files needed for the rest of the example . 
' admin' > ./username . txt 
'If2d1e2e67df' > 
. / pas sword . txt 

-n alt satıra geçmemesi için yazılıyor, password lerde genelde hata almamak için kullanılır.

biz büyük projede docker image ları Amazon ECR da tutacaz, ECR dockerhub benzeri bir repo ancak özel bir repo.

ECR dan direk image ı çekmek için docker run komutu ile bu image ın tam ismini de yazarsak çekmez. credentials gerekir, AWS ye giriş bilgilerini ister.

bu credentials ı secret a yazarız, deploymant a env tanımlayıp birebir yazmıyoruz. secret sayesinde hem bütün podlar çekebiliyor hem de güvenli tutulmuş oluyor.

• kubectl create secret —help 
Create a secret using specified subcownand . 
Available Cotrnands: 
docker-registry Create a secret for use with a Docker registry 
geneü•ic 
Create a secret from a local file, directory, or literal value 
Create a TLS secret 
Usage: 
kubectl create secret [flags] (options] 
Use "kubectl --help" for more information about a given connand. 
Use "kubeetl options" for' a list of global connand-line options (applies to all com-ands)_ 

kubectl create secret --help deyince neden generic kullandığımıza dair açıklama geliyor:

lokal bir dosya ya da klasör ya da literal değer ile secret oluşturacaksak generic kullanmamız gerekiyor.

kubectl create secret generic --help yazarsak da generic ile nasıl yazabildiğimizi gösterir.

kubectl create secret generic db-user-pass --from-file=./username.txt --from-file=./password.txt

komutuyla username.txt ve password.txt den çekmesini istiyoruz.

• kubectl create secret generic db-user-pas: 
secret/db-user-pass created 
• k get secret 
TYPE 
DATA AGE 
db-user-pass 
Opaque 2 
IIS 

secret ın kıslatması yoktur. k get secret ile görelim. opaque tip defaulttur.

yine dosyadan alıp secret oluşturmasını isteyecez ancak bu kez key-valu şeklinde tanımlayacaz.

kubectl create secret generic db-user-pass-key --from-file=username=./username.txt --from-file=password=./password.txt

burada araya username ekledik. eğer bunu yazmazsak key olarak dosya ismini yani username.txt yi alır. ancak key i bu şekilde kendimiz de yazabiliyoruz. value de git dosyadan al demiş oluyoruz.

yani env variable ları secret olarak gömüyoruz.

kubectl create secret generic dev-db-secret --from-literal=username=devuser --from-literal=password='S!B\\*d$zDsb='

burada da dosyadan al demedik from literal ile komut satırında tanımlıyoruz. burada dikkat etmemiz gereken konu şu;

password lerde linux ta reserve edilmiş wildcard karakterler olan $, !, gibi karakterler bulunabiliyor bu nedenle password u yazarken '' içinde yazmamız gerekiyor.

• kubectl describe secrets/db-user-pass 
db-user-pass 
Name : 
default 
Namespace: 
Labels : 
<none> 
Annotations : 
<none> 
Type: Opaque 
Data 
password . txt : 
username. txt : 
12 bytes 
5 bytes 

describe ile secret ın ismini girip detay istediğimizde value leri göstermediğini görürüz. çünkü gizil bilgidir. yaml formatını isteyelim:

• k get secret db-user-pass 
-o yaml 
apiVersion: VI 
data: 
password . txt : 
username. txt : 
kind: Secret 
metadata : 
FUYyZDF1mmU2N2Rm 
YWRtaW4= 
creation Timestamp: 
name: db-user-pass 
namespace: default 
resourceVersion : ' 
116140" 
uid: acc6ab9d-36fb-4bbø-946f-7Ø116b72e151 
type: Opaque 

burada da base64 ile kodlanmış halini gösteriyor value kısmında.

komut satırında secret oluştururken from literal dedik yazdık, devamında from file diyerek dosyadan başka veriler çekmesini de isteyebiliriz.

kendimiz yaml manfifest dosyasıyla da secret oluşturabiliriz ancak bunun için linuz te base64 ile encode ediyoruz:

echo -n 'admin' | base64

• echo -n ' admin' 
YWRtaW4= 
I base64 

base64 encode eder ancak encryption configuration ile encrypte de edilebilir. ancak asıl güvenlik önlemleri poilcylerle vb. alınıyor.

apiVersion: v1

kind: Secret

metadata:

name: mysecret

type: Opaque

data:

username: YWRtaW4=

password: MWYyZDFlMmU2N2Rm

bu secret oluşturmak için manifest yaml dosyası. data: altında keylerin karşılığına value leri direk yazamıyoruz base64 ile encode ettiğimizde aldığımız değerleri yazıyoruz.

