Openshift basics

Contents

Telepítés

Telepítsük fel a függ?ségeket, ha ezek hiányoznának
A minisfhit KVM virtuális gépet hoz létre, amit docker-machine driver-el vezérel.

```
# sudo dnf install libvirt qemu-kvm
```

Telepítsük föl a docker-machine driver-t:

```
# curl -L https://github.com/dhiltgen/docker-machine-kvm/releases/download/v0.10.0/docker-machine-driver-kvm-centos7 -o /usr/local/bin/docker
# chmod +x /usr/local/bin/docker-machine-driver-kvm
```

Külön fel kell telepítsük az oc scriptet, ami az OpenShfit parancssori eszköze:
https://github.com/CCI-MOC/moc-public/wiki/Installing-the-oc-CLI-tool-of-OpenShift

```
ln -s  /path to download/oc /usr/bin/oc
```

Harmadik lépésként le kell tölteni a minishift binárist. Ezzel indíthatjuk el majd a lokális, egy node-os clusterünket. Ez csak az indító script, induláskor le fogja tölteni az oc programot és a centOs-mini-t, majd libvirt-el létrehoz egy qemu virtuális gépet.
https://github.com/minishift/minishift/releases

```
# ln -s /path to download /minishift-1.34.0-linux-amd64/minishift /usr/bin/minishift
```

Indítsuk el a minishift-et.

```
# minishift start
-- Starting profile 'minishift'
-- Check if deprecated options are used ... OK
...
...
The server is accessible via web console at:
    https://192.168.42.185:8443/console

You are logged in as:
    User:       developer
    Password: developer

To login as administrator:
    oc login -u system:admin
```

Ha letöltötte a CentOS-t és feltelepítette rá az OpenShfit-et, ki fogja írni, hogy az OpenShift konzol milyen címen érhet? el. Ez a KVM VM IP címe. Ezen felül láthatjuk, hogy két user-t is létrehozott, egy admin-t és egy developer-t.

Virsh-val nézzük meg hogy fut e a VM:

```
# virsh list --all
 Id   Name         State
----------------------------
 1    minishift    running
```

Törlés

A **minishift delete** paranccsal az aktuális VM-et töröljük csak le, nem a minishift programot. A törlés után egy minishift start új VM-et fog készíteni.

```
# minishift delete
You are deleting the Minishift VM: 'minishift'. Do you want to continue [y/N]?: y
Removing entries from kubeconfig for cluster: 192-168-42-64:8443
Deleting the Minishift VM...
Minishift VM deleted.
```

Login

Az **oc login** paranccsal kell belépni a klaszerbe. A sikeres oc login inicializálja a kubernetes config-ot is.

```
# oc login -u system:admin
Logged into "https://192.168.42.185:8443" as "system:admin" using existing credentials.

You have access to the following projects and can switch between them with 'oc project <projectname>':

    default
    kube-dns
    kube-proxy
    kube-public
    kube-system
  * myproject
    ...
```

OpenShift-ben a projekt egyenl? a Kubernetes névtérrel. Láthatjuk, hogy a myproject névtérbe vagyunk belépve.

Listázzuk a Kubernetes konfigurációt:

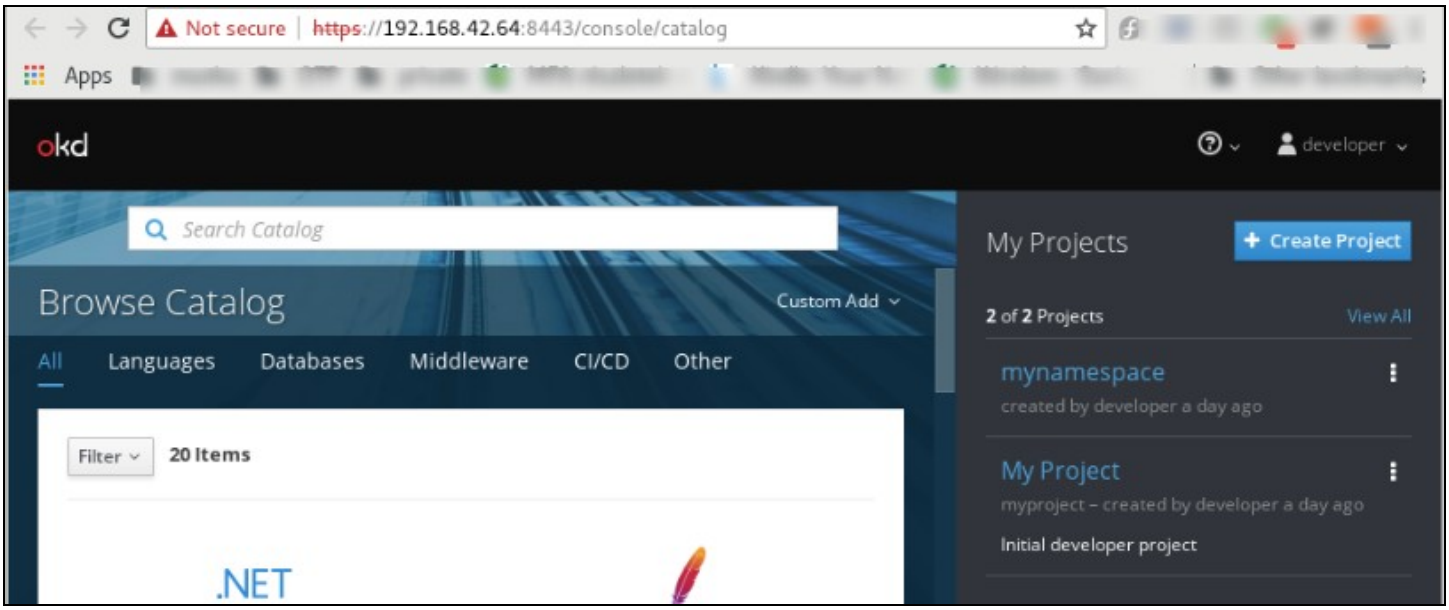
```
# kubectl config view
....
- context:
    cluster: 192-168-42-185:8443
    namespace: myproject
    user: system:admin/192-168-42-185:8443
  name: myproject/192-168-42-185:8443/system:admin
```

