Spring Boot Microservices with Spring Cloud Beginner to Guru

John Thompson
Udemy

Section 29: Deploying with Docker Swarm
- 318. Introduction
- 319. Create Digital Ocean Account
- 320. Deployment Design
  - Containerized Deployment
  - Deployment Goals
  - Summary
- 321. Provision Database Servers
- 322. Configure Database
- 323. Configure Java Truststore
- 324. Add Truststore file to Docker Image
- 325. Docker Image Release Process
- 326. Provision Service VMs
- 327. Configure JMS Server
- 328. Configure Elasticsearch Server
- 329. Configure Kibana Server
- 330. Configure Zipkin Server
- 331. Configure Eureka Server
- 332. Configure Spring Cloud Config Server
- 333. Spring Cloud Config Server IP Address Update
- 334. Provision Docker Swarm Cluster
- 335. Linux Troubleshooting Commands
- 336. Initialize Docker Swarm Cluster
- 337. Filebeat Swarm Configuration

[338. Eureka Swarm Configuration]
[339. Spring Cloud Configuration]
[340. Digital Ocean Profile]
[341. Running Microservices with Docker Swarm]
[342. Tracing Requests for Troubleshooting]
[343. Zipkin Tracing]
[344. Tasting Room Service Challenge]
[345. Retrospective]

Section 29: Deploying with Docker Swarm

318. Introduction

-

319. Create Digital Ocean Account

-

320. Deployment Design

Containerized Deployment

- microservices are well suited for containerized deployment
- typically they are compute only, and don’t persist data in instance
- scalability and reliability achieved with multiple instances
- deployments managed via container orchestration
    - Docker Swarm is among the simplest - uses extensions in Docker Compose
- other solutions - Kubernetes, OpenSHift, Mesos, AWS ECS
    - large and evolving area

- dbs persisting data typically are poor candidates for containerized deployments
- becomes a problem in managing disk storage
- while it can be done, typically not the optimal solution
- often you will see dedicated VMs or physical servers for dbs
    - nothing faster than physical servers
- this extends to any db like apps - JMS or other message brokers, Elasticsearch, Zipkin, etc

Deployment Goals

- use Digital Ocean to create a realistic deployment
- use Digital Ocean Managed MySQL dbs
    - Setup 3 - one per microservice
    - larger organizations will have a db administration team
- setup a dedicated JMS broker
    - high volume production would use a cluster
- setup dedicated Eureka Server
    - production would have a cluster for high availability
- setup dedicated ElasticSearch Server
    - production would have a cluster for high availability and scalability

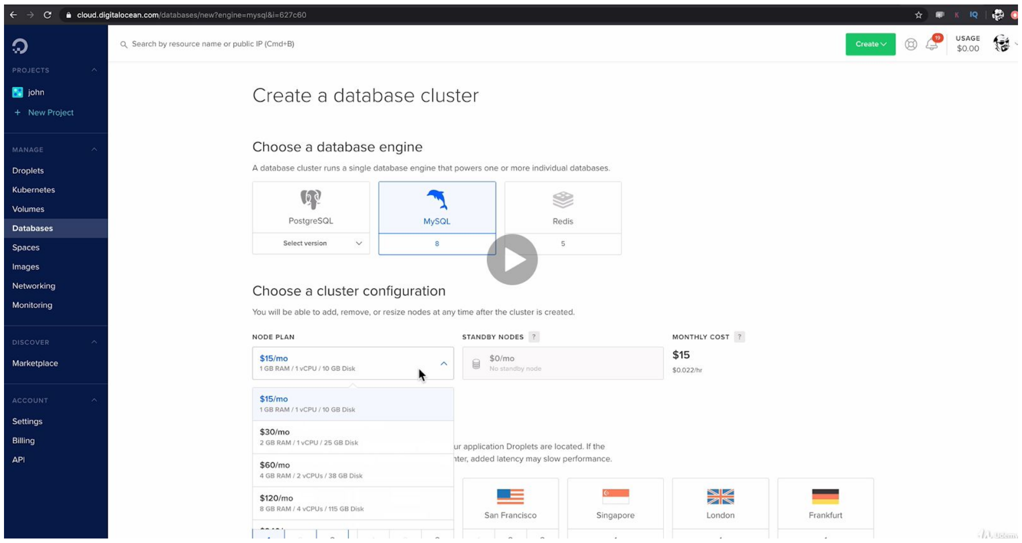
- setup a dedicated Zipkin Server
    - production would use a Cassandra or ElasticSearch data store
- setup dedicated Configuration Server
    - production would use a cluster for high availability
- Deploy Spring Boot Services to 3 node Swarm Cluster
    - Gateway, Beer Service, Inventory Service, Inventory Failover Service, Order Service
    - use Spring Cloud Config with new profile for cloud deployment
- Filebeat
    - deploy per node, use ‘extra_hosts’ to config ElasticSearch Server

Summary

- deployment needs 12 different servers (including 3 MySQL instances)
    - 1 GB RAM / 1 vCPU
- for simplicity we will use Docker deployments
- VMs for services - 6 VMs with 4GB
- VMs for Swarm Cluster - 3 VMs with 8GB

321. Provision Database Servers

- create a MySQL cluster
- set up the 3 dbs (shown the beer db)



cloud.digitalocean.com/databases/new?engine=mysql&i=627c60

Create a database cluster

Choose a database engine

A database cluster runs a single database engine that powers one or more individual databases.

PostgreSQL — Select version
MySQL — 8
Redis — 5

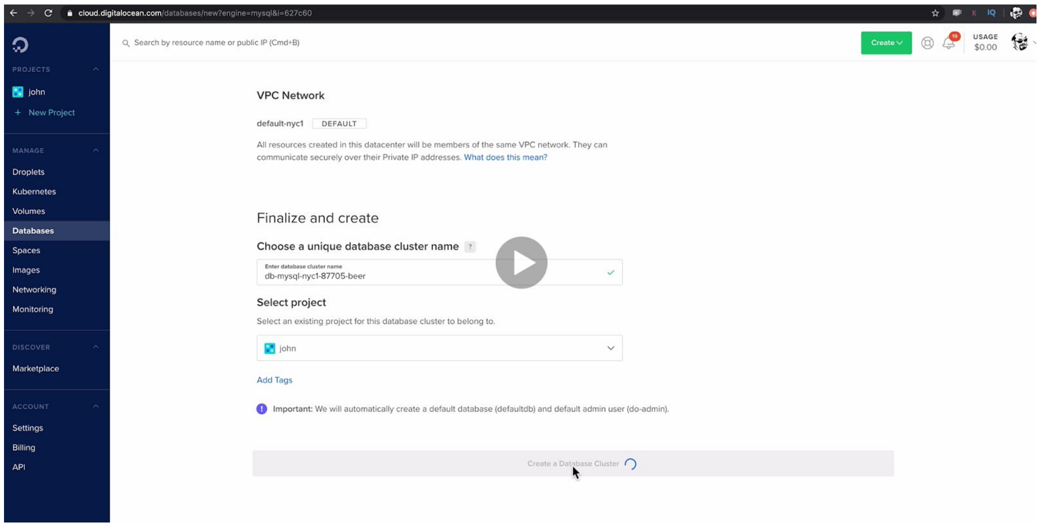
Choose a cluster configuration

You will be able to add, remove, or resize nodes at any time after the cluster is created.