G) README.mdX ! 
secret.yaml X 
secret.yaml > @ apiVersion 
1 
2 
3 
5 
6 
7 
8 
io.k8s.api.core.v1 .Secret (VI @secret.json) 
apiVersion: VI 
kind: Secret 
metadata : 
name: mysecret 
type: Opaque 
data: 
username: WRtaW4= 
password: WYyZDF1mmU2N2Rm 

secret.yaml dosyasını oluşturduk.

apply ile secret ı oluşturuyoruz.

base64 ü decode etmek için:

• ubuntu@kube-master : -„$ echo "WYyZDFIMmU2N2Rm" I base64 
-decode 

--dry-run kubernetes te hata varsa göstermesi için bir koddur ancak biz burada komut satırındaki env leri dosyaya yazdırması için de kullanabiliyoruz:

kubectl create secret generic my-secret --from-literal=username=devuser --from-literal=assword='1f2d1e2e67df' --dry-run=client -o yaml > mysecrets.yaml

O README.md 
mysecretsyam( 1 
mysecrets.yaml > apiVersion 
1 
2 
3 
5 
6 
7 
8 
io.k8s.api.core.v1 .Secret (VI @secret.json) 
apiVersion: VI 
data: 
assword: PMyZDFIPW2N2Rm 
username: ZGV2dXN1cg=— 
kind: Secret 
metadata : 
creationTimestamp: null 
name: my-secret 

ya da deploy için:

kubectl create deploy mydeploy --image=nginx --dry-run=client -o yaml > mydeploy.yaml

şimdi ise pod içinde secret nasıl kullanılıyor ona bakalım:

apiVersion: v1

kind: Pod

metadata:

name: secret-env-pod

spec:

containers:

- name: mycontainer

image: redis

env:

- name: SECRET\_USERNAME

value: admin

- name: SECRET\_PASSWORD

value: 1f2d1e2e67df

restartPolicy: Never

yaml dosyasında bu şekilde env yazabiliyoruz normalde.

apply ile oluşturalım ve pod un içine girip bakalım:

SECRET USERNAME-admin 
GOSU VERSION-1.16 
RE =http://download.redi! 
SHLVL=I 
KUBERNETES PORT 443 TCP AD0R=1e.96.e.1 
KUBERNETES SERVICE msT-1ø.96.e.1 
96 _ e _ 
KUBERNETES 443 TCP 
PATH=/usr/ local/sbin: /usr/local/bin : /usl 
SECRET 
_=/usr/bin/ env 
exit 
exit 

linux komutlarıyle env variable ları görebiliyoruz.

şimdi pod manifestinde env value lerini secret tan çekmesi için şu şekilde yazıyoruz:

env:

- name: SECRET\_USERNAME

valueFrom:

secretKeyRef:

name: mysecret

key: username

- name: SECRET\_PASSWORD

valueFrom:

secretKeyRef:

name: mysecret

key: password

restartPolicy: Never

-name: i manuel olarak girdik, valueFrom ile secretKeyRef ile mysecret isimli tan username ya da password keylerinin değerini çekmesini istedik.

bu değişiklikleri yaptıktan sonra apply diyelim ve içinde env leri görebileceğimize bakalım:

• kubectl exec 
-it secret-env-pod 
env 
PATH=/usr/10ca1/ sbin : /usr/local/bin: /usr/sbin: /usr/bin: /sbin: /bin 
FOSTNAME=secret -env-pod 
GOSU VERSION-I. 16 
REDIS VERSION=7.Ø.1Ø 
REDIS DOWNLOAD URL=http://down10ad.redis . io/re1eases/redis-7.Ø.1Ø.tE 
REDIS DOWNLOAD 
SECRET USERNAME=admin 
SECRET PASSWORD=1f2d1e2e67df 

secret ı volume olarak da pod da yazabiliriz, configmap te bunu yapacaz.

configmap handsonunda Hello,Clarusway yazdıran image ler kullanacağız.

k8s isimli folder oluşturuyoruz.

apiVersion: apps/v1

kind: Deployment

metadata:

name: demo

labels:

app: demo

spec:

replicas: 1

selector:

matchLabels:

app: demo

template:

metadata:

labels:

app: demo

spec:

containers:

- name: demo

image: clarusway/demo:hello

ports:

- containerPort: 8888

içerisinde deployment.yaml dosyamızı oluşturuyoruz.

bir de service.yaml oluşturuyoruz:

apiVersion: v1

kind: Service

metadata:

name: demo-service

labels:

app: demo

spec:

type: NodePort

ports:

- port: 80

targetPort: 8888

nodePort: 30001

selector:

app: demo

k8s içinde apply diyoruz.

curl ile ya da master ya da worker ın public ip sini port numarasıyla browser a yazarak uygulamayı alabiliriz.

C localhost:30001 
AWS Management... Course AWS 1 3-TU... 
Merhaba, CIE rusway ! 

deployment ı modifiye ediyoruz, image değiştirdik.

containers:

- name: demo

image: clarusway/demo:hello-config-env

ports:

- containerPort: 8888

env:

- name: GREETING

value: selam

image i yazarken Hello yerine GREETING değişkeni tanımlanmış, burada oraya hangi value yazarsak onu görürüz.

docker container run -d -p 80:8888 clarusway/demo:hello-config-env

komutuyla docker da çalıştıralım.

