Devops zajęcia nr 1 (23.02.2020r.)

- Konteneryzacja (Docker)
- Orkiestracja (Kubertenes)

Kontener – uruchomiony **proces** w systemie
Image – plik ze „źródłem" kontenera, „binarka"

$ docker run hello-world

- DOCKER DAEMON (server) – API restowe
- DOCKER CLI – komunikuje się z Daemonem wydając mu polecenia

$ docker ps –all → wyświetlenie wszystkich kontenerów
$ docker system info
$ docker system prune → czyści wszystkie kontenery

$ docker run busybox

Devops zajęcia nr 2 (1.03.2020r.)

$ docker info

Image:

FS (file system)	CMD (komenda)
/lan /lib /bin	*echo hello* itp.

Docker:

- DOCKER DAEMON (server) – właściwe centrum wydawania poleceń, coś co kontroluje wszystkie nasze kontenery (uruchamia, stopuje itp.)
- DOCKER CLI (polecenie *docker* z którym pracujemy, wydaje polecenia do DAEMON'a)

Za Wikipedia: Demon (ang. daemon) – program lub proces, wykonywany wewnątrz środowiska wielozadaniowego systemu operacyjnego, bez konieczności interakcji z użytkownikiem (jako proces drugoplanowy)

$ docker ps → pokazuje uruchomione kontenery
$ docker ps -a → pokazuje kontenery, które mieliśmy uruchomione oraz ich stan (ważne tu jest CONTAINER ID, bo potem tego ID będziemy korzystać). Natomiast jak jest STATUS i tam jest Exited to ta wartość w nawiasie to jest stan – gdy różny od 0 to jakiś błąd
$ docker run busybox echo Hello → tutaj *busybox* jest imagem

- https://hub.docker.com/search?q=&type=image
- Redis (https://redis.io/; https://hub.docker.com/_/redis)
- busybox (https://hub.docker.com/_/busybox)
- alpine (https://hub.docker.com/_/alpine)

$ docker run → gdy to jest odpalane to właśnie z docker huba pobierane są image
$ docker images → pobrane przeze mnie wylistowane image
$ docker run image MyCMD → wchodzi wtedy ta moja Komenda MyCMD podmienia tą inną z tabelki wyżej
$ docker system prune -a → czyści wszystkie kontenery, a dzięki „-a" również image. Tego nie ma co jakoś bardzo często odpalać, bo wtedy np. przy kolejnym „docker run busybox" będzie od nowa ściągał *busybox*

$ docker exec -it CONTAINER_ID CMD → wystarczą trzy pierwsze znaki ID kontenera I ogarnie, CMD – komenda np.: $ docker exec -it 9124016b1700 sh → no i to odpali shella, i możemy sobie go używać: np.: *du -hs* (sprawdzenie disk usage)

- **vi** – najlepszy edytor dla adminów, devops-ów itp. (do zadań administracyjnych)
  - **w domu nauczyć się podstawowych komend vi** !
  - https://vim-adventures.com/

- http://commandlinemac.blogspot.com/2008/12/vim.html

Postawiliśmy dwa kontenery i w jednym stworzyliśmy plik. Potem wchodzimy w drugi kontener i tam go nie ma. Tamten plik jest w tym pierwszym kontenerze.

$ docker run redis

- *redis* jest imagem typu server - cmd wystawia proces
- *$ docker ps* → tutaj kopiujemy ID kontenera z redisem (tutaj *08de9ba656fd i używamy na dole)*
- *docker exec -it 08de9ba656fd redis-cli*

```
(base) Jakubs-MacBook-Pro:~ jakublemka$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED              STATUS              PORTS               NAMES
3e8d99f546fc        busybox             "sleep 1000"        49 seconds ago       Up 48 seconds                           friendly_turing
9ce585c1dad5        busybox             "sleep 1000"        About a minute ago   Up About a minute                       xenodochial_diffie
(base) Jakubs-MacBook-Pro:~ jakublemka$ docker exec -it 3e8d99f546fc sh
/ # touch arka
/ # logout
sh: logout: not found
/ # exit
(base) Jakubs-MacBook-Pro:~ jakublemka$ docker exec -it 9ce585c1dad5 sh
/ # ls
bin   dev   etc   home  proc  root  sys   tmp   usr   var
/ #
```

```
(base) Jakubs-MacBook-Pro:~ jakublemka$ docker exec -it 08de9ba656fd redis-cli
127.0.0.1:6379> set message Hello
OK
127.0.0.1:6379> get message
"Hello"
127.0.0.1:6379>
```

127.0.0.1:6379> exit

(base) Jakubs-MacBook-Pro:~ jakublemka$ **docker stop** 08de9ba656fd

Gdy coś jest np. z tym sleep 10000 to docker stop 10 sekund zaczeka, a po 10 sekundach wywoła kill.

```
(base) Jakubs-MacBook-Pro:~ jakublemka$ docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS                      PORTS               NAMES
08de9ba656fd        redis               "docker-entrypoint.s…"   14 minutes ago      Exited (0) 5 minutes ago                        elastic_dewdney
3e8d99f546fc        busybox             "sleep 1000"             34 minutes ago      Exited (0) 6 minutes ago                        friendly_turing
9ce585c1dad5        busybox             "sleep 1000"             34 minutes ago      Exited (0) 6 minutes ago                        xenodochial_diffie
9124016b1700        busybox             "sleep 1000"             53 minutes ago      Exited (0) 36 minutes ago                       stoic_lalande
1cd7164e0050        busybox             "echo beeee"             59 minutes ago      Exited (0) 59 minutes ago                       pedantic_pascal
(base) Jakubs-MacBook-Pro:~ jakublemka$ docker start 08de9ba656fd
08de9ba656fd
(base) Jakubs-MacBook-Pro:~ jakublemka$ docker exec -it 08de9ba656fd redis-cli
127.0.0.1:6379> get message
"ArkaGdynia"
127.0.0.1:6379>
```

$ docker start CONTAINER_ID → startuje nam na nowo, potem *docker stop* CONTAINER_ID I znówgo nie ma, jak robimy *docker ps*

- *docker start -a 08de9ba656fd* → -a jak attach, jak startujemy I interesują nas komunikaty na konsoli
- *docker logs CONTAINER_ID* → wszystkie logi co robiliśmy w danym kontenerze

proces		
STDIN	STDOUT	STDERR

1. **Jak się buduje własny image? Budujemy!**
    a. vi Dockerfile → tworzymy plik

        FROM busybox

        CMD echo Hello from my busybox

    b. :wq → wychodzimy z zapisem (wcześniej ESC by wyjść z tworzenia tekstu pliku
    c. *docker build -t helloworldbusybox:latest .* → tutaj ta kropka to, gdzie szukamy Dockerfile, już byłem w folderze deveps wiec po prostu kropczeka
    d. docker run helloworldbusybox
2. A do dockerhuba:
    a. docker build -t jakublem/mybbhelloworld:latest .
    b. *docker push jakublem/mybbhelloworld:latest*
    c. *https://hub.docker.com/u/jakublem --> I tutaj jest*
3. Natomiast z dockerhuba np. czyjegoś – tutaj użytkownik juuzeff:
    a. docker run juuzeff/mybbox

Ogólny przepis:

1. FROM image_bazowej_dystrybucji
2. Doinstalowywanie (apt-get……)
3. Userzy, prawa…
4. CMD

Architektura do jakiej dążymy:

1. My App – front + backend, a. nad wszystkim NGINX – do serwowania (reverse proxy?)
2. Postgres – baza danych
3. Redis – cache

$ docker run nginx

Nginx to serwer webowy, taki frontowy, na porcie 80.

docker run -p 90:80 nginx

http://localhost:90/ → no i tu będzie *welcome to nginx itp.*
90 – local port; 80 – container port

VI BASICS:

- Tworzymy plik: vi nazwa pliku np. vi arka.txt
- i aby wejść w INSERT mode
- Wychodzimy zapisując plik: najpierw ESC po wpisaniu treści a potem :wq

VI Editing commands

- i - Insert at cursor (goes into insert mode)
- a - Write after cursor (goes into insert mode)
- A - Write at the end of line (goes into insert mode)
- ESC - Terminate insert mode
- u - Undo last change
- U - Undo all changes to the entire line
- o - Open a new line (goes into insert mode)
- dd - Delete line
- 3dd - Delete 3 lines.
- D - Delete contents of line after the cursor
- C - Delete contents of a line after the cursor and insert new text. Press ESC key to end insertion.
- dw - Delete word
- 4dw - Delete 4 words
- cw - Change word
- x - Delete character at the cursor
- r - Replace character
- R - Overwrite characters from cursor onward
- s - Substitute one character under cursor continue to insert
- S - Substitute entire line and begin to insert at the beginning of the line
- ~ - Change case of individual character

Moving within a file

- k - Move cursor up
- j - Move cursor down
- h - Move cursor left

- l - Move cursor right

You need to be in the command mode to move within a file. The default keys for navigation are mentioned below else; You can **also use the arrow keys on the keyboard**.

Saving and Closing the file

- Shift+zz - Save the file and quit
- :w - Save the file but keep it open
- :q - Quit without saving
- :wq - Save the file and quit

You should be in the **command mode to exit the editor and save changes** to the file.

Devops zajęcia nr 3 (21.03.2020r.)

POWTÓRZENIE:

$ docker run redis

$ docker exec -it 6e6b8cc99587 sh

$ redis-cli

$ set message Arka

I w innym terminalu:

$ docker exec -it 6e6b8cc99587 sh

$ redis-cli

$ get message

→ “Arka”

- https://hub.docker.com/_/nginx

FROM nginx:alpine

RUN echo "<html><body>Hello Arka Gdynia</body></html>" > /usr/share/nginx/html/hello.html

- ➔ Wchodzimy tam gdzie ten Dockerfile zapisaliśmy i „docker build .”
- ➔ Dostaliśmy ID i potem
- ➔ docker run -p 90:80 546acc08a2f6
- ➔ a potem np możemy na naszego dockerhuba to wysłać i push tak jak na poprzednich zajęciach

DOCKER COMPOSE:

```
version: '3'


services:
  my-redis-server:
    image: 'redis'
  my-webapp:
    build: .
    ports:
      - "9090:8080"
```

$ docker-compose up

Jak coś zmienimy, żeby przebudował to tak:
$ docker-compose up --build

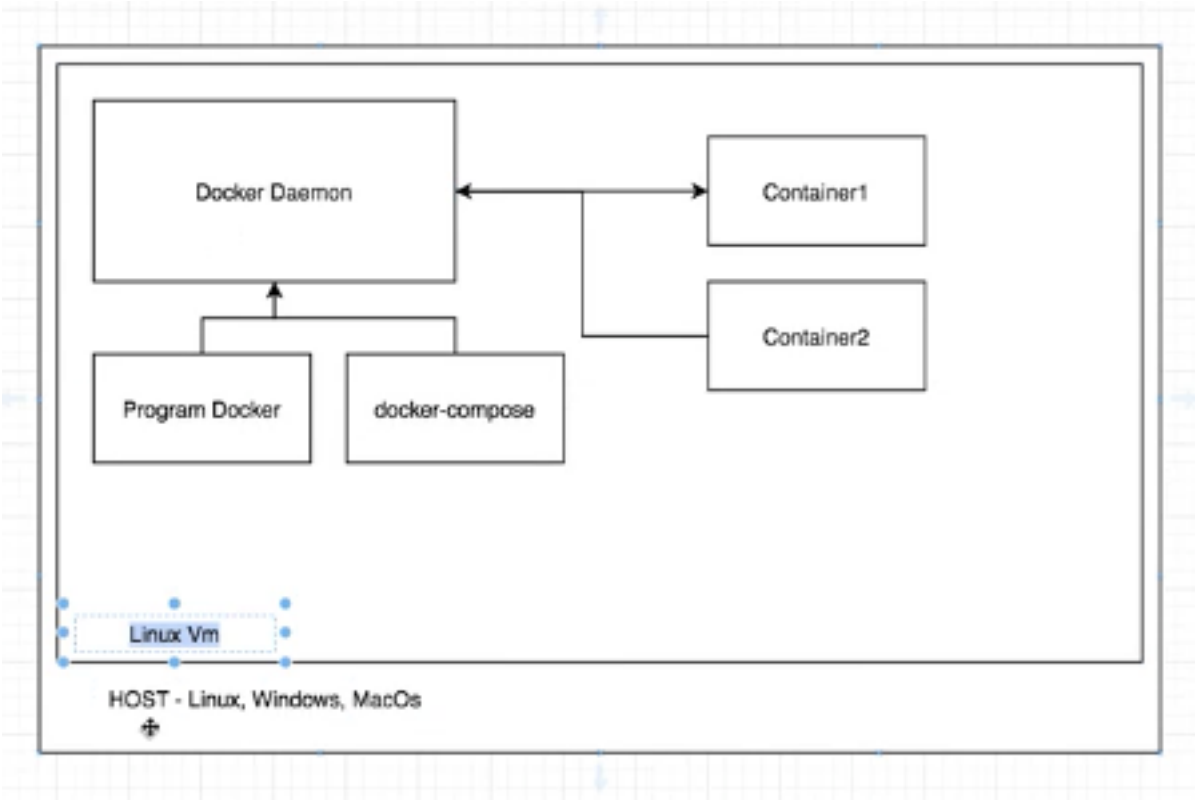
$ docker-compose ps

```
(base) Jakubs-MacBook-Pro:multicontapp jakublemka$ docker-compose ps
             Name                           Command               State           Ports
-------------------------------------------------------------------------------------------------
multicontapp_my-redis-server_1   docker-entrypoint.sh redis ...   Up      6379/tcp
multicontapp_my-webapp_1         docker-entrypoint.sh npm r ...   Up      0.0.0.0:9090->8080/tcp
```