NODE PLAN	STANDBY NODES	MONTHLY COST
$15/mo 1 GB RAM / 1 vCPU / 10 GB Disk	$0/mo No standby node	$15 $0.022/hr

$15/mo 1 GB RAM / 1 vCPU / 10 GB Disk
$30/mo 2 GB RAM / 1 vCPU / 25 GB Disk
$60/mo 4 GB RAM / 2 vCPUs / 38 GB Disk
$120/mo 8 GB RAM / 4 vCPUs / 115 GB Disk

ur application Droplets are located. If the
nter, added latency may slow performance.



VPC Network

default-nyc1 DEFAULT

All resources created in this datacenter will be members of the same VPC network. They can communicate securely over their Private IP addresses. What does this mean?

Finalize and create

Choose a unique database cluster name

Enter database cluster name: db-mysql-nyc1-87705-beer

Select project

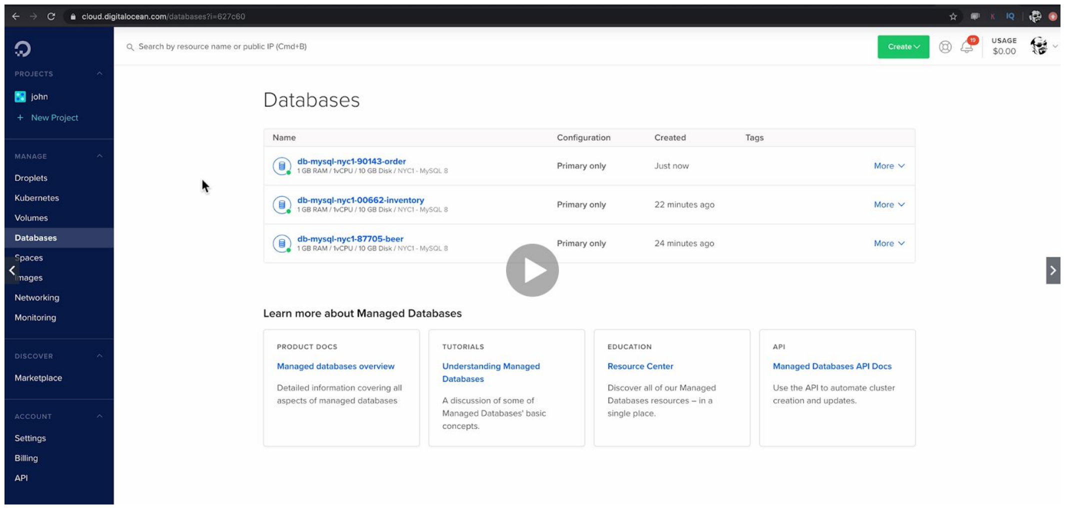
Select an existing project for this database cluster to belong to.

john

Add Tags

Important: We will automatically create a default database (defaultdb) and default admin user (do-admin).

- the dbs:



Databases

Name	Configuration	Created	Tags	
db-mysql-nyc1-90143-order 1 GB RAM / 1vCPU / 10 GB Disk / NYC1 - MySQL 8	Primary only	Just now		More
db-mysql-nyc1-00662-inventory 1 GB RAM / 1vCPU / 10 GB Disk / NYC1 - MySQL 8	Primary only	22 minutes ago		More
db-mysql-nyc1-87705-beer 1 GB RAM / 1vCPU / 10 GB Disk / NYC1 - MySQL 8	Primary only	24 minutes ago		More

Learn more about Managed Databases

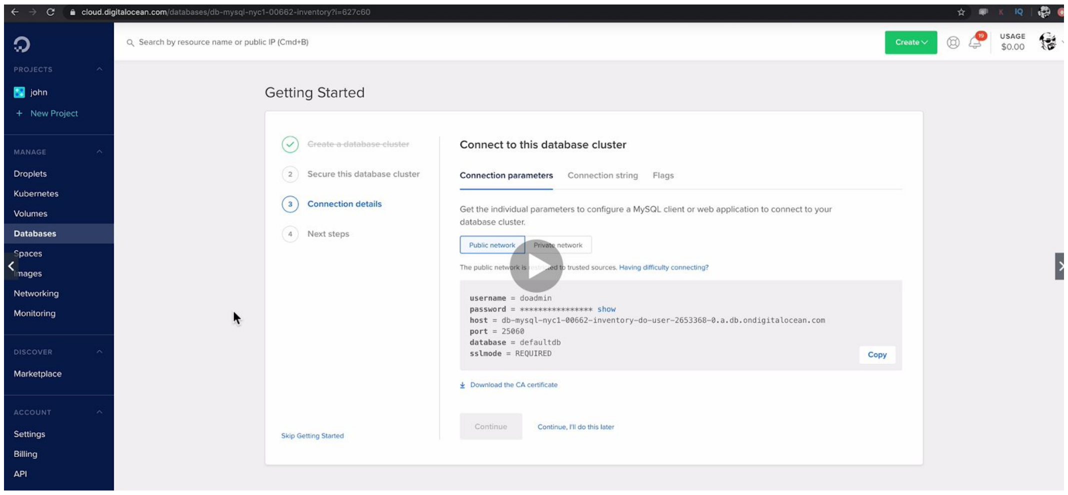
PRODUCT DOCS — Managed databases overview — Detailed information covering all aspects of managed databases

TUTORIALS — Understanding Managed Databases — A discussion of some of Managed Databases' basic concepts.

EDUCATION — Resource Center — Discover all of our Managed Databases resources – in a single place.

API — Managed Databases API Docs — Use the API to automate cluster creation and updates.

- connection details for one of the dbs:



Getting Started

1. ~~Create a database cluster~~
2. Secure this database cluster
3. Connection details
4. Next steps

Connect to this database cluster

Connection parameters | Connection string | Flags

Get the individual parameters to configure a MySQL client or web application to connect to your database cluster.

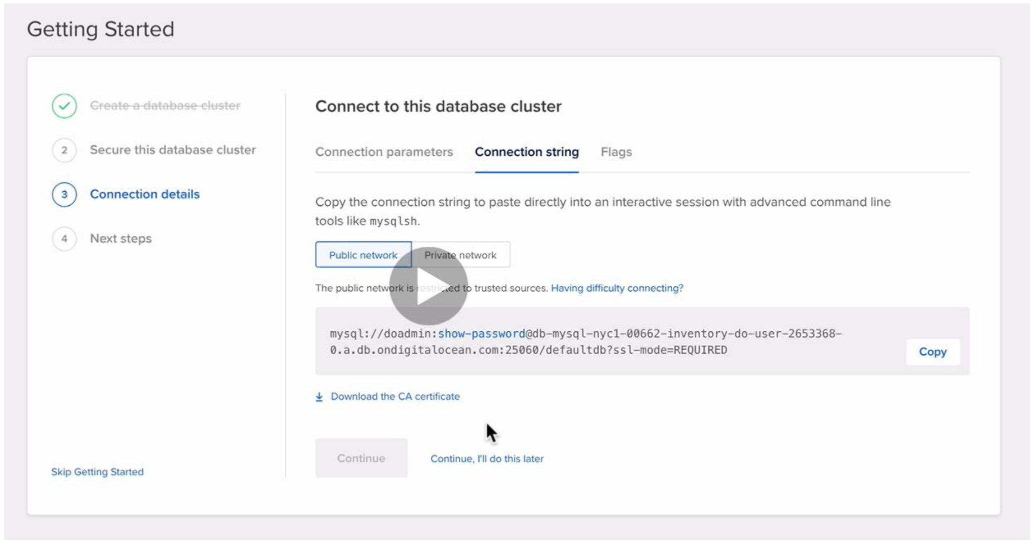
Public network | Private network

The public network is restricted to trusted sources. Having difficulty connecting?

```
username = doadmin
password = **************** show
host = db-mysql-nyc1-00662-inventory-do-user-2653368-0.a.db.ondigitalocean.com
port = 25060
database = defaultdb
sslmode = REQUIRED
```

Download the CA certificate

- connection string for one of the dbs:



Getting Started

~~Create a database cluster~~
2 Secure this database cluster
3 Connection details
4 Next steps

Skip Getting Started

Connect to this database cluster

Connection parameters | Connection string | Flags

Copy the connection string to paste directly into an interactive session with advanced command line tools like `mysqlsh`.