A 54.82.12.u 
, Clarusway ! 

image bu şekilde.

docker komutuyla değişken yazmak için:

docker container run -d -p 80:8888 -e GREETING=selam clarusway/demo:hello-config-env

komutunu kullanırız.

şimdi objeleri sileilm ve configmap oluşturalım.

465 
466 
467 
468 
469 
1. 
2. 
3. 
Use 
Use 
the contents of an entire directory with kubectl create configmap my-config 
-from-file= . /my /dir/path/ 
the contents of a file or specific set of files with 
my-config - -from-file=./my/file. txt- 
kubectl create configmap 
Use literal key-value pairs defined on the command line with 
my-config --from-literal=keyl=valuel 
kubectl create configmap 

bunlar komut satırında yazarak create etme yöntemleridir. manifest yaml ile de oluşturulabilir.

• kubectl create configmap demo-config 
configmap/demo-config created 
• k get cm 
DATA AGE 
demo-config 
kube-root -ca . crt 
1 
1 

from literal ile komut satırında cm oluşturduk.

describe ile value lerin yazdığını görüyoruz:

• k get cm demo-config 
apiVersion: VI 
data: 
GREETING: hola 
kind: ConfigMap 
metadata : 
creation Timestamp: 
name: demo-config 
namespace: default 
resourceVersion : ' 
122497" 
uid: deb69b47-fb16-467a-8øc1-a6eØ2b45492a 
-o yaml 

şimdi manifest dosyasıyla comfigmap oluşturalım.

apiVersion: v1

kind: ConfigMap

metadata:

name: demo-config

data:

greeting: Hola

deployment.yaml da modify yapıyoruz:

env:

- name: GREETING

valueFrom:

configMapKeyRef:

name: demo-config

key: greeting

env kısmını modifie ettik.

üstteki GREETING ile env i tanımlıyoruz. aşağıda ise configmap ten greeting key in değerini al demiş oluyoruz.

get cm, pod, deploy, svc 
configmap/demo-config 
configmap/kube-root-ca . crt 
pod/ demo-7dcc8bd9cf-tbpks 
DATA 
1 
1 
READY 
1/1 
1/1 
1/1 
AGE 
58s 
STATUS 
Running 
Running 
Running 
RESTARTS 
3 (8h ago) 
pod/nginx-pod 
pod/secret-env-pod 
deployment . apps / demo 
service/demo-service 
service/kubernetes 
AGE 
58s 
46h 
45m 
READY 
1/1 
TYPE 
UP-TO-DATE AVAILABLE 
1 
AGE 
58s 
1 
CLUSTER-IP 
10. 105.224. 
10.96.ø.1 
61 
EXTERNAL 
<none> 
<none> 
PORT(S) 
80 : 30001/TCP 
443/TCP 
AGE 
58s 
NodePort 
ClusterIP 

C localhost:30001 
AWS Management... Course AWS 1 3-TU. 
Bola, Clarusvvay 

configmap teki value yu değiştirip apply ettiğimizde deploymentta değişikliği göremiyoruz, pod un restart olması gerekiyor görebilmek için.

şimdi configmap te greeting i büyük yazarsak POSIX format deniyor buna kubernetes bunun env variable olduğunu anlıyor:

apiVersion: v1

kind: ConfigMap

metadata:

name: demo-config

data:

GREETING: Merhaba

deployment ta env: yerine envFrom diyoruz:

spec:

containers:

- name: demo

image: clarusway/demo:hello-config-env

ports:

- containerPort: 8888

envFrom:

- configMapRef:

name: demo-config

envFrom dediğimizde ise key girmemize gerek kalmıyor çünkü configmap te büyük harflerle POSIX formatında yazınca onun key olduğunu anlıyor.

bir configmap objesinde birden fazla env variable ları almak için onları da deployment ta yazıyoruz. birdne fazla env yazacağımız zaman da POSIX formatı işimizi kolaylaştırıyor.

ne kadar key-value varsa envFrom ile hepsini alır.

şimdi de configmap te dosyadan nasıl çekebiliriz onu görelim:

k8s içinde config diye bir dosya oluşturduk:

echo "greeting: Hei" > config

kubectl create configmap demo-config --from-file=./config

• kubectl create configmap demo-config 
configmap/demo-config created 
- -from-file=. / config 

bu komutla config file dan al demiş olduk.