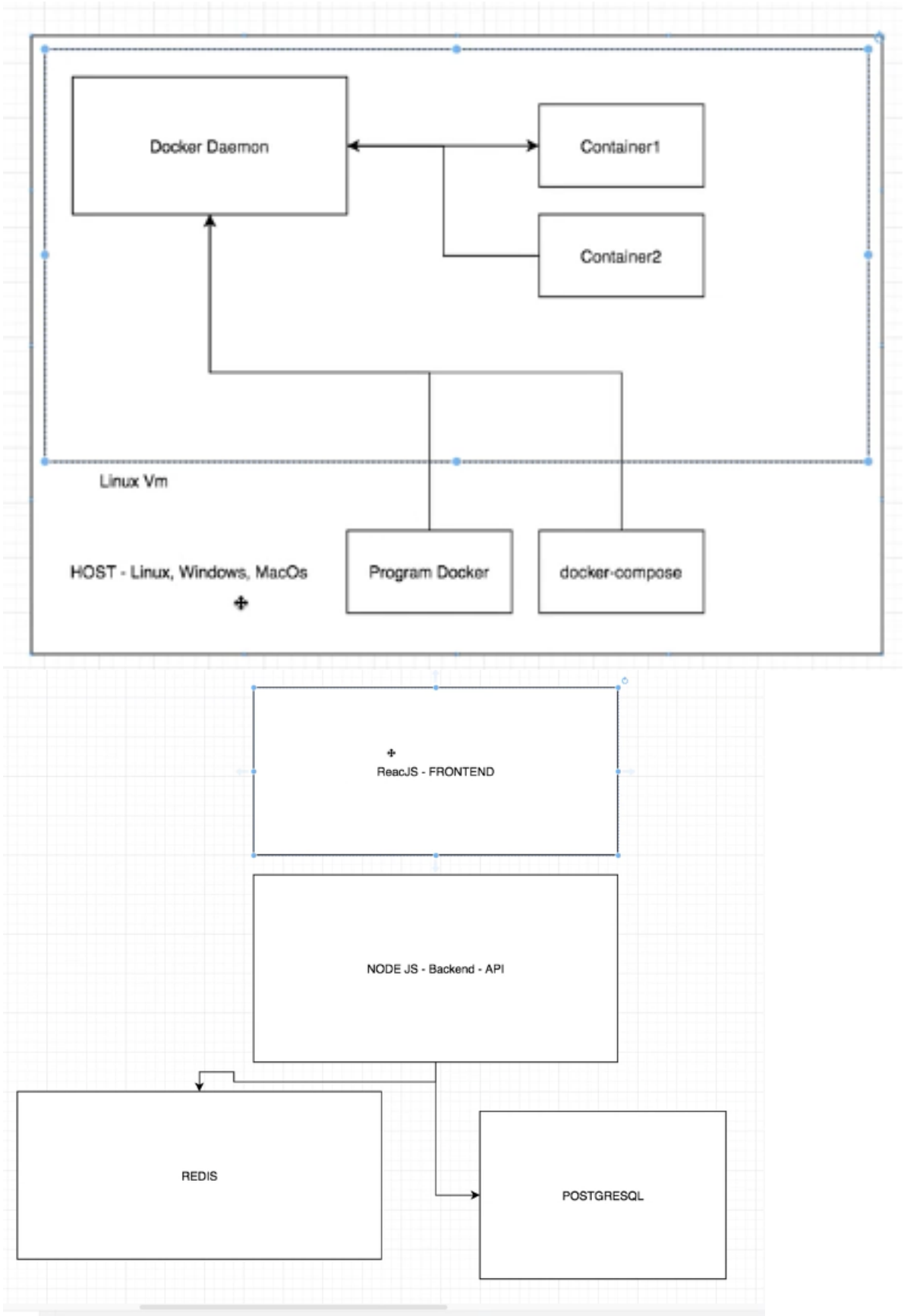
$ docker-compose down

Devops zajęcia nr 4 (05.04.2020r.)

Jak jest linux to tak:



A jak Windows/Mac to tak:



Jak jestem w folderze multicontapp to:

docker-compose up --build

i działa na localhost:9090
Np.: curl localhost:9090

process.exit(0); → na linux zakończenie ze statusem zero (0) to znaczy zakończenie BEZ BŁĘDU. Ze statusem innym niż zero – jakiś błąd

jak mamy politykę „restart:always” w docker-compose a w index.js „proces.exit(0)
to wtedy

curl localhost:9090

curl: (52) Empty reply from server

I za każdym *getem* na nowo się stawia appka

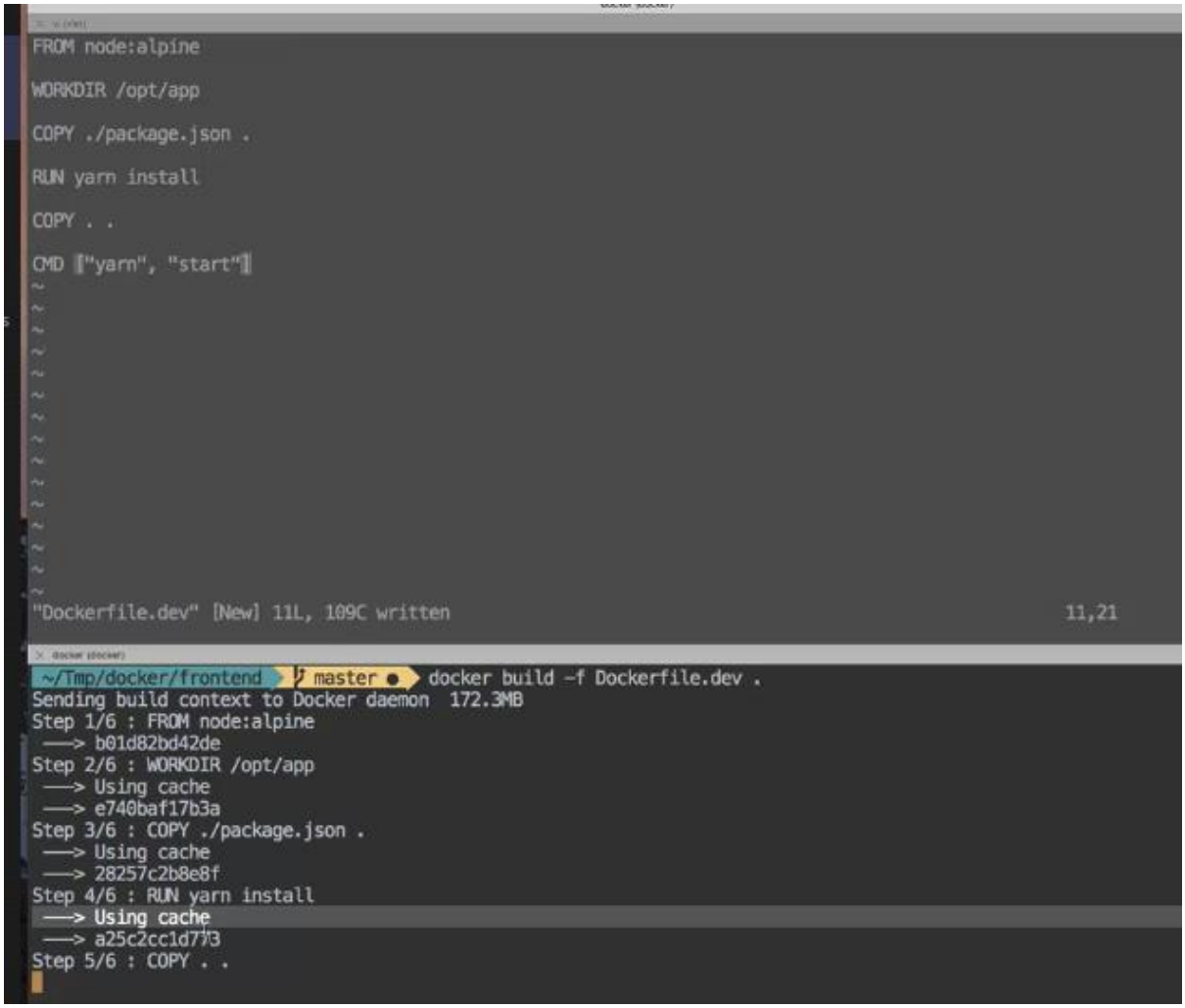
Polityki:

- restart: "always"
- restart: "no"
- restart: "on-failure"

ZADANIE:
Wielo (dwu) kontenerowa aplikacja:

- redis (cache)
- node (backend)

Backend:
Liczy NWD (l1, l2) – i wyliczone wcześniej (tj. do tej pory) wartości przechowuje w cache
Jeśli nie ma to wylicza NWD dla zadanych liczb.



We frontendzie:

docker build -f Dockerfile.dev .

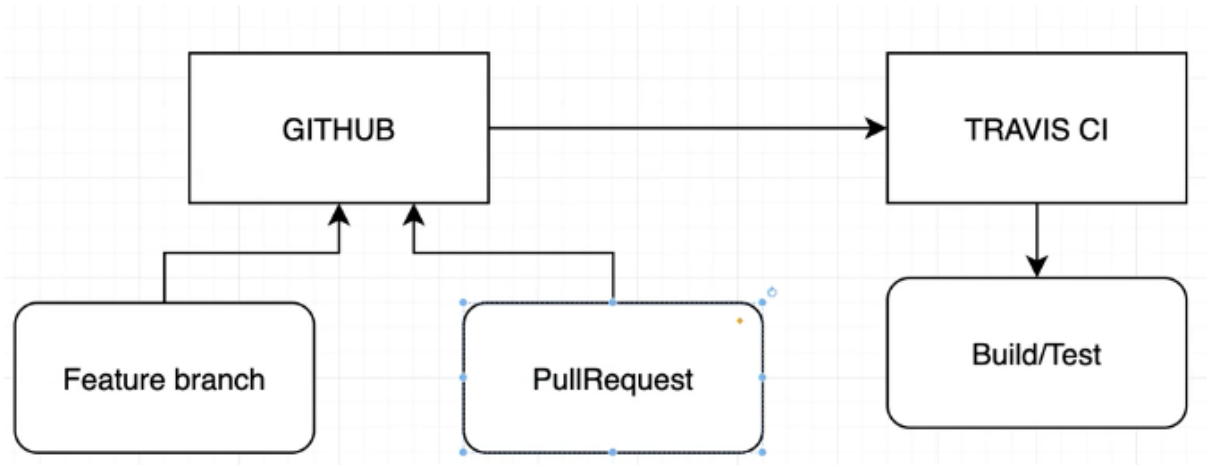
na koniec bedzie

Successfully built 2cd5cf06ca73

I my robimy:

docker run -it 2cd5cf06ca73 //to -it potrzebujemy bo jest póki co błąd

Devops zajęcia nr 5 (19.04.2020r.)



