Şimdi 7. adımdayız.

Dockerfile ile image leri oluşturduk. Şimdi ise bu image leri build edecek bir script oluşturma görevimiz var. Böyle birden fazla image i buid ederken best practice script yazmaktır.

git checkout dev

git branch feature/msp-7

git checkout feature/msp-7

Yeni bir brancha geçiyoruz.

./mvnw clean package

docker build --force-rm -t "petclinic-admin-server:dev" ./spring-petclinic-admin-server

docker build --force-rm -t "petclinic-api-gateway:dev" ./spring-petclinic-api-gateway

docker build --force-rm -t "petclinic-config-server:dev" ./spring-petclinic-config-server

docker build --force-rm -t "petclinic-customers-service:dev" ./spring-petclinic-customers-service

docker build --force-rm -t "petclinic-discovery-server:dev" ./spring-petclinic-discovery-server

docker build --force-rm -t "petclinic-hystrix-dashboard:dev" ./spring-petclinic-hystrix-dashboard

docker build --force-rm -t "petclinic-vets-service:dev" ./spring-petclinic-vets-service

docker build --force-rm -t "petclinic-visits-service:dev" ./spring-petclinic-visits-service

docker build --force-rm -t "petclinic-grafana-server:dev" ./docker/grafana

docker build --force-rm -t "petclinic-prometheus-server:dev" ./docker/prometheus

Komutlarını build-dev-docker-images.sh

dosyası içine yazdık. bu dosya proje klasöümüzdün rootunda olacak.

chmod +x build-dev-docker-images.sh

komutuyla da scripti executable hale getiriyoruz.

scriptin başındaki komut .jar file larını temizleyip yeniden oluşturması için ekleniyor. bu sayede .jar file ların çalıştığına emin oluruz.

--force-rm parametresi her zaman intermediate conainer ları siler.

bu ne demek?

Docker dockerfile ı gerçekleştirirken her aşamada komutu çalıştırmak için bir container oluşturur. komut başarılı olursa bu ara container ı siler. ancak bazen bu ara container lar silinmiyor. Dockerifle da bir satırda hata alırsa o satırda oluşturduğu container ı (hataları inceleyelim diye) silmiyor. --froce-rm dediğimizde biz bu ara container ı da silmesini istiyoruz.

Neden? yüzlerce yerde bu image çalışacak bu ara container kalırsa disklerde şişme oluşur. Oturmuş bir sistemde böyle kullanılır.

Şimdi ./build-dev-docker-images.sh

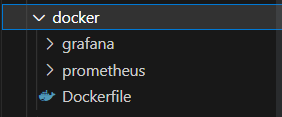
komutuyla script dosyasını çalıştıralım. Önce .jar file ları tekrar oluşturuyor sonrasında da tüm Dockerfile ları çalıştıracak.

Bu arada dev-serverın SECGRP ta 8080 portunu açalım.

image ler hazır:

ec2-user@petc1inic-dev-server# petclinic-microservices-with-db:( feature/msp-7)$ docker i 
mage Is 
REPOSITORY 
pet clinic-prometheus-server 
petclinic -grafana -server 
petclinic -visits -service 
petclinic -vets -service 
petclinic-hystrix-dashboard 
petclinic-discovery-server 
petclinic-customers-service 
petclinic-config-server 
petclinic-api-gateway 
petclinic-admin-server 
grafana/grafana 
openjdk 
prom/ prometheus 
TAG 
dev 
dev 
dev 
dev 
dev 
dev 
dev 
dev 
dev 
dev 
latest 
Il-jre 
v2.4.2 
IMAGE ID 
ØØa52f67e482 
gauge763Øaa 
ec59165cbU8 
IffØ95f 7 a 26f 
16a7412ØØ352 
9d5f8be43fe3 
90935957c5ff 
ecd9175143be 
Icae2822d692 
1b8626d6b7b2 
6c5313eaØØcc 
362cda5d27Øe 
4149712a7a11 
CREATED 
15 
18 
28 
34 
40 
45 
50 
56 
seconds 
seconds 
seconds 
seconds 
seconds 
seconds 
seconds 
seconds 
ago 
ago 
ago 
ago 
ago 
ago 
ago 
ago 
About a minute 
About a minute 
3 weeks ago 
10 months ago 
4 years ago 
ago 
ago 
SIZE 
1101B 
3Ø9MB 
384MB 
386MB 
355MB 
368MB 
384MB 
351MB 
373MB 
368MB 
3Ø9MB 
3Ø2MB 
1101B 

grafana ile prometheus da uygulama ile geliyor ve docker klasörü içerisinde bulunuyor:



şimdi komutlarıyla feature 7 yi push ve merge edelim.

git add .

git commit -m 'added script for building docker images'

git push --set-upstream origin feature/msp-7

git checkout dev

git merge feature/msp-7

git push origin dev

Şimdi 8. adımdayız.

Görev olarak docker-compose file hazırlamamız istendi bizden. yeni bir branch oluşturalım.

**git checkout -b feature/msp-8**

docker-compose-local.yml

isimli bir docker-compose dosyası oluşturuyoruz. bu dosyayı projenin root unda oluşturuyoruz.

bu dosyayo ççalıştırması için test-local-deployment.sh

isimli bir .sh dosyası yazıyoruz:

docker-compose -f docker-compose-local.yml up

-f ile dosya pth i belirttik çünkü uygulamada başka bir docker-compose dosyası daha var ve biz onun çalışmasını istemiyoruz.

chmod +x test-local-deployment.sh

ile script dosyasını executable etkisi veriyoruz.

feature/msp-8 i de push edip merge edelim.

git add .

git commit -m 'added docker-compose file and script for local deployment'

git push --set-upstream origin feature/msp-8

git checkout dev

git merge feature/msp-8

git push origin dev

./test-local-deployment.sh

komutyla script dosyasını çalıştıralım.

Şimdi docker-compose üzerinde biraz konuşalım:

version: '2'

neden version 2? docker-compose un 3 versionu var, 1. si kullanılmıyor artık.

Ref«mce 
Versi 3 
Version 2 
Version 1 (Deprecated) 
What chmged in fris vesbn 
Version 3 updates 
Version 2 updates 
Version 1 updates 

Version 3 e geçerken mem\_limit i kaldırmışlar:

Version 3 
Designed to be cross-compatible between Compose and the Docker Engines swarm mode, version 3 removes several options 
and adds several more. 
Removed: volume_driver , volumes_fron , cpu_shares , cpu_quota , cpuset , 
-em _ limit 
memswap_ , 
extends , group_add . Seethe upgrading guide for how to migrate awayfrom these. (For more information on 
extends , see Extending services.) 
• Added: deploy 

şu an test aşamasında olduğumuz için limit vermezsek container host un bütün memorysini çeker ve host çöker. host çökmesin container çöksün çökecekse. Version 3 te bu özelliği kullanmak için deploy kullanılıyor deploy da docker swarm ile kullanılıyor. Biz o yüzden version 2 kullanıyoruz.

services:

config-server:

image: petclinic-config-server:dev

container\_name: config-server

mem\_limit: 512M

ports:

- 8888:8888

burada isim verirken config\_server verdik? neden çünkü spring\_profile de bu config-server ismiyle yer alıyordu. Bunu developerlarla konuşuyoruz ne isim verelim diye.

config-server 8888 den yayın yapacak ve diğer container lar ondan bilgileri alacak.

Sadece 8888 yazarsak nhangi porttan bulursa o porttan yayın yapar.

discovery-server:

image: petclinic-discovery-server:dev

container\_name: discovery-server

mem\_limit: 512M

ports:

- 8761:8761

depends\_on:

- config-server

entrypoint: ["./dockerize", "-wait=tcp://config-server:8888", "-timeout=160s", "--", "java", "-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]

buradaki 8761 portunu da developerlar söylüyor kafamıza göre koymuyoruz.

discovery-server ın ayağa kalkmasından önce config-server ın ayağa kalkmasını istiyoruz bu nedenle depends\_on kullandık.