Láthatjuk, hogy létrehozott az **oc login** egy külön kontextust. Innent?l kezdve a **kubectl** parancsokat tudjuk használni.

Listázzuk ki a felhasználókat. Látni fogjuk hogy a fenti is látott system és developer felhasználók vannak a klaszterben:

```
# kubectl get users
NAME        UID                                    FULL NAME   IDENTITIES
developer   f59113f2-9f30-11e9-ba85-525400efb4ec               anypassword:developer
system      df29da3c-9f35-11e9-ba85-525400efb4ec               anypassword:system
```

A webes konzolon is be tudunk lépni: https://192.168.42.64:8443/console/catalog



Note
Ha kés?bb újra el akarjuk indítani a minishfit-et, akkor fontos, hogy indulás el?tt a kubectl kontextust átállítsuk a minishfit-es kontextusra ha több kontextusunk is van, különben nem fog elindulni a minishfit.

```
# kubectl config use-context myproject/192-168-42-185:8443/system:admin
```

Új névtér hozzáadása

Hozzuk létre az új névteret.

```
# kubectl create namespace mynamespace
```

Rendeljük hozzá az admin clusterrole-t a developer user-hez a mynamespace névtérben. Ezzel teljhatalmat kap a developer a mynamesapce-ben.

```
# kubectl create rolebinding developer-admin --namespace=mynamespace --clusterrole=admin --user=developer
rolebinding.rbac.authorization.k8s.io/developer-admin created
```

Ezzel létrejött a developer-admin rolebindig a mynamespace-ben, listázzuk ki:

```
# kubectl get rolebinding developer-admin -n mynamespace -o yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  creationTimestamp: "2019-09-22T10:51:01Z"
  name: developer-admin
  namespace: mynamespace
  resourceVersion: "35751"
  selfLink: /apis/rbac.authorization.k8s.io/v1/namespaces/mynamespace/rolebindings/developer-admin
  uid: de318108-dd26-11e9-bc53-52540074f436
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: admin
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: developer
```

Most lépjünk be a developer-el, láthatjuk, hogy megjelent a névtér listában a mynamespace is:

```
# oc login
Authentication required for https://192.168.42.214:8443 (openshift)
Username: developer
Password:
Login successful.

You have access to the following projects and can switch between them with 'oc project <projectname>':

    mynamespace
  * myproject

Using project "myproject".
```

Deployment és service létrehozása

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  revisionHistoryLimit:5
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge:1
      maxUnavailable: 1
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: bitnami/nginx
        ports:
        - containerPort: 80
```

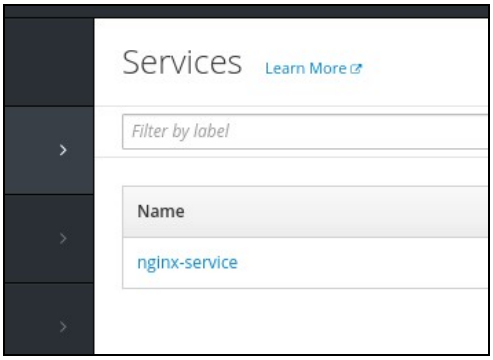
Warning
Az OpenShift-el nem kompatibilis a standard ngingx image, mert root felhasználóval kéne futtatni, ahhoz hogy létre tudja hozni a szükséges mappákat a konténerben. Ez ki van küszöbölve a **bitnami/nginx** image-ben.