Dla frontend nowy dockerfile:

```
FROM node:alpine as builder

WORKDIR /opt/app

COPY ./package.json .

RUN yarn install

COPY . .

RUN yarn build

FROM nginx

COPY --from=builder /opt/app/build /usr/share/nginx/html
```

najpierw robi sie yarn build (w kontenerze)

a potem te statyczne pliki sa wrzucane do folderu, ktory nginx domyslnie hostuje pliki

→ Successfully built 7000626fefec

docker run -p 90:80 7000626fefec

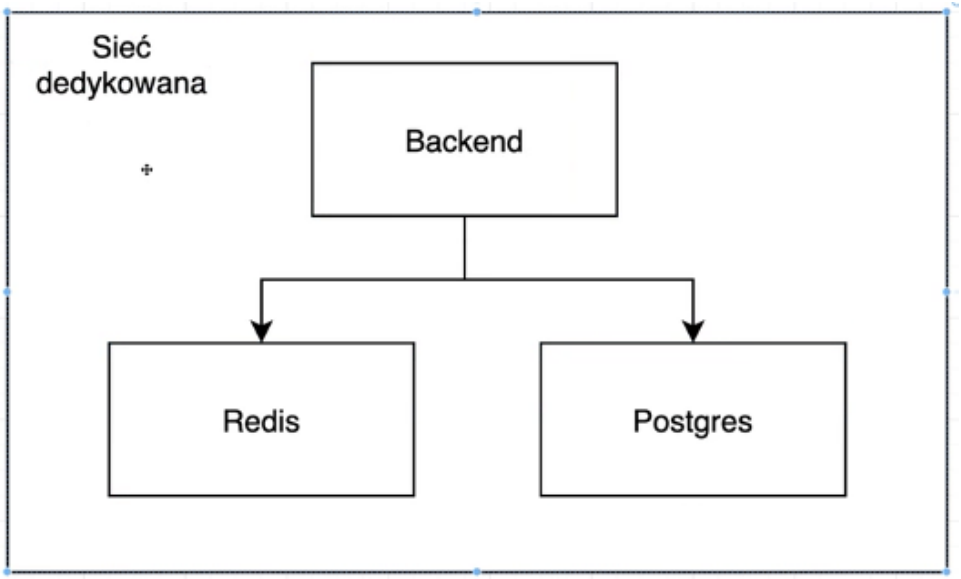
I teraz możemy wejść w port 90 i tam nasza apka będzie

W tym Dockerfile.dev możemy tez zrobić docker build. I potem
Docker run ID_BUILD yarn test

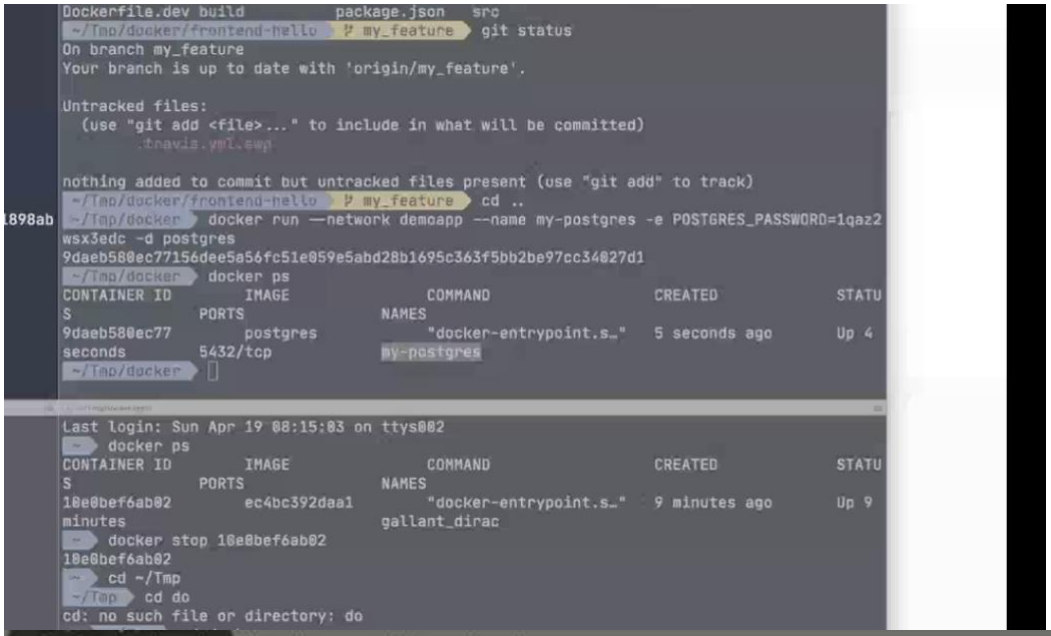
Podpięcie z travisem:

```
sudo: required

services:
  - docker

before_install:
  - docker build -t lemjak/frontend -f ./frontend/Dockerfile.dev ./frontend

script:
  - docker run -e CI=true lemjak/frontend yarn test
```

Teraz robimy persystentą bazę danych (Postgres)



- https://hub.docker.com/_/postgres
    - docker run –network demoapp --name my-postgres -e POSTGRES_PASSWORD=arka -d postgres



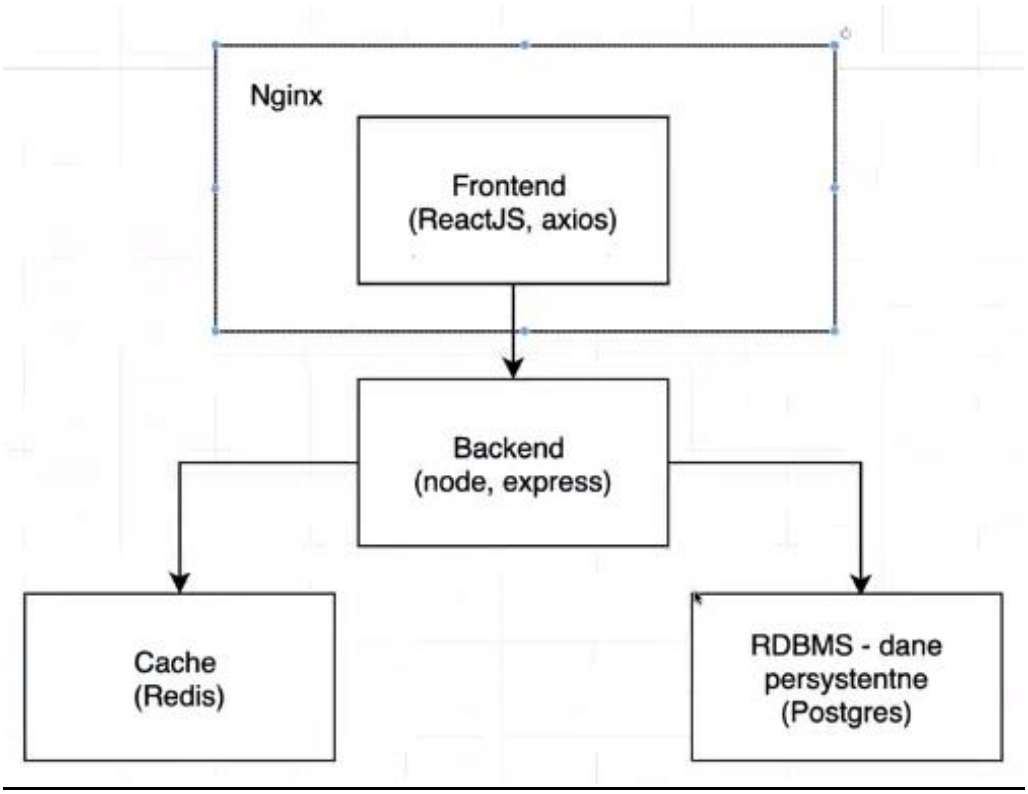

```
1. Zaznajomienie się z poleceniami
  docker network create nazwa
  docker network ls
  docker network rm nazwa
  docker network inspect nazwa

2. Uruchomienie kontenerów Redis i Postgresql przyłączonych do jednej sieci

3. Napisanie aplikacji webowej (node + express + redis + pg) łączącej się do
kontenerów Postgres i Redis

4. Wzbogacenie aplikacji o nast. funkcjonalność:
 - aplikacja na odpowiednich endpoitach
    a) liczy NWD dwóch liczb
    b) zwraca wszystkie dotychczas uzyskane wyniki tj. same rezultaty, wyniki
    obliczeń (wstawiane i przechowywane do postgresa)
Cache przechowuje obliczenia tak jak do tej pory.

yarn add {pg, express, cors, redis, body-parser, nodemon}

  "scripts": {
    "dev": "nodemon",
    "start": "node index.js"
  },

docker run --rm --name my-postgres -e POSTGRES_PASSWORD=123qaz123qaz --network
my-demo-app postgres

docker run --rm  --name my-redis --network my-demo-app redis

docker run --env REDIS_HOST=my-redis --rm --name my-backend --network my-demo-app
-v /opt/app/node_modules -v $(pwd):/opt/app -e PGHOST=my-postgres -e
PGUSER=postgres -e PGDATABASE=postgres -e PGPASSWORD=123qaz123qaz -e PGPORT=5432
DockerID
```

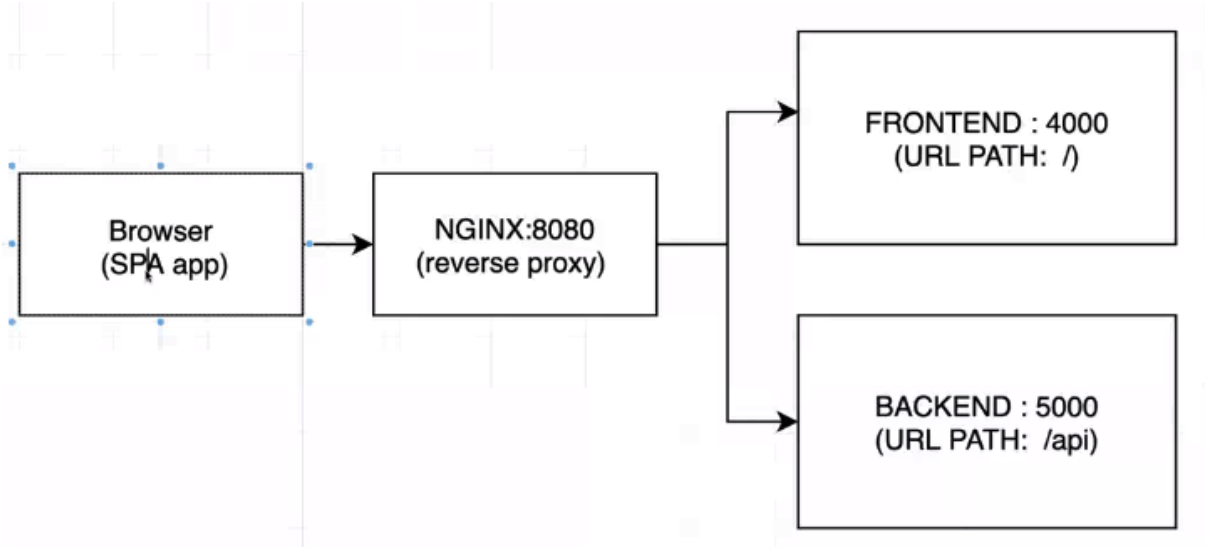
WYKONANIE:

- Stworzyć sieć
  - docker network create **my-network**

- Zbudować index.js poprzez Dockerfile.dev
  - docker build -f Dockerfile.dev .
    - otrzymujemy **CONTAINER_ID**
- Postawić redisa
  - docker run --rm --name **my-redis** --network **my-network** redis
- Postwaić postgresa
  - docker run --rm --name **my-postgres** --network **my-network** -e POSTGRES_PASSWORD=arka postgres
- Odpalić wszystko z odpowiednimi nazwami
  - docker run --rm --name **my-backend** --network **my-network** -e REDIS_HOST=**my-redis** -e REDIS_PORT=6379 -e PGHOST=**my-postgres** -e PGPORT=5432 -e PGUSER=postgres -e PGPASSWORD=arka -e PGDATABASE=postgres -p 5000:5000 **CONTAINER_ID**
- możemy odpytywać API na zasadzie:
  - curl localhost:5000/gcd/6/9
  - curl localhost:5000/gcd/values

Devops zajęcia nr 6 (26.04.2020r.)



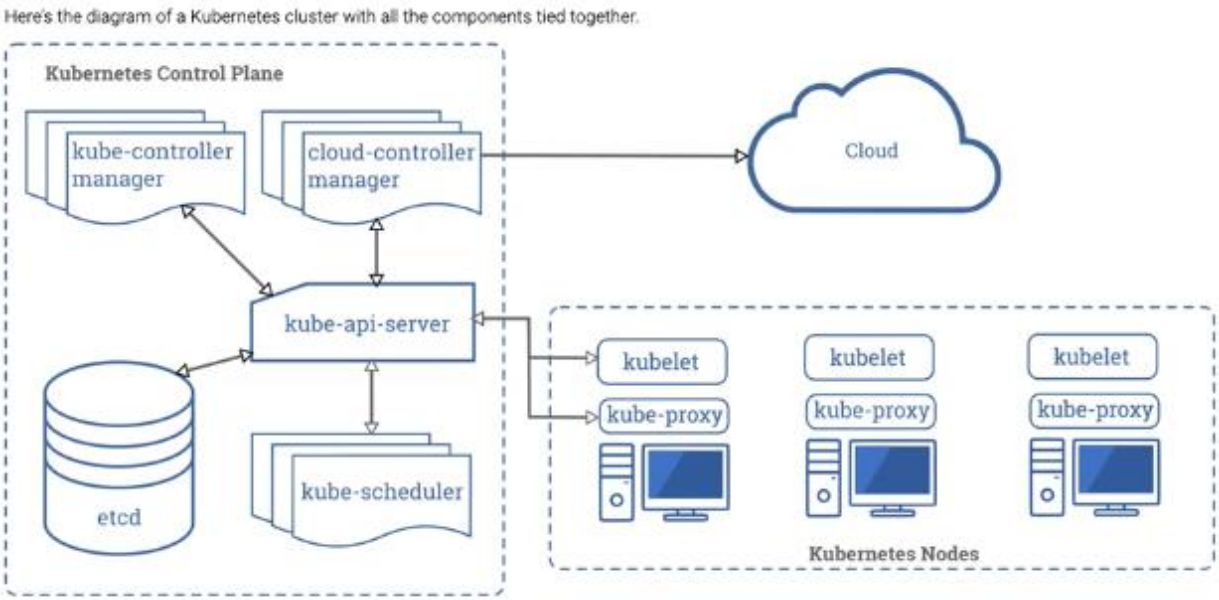


docker-compose up –-build

docker-compose down

Devops zajęcia nr 7 (10.05.2020r.)