Ancak depends\_on container ın ayağa kalkıp kalkmadığına bakar, yayın yapıp yapmadığına bakmaz. bu nedenle dockerize kullanıyoruz.

Dockerize çalıştırmak için de burada entrypoint kullandık.

Docker-compose daki entrypoint ile Dockerfile daki entrypoint i override yapabiliriz.

entrypoint: ["./dockerize", "-wait=tcp://config-server:8888", "-timeout=160s", "--", "java", "-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]

dockerize ı çalıştır, config-server:8888 in yayın yapmasını bekle, yapmıyorsa 160 sn bekle. Sonrasında "--" "java" nın "dockerize" ın bir parametresi olmadığını belirtmek için ekleniyor.

( Kubernetes teki readiness kendini kontrol eder. Buradaki amacımız başka bir servisi kontrol etmek. )

customers-service:

image: petclinic-customers-service:dev

container\_name: customers-service

mem\_limit: 512M

ports:

- 8081:8081

depends\_on:

- config-server

- discovery-server

entrypoint: ["./dockerize", "-wait=tcp://discovery-server:8761", "-timeout=160s", "--", "java", "-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar" ]

bu kez depends\_on da hem config-server hem discovery-server var. bu ve devamındaki serverlar aynı.

tracing server a bakalım:

tracing-server:

image: openzipkin/zipkin

container\_name: tracing-server

mem\_limit: 512M

ports:

- 9411:9411

bu server bağımsız çalışıyor ve discovery-server a çalıştığını rapor etmesine gerek yok. bu nedenle burada depends\_on eklenmiyor.

tracing-server logları tutar.

gelelim mysql-server a:

mysql-server:

image: mysql:5.7.8

container\_name: mysql-server

environment:

MYSQL\_ROOT\_PASSWORD: petclinic

MYSQL\_DATABASE: petclinic

mem\_limit: 256M

ports:

- 3306:3306

bu image i kullanmamızı söylediler, environment variable lar nereden geliyor? Developerlar söylüyor:

Start a MySql database 
You may start a MySqI database with docker: 
docker run -e MYSQL ROOT PASSWORD=petc1inic -e MYSQL DATABASE=petc1inic -p 3386:3306 mysq1:s.7.8 

docker-compose da sıra önemli değildir serviceler paralael şekilde çalışır.

Şimdi 9. adıma geçiyoruz.

Bizden istenen bir jenkins-server için terraform dosyası oluşturmamız.

Biz branch ı oluşturup hazır olan dosyamızı burada push edip branch ı kapatalım. Bu dosyayı infrastructure folderı altında oluşturuyoruz.

Bunun için de proje klasöründeki msp-9-jenkins-server-template klasörünü infrastructure klasörü altına kopyaladık.

git add .

git commit -m 'added jenkins server terraform files'

git push --set-upstream origin feature/msp-9

git checkout dev

git merge feature/msp-9

git push origin dev

bu komutlarla push ve merge ediyoruz.

Bir tane Jenkins server ayağa kaldıracaz. Oraya geçecez ve bu instance ı terminate edecez. Bu nedenle jenkins server ı lokalden ayağa kaldıracaz.

Proje klasöründeki msp-9-jenkins-server-template i masaüstüne alarak terraform init/apply ile çalıştırıyoruz. vs code ile bağlanıyoruz.

**Şimdi 10. adımdayız.**

Jenkins e browser da bağlanarak gerekli conf ları yapıyoruz.

GitHub Integration`, `Docker Plugin`, `Docker Pipeline`, and `Jacoco`

pluginlerini yüklüyoruz. (Locale de yüklenebilir ingilizce için)

Son olarak GitHub repomuza gidiyoruz. HTTPS kodunu alıyoruz ve jenkins-server a kullanıcı adı ve token ile clone luyoruz.

bash.rc ye de branch için gerekli kodu yapıştırıyoruz.

normalde dev-server ayrı jenkins-server ayrıdır. ancak biz maliyet olmasın diye şimdi dev-server ı terminate ediyoruz.