Public network | Private network

The public network is restricted to trusted sources. Having difficulty connecting?

```
mysql://doadmin:show-password@db-mysql-nyc1-00662-inventory-do-user-2653368-
0.a.db.ondigitalocean.com:25060/defaultdb?ssl-mode=REQUIRED
```

Copy

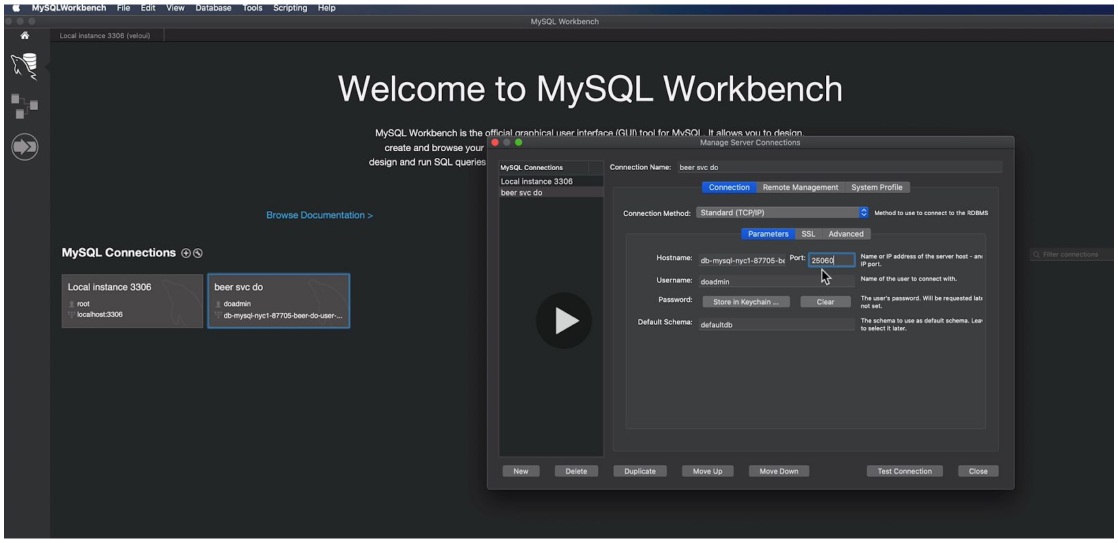
Download the CA certificate

Continue | Continue, I'll do this later

- download the CA certificates

322. Configure Database

- configure to MySQL Workbench:



Welcome to MySQL Workbench

Manage Server Connections

MySQL Connections: Local instance 3306, beer svc do

Connection Name: beer svc do

Connection | Remote Management | System Profile

Connection Method: Standard (TCP/IP) — Method to use to connect to the RDBMS

Parameters | SSL | Advanced

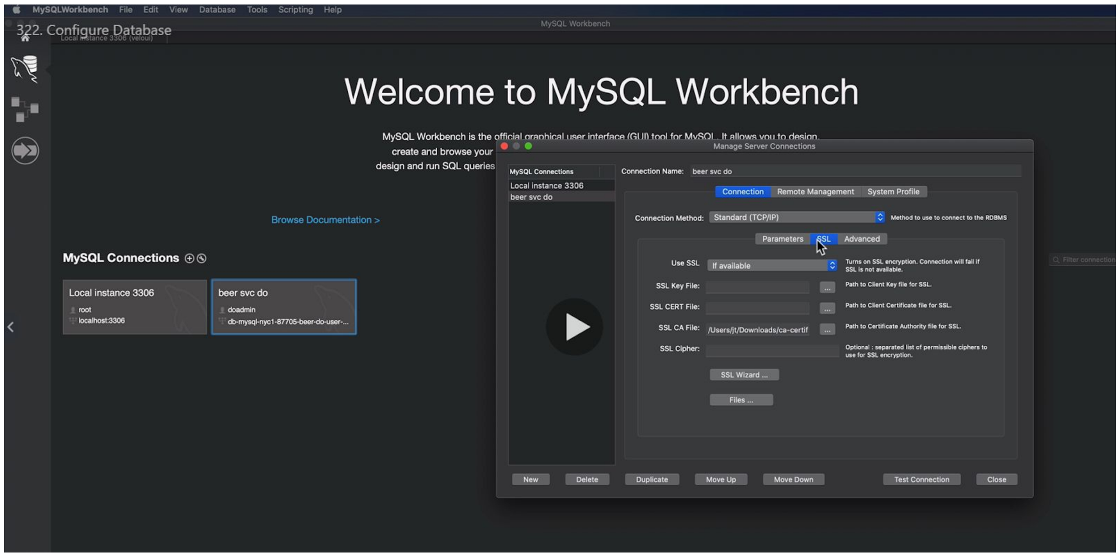
Hostname: db-mysql-nyc1-87705-be Port: 25060 — Name or IP address of the server host - and IP port.

Username: doadmin — Name of the user to connect with.

Password: Store in Keychain ... | Clear — The user's password. Will be requested later if it's not set.

Default Schema: defaultdb — The schema to use as default schema. Leave blank to select it later.

New | Delete | Duplicate | Move Up | Move Down | Test Connection | Close



322. Configure Database

Welcome to MySQL Workbench

Manage Server Connections

Connection Name: beer svc do

Connection Method: Standard (TCP/IP)

Parameters | SSL | Advanced

Use SSL: If available — Turns on SSL encryption. Connection will fail if SSL is not available.

SSL Key File: — Path to Client Key file for SSL.

SSL CERT File: — Path to Client Certificate file for SSL.

SSL CA File: /Users/jt/Downloads/ca-certif — Path to Certificate Authority file for SSL.

SSL Cipher: — Optional : separated list of permissible ciphers to use for SSL encryption.

SSL Wizard ... | Files ...