• k get cm demo-config 
-o Json 
"apiVersion 
"VI", 
"data " : 
"config" • "greeting: 
Hei\n" 
"kind": "ConfigMap" , 
"metadata " : 
"creation Timestamp 
"name": "demo-config" , 
"namespace": "default" , 
"resourceVersion 
"125771", 
"uid" "b681d21ø-7f7d-4f09-8c35-a7føae29c49e" 

burada dosya ismi key gibi oldu: config: greeting: Hei

bu cm yi silelim ve yaml dosyası ile oluşturalım:

apiVersion: v1

kind: ConfigMap

metadata:

name: demo-config

data:

config: |

greeting: Buongiorno

data: altında bu kez key olarak config girdik, yukarıda söylediğimiz gibi burada key dosya ismi oldu.

burada | işaretini multiline gibi düşünelim.

daha fazla key value eklemek istiyorsak buradaki yöntemi budur.

simdi deployment ı da modifie etmemiz gerekiyor. burada configmap i volume e bağlayalım.

apiVersion: apps/v1

kind: Deployment

metadata:

name: demo

spec:

replicas: 1

selector:

matchLabels:

app: demo

template:

metadata:

labels:

app: demo

spec:

containers:

- name: demo

image: clarusway/demo:hello-config-file

ports:

- containerPort: 8888

volumeMounts:

- mountPath: /config/

name: demo-config-volume

readOnly: true

volumes:

- name: demo-config-volume

configMap:

name: demo-config

items:

- key: config

path: demo.yaml

bir volume oluştur, ismi demo-config-volume olacak.

configMap ismini giriyoruz. altında items parametresi geldi. burada key config di zaten onu yazdık. path geldi, config teki valueyi al volume bağladığımız yerde demo.yaml oluştur ve value yi oraya yaz.

VolumeMounts: altında ise mounthpath te belirlenen config de container içinde bir config klasörü oluştur ve buraya mount et demek.

apply dedik uygulamayı gördük:

C A Güvenli deéil 54.196.0.48:30001 
AWS Management... Course AWS 13-TU... Google 
Buongiorno, Clarusvvay ! 

pod a bağlanalım:

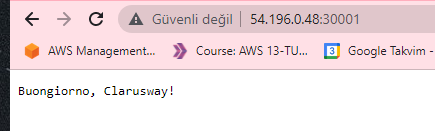
o k exec -it 
/ # Is 
demo-5bf545f76-vwz76 
bin 
dev 
horne 
lib 
media 
mnt 
opt 
proc 
root 
run 
sbin 
config etc 
sys 
tmp 
sh 
usr 
var 

config isimli bir klasör oluşturmuş.

/ # cd config/ 
/config # Is 
demo. yaml 
/config # cat demo.yaml 
greeting: Buongiorno 
/config # 

config içinde demo.yaml ı oluşturmuş onun içine de key ve value mizi yazdırmış.

bu örnekteki deployment ta image ın da değiştiğini gözden kaçırmayalım. image da yapılan conf ile



bu çıktıyı görüyoruz, bu çıktıyı demo.yaml dan çekiyor.

cret ile değilse configmap ile tutuyoruz.

Hassas bilgileri secret objesi tutuyor, bu sayede güvenli oluyor ve cluster içinde dolaştırabiliyoruz. Veri tuttuğu için volum type olarak geçiyor, kubernetes in temel objelerindendir.

Built-in Type 
Opaque is the default Secret type 
kubernetes. io/seNice-account -token 
if omitted from a Secret 
configuration file. When you 
ku 10/ dockercfg 
create a Secret using kubectl, you 
will use the generic subcommand 
-auth 
to indicate an Opaque Secret 
kubernetes. io/ssh-aueh 
type. 
io/toke 
arbitrary user-defined data 
ServiceAccount to ken 
serialized -dockercfg file 
serialized —%docker/config. "son file 
credentials for basic authentication 
credentials for SSH authentication 
data for a TLS client or sewer 
bootstrap token data 

Default secret tipi opawue tır. Diğerleri de secret tipleridir. Dockerconfigjson u docker crederntials için kullanıyoruz, cpastone da kullanacağız.

Ssh bağlantı bilgilerini ssh-auth ta kullanabiliyoruz. generic in kullanımını handsonda göreceğiz.

komut satırından da, yaml dosyasından da secret oluşturabiliyoruz.

K8s Master 
Secrets can be mounted as 
data volumes or exposed as 
environment variables to be 
used by a container in a Pod. 

secret ı pod a env olarak ya da volume olarak iki türlü bağlayabiliriz.

Hassas olmayan ve env şeklindeki bilgileri (örn db ismi) configmap ile key-value şeklinde tutuyoruz.

ConfigMaps 
A ConfigMap is an API object used to 
store non-confidential data in 
key-value pairs. Pods can consume 
DEV 
TEST 
PROD 
ConfigMaps 
variables, 
environment 
as 
command-line 
arguments, or asl configuration files 
in a volume. 

farklı ns için configmap ler oluşturulmuş, daha sonra clusterda bu configmap kullanılabiliyor.

o EV-NAMESPACE : 
A ConfigMap allows you to 
decouple environment-specific 
my-app 
configuration from your container 
images, so that your applications 
are easily portable. 
CONFIGMAP 
TEST*' 
. Environn•.ent Specific Cor@rauonsL• 
my-a A) 
no-con 

configmap te tutulan örneğin db ismini farklı ns ler için farklı isimlerle kullanabilmek için değişikliği sadece configmapte yapmamız yetecektir

secret bilgi tutarken base64 le encode ederek saklar. base64 aslında ekran koruyucusu gibidir, ekranda okunmamasını sağlar. encode için çok daha farklı yöntemler kullanılıyor.