```
# kubectl apply -f deployment-demo.yaml

# kubectl get deployment
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   3         3         3            3           2d
```

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
  namespace: my-namespace
spec:
  ports:
  - nodePort: 32730
    port: 8080
    protocol: TCP
```

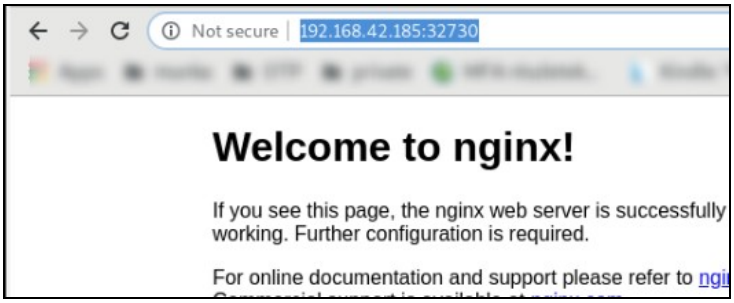
```
    targetPort: 8080
  selector:
    app: nginx
  type: NodePort

# kubectl apply -f service-demo.yaml
service/http created

# kubectl get svc
NAME    TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
nginx-service   NodePort   172.30.198.61   <none>        8080:32730/TCP   31s
```

Openshfit console:




```
# minishfit ip
192.168.42.185
```

http://192.168.42.185:32730/




```
# kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-58b5fbbff4-6cztz   1/1     Running   0          7m
nginx-deployment-58b5fbbff4-bsq6x   1/1     Running   0          7m
nginx-deployment-58b5fbbff4-gzr8g   1/1     Running   0          7m

# kubectl exec -it nginx-deployment-58b5fbbff4-6cztz /bin/bash
$ curl nginx-service:8080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
```

Openshift sepcific commands

Openshift specific object

DeploymentConfig

Service with ssl encryption

Az egésznek a mozgató rugója, hogy a service objektumban szerepeltetjük a **service.alpha.openshift.io/serving-cert-secret-name** annotációt. Az itt megadott értékkel (a példában **my-app-internal-cert**) létre fog jönni automatikusan egy TLS típusú secret mikor kiadjuk a svc-re az apply parancsot.
service.yaml

```
apiVersion: v1
kind: Service
```

```
metadata:
  annotations:
    service.alpha.openshift.io/serving-cert-secret-name: my-app-internal-cert
  labels:
    app: my-app
    configurable: "true"
  name: my-app
  namespace: mynamespace
...
```

Ezt a kulcsot fogja a service titkosításra használni.

```
# kubectl get secret pod-monitor-internal-cert -o yaml
-----------------------------------------------------
apiVersion: v1
data:
  tls.crt: SDGSDFGsda...
  tls.key: AFGGSDGcs....
kind: Secret
metadata:
  name: my-app-internal-cert
  namespace: mynamespace
  ownerReferences:
  - apiVersion: v1
    kind: Service
    name: my-app
type: kubernetes.io/tls
```

A **my-app-internal-cert** secret-et bele kell tenni a Deployment-hez tartozó service-account-ba is, hogy mount-olni tudjuk majd a pod-ba:
service-account.yaml

```
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app: my-app
  name: my-app
secrets:
...
- name: my-app-internal-cert
```

deployment.yaml

```
- mountPath: /var/run/secrets/https-internal-cert
          name: https-internal-cert
          readOnly: true
...
      volumes:
      - name: https-internal-cert
        secret:
          defaultMode: 420
          secretName: my-app-internal-cert
```

Router

http://people.redhat.com/jrivera/openshift-docs_preview/openshift-origin/glusterfs-review/install_config/router/index.html
http://people.redhat.com/jrivera/openshift-docs_preview/openshift-origin/glusterfs-review/architecture/networking/routes.html
https://docs.openshift.com/container-platform/3.11/architecture/networking/routes.html#route-types
Default: HAProxy

The controller and HAProxy are housed inside a pod, which is managed by a deployment configuration. The process of setting up the router is automated by the oc adm router command.

The controller watches the routes and endpoints for changes, as well as HAProxy?s health. When a change is detected, it builds a new haproxy-config file and restarts HAProxy. The haproxy-config file is constructed based on the router?s template file and information from OpenShift Origin.

Ez az Ingress megfelel?je az OpenShfit-ben. Az alapértelmezett implementációja a HAProxy. A load-balancing-ot OpenShift-ben a szabványos Kubernetes-el szemben a router végzi, nem a service. Alapértelmezetten a router-ek a node-on a 80-as ill a 443-as portokra fognak kapcsolódni.

```
apiVersion: v1
kind: Route
metadata:
  name: nginx-route
  namespace: my-namespace
spec:
  path: /
  to:
    kind: Service
    name: nginx-service
    weight: 100
  port:
    targetPort: 8080
  wildcardPolicy: None
  tls:
    insecureEdgeTerminationPolicy: Allow
    termination: edge
```

- Edge Termination: a TLS terminálva lesz még a load-balancing el?tt. A TLS beállításokat ebben az esetben a router-ben kell megadni. Ha nem adjuk meg, akkor az alapértelmezett tanúsítványt fogja használni.
- Passthrough Termination: ebben az esetben a router nem választja le a TLS-t, a kérést egy az egyben továbbküldi a service-nek, aki meg a POD-oknak. A POD-ok felel?ssége, hogy a megfelel? tanúsítványt kezelj?k.

- Re-encryption Termination: A router végez TLS terminálást, de a router és a service közötti kapcsolat megint csak titkosítva történik.

Ingress definíciót is meg lehet adni, ezt konvertálni fogja az OpenShift Route objektumra.