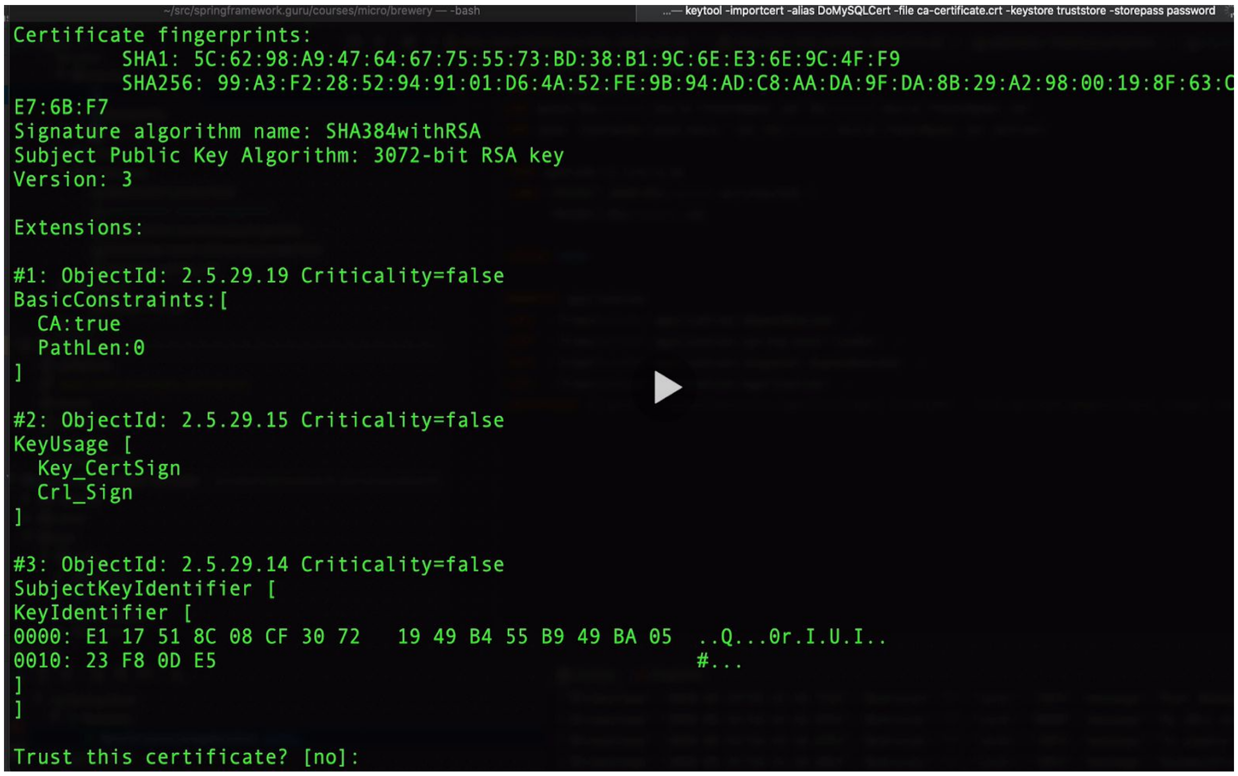
New | Delete | Duplicate | Move Up | Move Down | Test Connection | Close

- for each db, run the sql db creation script from MySQL Workbench

323. Configure Java Truststore

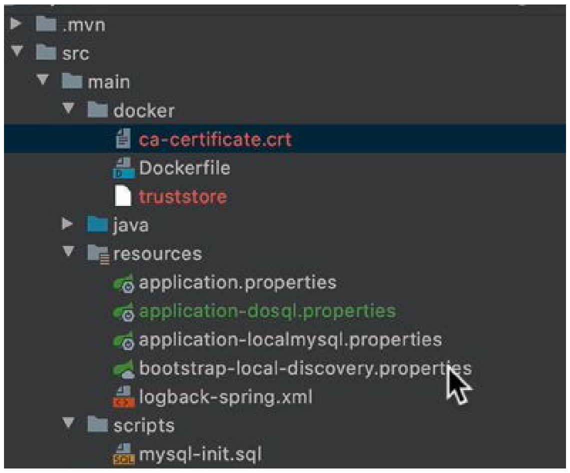
- convert the certificate into a Java Truststore
- run the command in the Docker folder / directory (as the certificate was copied there)

```
keytool -importcert -alias DoMySQLCert -file ca-certificate.crt -keystore
truststore -storepass password
```



```
~/src/springframework.guru/courses/micro/brewery — -bash        ...— keytool -importcert -alias DoMySQLCert -file ca-certificate.crt -keystore truststore -storepass password
Certificate fingerprints:
         SHA1: 5C:62:98:A9:47:64:67:75:55:73:BD:38:B1:9C:6E:E3:6E:9C:4F:F9
         SHA256: 99:A3:F2:28:52:94:91:01:D6:4A:52:FE:9B:94:AD:C8:AA:DA:9F:DA:8B:29:A2:98:00:19:8F:63:C
E7:6B:F7
Signature algorithm name: SHA384withRSA
Subject Public Key Algorithm: 3072-bit RSA key
Version: 3

Extensions:

#1: ObjectId: 2.5.29.19 Criticality=false
BasicConstraints:[
  CA:true
  PathLen:0
]

#2: ObjectId: 2.5.29.15 Criticality=false
KeyUsage [
  Key_CertSign
  Crl_Sign
]

#3: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: E1 17 51 8C 08 CF 30 72   19 49 B4 55 B9 49 BA 05  ..Q...0r.I.U.I..
0010: 23 F8 0D E5                                        #...
]
]

Trust this certificate? [no]:
```

-> creates the truststore file



```
▶ .mvn
▼ src
  ▼ main
    ▼ docker
        ca-certificate.crt
        Dockerfile
        truststore
    ▶ java
    ▼ resources
        application.properties
        application-dosql.properties
        application-localmysql.properties
        bootstrap-local-discovery.properties
        logback-spring.xml
    ▼ scripts
        mysql-init.sql
```

-> Edit Configuration for each app that uses one of these dbs
- under "**Program arguments**" add:
`-Djavax.net.ssl.trustStore=/path/to/truststore`
`-Djavax.net.ssl.trustStorePassword=password`

- add new .properties file for DigitalOcean dbs: **application-dosql.properties**
- update the url with the host and port from DO:
spring.datasource.url= jdbc:mysql://host:port/db?...
- update the active profiles from **local-discovery** and **local** to **local** and **dosql**

324. Add Truststore file to Docker Image

- copy the truststore file in the Docker image:

```
COPY truststore ./
```

325. Docker Image Release Process

- the profile is automatically activated if the Dockerfile exists

```
<profiles>
    <profile>
        <id>dockerbuild</id>
        <activation>
            <file>
                <exists>src/main/docker/Dockerfile</exists>
            </file>
        </activation>
        <build>
            <plugins>
                <!--   push to docker with release-->
                <plugin>
                <groupId>io.fabric8</groupId>
                <artifactId>docker-maven-plugin</artifactId>
                <executions>
                    <execution>
                    <id>push-to-docker</id>
                    <phase>deploy</phase>
                    <goals>
                        <goal>build</goal>
                        <goal>push</goal>
                    </goals>
                    </execution>
                </executions>
                </plugin>
            </plugins>
        </build>
    </profile>
 <profiles>
```

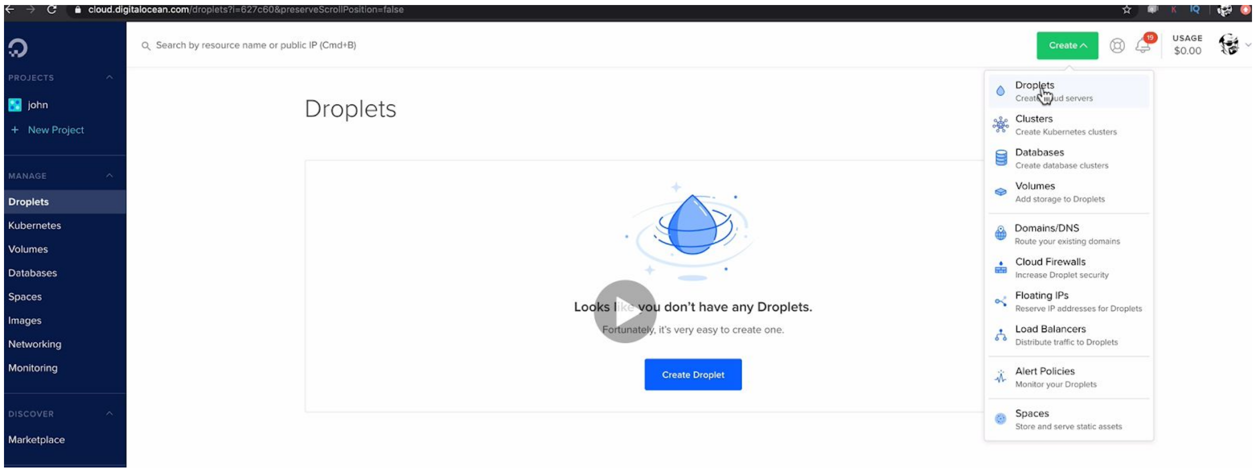
Release to Maven Central:

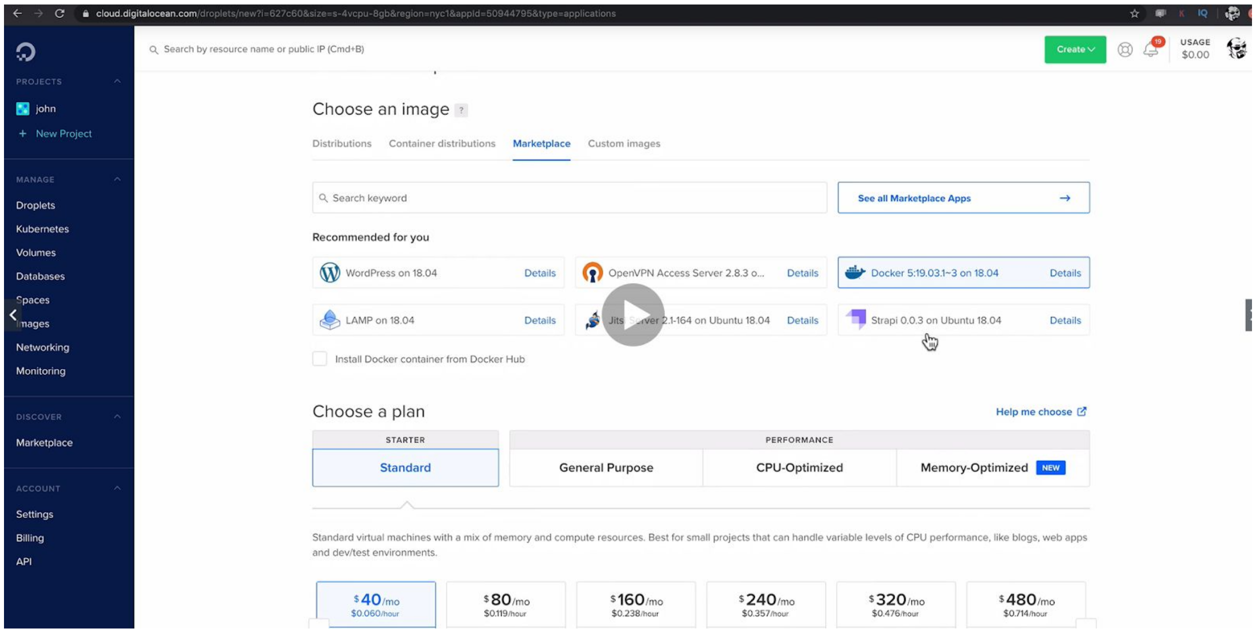
```
mvn release:prepare -P ossrh
mvn release:perform -P ossrh
```

// push the image and build the Docker image

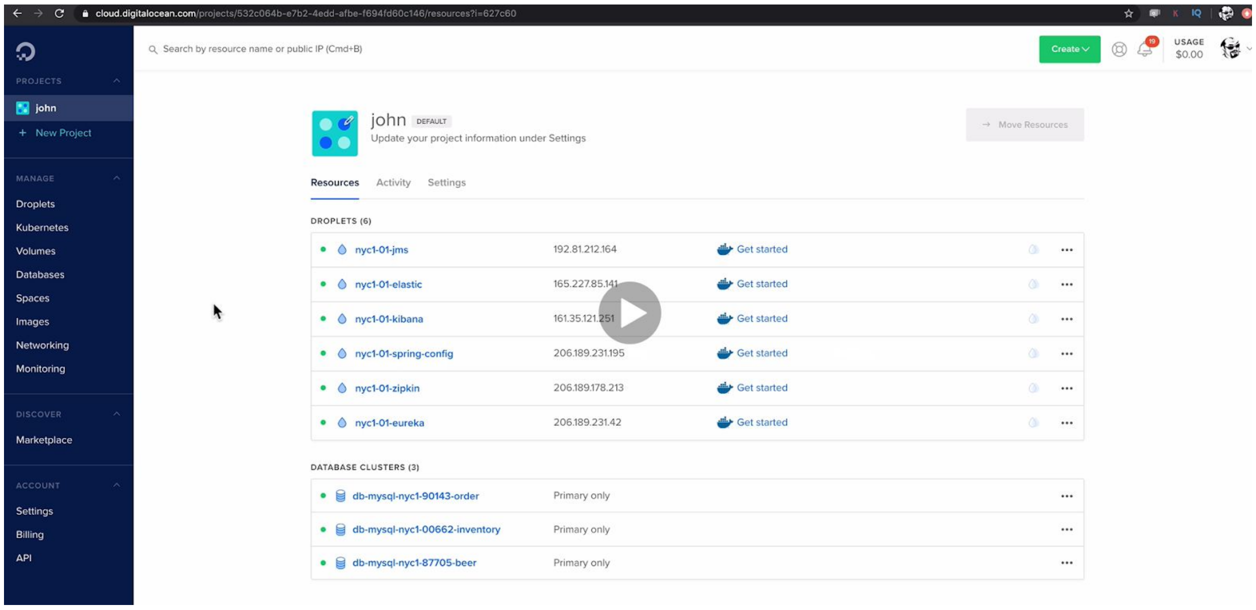
326. Provision Service VMs

- provision the service vms for our services
- they are called "**droplets**" by DigitalOcean


cloud.digitalocean.com/droplets?i=627c60&preserveScrollPosition=false
Search by resource name or public IP (Cmd+B)
Create
USAGE $0.00
PROJECTS
john
New Project
MANAGE
Droplets
Kubernetes
Volumes
Databases
Spaces
Images
Networking
Monitoring
DISCOVER
Marketplace
Droplets
Looks like you don't have any Droplets.
Fortunately, it's very easy to create one.
Create Droplet
Droplets
Create cloud servers
Clusters
Create Kubernetes clusters
Databases
Create database clusters
Volumes
Add storage to Droplets
Domains/DNS
Route your existing domains
Cloud Firewalls
Increase Droplet security
Floating IPs
Reserve IP addresses for Droplets
Load Balancers
Distribute traffic to Droplets
Alert Policies
Monitor your Droplets
Spaces
Store and serve static assets


cloud.digitalocean.com/droplets/new?i=627c60&size=s-4vcpu-8gb&region=nyc1&appId=50944795&type=applications
Choose an image
Distributions
Container distributions
Marketplace
Custom images
Search keyword
See all Marketplace Apps
Recommended for you
WordPress on 18.04
OpenVPN Access Server 2.8.3 o...
Docker 5:19.03.1~3 on 18.04
LAMP on 18.04
Jitsi Server 2.1-164 on Ubuntu 18.04
Strapi 0.0.3 on Ubuntu 18.04
Details
Install Docker container from Docker Hub
Choose a plan
Help me choose
STARTER
PERFORMANCE
Standard
General Purpose
CPU-Optimized
Memory-Optimized NEW
Standard virtual machines with a mix of memory and compute resources. Best for small projects that can handle variable levels of CPU performance, like blogs, web apps and dev/test environments.
$40/mo $0.060/hour
$80/mo $0.119/hour
$160/mo $0.238/hour
$240/mo $0.357/hour
$320/mo $0.476/hour
$480/mo $0.714/hour
ACCOUNT
Settings
Billing
API