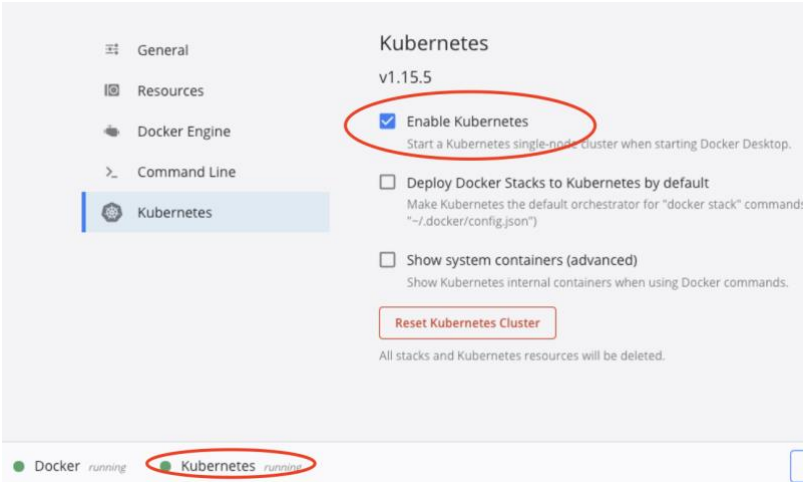
KUBERNETES (K8s)



- Orkiestrator – poziom wyżej niż Docker w budowaniu **skalowanych** aplikacji
- Jak sama nazwa wskazuje orkiestrator zajmuje się udostępnianiem, **skalowaniem**
- Zarządzenie kontenerami

- Główny podział:
  - Control Plane (inaczej Kubernetes Master)
    - etdc (baza danych klucz-wartość) – tutaj przechowywana jest cała konfiguracja naszego klastra
    - kube-api-server – punkt, do którego komunikują się pozostałe komponenty, także z węzłów roboczych. Pełni podobną rolę jak daemon Dockera
    - kontrolery – na przykład my będziemy korzystać z kontrolera replikacji, który będzie dbał o to, by nasza aplikacja zachowała skalowalność
  - Nody (tutaj node jest rozumiany jako serwer, to są jakieś instancje maszyny/serwera)
    - kubelet – wykonawca wszystkich poleceń, które przychodzą z control plane'a. Kubelet fizycznie odpowiada, żeby na danym węźle uruchomić naszą aplikację.
    - kube-proxy – proces związany z adresacją IP. Wykorzystując mechanizmy związane z IP tables dokonywać różnych wpisów na nodach

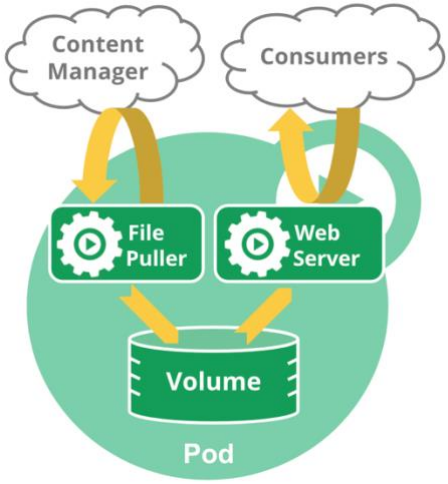
Uruchamiamy sobie z poziomu Docker Desktop:




```
(base) Jakubs-MacBook-Pro:multi3 jakublemka$ kubectl cluster-info
Kubernetes master is running at https://kubernetes.docker.internal:6443
KubeDNS is running at https://kubernetes.docker.internal:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

$ kubectl config current-context

docker-desktop



Pod – najmniejsza jednostka, w którą pakujemy aplikacji i zarządzamy kubernetesem:
→ https://kubernetes.io/docs/concepts/workloads/pods/pod/

Taki Pod tworzymy poprzez definiowanie plików.

$ kubectl get pods

No resources found.

Jak zmienimy przestrzeń nazw to będą:

```
(base) Jakubs-MBP:tmp jakublemka$ kubectl get pods --namespace=kube-system
NAME                                     READY   STATUS    RESTARTS   AGE
coredns-5c98db65d4-m8tf8                 1/1     Running   0          46m
coredns-5c98db65d4-sc2wn                 1/1     Running   0          46m
etcd-docker-desktop                      1/1     Running   0          45m
kube-apiserver-docker-desktop            1/1     Running   0          45m
kube-controller-manager-docker-desktop   1/1     Running   0          45m
kube-proxy-mfkqs                         1/1     Running   0          46m
kube-scheduler-docker-desktop            1/1     Running   0          45m
```

Tworzymy pod-template.yml

```yaml
apiVersion: v1

kind: Pod

metadata:
  name: myapp-pod
  labels:
    app: myapp
    type: frontend

spec:
  containers:
    - name: my-nginx-container
      image: nginx
```

$ kubectl create -f pod-template.yml

pod/myapp-pod created

$ kubectl get pod

NAME READY STATUS RESTARTS AGE

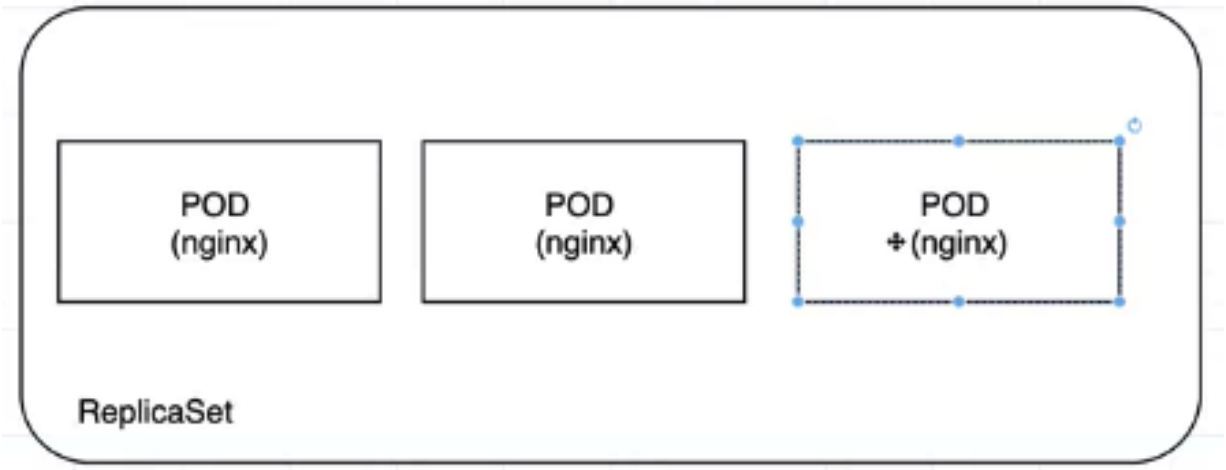
myapp-pod 1/1 Running 0 92s

```
(base) Jakubs-MBP:tmp jakublemka$ kubectl describe pod myapp-pod
Name:         myapp-pod
Namespace:    default
Priority:     0
Node:         docker-desktop/192.168.65.3
Start Time:   Sun, 10 May 2020 10:10:45 +0200
Labels:       app=myapp
              type=frontend
Annotations:  <none>
Status:       Running
IP:           10.1.0.5
Containers:
  my-nginx-container:
    Container ID:   docker://0bf5ffe2dff66b2953d0055f51ab851fa60ba1ca74c2c080ad69766db2f1868e
    Image:          nginx
    Image ID:       docker-pullable://nginx@sha256:86ae264c3f4acb99b2dee4d0098c40cb8c46dcf9e1148f05d3a51c4df6758c12
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Sun, 10 May 2020 10:11:06 +0200
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-djjpk (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  default-token-djjpk:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-djjpk
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type    Reason     Age    From                     Message
  ----    ------     ----   ----                     -------
  Normal  Scheduled  3m54s  default-scheduler        Successfully assigned default/myapp-pod to docker-desktop
  Normal  Pulling    3m53s  kubelet, docker-desktop  Pulling image "nginx"
  Normal  Pulled     3m33s  kubelet, docker-desktop  Successfully pulled image "nginx"
  Normal  Created    3m33s  kubelet, docker-desktop  Created container my-nginx-container
  Normal  Started    3m33s  kubelet, docker-desktop  Started container my-nginx-container
```

Ta ostatnia sekcja **events** jest ważna, bo tak jest wypisane co po kolei się działo z tym kontenerem.

Usuwanie:

$ kubectl delete pod myapp-pod
pod "myapp-pod" deleted




```
apiVersion: apps/v1
```

```yaml
kind: ReplicaSet

metadata:
  name: myapp-rs
  labels:
    app: myapp
    type: frontend

spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: frontend
    spec:
      containers:
        - name: my-nginx-container
          image: nginx
  replicas: 3
  selector:
    matchLabels:
      type: frontend
```

ReplicaSet upewnia się, że dany stan naszego rozwiązania jest taki jak go opisaliśmy – czy mamy odpowiednią liczbę podów. Elementem nadrzędnym wykorzystującym ReplicaSet i dającym więcej możliwości jest deploymnet.

$ kubectl create -f rs-template.yml

replicaset.apps/myapp-rs created

(base) Jakubs-MBP:tmp jakublemka$ kubectl get pods

NAME	READY	STATUS	RESTARTS	AGE
myapp-pod	1/1	Running	0	32m
myapp-rs-2qgv5	0/1	ContainerCreating	0	3s
myapp-rs-ncpdz	0/1	ContainerCreating	0	3s

(base) Jakubs-MBP:tmp jakublemka$ kubectl delete pod myapp-pod

```
pod "myapp-pod" deleted
(base) Jakubs-MBP:tmp jakublemka$ kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
myapp-rs-2qgv5   1/1     Running   0          47s
myapp-rs-ncpdz   1/1     Running   0          47s
myapp-rs-zms7p   1/1     Running   0          6s
```

Czyli mieliśmy ten myapp-pod, odpaliliśmy nasz replicaset i 2 się stworzyły. Jak usunęliśmy nasz myapp-pod automatycznie trzeci się od razu stworzył.

```
$ kubectl get rs
NAME      DESIRED   CURRENT   READY   AGE
myapp-rs   3         3         3       2m52s
```

```
$ kubectl get all
NAME              READY   STATUS    RESTARTS   AGE
pod/myapp-rs-2qgv5   1/1     Running   0          3m5s
pod/myapp-rs-ncpdz   1/1     Running   0          3m5s
pod/myapp-rs-zms7p   1/1     Running   0          2m24s

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   105m

NAME                    DESIRED   CURRENT   READY   AGE
replicaset.apps/myapp-rs   3         3         3       3m5s
```

Kiedy używanie takich replicasetów ma sens? → Aplikacja musi być bezstanowa, żeby dała się skalować. Do tego należy pisać aplikacje niemutowalne, żeby obraz był niemutowalny.

```
$ kubectl get rs
```

NAME DESIRED CURRENT READY AGE

myapp-rs 3 3 3 13m

$ kubectl **scale** --replicas=5 replicaset myapp-rs

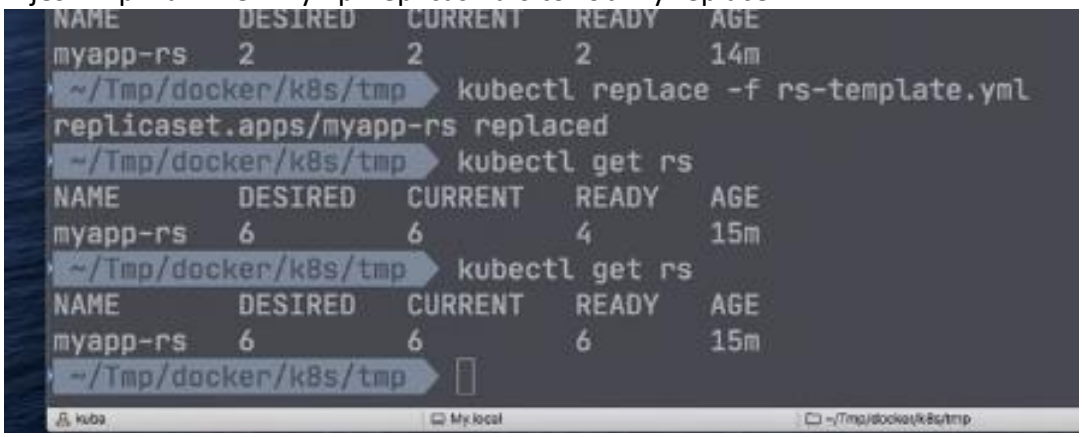
replicaset.extensions/myapp-rs scaled

$ kubectl get rs

NAME DESIRED CURRENT READY AGE

myapp-rs 5 5 5 13m

A jeśli w pliku zmienimy np. replicas na 6 to robimy replace:



deploment-template.yml