Telepítés:
A kubectl parancsot nem használhatjuk a router telepítésére, nem nem ismeri ezt a típust, kizárólag az oc paranccsal tudjuk telepíteni:

```
# oc apply -f route-nginx.yaml
route.route.openshift.io/nginx-route created
```

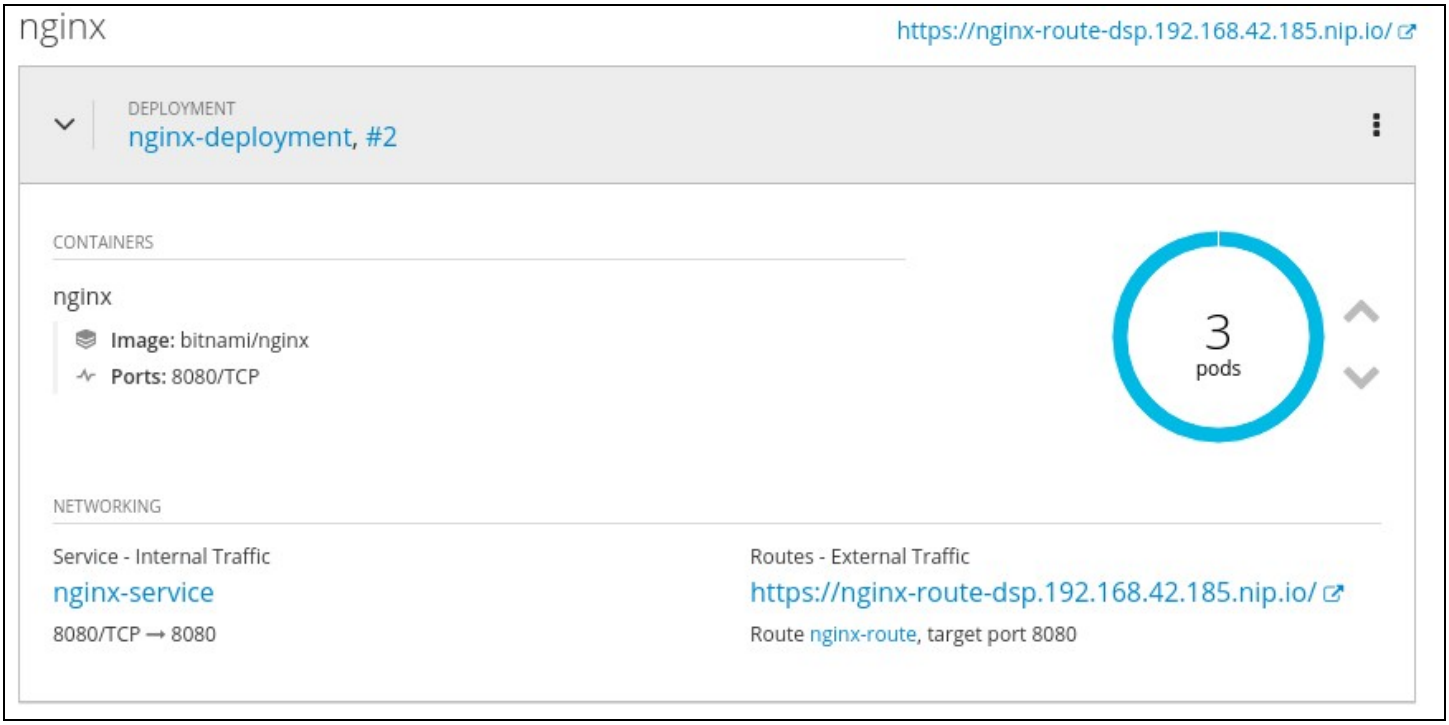
Lekérdezni már kubectl -el is lehet :

```
# kubectl get route -o wide
NAME          HOST/PORT                              PATH   SERVICES        PORT   TERMINATION   WILDCARD
nginx-route   nginx-route-dsp.192.168.42.185.nip.io   /      nginx-service   8080   edge/Allow    None
```

Mivel nem adtunk meg host nevet az OpenShift generált egyet a route-hoz. Az openShift alapértelmezetten a **nip.io** szolgáltatást használja host név generáláshoz. Ez egy ingyenes DNS szolgáltatás, ahol a <anything>[.-]<IP Address>.nip.io szerkezet? host neveket mindig a megadott IP címre oldja fel a nip.io DNS szervere.

```
# nslookup nginx-route-dsp.192.168.42.185.nip.io
Server:         192.168.0.1
Address:        192.168.0.1#53

Non-authoritative answer:
Name:   nginx-route-dsp.192.168.42.185.nip.io
Address: 192.168.42.185
```

Nézzük meg az OpenShift konzolon a szolgáltatásunkat:



Láthatjuk, hogy létrejött hozzá egy route az nginx-route-dsp.192.168.42.185.nip.io URL-el.