- need Docker 19 or higher
- need the 4 GB / 2CPUs; 80 GB SSD disk; 4 TB transfer for $20 / month
- without storage - no volumes will be added
- need to create a new SSH key
- create 6 droplets for the 6 service instances - jms, elastic, kibana, zipnik, eureka, spring-config



john DEFAULT
Update your project information under Settings

DROPLETS (6)

nyc1-01-jms	192.81.212.164	Get started
nyc1-01-elastic	165.227.85.141	Get started
nyc1-01-kibana	161.35.121.251	Get started
nyc1-01-spring-config	206.189.231.195	Get started
nyc1-01-zipkin	206.189.178.213	Get started
nyc1-01-eureka	206.189.231.42	Get started

DATABASE CLUSTERS (3)

db-mysql-nyc1-90143-order	Primary only
db-mysql-nyc1-00662-inventory	Primary only
db-mysql-nyc1-87705-beer	Primary only

327. Configure JMS Server

(From Linux / Mac:)

```
ssh root@ip_of_jms_server -i ~/.ssh/credentials_for_do


docker run -d --rm -p 8161:8161 -p 61616:61616 vromero/activemq-artemis
```

Check the logs:

```
docker logs -f container_id
```

328. Configure Elasticsearch Server

Link.
(From Linux / Mac:)

```
ssh root@ip_of_elasticsearch_server -i ~/.ssh/credentials_for_do


docker run -d -p 9200:9200 -p 9300:9300 -e "discovery.type=single-node"
docker.elastic.co/elasticsearch/elasticsearch:7.9.1
```

329. Configure Kibana Server

Link.
(From Linux / Mac:)

```
ssh root@ip_of_kibana_server -i ~/.ssh/credentials_for_do
```

```
docker run -d --add-host elasticsearch:ip_for_elasticsearch_server -p
5601:5601 docker.elastic.co/kibana/kibana:7.9.1
```

330. Configure Zipkin Server

Will be run for localhost purposes, not for production (with a db).

(From Linux / Mac:)

```
ssh root@ip_of_zipkin_server -i ~/.ssh/credentials_for_do
```

```
docker run -d -p 9411:9411 openzipkin/zipkin
```

331. Configure Eureka Server

(From Linux / Mac:)

```
ssh root@ip_of_eureka_server -i ~/.ssh/credentials_for_do
```

```
docker run -d -p 8761:8761 mariamihai/sbm-brewery-gateway
```

332. Configure Spring Cloud Config Server

(From Linux / Mac:)

```
ssh root@ip_of_config_server -i ~/.ssh/credentials_for_do
```

```
docker run -d -p 8888:8888 \
-e EUREKA_CLIENT_SERVICEURL_DEFAULTZONE=\
http://netflix:eureka@ip_for_eureka_service:8761/eureka \
-e EUREKA_INSTANCE_PREFER_IP_ADDRESS=true \
mariamihai/sbm-config-server

docker run -d -p 8888:8888 \
-e eureka.client.service-url.defaultZone=\
http://netflix:eureka@ip_for_eureka_service:8761/eureka \
-e eureka.instance.prefer-ip-address=true \
mariamihai/sbm-config-server
```

333. Spring Cloud Config Server IP Address Update

Register the Config Server with the public IP not the one exposed by Docker:

```
docker run -d -p 8888:8888 \
-e eureka.client.service-url.defaultZone=\
http://netflix:eureka@ip_for_eureka_service:8761/eureka \
-e eureka.instance.prefer-ip-address=true \
-e eureka.instance.ip_address=public_ip_address \
mariamihai/sbm-config-server
```

334. Provision Docker Swarm Cluster

- create 3 droplets for the cluster
- pick the Docker image from the Marketplace when creating the new droplet
- use the $40/month plan, with 8GB / 4 CPUs; 160 GB SSD disk; 5 TB transfer

335. Linux Troubleshooting Commands

- view network information: `ifconfig -a`
- view processes listening on ports: `netstat -tulpn | grep LISTEN`
- see if the remote machine can 'see' the target machine: `ping <hostname or IP>`
- docker communicates via HTTP, thus you can use telnet: `telnet <IP Address> port`
(Use ctrl + c to exit)

- update the firewall settings (for all 3 nodes, to enable communication between them):

```
ufw allow 22/tcp
ufw allow 2376/tcp
ufw allow 7946/tcp
ufw allow 7946/udp
ufw allow 4789/udp
ufw reload
ufw enable
systemctl restart docker
```

336. Initialize Docker Swarm Cluster

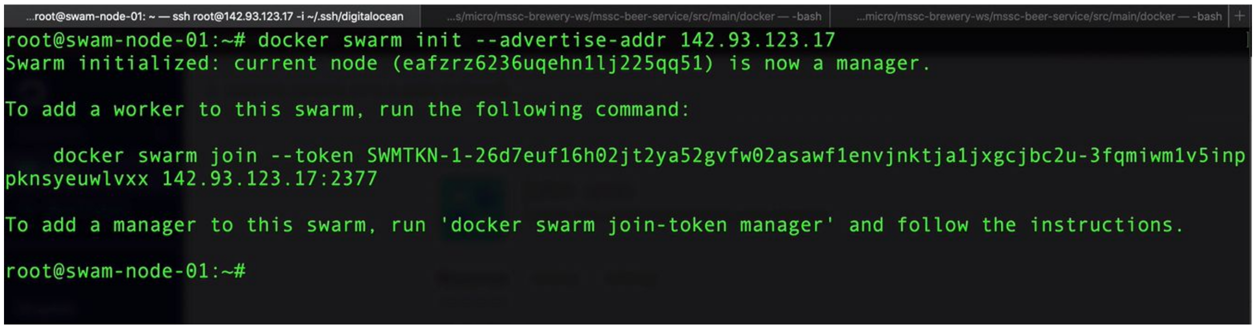
(From Linux / Mac:)

```
ssh root@ip_of_swarm_node_1_server -i ~/.ssh/credentials_for_do
```

- create a swarm

- create a manager node:

```
docker swarm init --advertise-addr ip_of_node
```


...root@swam-node-01: ~ — ssh root@142.93.123.17 -i ~/.ssh/digitalocean
...s/micro/mssc-brewery-ws/mssc-beer-service/src/main/docker — -bash
...micro/mssc-brewery-ws/mssc-beer-service/src/main/docker — -bash

```
root@swam-node-01:~# docker swarm init --advertise-addr 142.93.123.17
Swarm initialized: current node (eafzrz6236uqehn1lj225qq51) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-26d7euf16h02jt2ya52gvfw02asawf1envjnktja1jxgcjbc2u-3fqmiwm1v5inp
pknsyeuwlvxx 142.93.123.17:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

root@swam-node-01:~#
```

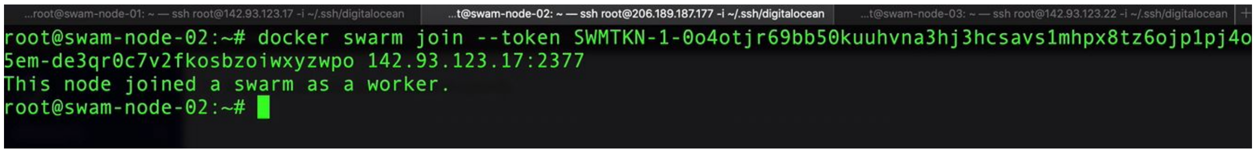
(From Linux / Mac:)

```
ssh root@ip_of_swarm_node_2_server -i ~/.ssh/credentials_for_do
```

```
docker swarm join --token ... ip_of_node:2377
```

- to add a worker - shows the command with the token to be copied and run on the other nodes:

```
docker swarm join-token worker
```