Şimdi 7. adımdayız.

Dockerfile ile image leri oluşturduk. Şimdi ise bu image leri build edecek bir script oluşturma görevimiz var. Böyle birden fazla image i buid ederken best practice script yazmaktır.

git checkout dev

git branch feature/msp-7

git checkout feature/msp-7

Yeni bir brancha geçiyoruz.

./mvnw clean package

docker build --force-rm -t "petclinic-admin-server:dev" ./spring-petclinic-admin-server

docker build --force-rm -t "petclinic-api-gateway:dev" ./spring-petclinic-api-gateway

docker build --force-rm -t "petclinic-config-server:dev" ./spring-petclinic-config-server

docker build --force-rm -t "petclinic-customers-service:dev" ./spring-petclinic-customers-service

docker build --force-rm -t "petclinic-discovery-server:dev" ./spring-petclinic-discovery-server

docker build --force-rm -t "petclinic-hystrix-dashboard:dev" ./spring-petclinic-hystrix-dashboard

docker build --force-rm -t "petclinic-vets-service:dev" ./spring-petclinic-vets-service

docker build --force-rm -t "petclinic-visits-service:dev" ./spring-petclinic-visits-service

docker build --force-rm -t "petclinic-grafana-server:dev" ./docker/grafana

docker build --force-rm -t "petclinic-prometheus-server:dev" ./docker/prometheus

Komutlarını build-dev-docker-images.sh

dosyası içine yazdık. bu dosya proje klasöümüzdün rootunda olacak.

chmod +x build-dev-docker-images.sh

komutuyla da scripti executable hale getiriyoruz.

scriptin başındaki komut .jar file larını temizleyip yeniden oluşturması için ekleniyor. bu sayede .jar file ların çalıştığına emin oluruz.

--force-rm parametresi her zaman intermediate conainer ları siler.

bu ne demek?

Docker dockerfile ı gerçekleştirirken her aşamada komutu çalıştırmak için bir container oluşturur. komut başarılı olursa bu ara container ı siler. ancak bazen bu ara container lar silinmiyor. Dockerifle da bir satırda hata alırsa o satırda oluşturduğu container ı (hataları inceleyelim diye) silmiyor. --froce-rm dediğimizde biz bu ara container ı da silmesini istiyoruz.

Neden? yüzlerce yerde bu image çalışacak bu ara container kalırsa disklerde şişme oluşur. Oturmuş bir sistemde böyle kullanılır.

Şimdi ./build-dev-docker-images.sh

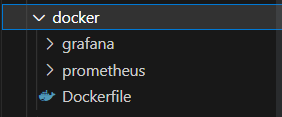
komutuyla script dosyasını çalıştıralım. Önce .jar file ları tekrar oluşturuyor sonrasında da tüm Dockerfile ları çalıştıracak.

Bu arada dev-serverın SECGRP ta 8080 portunu açalım.

image ler hazır:

ec2-user@petc1inic-dev-server# petclinic-microservices-with-db:( feature/msp-7)$ docker i 
mage Is 
REPOSITORY 
pet clinic-prometheus-server 
petclinic -grafana -server 
petclinic -visits -service 
petclinic -vets -service 
petclinic-hystrix-dashboard 
petclinic-discovery-server 
petclinic-customers-service 
petclinic-config-server 
petclinic-api-gateway 
petclinic-admin-server 
grafana/grafana 
openjdk 
prom/ prometheus 
TAG 
dev 
dev 
dev 
dev 
dev 
dev 
dev 
dev 
dev 
dev 
latest 
Il-jre 
v2.4.2 
IMAGE ID 
ØØa52f67e482 
gauge763Øaa 
ec59165cbU8 
IffØ95f 7 a 26f 
16a7412ØØ352 
9d5f8be43fe3 
90935957c5ff 
ecd9175143be 
Icae2822d692 
1b8626d6b7b2 
6c5313eaØØcc 
362cda5d27Øe 
4149712a7a11 
CREATED 
15 
18 
28 
34 
40 
45 
50 
56 
seconds 
seconds 
seconds 
seconds 
seconds 
seconds 
seconds 
seconds 
ago 
ago 
ago 
ago 
ago 
ago 
ago 
ago 
About a minute 
About a minute 
3 weeks ago 
10 months ago 
4 years ago 
ago 
ago 
SIZE 
1101B 
3Ø9MB 
384MB 
386MB 
355MB 
368MB 
384MB 
351MB 
373MB 
368MB 
3Ø9MB 
3Ø2MB 
1101B 

grafana ile prometheus da uygulama ile geliyor ve docker klasörü içerisinde bulunuyor:



şimdi komutlarıyla feature 7 yi push ve merge edelim.

git add .

git commit -m 'added script for building docker images'

git push --set-upstream origin feature/msp-7

git checkout dev

git merge feature/msp-7

git push origin dev

Şimdi 8. adımdayız.

Görev olarak docker-compose file hazırlamamız istendi bizden. yeni bir branch oluşturalım.

**git checkout -b feature/msp-8**

docker-compose-local.yml

isimli bir docker-compose dosyası oluşturuyoruz. bu dosyayı projenin root unda oluşturuyoruz.

bu dosyayo ççalıştırması için test-local-deployment.sh

isimli bir .sh dosyası yazıyoruz:

docker-compose -f docker-compose-local.yml up

-f ile dosya pth i belirttik çünkü uygulamada başka bir docker-compose dosyası daha var ve biz onun çalışmasını istemiyoruz.

chmod +x test-local-deployment.sh

ile script dosyasını executable etkisi veriyoruz.

feature/msp-8 i de push edip merge edelim.

git add .

git commit -m 'added docker-compose file and script for local deployment'

git push --set-upstream origin feature/msp-8

git checkout dev

git merge feature/msp-8

git push origin dev

./test-local-deployment.sh

komutyla script dosyasını çalıştıralım.

Şimdi docker-compose üzerinde biraz konuşalım:

version: '2'

neden version 2? docker-compose un 3 versionu var, 1. si kullanılmıyor artık.

Ref«mce 
Versi 3 
Version 2 
Version 1 (Deprecated) 
What chmged in fris vesbn 
Version 3 updates 
Version 2 updates 
Version 1 updates 

Version 3 e geçerken mem\_limit i kaldırmışlar:

Version 3 
Designed to be cross-compatible between Compose and the Docker Engines swarm mode, version 3 removes several options 
and adds several more. 
Removed: volume_driver , volumes_fron , cpu_shares , cpu_quota , cpuset , 
-em _ limit 
memswap_ , 
extends , group_add . Seethe upgrading guide for how to migrate awayfrom these. (For more information on 
extends , see Extending services.) 
• Added: deploy 

şu an test aşamasında olduğumuz için limit vermezsek container host un bütün memorysini çeker ve host çöker. host çökmesin container çöksün çökecekse. Version 3 te bu özelliği kullanmak için deploy kullanılıyor deploy da docker swarm ile kullanılıyor. Biz o yüzden version 2 kullanıyoruz.

services:

config-server:

image: petclinic-config-server:dev

container\_name: config-server

mem\_limit: 512M

ports:

- 8888:8888

burada isim verirken config\_server verdik? neden çünkü spring\_profile de bu config-server ismiyle yer alıyordu. Bunu developerlarla konuşuyoruz ne isim verelim diye.

config-server 8888 den yayın yapacak ve diğer container lar ondan bilgileri alacak.

Sadece 8888 yazarsak nhangi porttan bulursa o porttan yayın yapar.

discovery-server:

image: petclinic-discovery-server:dev

container\_name: discovery-server

mem\_limit: 512M

ports:

- 8761:8761

depends\_on:

- config-server

entrypoint: ["./dockerize", "-wait=tcp://config-server:8888", "-timeout=160s", "--", "java", "-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]

buradaki 8761 portunu da developerlar söylüyor kafamıza göre koymuyoruz.

discovery-server ın ayağa kalkmasından önce config-server ın ayağa kalkmasını istiyoruz bu nedenle depends\_on kullandık.