Íjuk be a böngész?be: https://nginx-route-dsp.192.168.42.185.nip.io

```
# kubectl get dc -n default
NAME              REVISION   DESIRED   CURRENT   TRIGGERED BY
router            1          1         1         config
```

```
# kubectl get dc router -n default -o yaml
apiVersion: apps.openshift.io/v1
kind: DeploymentConfig
spec:
      containers:
        image: openshift/origin-haproxy-router:v3.11.0
        name: router
```

SecurityContextConstraints

https://medium.com/bitnami-perspectives/running-containers-in-openshift-629af79945b5

Névtér független

```
# kubectl get SecurityContextConstraints anyuid -o yaml

allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: true
allowPrivilegedContainer: false
allowedCapabilities: null
apiVersion: security.openshift.io/v1
defaultAddCapabilities: null
fsGroup:
  type: RunAsAny
groups:
- system:cluster-admins
kind: SecurityContextConstraints
metadata:
  annotations:
    kubernetes.io/description: anyuid provides all features of the restricted SCC
      but allows users to run with any UID and any GID.
  name: anyuid
priority: 10
readOnlyRootFilesystem: false
requiredDropCapabilities:
- MKNOD
runAsUser:
  type: RunAsAny
seLinuxContext:
  type: MustRunAs
supplementalGroups:
  type: RunAsAny
users:
- system:serviceaccount:mynamespace:default
volumes:
- configMap
- downwardAPI
- emptyDir
- persistentVolumeClaim
- projected
- secret
```

Run container as root
Azt hogy melyik pod futtathat root nevében konténert a **SecurityContextConstraints** objektumok írják le. Itt meg lehet határozni, hogy milyen user-nek és group-nak engedjük meg a konténer futtatását a konténeren belül. A **SecurityContextConstraints** definícióban a user szekcióban meg lehet adni service account-okat, és akkor az adott serviceAccount-hoz rendelt deploymenthez tartozó pod-oknak meg lehet engedni hogy a nekik tetsz? user-el futtassanak process-eket. Ha nincs megengedi a root futtatás egy pod-nak, akkor egy nem privilegizált user-el fog futni.

```
# oc edit scc anyuid
securitycontextconstraints.security.openshift.io/anyuid edited
```

A **mynamespace** névtérben lév? **default** nev? service account-al futtatott pod-oknak megengedjük a tetsz?leges user választást.

```
- system:serviceaccount:mynamespace:default
```

Ha egy deployment-nek külön nem adjuk meg, akkor az adott névtér default serviceAccount-jával fog futni.

Template

https://docs.openshift.com/container-platform/3.11/dev_guide/templates.html

```
# oc get template -n mynamespace
NAME                         DESCRIPTION                                                                   PARAMETERS     OBJECTS
mytemplate-1                 This template contains the DeploymentConfig, Service, Route and ServiceAccoun...   11 (5 blank)   5
```

```
# oc get template mytemplate-1 -n mynamespace -o yaml
apiVersion: template.openshift.io/v1
kind: Template
metadata:
....
```

Fájl másolása konténerekb?l

```
kubectl cp <névtér>/<pod-név>:<fájl elérési út> <cél-mappa-neve>
```

Pl:

```
kubectl cp mynamespace/example-pod:/tmp/example.txt /home/adam/
```

Note
Fontos, hogy a konténerben legyen rsync vagy tar telepítve, ezek valamelyikével fogja a Kubernetes végrehajtani a másolást

A cp OpenShfit megfelel?je a oc rsync :

```
 oc rsync mynamespace/example-pod:/tmp/example.txt /home/adam/
```

Minishfit docker registry

https://torstenwalter.de/minishift/openshift/docker/registry/2017/07/25/build-docker-image-and-upload-to-openshift-registry.html
https://github.com/minishift/minishift/issues/817

Áttekintés

Ahhoz hogy egy általunk készült docker image-et futtatni tudjunk a minishift klaszteren, els?ként fel kell tölteni azt a minshift image repository-ba, ami nem is annyira egyszer?. A minishift docker registry-t csak a minishfit-et futtató VM-r?l lehet elérni, az anyagépr?l nem.

Az openshfit saját docker registry-je egy deploymentConfig-al definiált pod-ban fut:

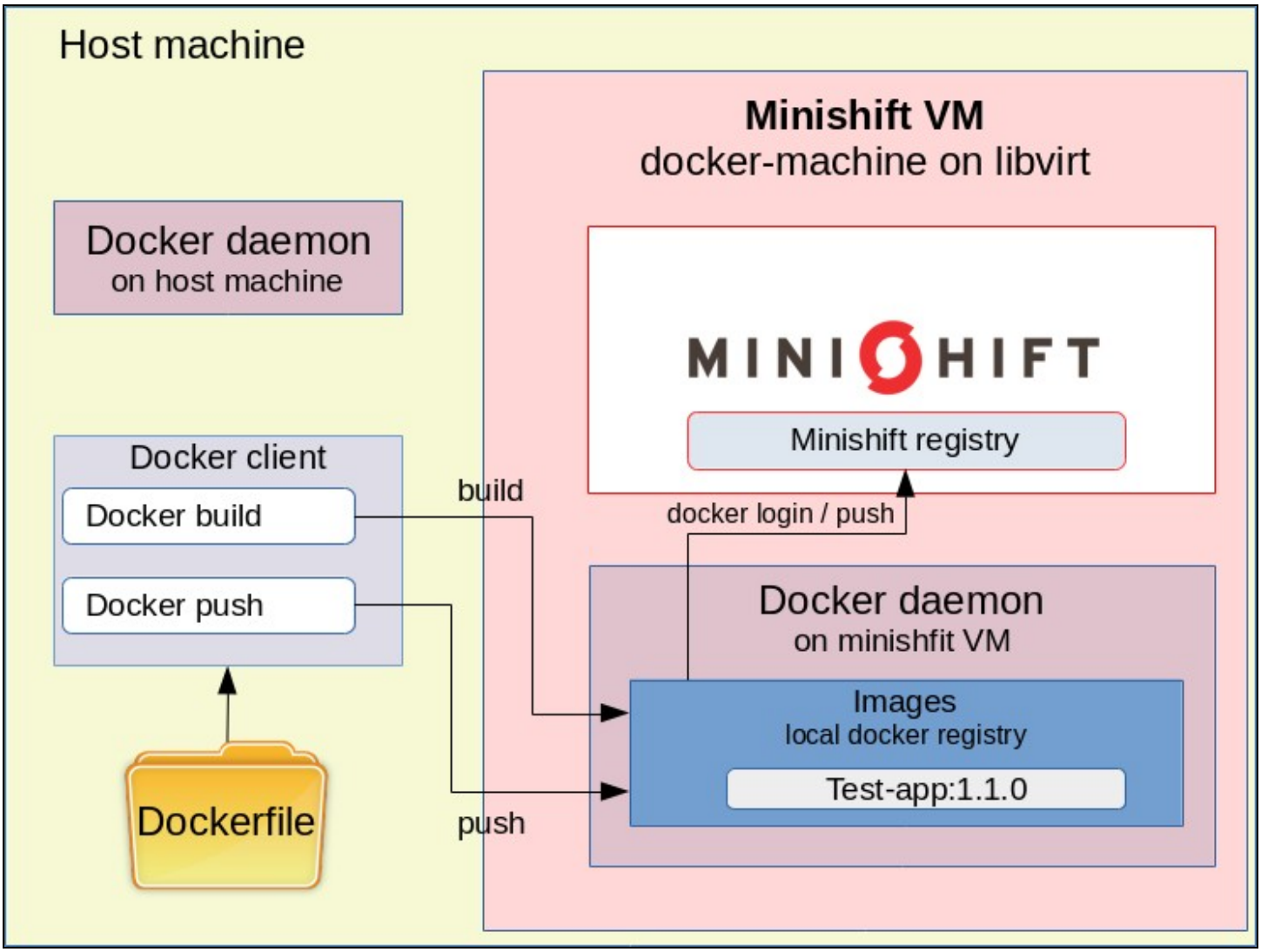
```
# kubectl get dc -n default
NAME              REVISION   DESIRED   CURRENT   TRIGGERED BY
docker-registry   1          1         1         config
...
```

A registry pod ebb?l az image-b?l készül: **openshift/origin-docker-registry:v3.11.0**

Lépésel:

1. A minishit-et futó VM-en a docker klienssel be kell lépni a minisfht-ben futó docker registry-be az els? pontban szerzett token-el.
2. Docker-el build-elni kell az image-t a minishfit VM-en.
3. A minisfhit-et futtató VM-r?l push-olni kell az image-t a minisfhit image registry-be.
4. kubectl run-al el lehet indítani az általunk létrehozott image-et.

Az egyik lehet?ségünk az, hogy ssh-val belépünk a minisfhit VM-re (minisfhit ssh) és ott lépünk be a registry-be (docker login) és ott build-eljük és push-oljuk az image-et. A másik sokkal kényelmesebb lehet?ség, hogy a minishift VM docker démon-ját átirányítjuk a lokális docker kliensünkre a **minishift docker-env** paranccsal. Ezt azért tudjuk megtenni, mert a minisfhit is **docker-machine**-t használ. Ugyan ezt a parancsot használjuk docker swarm esetén is, hogy a **swarm master**-en adjuk ki a swarm parancsokat. Ennek az a nagy el?nye, hogy a docker build-hez szükséges fájlokat vehetjük a lokáli gépr?l, de a build-elt docker image már a minishfit VM docker lokális repository-ban fog landolni, ahonnan már push-olni tudjuk a minishfit repositroy-ba.



Kísérlet képen írtunk egy olyan Java webalkalmazást, ami a GET paraméterben kapott milliszekundum után adja vissza a 200 : 'OK' választ. Az alkalmazás embedded jetty-vel készült, a Docker fájlja az alábbi:

```
FROM openjdk:11-jdk
RUN mkdir /home/lib
ADD ./*.jar /home
ADD ./lib/* /home/lib/
EXPOSE 8080
CMD ["java", "-cp", "/home/lib/*:/home/testapp.jar", "com.adam.testapp.App"]
```

Az openjdk-11 base image-b?l indulunk ki, majd belemásoljuk az összes függ?séget, rárakjuk classPath-ra az összes bemásolt jar-t, amjd elindítjuk az alkalmazást a **com.adam.testapp.App** main osztállyal. A cél, hogy az ebb?l készült image-et másoljuk fel az opensfhit repository-ba, majd készítsünk bel?le egy Kubernetes deploymentet-et service-el együtt. A Dokcerfile és az abban hivatkozott összes további fájl a host gépen lesz, de mivel a docker-kliens a minishift-en lév? démonhoz csatlakozik, mikor kiadjuk majd a build parancsot, az image a minishfit VM ottani lokális docker repository-ába fog bekerülni.