```yaml
apiVersion: apps/v1

kind: Deployment

metadata:
  name: myapp-rs
  labels:
    app: myapp
    type: frontend

spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: frontend
    spec:
```

```
    containers:
      - name: my-nginx-container
        image: nginx
replicas: 3
selector:
  matchLabels:
    type: frontend
```

On się różni tylko tym, że **kind jest teraz Deployment**

$ kubectl create -f deployment-template.yml

deployment.apps/myapp-rs created

(base) Jakubs-MBP:tmp jakublemka$ kubectl get deployments

NAME READY UP-TO-DATE AVAILABLE AGE

myapp-rs 3/3 3 3 12s

$ kubectl get all

NAME READY STATUS RESTARTS AGE

pod/myapp-rs-6b6ddd9b76-9jgtn 1/1 Running 0 31s

pod/myapp-rs-6b6ddd9b76-dcbmx 1/1 Running 0 31s

pod/myapp-rs-6b6ddd9b76-llz2d 1/1 Running 0 31s

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

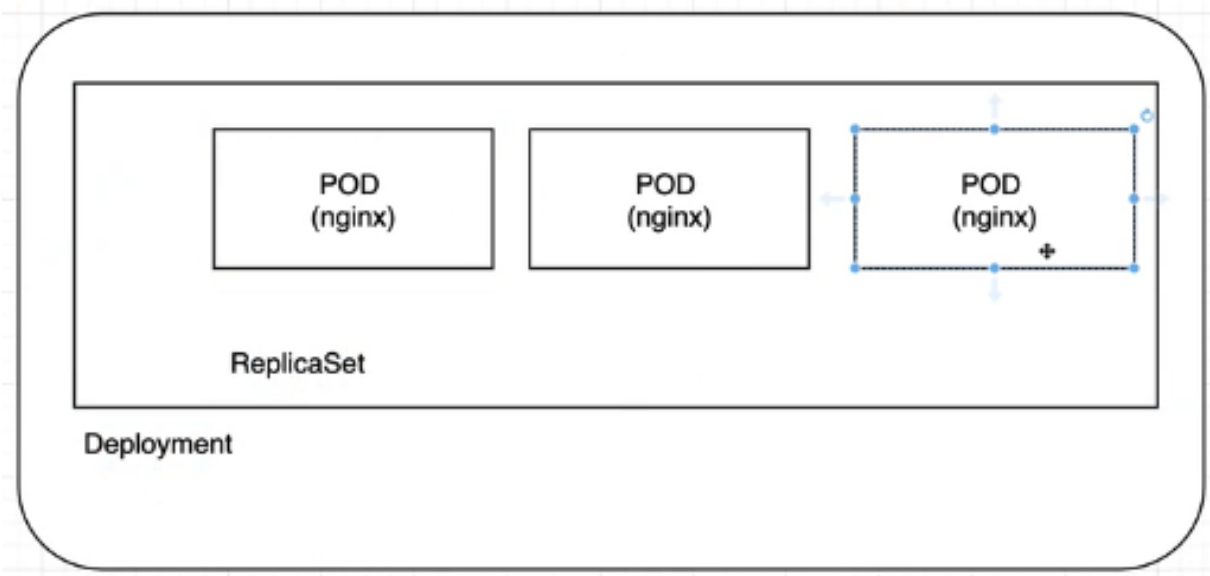
service/kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 122m

NAME READY UP-TO-DATE AVAILABLE AGE

deployment.apps/myapp-rs 3/3 3 3 31s

NAME DESIRED CURRENT READY AGE

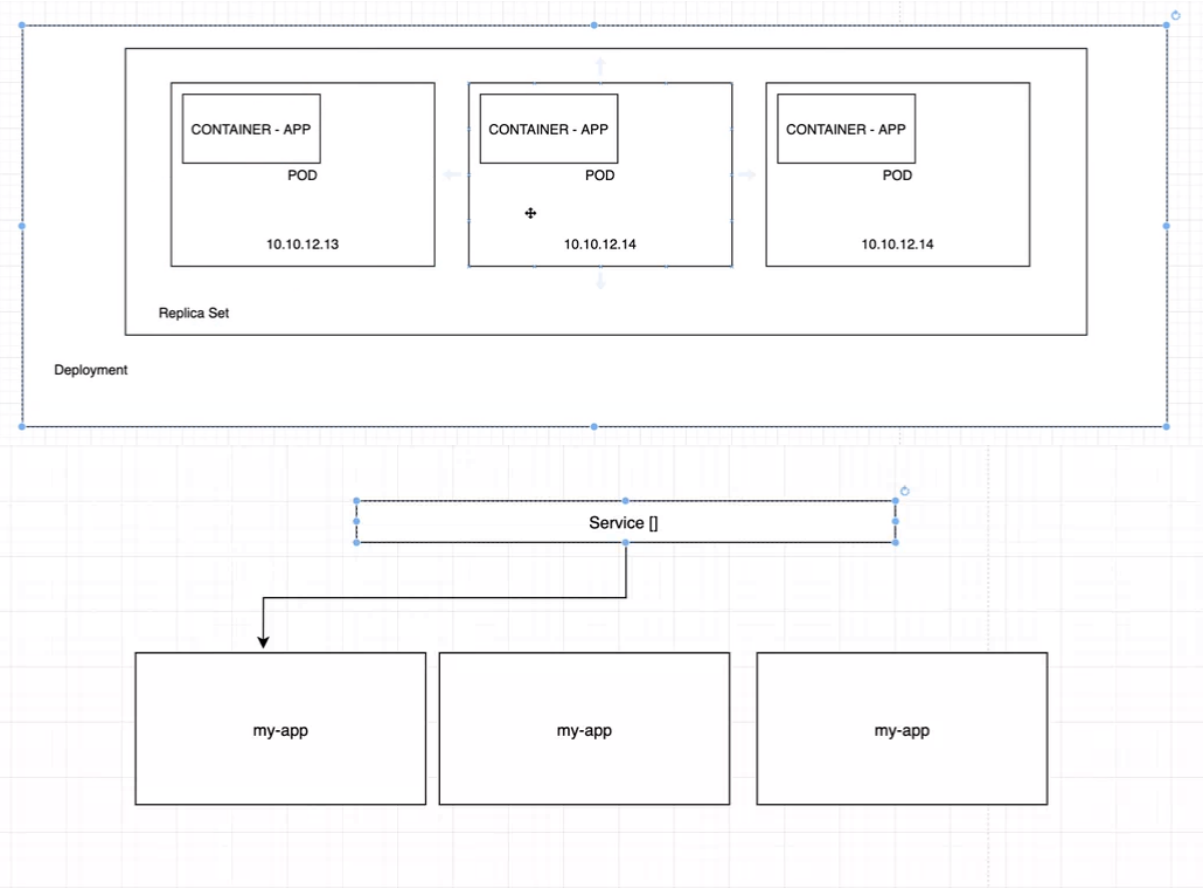
replicaset.apps/myapp-rs-6b6ddd9b76 3 3 3 31s



Najważniejsze komendy z zajęć do opanowania:

```
kubectl replace -f rs-template.yml
kubectl scale --replicas=6 -f rs-template.yml
kubectl scale --replicas=6 replicaset rs-name

kubectl get {pods, rs, deployments}

kubectl describe {pod, rs, deployment} name

kubectl delete pod myapp-pod

kubectl get all
```

Devops zajęcia nr 8 (24.05.2020r.)



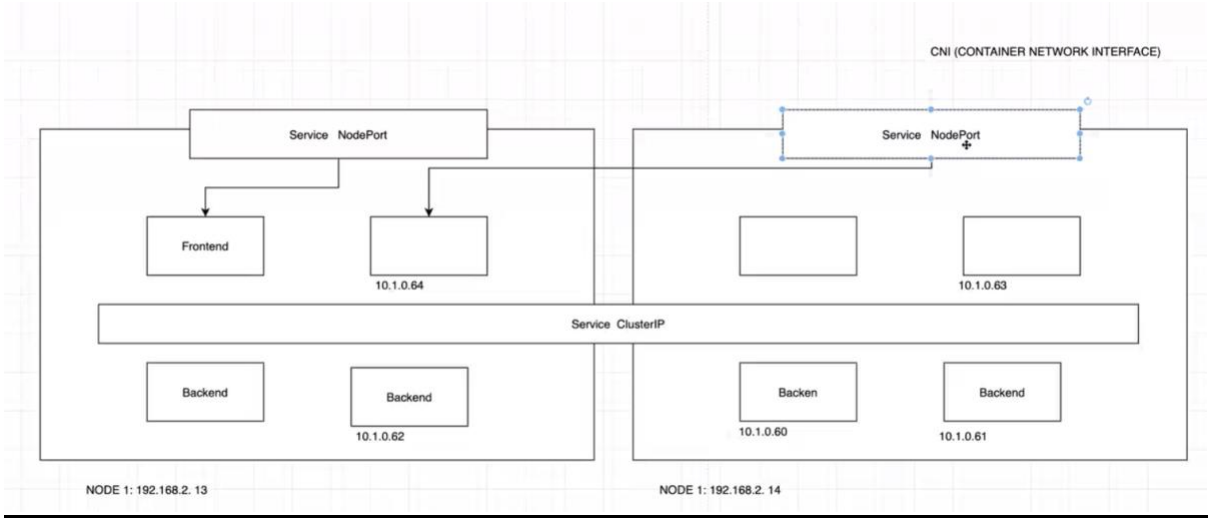
(base) Jakubs-MBP:k8sLab8 jakublemka$ kubectl create -f nginx-deployment.yml
deployment.apps/my-nginx-deployment created

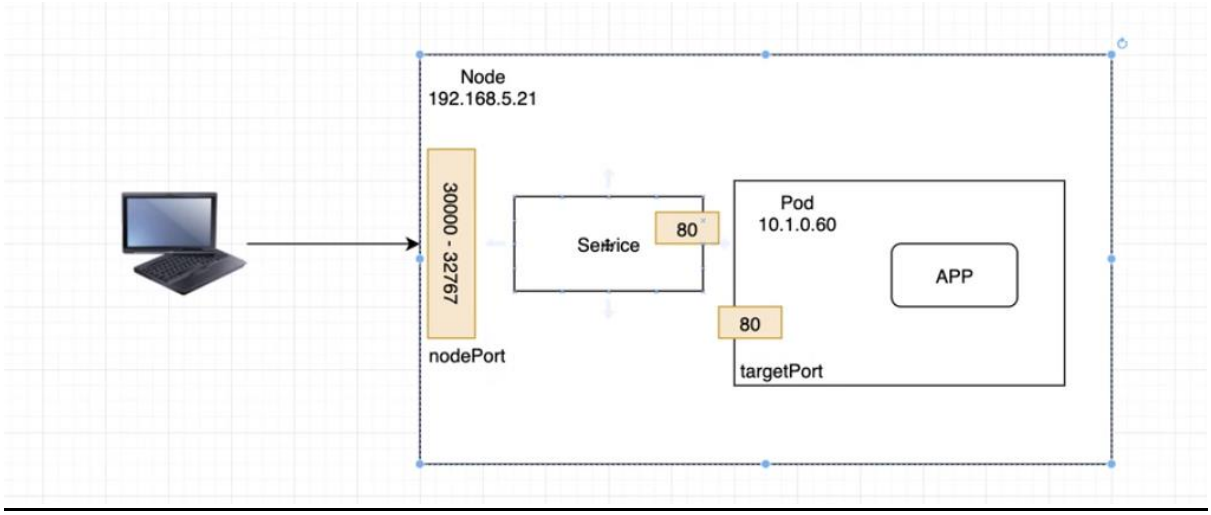
(base) Jakubs-MBP:k8sLab8 jakublemka$ kubectl get deploy

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
my-nginx-deployment	3/3	3	3	95s
myapp-rs	3/3	3	3	13d

kubectl describe deploy my-nginx-deployment

SERWISY – byty, które służą by udostępnić nasz klaster na zewnątrz:





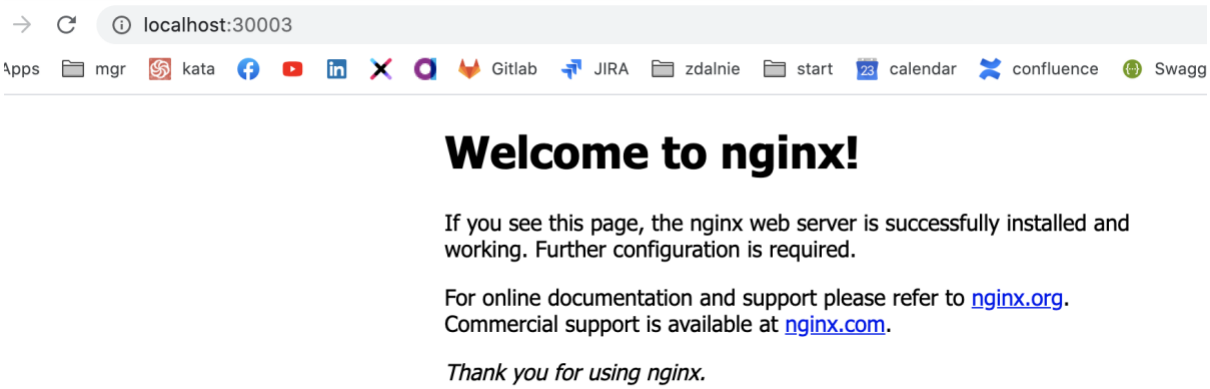
devopsug > k8sLab8 > nginx-deployment.yml