"Secret" 
1385guD+qpvKOjLt 
zrucscMk8qqüQk 
*YINy2E04ttgRT 
yblRn4u7QiUDyPQP 
xw3LhTaHzYJsn/ p 
Bcw34m9KnHykN 
kRMCn8GOHvcaj8 
vars 
"Config map" 
Confor.up 
ran-e: srcial-config 
defauR 
SPECIAL LEVEL: S 

Confi Ma s Sa ples 
User can create configMap via Literal or from Files. 
1 
Via File: A path to a directory containing one or more configuration 
files, indicated using the —from-file flag. 
kubectl create configmap th0ÄMÉj;--from-file [/PATH/TO/FILE.PROPERTIES] 
--from-file [/PATH/TO/FlLE2.PROPERTIES] 
User can also put complete directory, containing multiple files. 
kubectl create configmap [NAME] --from-file [/PATH/TO/DIRECTORY] 

configmap dosyadan klasörden de oluşturuluyor.

Confi Ma s Sa ples 
Via Literal Values: To create a ConfigMap from literal values 
—from-literal. 
kubectl create configmap literal-data rfrom-literalkeyl=valuel 
from-literal key2=va/ue2 
kubectl create configmap special config 
- -from- literal—special . type—charm 
Get ConfigMap via CLI. 
kubectl get configmaps <confifig-map_Name> -o yaml/json 

form literal diyerek komut satırında da env tanımayabiliyoruz.

imperative 
(cli) 
declarative 
(yaml) 
key-value 
file/directory 
key-value 
file/directory 
env. var. 
volume 
env. var. 
volume 

önce user ve password bilgilerini içeren iki farklı dosya oluşturup bunları secret ta tutacaz.

handsona gçeiyoruz, dosyaları oluşturalım:

bash 
# Create 
echo -n 
echo -n 
files needed for the rest of the example . 
' admin' > ./username . txt 
'If2d1e2e67df' > 
. / pas sword . txt 

-n alt satıra geçmemesi için yazılıyor, password lerde genelde hata almamak için kullanılır.

biz büyük projede docker image ları Amazon ECR da tutacaz, ECR dockerhub benzeri bir repo ancak özel bir repo.

ECR dan direk image ı çekmek için docker run komutu ile bu image ın tam ismini de yazarsak çekmez. credentials gerekir, AWS ye giriş bilgilerini ister.

bu credentials ı secret a yazarız, deploymant a env tanımlayıp birebir yazmıyoruz. secret sayesinde hem bütün podlar çekebiliyor hem de güvenli tutulmuş oluyor.

• kubectl create secret —help 
Create a secret using specified subcownand . 
Available Cotrnands: 
docker-registry Create a secret for use with a Docker registry 
geneü•ic 
Create a secret from a local file, directory, or literal value 
Create a TLS secret 
Usage: 
kubectl create secret [flags] (options] 
Use "kubectl --help" for more information about a given connand. 
Use "kubeetl options" for' a list of global connand-line options (applies to all com-ands)_ 

kubectl create secret --help deyince neden generic kullandığımıza dair açıklama geliyor:

lokal bir dosya ya da klasör ya da literal değer ile secret oluşturacaksak generic kullanmamız gerekiyor.

kubectl create secret generic --help yazarsak da generic ile nasıl yazabildiğimizi gösterir.

kubectl create secret generic db-user-pass --from-file=./username.txt --from-file=./password.txt

komutuyla username.txt ve password.txt den çekmesini istiyoruz.

• kubectl create secret generic db-user-pas: 
secret/db-user-pass created 
• k get secret 
TYPE 
DATA AGE 
db-user-pass 
Opaque 2 
IIS 

secret ın kıslatması yoktur. k get secret ile görelim. opaque tip defaulttur.

yine dosyadan alıp secret oluşturmasını isteyecez ancak bu kez key-valu şeklinde tanımlayacaz.

kubectl create secret generic db-user-pass-key --from-file=username=./username.txt --from-file=password=./password.txt

burada araya username ekledik. eğer bunu yazmazsak key olarak dosya ismini yani username.txt yi alır. ancak key i bu şekilde kendimiz de yazabiliyoruz. value de git dosyadan al demiş oluyoruz.

yani env variable ları secret olarak gömüyoruz.

kubectl create secret generic dev-db-secret --from-literal=username=devuser --from-literal=password='S!B\\*d$zDsb='

burada da dosyadan al demedik from literal ile komut satırında tanımlıyoruz. burada dikkat etmemiz gereken konu şu;

password lerde linux ta reserve edilmiş wildcard karakterler olan $, !, gibi karakterler bulunabiliyor bu nedenle password u yazarken '' içinde yazmamız gerekiyor.