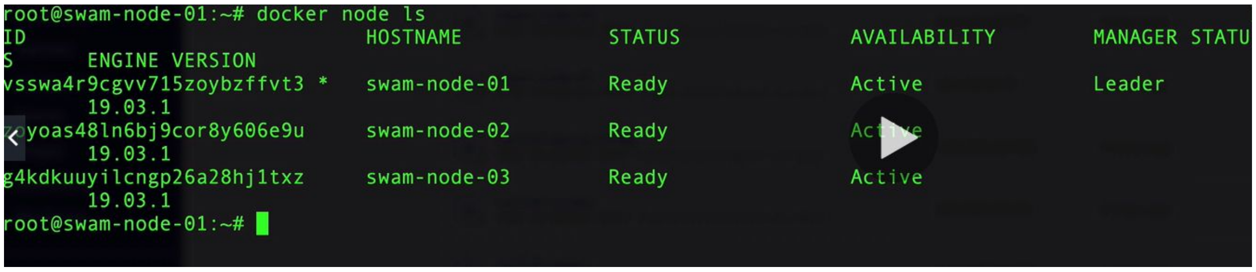

...root@swam-node-01: ~ — ssh root@142.93.123.17 -i ~/.ssh/digitalocean
...t@swam-node-02: ~ — ssh root@206.189.187.177 -i ~/.ssh/digitalocean
...t@swam-node-03: ~ — ssh root@142.93.123.22 -i ~/.ssh/digitalocean

```
root@swam-node-02:~# docker swarm join --token SWMTKN-1-0o4otjr69bb50kuuhvna3hj3hcsavs1mhpx8tz6ojp1pj4o
5em-de3qr0c7v2fkosbzoiwxyzwpo 142.93.123.17:2377
This node joined a swarm as a worker.
root@swam-node-02:~#
```

With this setup, there will be 1 manager and 2 worker nodes. For a production environment a 3 manager setup is recommended.

In the first node:

```
docker node ls
```

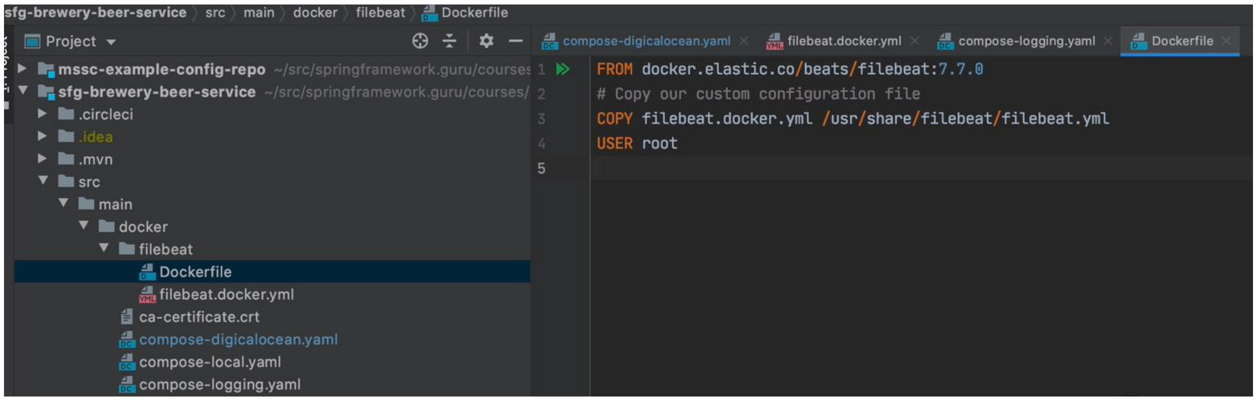


```
root@swam-node-01:~# docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATU
S          ENGINE VERSION
vsswa4r9cgvv715zoybzffvt3 *   swam-node-01        Ready               Active              Leader
           19.03.1
zoyoas48ln6bj9cor8y606e9u     swam-node-02        Ready               Active
           19.03.1
g4kdkuuyilcngp26a28hj1txz     swam-node-03        Ready               Active
           19.03.1
root@swam-node-01:~#
```

337. Filebeat Swarm Configuration

- create a Docker image with the .yml file needed for Filebeat (added as filebeat.docker.yml in the overview project, under docker/local-logging/filebeat/)

The Dockerfile:



```
FROM docker.elastic.co/beats/filebeat:7.7.0
# Copy our custom configuration file
COPY filebeat.docker.yml /usr/share/filebeat/filebeat.yml
USER root
```

The .yml file is mounted to the same folder we were mounting it in the docker compose file:

```
# ...
  filebeat:
    image: docker.elastic.co/beats/filebeat:7.7.0
    Volumes:
      # Configuration file
      - ./filebeat/filebeat.docker.yml:/usr/share/filebeat/filebeat.yml:ro
# ...
```

The image for Filebeat: **sfgbeerworks/filebeat**.