```yaml
apiVersion: apps/v1

kind: Deployment

metadata:
  name: my-nginx-deployment
  labels:
    app: my-nginx
    type: my-frontend

spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: my-nginx
        type: my-frontend
    spec:
      containers:
        - name: my-nginx-container
          image: nginx
  replicas: 3
  selector:
    matchLabels:
      app: my-nginx
      type: my-frontend
```

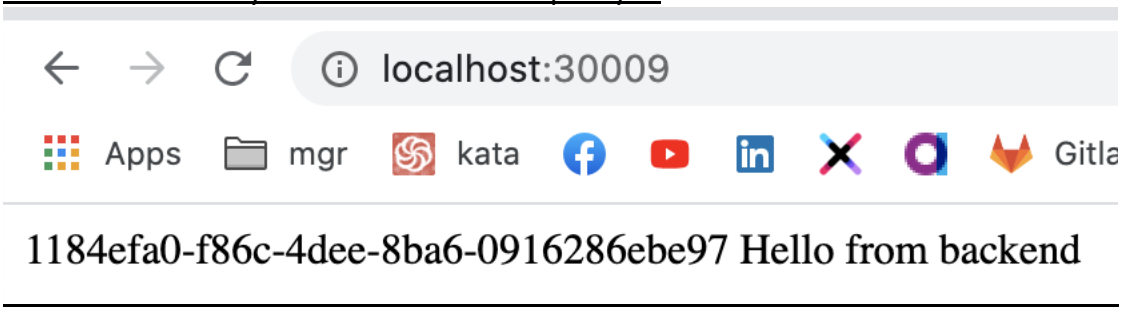
devopsug > k8sLab8 > myapp-service-nodeport.yml

```yaml
apiVersion: v1

kind: Service

metadata:
  name: myapp-service

spec:
  type: NodePort
  ports:
    - targetPort: 80
      port: 80
      nodePort: 30003

  selector:
    app: my-nginx
    type: my-frontend
```

Localhost:30003

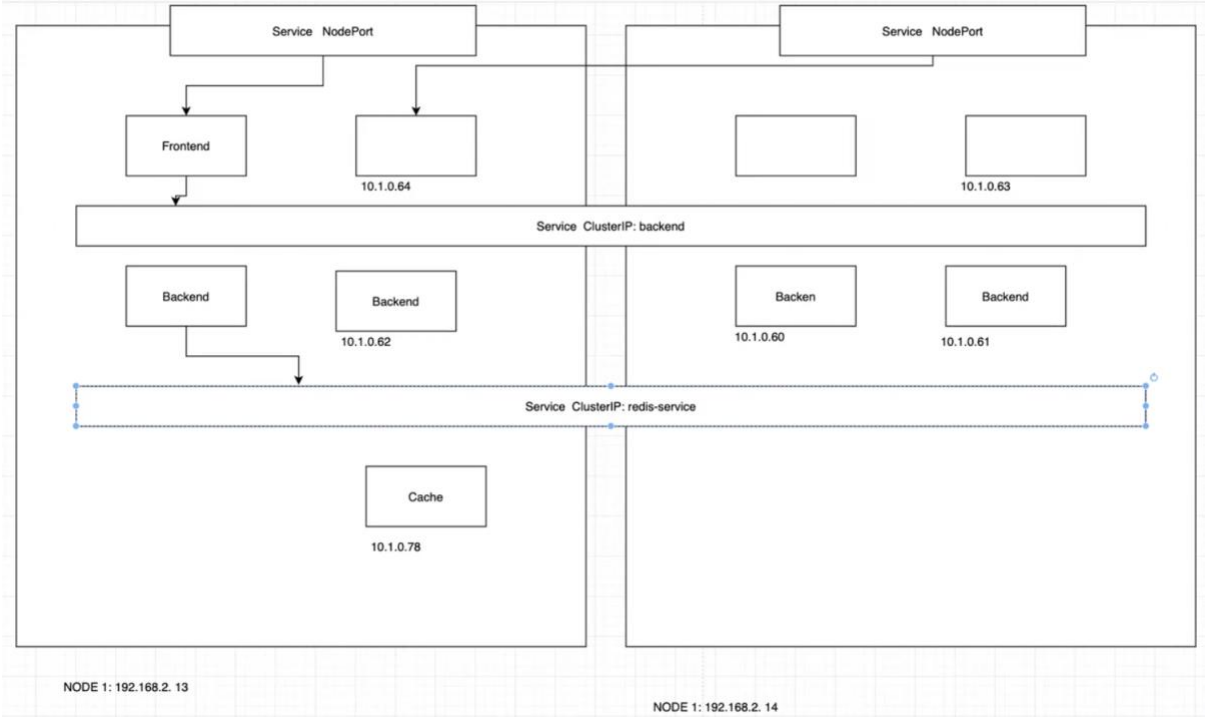


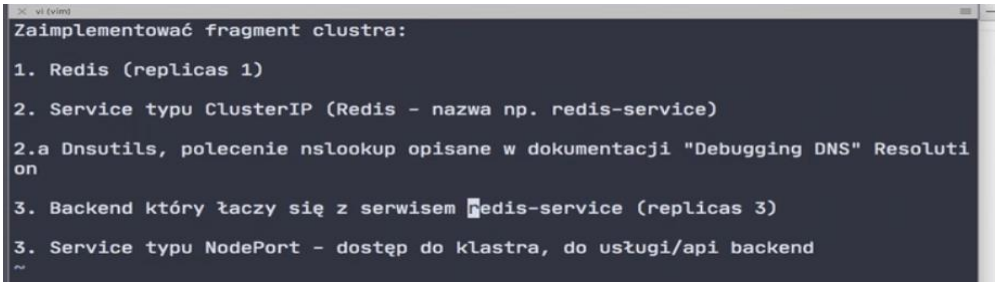
Postawiliśmy deployment + service

- docker build -t jakublem/mybackend .
- docker push jakublem/mybackend:latest
- kubectl create -f mybackend-deployment.yml
- kubectl create -f mybackend-service-nodeport.yml
- 

kubectl exec -ti dnsutils -- nslookup redis-service
Server: 10.96.0.10
Address: 10.96.0.10#53

Name: redis-service.default.svc.cluster.local
Address: 10.111.156.242



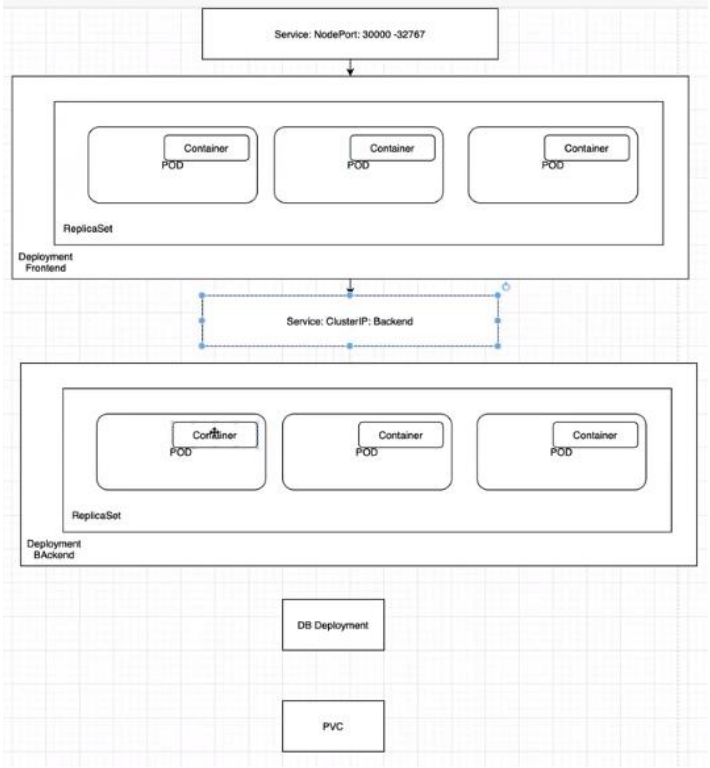


← zadanie

Devops zajęcia nr 9 (7.06.2020r.)

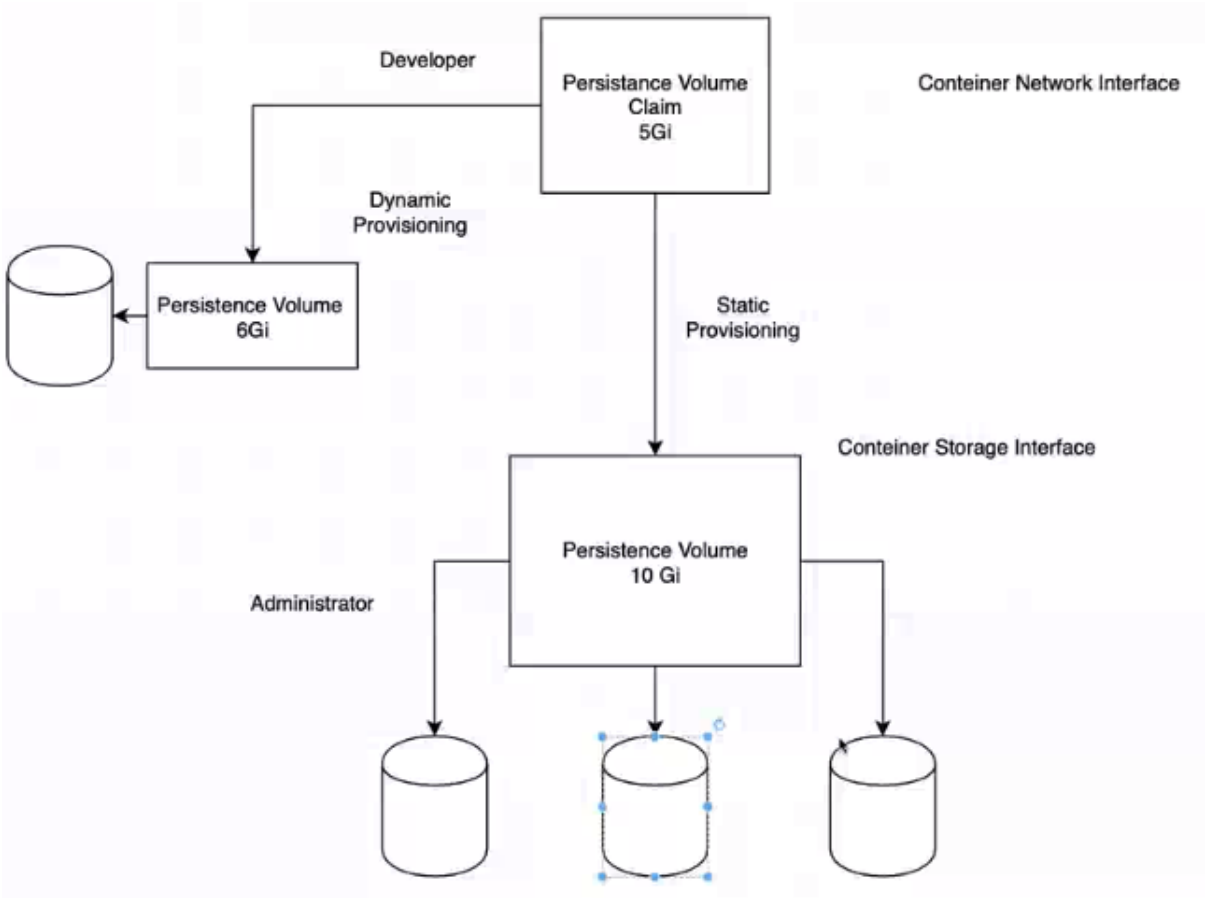
Podsumowanie:

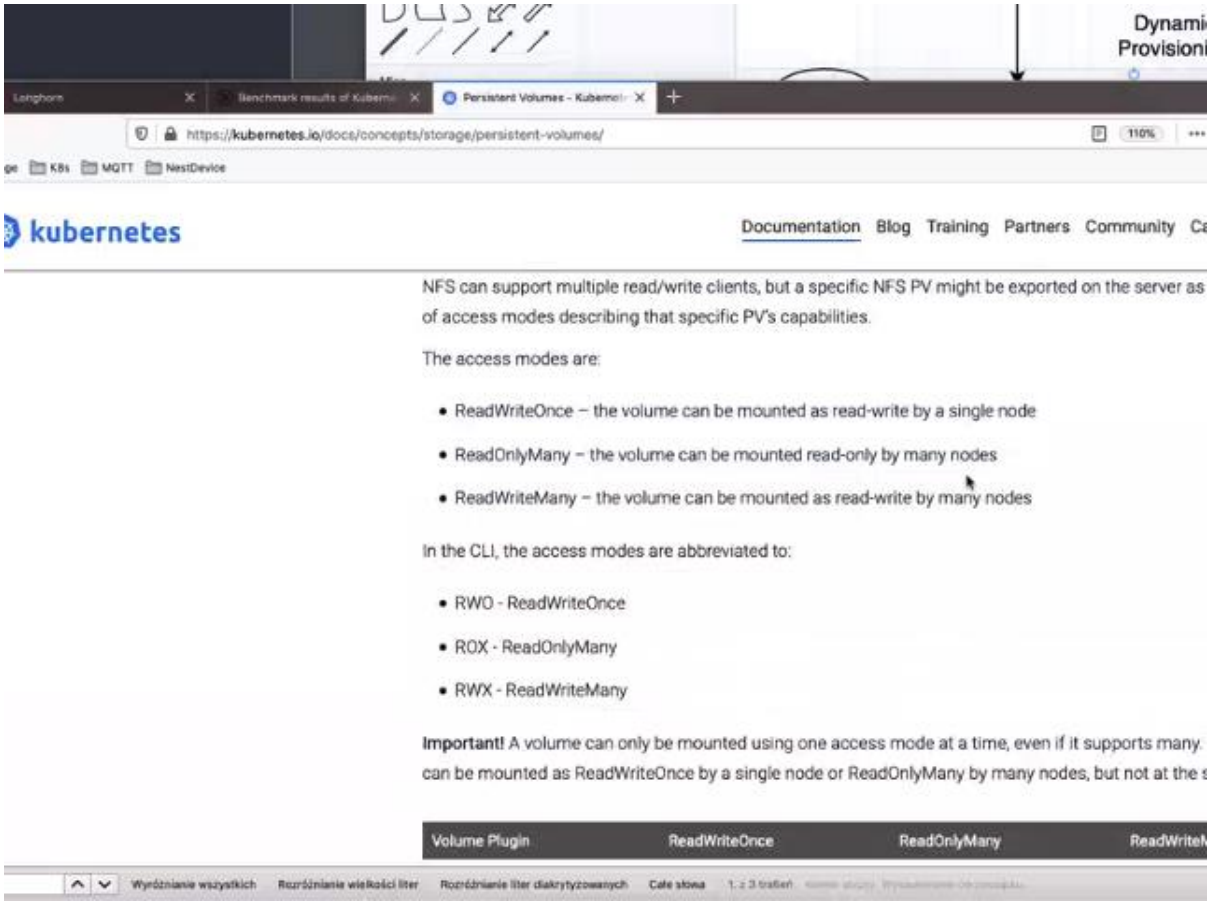
- POD – najmniejszy byt w klastrze, podstawowa jednostka
- Replikasety – w ten sposób pody potrafią się replikować
- Deployment

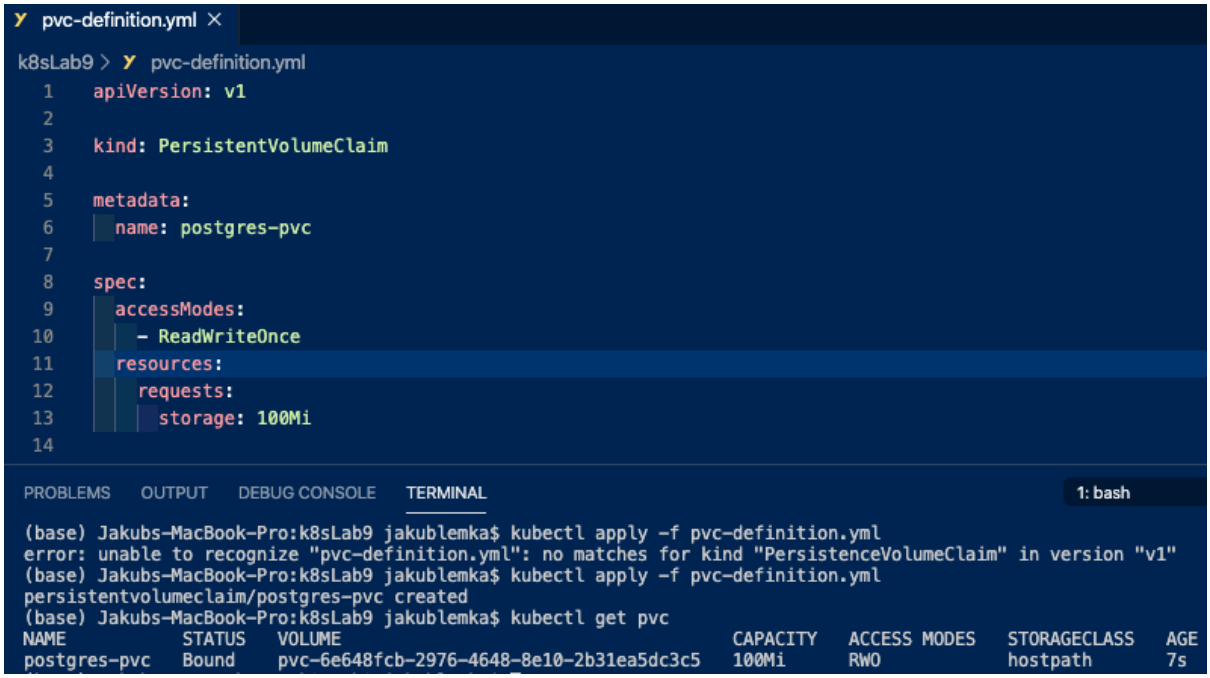


Teraz potrzebujemy persystencji (PVC). W tym świecie ulotnych bytów chcemy coś zapisywać.

Persistence Volume odwołuje się na jakimś niskim poziomie do jakiś fizycznych danych.







Mi – to są **mebibajty.** → https://pl.wikipedia.org/wiki/Przedrostek_dw%C3%B3jkowy
(mebibyte or MiB is the newer unit of measurement of size.
1MB = 1000KB or 1024 KB
1MiB = **only and only** 1024KB)

pvc-definition.yml | test-pvc-deployment.yml

k8sLab9 > test-pvc-deployment.yml