• kubectl describe secrets/db-user-pass 
db-user-pass 
Name : 
default 
Namespace: 
Labels : 
<none> 
Annotations : 
<none> 
Type: Opaque 
Data 
password . txt : 
username. txt : 
12 bytes 
5 bytes 

describe ile secret ın ismini girip detay istediğimizde value leri göstermediğini görürüz. çünkü gizil bilgidir. yaml formatını isteyelim:

• k get secret db-user-pass 
-o yaml 
apiVersion: VI 
data: 
password . txt : 
username. txt : 
kind: Secret 
metadata : 
FUYyZDF1mmU2N2Rm 
YWRtaW4= 
creation Timestamp: 
name: db-user-pass 
namespace: default 
resourceVersion : ' 
116140" 
uid: acc6ab9d-36fb-4bbø-946f-7Ø116b72e151 
type: Opaque 

burada da base64 ile kodlanmış halini gösteriyor value kısmında.

komut satırında secret oluştururken from literal dedik yazdık, devamında from file diyerek dosyadan başka veriler çekmesini de isteyebiliriz.

kendimiz yaml manfifest dosyasıyla da secret oluşturabiliriz ancak bunun için linuz te base64 ile encode ediyoruz:

echo -n 'admin' | base64

• echo -n ' admin' 
YWRtaW4= 
I base64 

base64 encode eder ancak encryption configuration ile encrypte de edilebilir. ancak asıl güvenlik önlemleri poilcylerle vb. alınıyor.

apiVersion: v1

kind: Secret

metadata:

name: mysecret

type: Opaque

data:

username: YWRtaW4=

password: MWYyZDFlMmU2N2Rm

bu secret oluşturmak için manifest yaml dosyası. data: altında keylerin karşılığına value leri direk yazamıyoruz base64 ile encode ettiğimizde aldığımız değerleri yazıyoruz.

G) README.mdX ! 
secret.yaml X 
secret.yaml > @ apiVersion 
1 
2 
3 
5 
6 
7 
8 
io.k8s.api.core.v1 .Secret (VI @secret.json) 
apiVersion: VI 
kind: Secret 
metadata : 
name: mysecret 
type: Opaque 
data: 
username: WRtaW4= 
password: WYyZDF1mmU2N2Rm 

secret.yaml dosyasını oluşturduk.

apply ile secret ı oluşturuyoruz.

base64 ü decode etmek için:

• ubuntu@kube-master : -„$ echo "WYyZDFIMmU2N2Rm" I base64 
-decode 

--dry-run kubernetes te hata varsa göstermesi için bir koddur ancak biz burada komut satırındaki env leri dosyaya yazdırması için de kullanabiliyoruz:

kubectl create secret generic my-secret --from-literal=username=devuser --from-literal=assword='1f2d1e2e67df' --dry-run=client -o yaml > mysecrets.yaml

O README.md 
mysecretsyam( 1 
mysecrets.yaml > apiVersion 
1 
2 
3 
5 
6 
7 
8 
io.k8s.api.core.v1 .Secret (VI @secret.json) 
apiVersion: VI 
data: 
assword: PMyZDFIPW2N2Rm 
username: ZGV2dXN1cg=— 
kind: Secret 
metadata : 
creationTimestamp: null 
name: my-secret 

ya da deploy için:

kubectl create deploy mydeploy --image=nginx --dry-run=client -o yaml > mydeploy.yaml

şimdi ise pod içinde secret nasıl kullanılıyor ona bakalım:

apiVersion: v1

kind: Pod

metadata:

name: secret-env-pod

spec:

containers:

- name: mycontainer

image: redis

env:

- name: SECRET\_USERNAME

value: admin

- name: SECRET\_PASSWORD

value: 1f2d1e2e67df

restartPolicy: Never

yaml dosyasında bu şekilde env yazabiliyoruz normalde.

apply ile oluşturalım ve pod un içine girip bakalım:

SECRET USERNAME-admin 
GOSU VERSION-1.16 
RE =http://download.redi! 
SHLVL=I 
KUBERNETES PORT 443 TCP AD0R=1e.96.e.1 
KUBERNETES SERVICE msT-1ø.96.e.1 
96 _ e _ 
KUBERNETES 443 TCP 
PATH=/usr/ local/sbin: /usr/local/bin : /usl 
SECRET 
_=/usr/bin/ env 
exit 
exit 

linux komutlarıyle env variable ları görebiliyoruz.