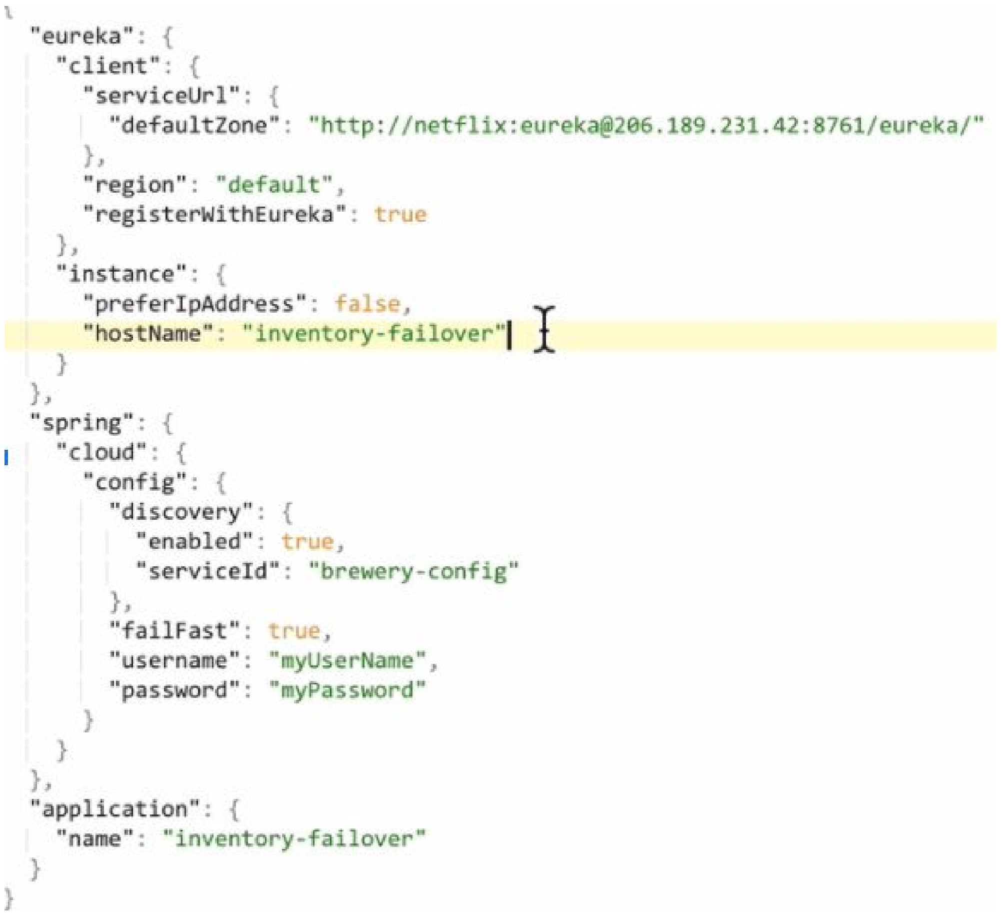
The profile for Digital Ocean: **compose-digicalocean.yaml** (specific to Docker Swarm):

```
version: '3.8'
services:
    filebeat:
        image: sfgbeerworks/filebeat:7.7.0
        Volumes:
            # Docker logs
            - /var/lib/docker/containers:/var/lib/docker/containers:ro
            # Additional information about containers
            - /var/run/docker.sock:/var/run/docker.sock:ro
        extra_hosts:
            - "elasticsearch:165.227.85.141"
        deploy:
            # Deploys one of these images on every node
            mode: global
            restart_policy:
                condition: on-failure

    inventory-service:
# Set version if needed, keep in mind 'latest' tag will only be
# pulled the first time, updates will not automatically get pulled
        image: sfgbeerworks/sfg-brewery-inventory-service
        ports:
            - 8082:8082
        environment:
            SPRING_PROFILES_ACTIVE: digitalocean
            SPRING_APPLICATION_JSON:
'{"eureka":{"client":{"serviceUrl":{"defaultZone":"http://netflix:eureka@206.189.23
1.42:8761/eureka/"},"region":"default","register-with-eureka":true},"instance":{"pr
eferIpAddress":false,"hostName":"inventory-service"}},"spring":{"cloud":{"config":{
"discovery":{"enabled":true,"serviceId":"brewery-config"},"failFast":true,"username
":"myUserName","password":"myPassword"}}},"application":{"name":"inventory-service"
}}'
        restart: on-failure
        labels:
            collect_logs_with_filebeat: "true"
            decode_log_event_to_json_object: "true"
        deploy:
            replicas: 2

    inventory-failover:
        image: sfgbeerworks/sfg-brewery-inventory-failover
        ports:
            - 8083:8083
        environment:
            SPRING_PROFILES_ACTIVE: digitalocean
            SPRING_APPLICATION_JSON:
'{"eureka":{"client":{"serviceUrl":{"defaultZone":"http://netflix:eureka@206.189.23
1.42:8761/eureka/"},"region":"default","register-with-eureka":true},"instance":{"pr
eferIpAddress":false,"hostName":"inventory-failover"}},"spring":{"cloud":{"config":
{"discovery":{"enabled":true,"serviceId":"brewery-config"},"failFast":true,"usernam
e":"myUserName","password":"myPassword"}}},"application":{"name":"inventory-failove
r"}}'
        deploy:
            replicas: 2

    beer-service:
        image: sfgbeerworks/sfg-brewery-beer-service
        ports:
            - 8080:8080
        restart: on-failure
        environment:
            SPRING_PROFILES_ACTIVE: digitalocean
            SPRING_APPLICATION_JSON:
'{"eureka":{"client":{"serviceUrl":{"defaultZone":"http://netflix:eureka@206.189.23
1.42:8761/eureka/"},"region":"default","register-with-eureka":true},"instance":{"pr
eferIpAddress":false,"hostName":"beer-service"}},"spring":{"cloud":{"config":{"disc
overy":{"enabled":true,"serviceId":"brewery-config"},"failFast":true,"username":"my
UserName","password":"myPassword"}}},"application":{"name":"beer-service"}}'
        labels:
            collect_logs_with_filebeat: "true"
            decode_log_event_to_json_object: "true"
        deploy:
            replicas: 2

    order-service:
        image: sfgbeerworks/sfg-brewery-order-service
        ports:
            - 8081:8081
        restart: on-failure
        environment:
            SPRING_PROFILES_ACTIVE: digitalocean
            SPRING_APPLICATION_JSON:
'{"eureka":{"client":{"serviceUrl":{"defaultZone":"http://netflix:eureka@206.189.23
1.42:8761/eureka/"},"region":"default","register-with-eureka":true},"instance":{"pr
eferIpAddress":false,"hostName":"order-service"}},"spring":{"cloud":{"config":{"dis
covery":{"enabled":true,"serviceId":"brewery-config"},"failFast":true,"username":"m
yUserName","password":"myPassword"}}},"application":{"name":"order-service"}}'
            SFG_BREWERY_BEER-SERVICE-HOST: http://beer-service:8080
        labels:
            collect_logs_with_filebeat: "true"
            decode_log_event_to_json_object: "true"
        deploy:
            replicas: 2

    gateway:
        image: sfgbeerworks/sfg-brewery-gateway
        ports:
            - 9090:9090
        restart: on-failure
        environment:
            SPRING_PROFILES_ACTIVE: digitalocean
            SPRING_APPLICATION_JSON:
'{"eureka":{"client":{"serviceUrl":{"defaultZone":"http://netflix:eureka@206.189.23
1.42:8761/eureka/"},"region":"default","register-with-eureka":false},"instance":{"p
referIpAddress":true}}}'
            SPRING_ZIPKIN_BASEURL: http://206.189.178.213:9411
```

```
labels:
    collect_logs_with_filebeat: "true"
    decode_log_event_to_json_object: "true"
deploy:
    replicas: 2
```

338. Eureka Swarm Configuration

- let Docker Swarm do the load balancing (the hostName is the same as the service name in the docker compose file)



```
{
  "eureka": {
    "client": {
      "serviceUrl": {
        "defaultZone": "http://netflix:eureka@206.189.231.42:8761/eureka/"
      },
      "region": "default",
      "registerWithEureka": true
    },
    "instance": {
      "preferIpAddress": false,
      "hostName": "inventory-failover"
    }
  },
  "spring": {
    "cloud": {
      "config": {
        "discovery": {
          "enabled": true,
          "serviceId": "brewery-config"
        },
        "failFast": true,
        "username": "myUserName",
        "password": "myPassword"
      }
    }
  },
  "application": {
    "name": "inventory-failover"
  }
}
```

- docker compose .yaml file for the inventory-failover service

```
inventory-failover:
        image: sfgbeerworks/sfg-brewery-inventory-failover
        ports:
            - 8083:8083
        environment:
            SPRING_PROFILES_ACTIVE: digitalocean
            SPRING_APPLICATION_JSON:
'{"eureka":{"client":{"serviceUrl":{"defaultZone":"http://netflix:eureka@206.189.231.42:8761/eureka/"},"region":"default","register-with-eureka":true},"instance":{"prefer
IpAddress":false,"hostName":"inventory-failover"}},"spring":{"cloud":{"config":{"disc
overy":{"enabled":true,"serviceId":"brewery-config"},"failFast":true,"username":"myUs
erName","password":"myPassword"}}},"application":{"name":"inventory-failover"}}'
        deploy:
            replicas: 2
```

339. Spring Cloud Configuration

340. Digital Ocean Profile

- enable the digitalocean profile with the BeerInventoryServiceFeign - `@Profile({"local-discovery", "digitalocean"})`

- make sure BeerInventoryServiceRestTemplateImpl is not used with local-discovery and digitalocean profiles (this feature was added in 5.1) - `@Profile("!local-discovery & !digitalocean")`

341. Running Microservices with Docker Swarm

- initialize the Docker Swarm

```
docker swarm deploy --compose-file file_name.yaml beerstack
```

- bring down the Swarm cluster

```
docker swarm rm beerstack
```

342. Tracing Requests for Troubleshooting

Check what is running across the entire swarm network:

```
docker stack ps beerstack
```

343. Zipkin Tracing

```
spring.zipkin.enabled=true
spring.zipkin.base-url=http://ip:port/ # defaults to port 9411
```

344. Tasting Room Service Challenge

-

345. Retrospective

-