```yaml
apiVersion: v1

kind: Pod

metadata:
  name: test-pvc-pod

spec:
  containers:
    - image: alpine
      name: alpine
      command: ["/bin/sh", "-c"]
      args: ["echo Hello > /opt/data/hello.txt"]

      volumeMounts:
        - mountPath: /opt/data
          name: data-volume

  volumes:
    - name: data-volume
      persistentVolumeClaim:
        claimName: postgres-pvc
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: bash

```
(base) Jakubs-MacBook-Pro:k8sLab9 jakublemka$ kubectl apply -f test-pvc-deployment.yml
pod/test-pvc-pod created
```

Na maszynie jest plik hello.txt. Możemy wejść w ten kontener i go podejrzeć:

```
(base) Jakubs-MacBook-Pro:k8sLab9 jakublemka$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
9c10d90f1535        alpine              "/bin/sh -c 'sleep 3…"   43 seconds ago      Up 42 seconds                           k8s_alpine_test-pvc-pod_de
fault_469056e0-fa2f-4f9d-b5a6-f4782a6363e9_0
(base) Jakubs-MacBook-Pro:k8sLab9 jakublemka$ docker exec it 9c10d90f1535 sh
Error: No such container: it
(base) Jakubs-MacBook-Pro:k8sLab9 jakublemka$ docker exec -it 9c10d90f1535 sh
/ # cd opt/
/opt # cd data/
/opt/data # ls
hello.txt
/opt/data # cat hello.txt
Hello
/opt/data # echo World >> hello.txt
/opt/data # cat hello.txt
Hello
World
/opt/data #
```

- Get storage class:
  - (base) Jakubs-MacBook-Pro:k8sLab9 jakublemka$ **kubectl get sc**
  - NAME PROVISIONER AGE
  - hostpath (default) docker.io/hostpath 28d
- Get persistence volume:
  - (base) Jakubs-MacBook-Pro:k8sLab9 jakublemka$ kubectl get pv
  - NAME CAPACITY ACCESS MODES RECLAIM POLICY STATUS CLAIM STORAGECLASS REASON AGE
  - pvc-6e648fcb-2976-4648-8e10-2b31ea5dc3c5 100Mi RWO Delete Bound default/postgres-pvc hostpath 20m

- Get persistence volume claim:
  - (base) Jakubs-MacBook-Pro:k8sLab9 jakublemka$ kubectl get pvc
  - NAME STATUS VOLUME CAPACITY ACCESS MODES STORAGECLASS AGE
  - postgres-pvc Bound pvc-6e648fcb-2976-4648-8e10-2b31ea5dc3c5 100Mi RWO hostpath 21m

POLECONE DO WYPRÓBOWANIA: LONGHORN: https://longhorn.io/

Postgres-secret.yml:

- używamy Base64 by kodowane znaki były ASCII
  - (base) Jakubs-MacBook-Pro:k8sLab9 jakublemka$ echo pgpassword123 | base64
  - cGdwYXNzd29yZDEyMwo=
- (base) Jakubs-MacBook-Pro:k8sLab9 jakublemka$ kubectl apply -f postgres-secret.yml
- secret/postgres-secret created
- (base) Jakubs-MacBook-Pro:k8sLab9 jakublemka$ kubectl get secret
- NAME TYPE DATA AGE
- default-token-djjpk kubernetes.io/service-account-token 3 28d
- postgres-secret Opaque

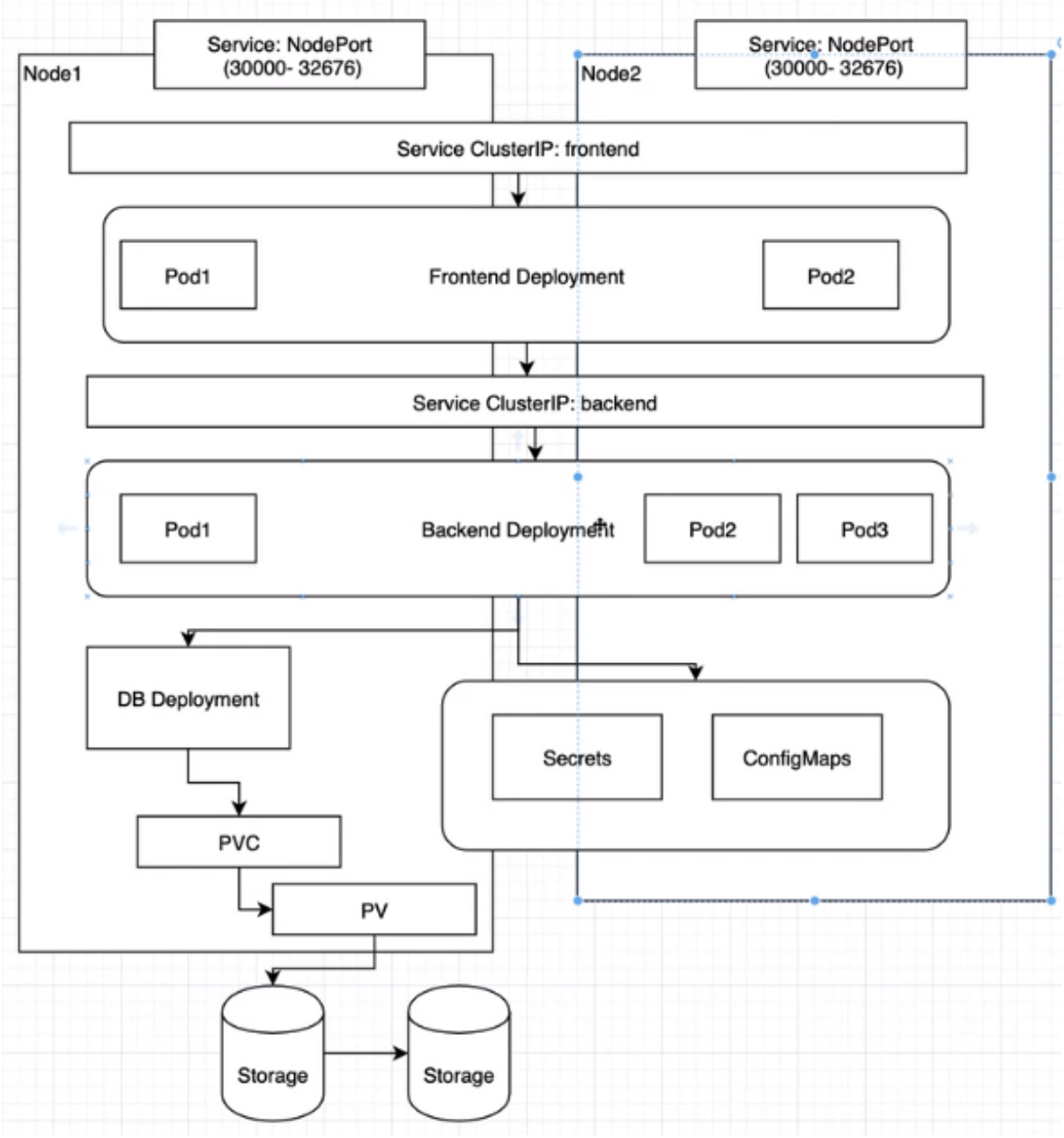
Jak tworzymy z pliku – to *apply* albo *create*. Różnice:

S. No.	Kubectl apply	Kubectl create
1.	It directly updates in the current live source, only the attributes which are given in the file.	It first deletes the resources and then creates it from the file provided.
2.	The file used in apply can be an incomplete spec	The file used in create should be complete
3.	Apply works only on some properties of the resources	Create works on every property of the resources
4.	You can apply a file that changes only an annotation, without specifying any other properties of the resource.	If you will use the same file with a replace command, the command would fail, due to the missing information.

Devops zajęcia nr 10 (14.06.2020r.)