şimdi pod manifestinde env value lerini secret tan çekmesi için şu şekilde yazıyoruz:

env:

- name: SECRET\_USERNAME

valueFrom:

secretKeyRef:

name: mysecret

key: username

- name: SECRET\_PASSWORD

valueFrom:

secretKeyRef:

name: mysecret

key: password

restartPolicy: Never

-name: i manuel olarak girdik, valueFrom ile secretKeyRef ile mysecret isimli tan username ya da password keylerinin değerini çekmesini istedik.

bu değişiklikleri yaptıktan sonra apply diyelim ve içinde env leri görebileceğimize bakalım:

• kubectl exec 
-it secret-env-pod 
env 
PATH=/usr/10ca1/ sbin : /usr/local/bin: /usr/sbin: /usr/bin: /sbin: /bin 
FOSTNAME=secret -env-pod 
GOSU VERSION-I. 16 
REDIS VERSION=7.Ø.1Ø 
REDIS DOWNLOAD URL=http://down10ad.redis . io/re1eases/redis-7.Ø.1Ø.tE 
REDIS DOWNLOAD 
SECRET USERNAME=admin 
SECRET PASSWORD=1f2d1e2e67df 

secret ı volume olarak da pod da yazabiliriz, configmap te bunu yapacaz.

configmap handsonunda Hello,Clarusway yazdıran image ler kullanacağız.

k8s isimli folder oluşturuyoruz.

apiVersion: apps/v1

kind: Deployment

metadata:

name: demo

labels:

app: demo

spec:

replicas: 1

selector:

matchLabels:

app: demo

template:

metadata:

labels:

app: demo

spec:

containers:

- name: demo

image: clarusway/demo:hello

ports:

- containerPort: 8888

içerisinde deployment.yaml dosyamızı oluşturuyoruz.

bir de service.yaml oluşturuyoruz:

apiVersion: v1

kind: Service

metadata:

name: demo-service

labels:

app: demo

spec:

type: NodePort

ports:

- port: 80

targetPort: 8888

nodePort: 30001

selector:

app: demo

k8s içinde apply diyoruz.

curl ile ya da master ya da worker ın public ip sini port numarasıyla browser a yazarak uygulamayı alabiliriz.

C localhost:30001 
AWS Management... Course AWS 1 3-TU... 
Merhaba, CIE rusway ! 

deployment ı modifiye ediyoruz, image değiştirdik.

containers:

- name: demo

image: clarusway/demo:hello-config-env

ports:

- containerPort: 8888

env:

- name: GREETING

value: selam

image i yazarken Hello yerine GREETING değişkeni tanımlanmış, burada oraya hangi value yazarsak onu görürüz.

docker container run -d -p 80:8888 clarusway/demo:hello-config-env

komutuyla docker da çalıştıralım.

A 54.82.12.u 
, Clarusway ! 

image bu şekilde.

docker komutuyla değişken yazmak için:

docker container run -d -p 80:8888 -e GREETING=selam clarusway/demo:hello-config-env

komutunu kullanırız.

şimdi objeleri sileilm ve configmap oluşturalım.

465 
466 
467 
468 
469 
1. 
2. 
3. 
Use 
Use 
the contents of an entire directory with kubectl create configmap my-config 
-from-file= . /my /dir/path/ 
the contents of a file or specific set of files with 
my-config - -from-file=./my/file. txt- 
kubectl create configmap 
Use literal key-value pairs defined on the command line with 
my-config --from-literal=keyl=valuel 
kubectl create configmap 

bunlar komut satırında yazarak create etme yöntemleridir. manifest yaml ile de oluşturulabilir.

• kubectl create configmap demo-config 
configmap/demo-config created 
• k get cm 
DATA AGE 
demo-config 
kube-root -ca . crt 
1 
1 

from literal ile komut satırında cm oluşturduk.

describe ile value lerin yazdığını görüyoruz:

• k get cm demo-config 
apiVersion: VI 
data: 
GREETING: hola 
kind: ConfigMap 
metadata : 
creation Timestamp: 
name: demo-config 
namespace: default 
resourceVersion : ' 
122497" 
uid: deb69b47-fb16-467a-8øc1-a6eØ2b45492a 
-o yaml 

şimdi manifest dosyasıyla comfigmap oluşturalım.

apiVersion: v1

kind: ConfigMap

metadata:

name: demo-config

data:

greeting: Hola

deployment.yaml da modify yapıyoruz:

env:

- name: GREETING

valueFrom:

configMapKeyRef:

name: demo-config

key: greeting

env kısmını modifie ettik.

üstteki GREETING ile env i tanımlıyoruz. aşağıda ise configmap ten greeting key in değerini al demiş oluyoruz.

get cm, pod, deploy, svc 
configmap/demo-config 
configmap/kube-root-ca . crt 
pod/ demo-7dcc8bd9cf-tbpks 
DATA 
1 
1 
READY 
1/1 
1/1 
1/1 
AGE 
58s 
STATUS 
Running 
Running 
Running 
RESTARTS 
3 (8h ago) 
pod/nginx-pod 
pod/secret-env-pod 
deployment . apps / demo 
service/demo-service 
service/kubernetes 
AGE 
58s 
46h 
45m 
READY 
1/1 
TYPE 
UP-TO-DATE AVAILABLE 
1 
AGE 
58s 
1 
CLUSTER-IP 
10. 105.224. 
10.96.ø.1 
61 
EXTERNAL 
<none> 
<none> 
PORT(S) 
80 : 30001/TCP 
443/TCP 
AGE 
58s 
NodePort 
ClusterIP 

C localhost:30001 
AWS Management... Course AWS 1 3-TU. 
Bola, Clarusvvay 

configmap teki value yu değiştirip apply ettiğimizde deploymentta değişikliği göremiyoruz, pod un restart olması gerekiyor görebilmek için.