Belépés az openshfit registriy-be

Ahogy azt már korábban írtuk, át fogjuk irányítani a host gépen futó docker klienst, hogy a minisfhit VM-en futó docker démonhoz kapcsolódjon, így a lokálisan kiadott docker parancsok mind a minishift VM-en futó docker-en fognak végrehajtódni. Ez megegyezik azzal, ahogy a docker swarm esetében is csatlakozunk a swarm master node-on futó démonhoz:

```
# minishift docker-env
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.42.185:2376"
export DOCKER_CERT_PATH="/root/.minishift/certs"

# eval $(minishift docker-env)
```

Innent?l kezdve minden lokálisan kiadott docker parancs a minishfit vm-en fog futni.

Ezt le tudjuk ellen?rizni, ha listázzuk vagy a futó konténereket vagy az elérhet? image-eket a docker démonon, ahova a docker kliens csatlakozik:

```
# docker image ls
REPOSITORY                                  TAG                 IMAGE ID            CREATED             SIZE
bitnami/nginx                               latest              ab5cdb60157c        11 hours ago        78.3MB
172.30.1.1:5000/mynamespace/test-app        1.1.0               f984e2105039        23 hours ago        620MB
```

```
...
openshift/origin-control-plane                v3.11.0             2905f7137f05        4 days ago          829MB
openshift/origin-hyperkube                    v3.11.0             7f2cb107435c        4 days ago          509MB
openshift/origin-hypershift                   v3.11.0             7507661b73ff        4 days ago          549MB
...
```

Láthatjuk, hogy csupa openshift-es image van a repóban, tehát már nem az anyagép lokális repository-ját nézzük.

A célunk, hogy a minisfhit VM-en futó docker démon belépjen a minshift saját docker registry-ébe, és oda push-olja a saját lokális repository-ban lév? image-et. Ehhez szereznünk kell egy openShift login toke-ent és meg kell szerezzük a repository címét.

Lépjünk be az **oc login**-al az OpenShift-be. Fontos hogy ne a system hanem a **developer** felhasználóval lépjünk be, mert csak neki lesz push joga.

Note
Ha így lépünk be nem jó: '*oc login -u system:admin*, mert nem keletkezik token.

```
# oc login
Authentication required for https://192.168.42.185:8443 (openshift)
Username: developer
Password:
Login successful.

You have access to the following projects and can switch between them with 'oc project <projectname>':

  * mynamespace
    myproject
```

Ellen?rizzük, hogy kaptunk e tokent. Ezt az **oc whoami** parancs **-t** (token) kapcsolójával lehet megtenni. Ha itt nem látjuk a toke-nt akkor dolgozni kell még a belépésen.

```
# oc whoami -t
JbdYzUHpsyJCBt3JQ7XBahLq6UAp0hiYsOTYTiD1arE
```

A minishift -nek a **openshift registry** parancsával kérhetjük le a docker registry címét.

```
# minishift openshift registry
172.30.1.1:5000
```

Ez a cím a minishift-et futtató VM-en értelmezett IP cím, ezért csak a minishift VM-en futó docker démon-al tudunk rácsatlakozni a repository-ra.

A **docker login** paranccsal léphetünk be a minishift registriy-be. Ha ezt lokálisan kiadjuk, akkor már a minishit VM-en fog lefutni így el fogjuk érni a **172.30.1.1:5000** címen a registry-t (A docker kliens átirányítása nélkül nem érnénk el a lokális gépr?l ezt a címet). Ezzel a minisfhit VM-en futó docker démon csatlakozik a minisfhit registry-hez, tehát nem az anyagépen futó docker démon.

```
# docker login -u developer -p $(oc whoami -t) $(minishift openshift registry)
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

Fontos, hogy csak token-el engedi a belépést az openshfit registry, jelszóval nem.

docker build és push

Az openshfit registry-be szánt image-nek a neve az alábbi szintaxist kell kövesse:

```
<registry-host>:<port>/<névtér>/<image-név>:<tag>
```

Ahhoz hogy push-olni tudjuk az OpenShift registry-be az image-et a "172.30.1.1:5000/" prefixel kell kezd?djön a neve. Innen fogja tudni a docker, hogy melyik registry-be akarjuk push-olni. A második elem a névtér neve, amihez az image tartozni fog. Az image-ek ugyan úgy névterekbe kerülnek mint minden kubernetes objektum.

A **docker build** paranccsal építsük meg az image-t. Ezt az image-et is tegyük a **mynamespace** névtérebe.