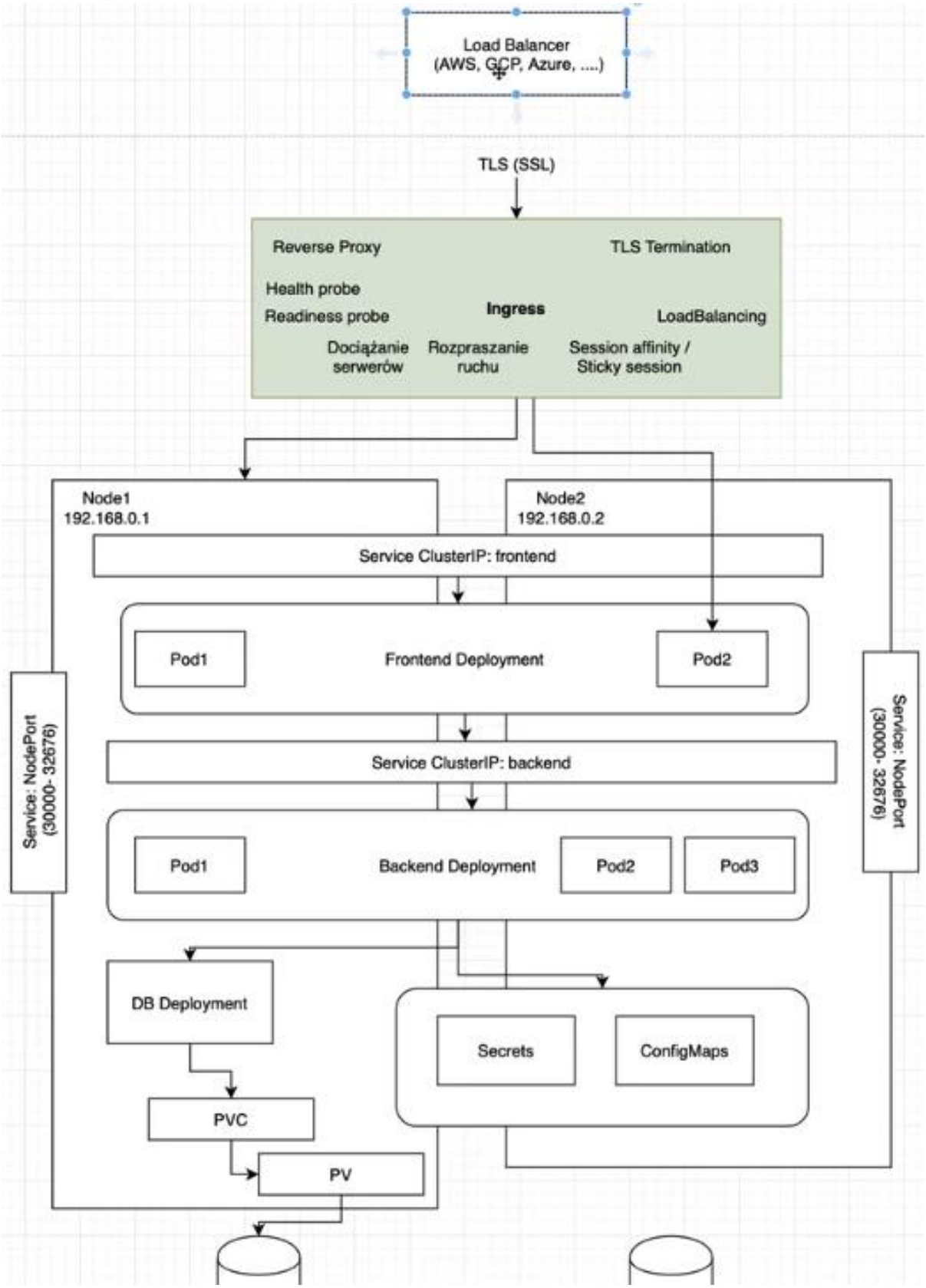
Podsumowanie naszego klastra:

PVC dzięki dostawcy usług storage'owych provisionerowi dostajemy dostęp do storage'a.



Teraz potrzebujemy takie bytu, który wystawia nasz klaster na zewnątrz na niskich portach i pozwala na różne zaawansowane rzeczy związane z wystawieniem usługi na zewnątrz. Należy pamiętać, że każdy NODE ma inny adres IP.

INGRES – ruch wchodzący

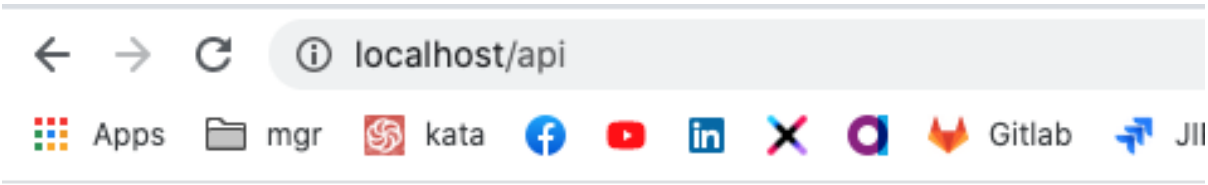


Load Balancer udostępnia naszą usługę na zewnątrz wykorzystując load balancer udostępniony przez danego dostawcy chmury (azure, aws, google itp.).

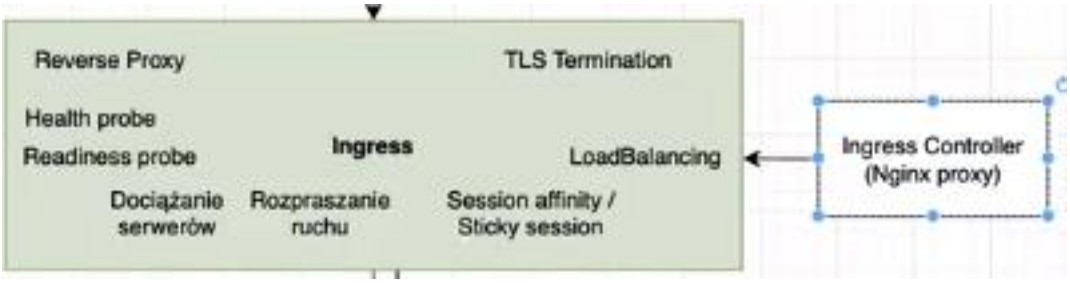
```
k8s/my-cluster4 kubectl get namespaces
NAME              STATUS   AGE
default           Active   30d
docker            Active   30d
ingress-nginx     Active   17d
kube-node-lease   Active   30d
kube-public       Active   30d
kube-system       Active   30d
k8s/my-cluster4 kubectl get pods -n ingress-nginx
NAME                                        READY   STATUS      RESTARTS   AGE
ingress-nginx-admission-create-grbft        0/1     Completed   0          17d
ingress-nginx-admission-patch-ddrjd         0/1     Completed   0          17d
ingress-nginx-controller-5cc4589cc8-f2642   1/1     Running     0          17d
k8s/my-cluster4 kubectl exec -it ingress-nginx-controller-5cc4589cc8-f2642 -n ingress-n
ginx -- /nginx-ingress-controller --version
-------------------------------------------------------------------------------
NGINX Ingress controller
  Release:       0.32.0
  Build:         git-446845114
  Repository:    https://github.com/kubernetes/ingress-nginx
  nginx version: nginx/1.17.10

-------------------------------------------------------------------------------
```

```yaml
apiVersion: extensions/v1beta1

kind: Ingress

metadata:
  name: ingress-service
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/rewrite-target: /$1

spec:
  rules:
    - http:
        paths:
          - path: /api/?(.*)
            backend:
              serviceName: mybackend-service
              servicePort: 5000
```

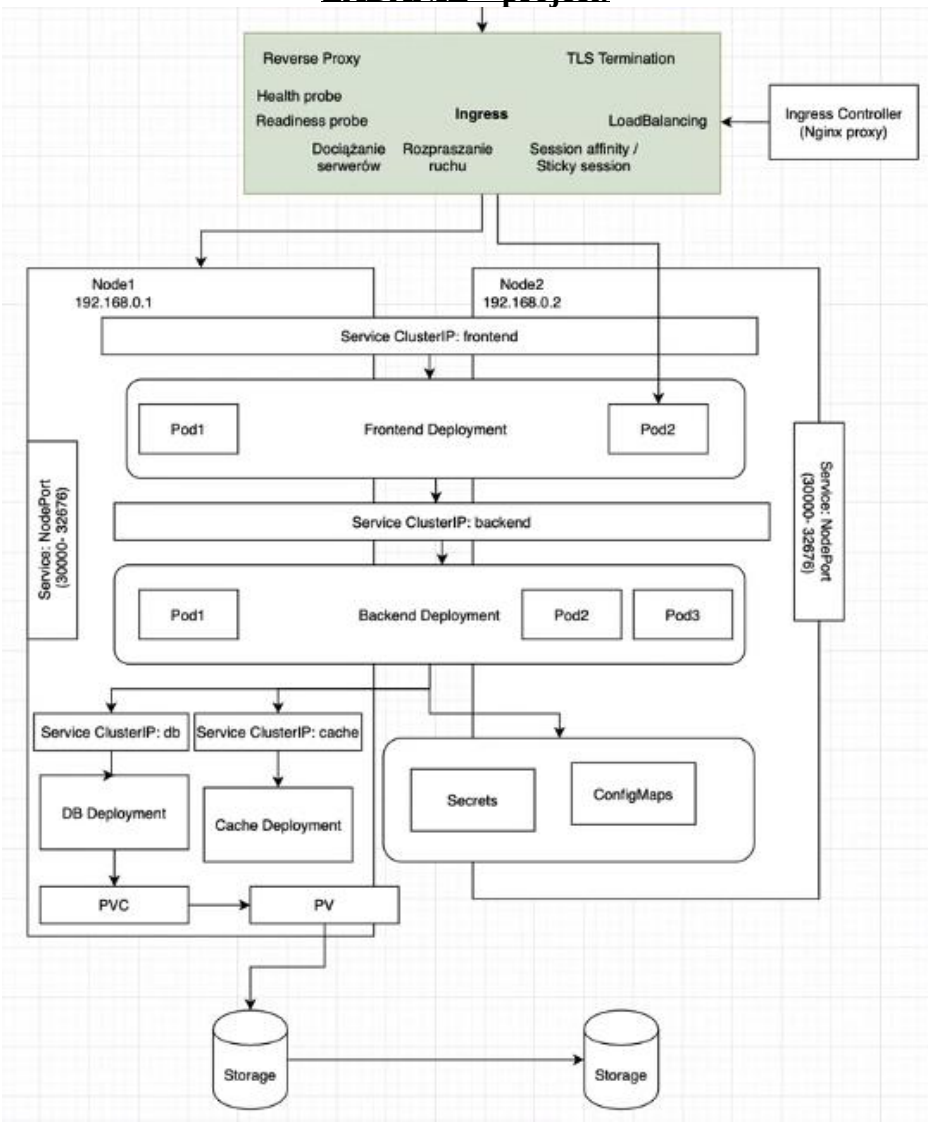
(base) Jakubs-MacBook-Pro:k8sLab10 jakublemka$ kubectl apply -f ingress-service.yml
ingress.extensions/ingress-service created
(base) Jakubs-MacBook-Pro:k8sLab10 jakublemka$ kubectl get ingress
NAME HOSTS ADDRESS PORTS AGE
ingress-service * localhost 80 8s



[7004ec36-076e-4147-816f-670a7e169fc4] Hello from my backend app



ZADANIE – project:



SŁOWNIK NAJWAŻNIEJSZYCH KOMEND - PODSUMOWANIE:

1. Docker

- docker version → show installed docker version
- docker stats → show stats of running containers
- docker logs CONTAINER → show the logs of a container
- docker images → show a list of all images
- docker pull IMAGE[:TAG] → download an image
- docker push IMAGE[:TAG] -> upload an image to repository
- docker build -t IMAGE[:TAG] DIR -> build and tag an image from Dockerfile
- docker ps → show a list of running containers
- docker ps -a → show a list of all containers
- docker run IMAGE → start a new container from an image
- docker run --name CONTAINER IMAGE → start a new container with passed name from an image
- docker tag IMAGE NEWIMAGE → tag an image
- docker rmi IMAGE → delete an image
- docker image prune → delete dangling images
- docker image prune -a → delete all unused images
- docker save IMAGE > FILE → save an image to a file
- docker load -i FILE → load an image from a file
- docker run -d IMAGE → start a new container from an image in a background
- docker run -it IMAGE CMD → start a new container from an image and run a command
- docker rm CONTAINER → delete a container
- docker rm -f CONTAINER → delete a running container
- docker container prune → delete stopped containers
- docker stop CONTAINER → stop container
- docker start CONTAINER → start container
- docker rename OLD NEW → rename container

2. Kubenetes

- kubectl version → get version information
- kubectl cluster-info → get cluster information
- kubectl get TYPE → get all elements in passed type
    - for example
        - kubectl get pods
        - kubectl get deploy
        - kubectl get svn
- kubectl get all → get all elements in all types
- kubectl describe TYPE NAME → get information about passed type with passed name
- kubectl logs TYPE NAME → get logs for passed type with passed name
- kubectl create -f FILE → create from file
- kubectl replace -f FILE → replace elements from file

- kubectl apply -f FILE → apply from file
- kubectl delete TYPE NAME → delete type with passed name
- kubectl scale --replicas=VALUE rs RSNAME → set passed number (VALUE) of replicas for passed replicaset name