şimdi configmap te greeting i büyük yazarsak POSIX format deniyor buna kubernetes bunun env variable olduğunu anlıyor:

apiVersion: v1

kind: ConfigMap

metadata:

name: demo-config

data:

GREETING: Merhaba

deployment ta env: yerine envFrom diyoruz:

spec:

containers:

- name: demo

image: clarusway/demo:hello-config-env

ports:

- containerPort: 8888

envFrom:

- configMapRef:

name: demo-config

envFrom dediğimizde ise key girmemize gerek kalmıyor çünkü configmap te büyük harflerle POSIX formatında yazınca onun key olduğunu anlıyor.

bir configmap objesinde birden fazla env variable ları almak için onları da deployment ta yazıyoruz. birdne fazla env yazacağımız zaman da POSIX formatı işimizi kolaylaştırıyor.

ne kadar key-value varsa envFrom ile hepsini alır.

şimdi de configmap te dosyadan nasıl çekebiliriz onu görelim:

k8s içinde config diye bir dosya oluşturduk:

echo "greeting: Hei" > config

kubectl create configmap demo-config --from-file=./config

• kubectl create configmap demo-config 
configmap/demo-config created 
- -from-file=. / config 

bu komutla config file dan al demiş olduk.

• k get cm demo-config 
-o Json 
"apiVersion 
"VI", 
"data " : 
"config" • "greeting: 
Hei\n" 
"kind": "ConfigMap" , 
"metadata " : 
"creation Timestamp 
"name": "demo-config" , 
"namespace": "default" , 
"resourceVersion 
"125771", 
"uid" "b681d21ø-7f7d-4f09-8c35-a7føae29c49e" 

burada dosya ismi key gibi oldu: config: greeting: Hei

bu cm yi silelim ve yaml dosyası ile oluşturalım:

apiVersion: v1

kind: ConfigMap

metadata:

name: demo-config

data:

config: |

greeting: Buongiorno

data: altında bu kez key olarak config girdik, yukarıda söylediğimiz gibi burada key dosya ismi oldu.

burada | işaretini multiline gibi düşünelim.

daha fazla key value eklemek istiyorsak buradaki yöntemi budur.

simdi deployment ı da modifie etmemiz gerekiyor. burada configmap i volume e bağlayalım.

apiVersion: apps/v1

kind: Deployment

metadata:

name: demo

spec:

replicas: 1

selector:

matchLabels:

app: demo

template:

metadata:

labels:

app: demo

spec:

containers:

- name: demo

image: clarusway/demo:hello-config-file

ports:

- containerPort: 8888

volumeMounts:

- mountPath: /config/

name: demo-config-volume

readOnly: true

volumes:

- name: demo-config-volume

configMap:

name: demo-config

items:

- key: config

path: demo.yaml

bir volume oluştur, ismi demo-config-volume olacak.

configMap ismini giriyoruz. altında items parametresi geldi. burada key config di zaten onu yazdık. path geldi, config teki valueyi al volume bağladığımız yerde demo.yaml oluştur ve value yi oraya yaz.

VolumeMounts: altında ise mounthpath te belirlenen config de container içinde bir config klasörü oluştur ve buraya mount et demek.

apply dedik uygulamayı gördük:

C A Güvenli deéil 54.196.0.48:30001 
AWS Management... Course AWS 13-TU... Google 
Buongiorno, Clarusvvay ! 

pod a bağlanalım:

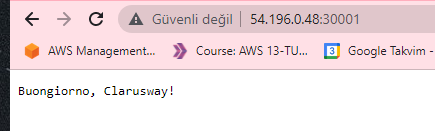
o k exec -it 
/ # Is 
demo-5bf545f76-vwz76 
bin 
dev 
horne 
lib 
media 
mnt 
opt 
proc 
root 
run 
sbin 
config etc 
sys 
tmp 
sh 
usr 
var 

config isimli bir klasör oluşturmuş.

/ # cd config/ 
/config # Is 
demo. yaml 
/config # cat demo.yaml 
greeting: Buongiorno 
/config # 

config içinde demo.yaml ı oluşturmuş onun içine de key ve value mizi yazdırmış.

bu örnekteki deployment ta image ın da değiştiğini gözden kaçırmayalım. image da yapılan conf ile



bu çıktıyı görüyoruz, bu çıktıyı demo.yaml dan çekiyor.