Ancak depends\_on container ın ayağa kalkıp kalkmadığına bakar, yayın yapıp yapmadığına bakmaz. bu nedenle dockerize kullanıyoruz.

Dockerize çalıştırmak için de burada entrypoint kullandık.

Docker-compose daki entrypoint ile Dockerfile daki entrypoint i override yapabiliriz.

entrypoint: ["./dockerize", "-wait=tcp://config-server:8888", "-timeout=160s", "--", "java", "-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]

dockerize ı çalıştır, config-server:8888 in yayın yapmasını bekle, yapmıyorsa 160 sn bekle. Sonrasında "--" "java" nın "dockerize" ın bir parametresi olmadığını belirtmek için ekleniyor.

( Kubernetes teki readiness kendini kontrol eder. Buradaki amacımız başka bir servisi kontrol etmek. )

customers-service:

image: petclinic-customers-service:dev

container\_name: customers-service

mem\_limit: 512M

ports:

- 8081:8081

depends\_on:

- config-server

- discovery-server

entrypoint: ["./dockerize", "-wait=tcp://discovery-server:8761", "-timeout=160s", "--", "java", "-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar" ]

bu kez depends\_on da hem config-server hem discovery-server var. bu ve devamındaki serverlar aynı.

tracing server a bakalım:

tracing-server:

image: openzipkin/zipkin

container\_name: tracing-server

mem\_limit: 512M

ports:

- 9411:9411

bu server bağımsız çalışıyor ve discovery-server a çalıştığını rapor etmesine gerek yok. bu nedenle burada depends\_on eklenmiyor.

tracing-server logları tutar.

gelelim mysql-server a:

mysql-server:

image: mysql:5.7.8

container\_name: mysql-server

environment:

MYSQL\_ROOT\_PASSWORD: petclinic

MYSQL\_DATABASE: petclinic

mem\_limit: 256M

ports:

- 3306:3306

bu image i kullanmamızı söylediler, environment variable lar nereden geliyor? Developerlar söylüyor:

Start a MySql database 
You may start a MySqI database with docker: 
docker run -e MYSQL ROOT PASSWORD=petc1inic -e MYSQL DATABASE=petc1inic -p 3386:3306 mysq1:s.7.8 

docker-compose da sıra önemli değildir serviceler paralael şekilde çalışır.

Şimdi 9. adıma geçiyoruz.

Bizden istenen bir jenkins-server için terraform dosyası oluşturmamız.

Biz branch ı oluşturup hazır olan dosyamızı burada push edip branch ı kapatalım. Bu dosyayı infrastructure folderı altında oluşturuyoruz.

Bunun için de proje klasöründeki msp-9-jenkins-server-template klasörünü infrastructure klasörü altına kopyaladık.

git add .

git commit -m 'added jenkins server terraform files'

git push --set-upstream origin feature/msp-9

git checkout dev

git merge feature/msp-9

git push origin dev

bu komutlarla push ve merge ediyoruz.

Bir tane Jenkins server ayağa kaldıracaz. Oraya geçecez ve bu instance ı terminate edecez. Bu nedenle jenkins server ı lokalden ayağa kaldıracaz.

Proje klasöründeki msp-9-jenkins-server-template i masaüstüne alarak terraform init/apply ile çalıştırıyoruz. vs code ile bağlanıyoruz.

**Şimdi 10. adımdayız.**

Jenkins e browser da bağlanarak gerekli conf ları yapıyoruz.

GitHub Integration`, `Docker Plugin`, `Docker Pipeline`, and `Jacoco`

pluginlerini yüklüyoruz. (Locale de yüklenebilir ingilizce için)

Son olarak GitHub repomuza gidiyoruz. HTTPS kodunu alıyoruz ve jenkins-server a kullanıcı adı ve token ile clone luyoruz.

bash.rc ye de branch için gerekli kodu yapıştırıyoruz.

normalde dev-server ayrı jenkins-server ayrıdır. ancak biz maliyet olmasın diye şimdi dev-server ı terminate ediyoruz.