```
# docker build -t 172.30.1.1:5000/mynamespace/test-app:1.1.0 .
Sending build context to Docker daemon  14.99MB
Step 1/6 : FROM openjdk:11-jdk
 ---> f06fdc42d01a
...
Successfully built ecc484546662
Successfully tagged 172.30.1.1:5000/mynamespace/test-app:1.1.0
```

Ellen?rizzük le, hogy bekerült e a test-app a minisfhit VM lokális docker repository-jába: (mivel az anyagépen futó docker kliens át van irányítva, ezért az anyagépen kiadott docker parancsokkal elvégezhet? az ellen?rzés)

```
# docker image ls
REPOSITORY                                    TAG                 IMAGE ID            CREATED             SIZE
...
172.30.1.1:5000/mynamespace/test-app          1.1.0               f984e2105039        23 hours ago        620MB
```

Nem maradt más dolgunk, mint hogy a **docker push** paranccsal felküldjük az image-t az opensfhit registry-be. A nevéb?l fogja tudni a **docker push** hogy melyik repository-ba kell küldeni és ott melyik névtérbe.

```
# docker push 172.30.1.1:5000/mynamespace/test-app:1.1.0
```

A minishift registry-ben úgynevezett **imageScream**-ek vannak. A frissen push-olt **test-api** image-hez is létrejött egy imageScream. Az oc paranccsal listázhatjuk a imageScream-eket:

```
# oc get is -n mynamespace
NAME       DOCKER REPO                            TAGS      UPDATED
test-app   172.30.1.1:5000/mynamespace/test-app   1.1.0     8 minutes ago
```

Teszt alkalmazás futtatása

Az egyszer?ség kedvéért a **kubectl run** paranccsal fogjuk futtatni az alkalmazást. Ha nem adjuk meg a -it kapcsolót, akkor létre fog hozzá hozni egy deployment-et és egy replicaSet-et is, nem csak a pod-ot fogja elindítani.

```
# kubectl run test-app --image=172.30.1.1:5000/mynamespace/test-app:1.1.0 --replicas=1 --port=8080 -n mynamespace
```

Ellen?rizzük, hogy létrejött e a test-app nev? deployment.

```
# kubectl get deployment -n mynamespace
NAME                          DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
..
test-app                      1         1         1            1           2m
```

Nézzük meg a deployment definícióját (csak a releváns részeket tüntetem fel):

```
# kubectl get deployment test-app -n mynamespace -o yaml

apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    run: test-app
  name: test-app
  namespace: mynamespace
spec:
  replicas: 1
  selector:
    matchLabels:
      run: test-app
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      creationTimestamp: null
      labels:
        run: test-app
    spec:
      containers:
      - image: 172.30.1.1:5000/mynamespace/test-app:1.1.0
        imagePullPolicy: IfNotPresent
        name: test-app
        ports:
        - containerPort: 8080
          protocol: TCP
      restartPolicy: Always
```

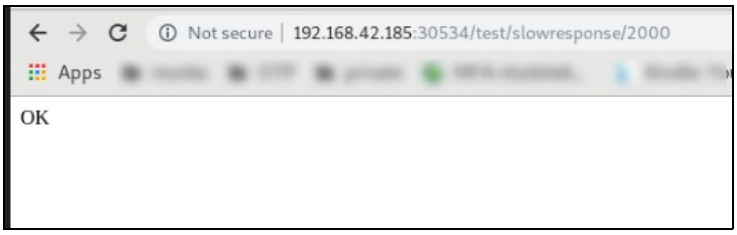
A **kubectl run** parancshoz hasonlóan a **kubectl expose** paranccsal instant módon készíthetünk service-t a deployment-hez. A deployment-ben megadott portot (8080) fogja kinyitni a pod-ok felé:

```
# kubectl expose deployment test-app --type=LoadBalancer --name=test-app-service -n mynamespace
service/test-app-service exposed
```

Ellen?rizzük, hogy létrejött e a service. Láthatjuk, hogy a **30534** porton érhet? el kívülr?l a service.

```
# kubectl get svc -n mynamespace
NAME                          TYPE           CLUSTER-IP      EXTERNAL-IP                     PORT(S)          AGE
...
test-app-service              LoadBalancer   172.30.254.3    172.29.192.241,172.29.192.241   8080:30534/TCP   2m
```

Ha a minishift VM publikus címét beírjuk a fenti port számmal, akkor meg kell tudjuk szólítani az alkalmazásunkat:
http://192.168.42.185:30534/test/slowresponse/2000



Hozzunk létre egy opensfhit route-ot is az imént létrehozott service-hez ugyan úgy instant módon az **oc create route** paranccsal.

```
# oc create route -n mynamespace edge --service test-app-service --insecure-policy Allow
route.route.openshift.io/test-app-service created
```

Listázzuk ki az imént létrehozott route-ot **wide** módban, hogy megtudjuk mi lett a host neve. A minisfhit alapértelmezetten készít egy **nip.io**-s host nevet minden route-hoz, ahol nem adjuk meg mi legyen a host név:

```
# kubectl get route test-app-service -n mynamespace
NAME               HOST/PORT                                          PATH   SERVICES           PORT    TERMINATION   WILDCARD
test-app-service   test-app-service-mynamespace.192.168.42.185.nip.io         test-app-service   <all>   edge/Allow    None
```

http://test-app-service-mynamespace.192.168.42.185.nip.io/test/slowresponse/2000