Deploy Application using Swarm Mode

Docker Engine includes swarm mode for natively managing a cluster of Docker Engines called a Swarm. Use the Docker CLI to create a swarm, deploy application services to a swarm, and manage swarm behavior. Complete details at: https://docs.docker.com/engine/swarm/. It's important to understand the Swarm mode key concepts of Swarm, Node, Service, Task and Load Balancing.

Swarm is typically a cluster of multiple Docker Engines. But for simplicity we'll run a single node Swarm.

This section will deploy an application that will provide a CRUD/REST interface on a data bucket in Couchbase. This is achieved by using a Java EE application deployed on WildFly to access the database.

Initialize a Swarm

Initialize Swarm mode using the following command:

```
docker swarm init
```

This starts a Swarm Manager. By default, manager node are also worker but can be configured to be manager-only.

Find some information about this one-node cluster using the command `docker info`

```
Containers: 0
 Running: 0
 Paused: 0
 Stopped: 0
Images: 17
Server Version: 1.13.0
Storage Driver: overlay2
 Backing Filesystem: extfs
 Supports d_type: true
 Native Overlay Diff: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
 Volume: local
 Network: bridge host ipvlan macvlan null overlay
Swarm: active
 NodeID: 92mydh0e09ba5hx3wtmcmvktz
 Is Manager: true
 ClusterID: v68ikyaff7rdxpaw1j0c9i60s
 Managers: 1
 Nodes: 1
 Orchestration:
  Task History Retention Limit: 5
 Raft:
  Snapshot Interval: 10000
  Number of Old Snapshots to Retain: 0
  Heartbeat Tick: 1
  Election Tick: 3
```

```
 Dispatcher:
  Heartbeat Period: 5 seconds
 CA Configuration:
  Expiry Duration: 3 months
 Node Address: 192.168.65.2
 Manager Addresses:
  192.168.65.2:2377
Runtimes: runc
Default Runtime: runc
Init Binary: docker-init
containerd version: 03e5862ec0d8d3b3f750e19fca3ee367e13c090e
runc version: 2f7393a47307a16f8cee44a37b262e8b81021e3e
init version: 949e6fa
Security Options:
 seccomp
  Profile: default
Kernel Version: 4.9.5-moby
Operating System: Alpine Linux v3.5
OSType: linux
Architecture: x86_64
CPUs: 4
Total Memory: 1.952 GiB
Name: moby
ID: SGCM:KDRD:G3M7:PZHN:J4RL:VFFR:G2SR:EKD5:JV4J:RL3X:LF7T:XF6V
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): true
 File Descriptors: 31
 Goroutines: 124
 System Time: 2017-01-27T08:25:58.032295342Z
 EventsListeners: 1
No Proxy: *.local, 169.254/16
Username: arungupta
Registry: https://index.docker.io/v1/
Experimental: true
Insecure Registries:
 127.0.0.0/8
Live Restore Enabled: false
```

This cluster has 1 node, and that is a manager.

Multi-container Application

This section describes how to deploy a multi-container application using Docker Compose to Swarm mode use Docker CLI.

Create a new directory and `cd` to it:

```
mkdir webapp
cd webapp
```

Create a new Compose definition `docker-compose.yml` using the configuration file shown below.

```
version: '3'
services:
  web:
    image: arungupta/couchbase-javaee:travel
    environment:
      - COUCHBASE_URI=db
    ports:
      - 8080:8080
      - 9990:9990
    depends_on:
      - db
  db:
    image: arungupta/couchbase:travel
    ports:
      - 8091:8091
      - 8092:8092
      - 8093:8093
      - 11210:11210
```

In this Compose file:

1. `arungupta/couchbase-javaee:travel` image is WildFly application server with a Java EE application predeployed. `COUCHBASE_URI` environment variable is used to read the host address of Couchbase server. `depends_on` ensures that `web` service starts before `db` service. This only ensures that the container is started first but the application readiness needs to be ensured within the application itself.

2. `arungupta/couchbase-javaee:travel` image is used for Couchbase server. This image pre-configures the Couchbase server and loads `travel-sample` bucket with ~32k JSON documents.

This application can be started as:

```
docker stack deploy --compose-file=docker-compose.yml webapp
```

This shows the output as:

```
Creating network webapp_default
Creating service webapp_web
Creating service webapp_db
```

WildFly and Couchbase services are started on this node. Each service has a single container. If the Swarm mode is enabled on multiple nodes then the containers will be distributed across multiple nodes.

A new overlay network is created. This allows multiple containers on different hosts to communicate with each other.

Verify Service/Containers in Application

Verify that the WildFly and Couchbase services are running using `docker service ls`:

```
ID            NAME        MODE        REPLICAS  IMAGE
a9pkiziw3vgw  webapp_db   replicated  1/1       arungupta/couchbase:travel
hr5s6ue54kwj  webapp_web  replicated  1/1       arungupta/couchbase-javaee:travel
```

More details about the service can be obtained using `docker service inspect webapp_web`:

```
[
    {
        " ID " : " z4ljnqteeqwnqcd8kwc5pod5k " ,
        " Version " : {
            " Índice " : 2611
        },
        " CreatedAt " : " 2017-02-01T23: 12: 34.84286773Z " ,
        " UpdatedAt " : " 2017-02-01T23: 12: 34.86880268Z " ,
        " Spec " : {
            " Nombre " : "Webapp_web " ,
            " Etiquetas " : {
                " com.docker.stack.namespace " : " webapp "
             },
            " TaskTemplate " : {
                " ContainerSpec " : {
                    " Imagen " : " arungupta / couchbase-javaee: travel@sha256: 7749f40f590108b76e0cbb7967d39e0a9d363
                    " Etiquetas " : {
                        " com.docker.stack.namespace ": " Webapp "
                     },
                    " Env " : [
                        " COUCHBASE_URI = db "
                     ]
                },
                " Recursos " : {},
                " Colocación " : {},
                " ForceUpdate " : 0
            },
            " Mode " : {
                " replicados " : {
                    " Replicas " : 1
                }
            },
            "Redes " : [
                {
                    " objetivo " : " s2b2qevxus11a21ebjseaj5g6 " " Alias " : [ " web " ] } ], " EndpointSpec " : { "
```

Los registros para el servicio se pueden ver usando `docker service logs -f webapp_web` :

```
webapp_web.1.wby0b04t7bap@moby     | ========================================================================
webapp_web.1.wby0b04t7bap@moby     |
```

```
webapp_web.1.wby0b04t7bap@moby    |   JBoss Bootstrap Environment
webapp_web.1.wby0b04t7bap@moby    |
webapp_web.1.wby0b04t7bap@moby    |   JBOSS_HOME: /opt/jboss/wildfly
webapp_web.1.wby0b04t7bap@moby    |
webapp_web.1.wby0b04t7bap@moby    |   JAVA: /usr/lib/jvm/java/bin/java
webapp_web.1.wby0b04t7bap@moby    |
webapp_web.1.wby0b04t7bap@moby    |   JAVA_OPTS:  -server -Xms64m -Xmx512m -XX:MetaspaceSize=96M -XX:MaxMetaspaceSize
webapp_web.1.wby0b04t7bap@moby    |
webapp_web.1.wby0b04t7bap@moby    | =========================================================================

. . .

webapp_web.1.wby0b04t7bap@moby    | 23:14:15,811 INFO  [org.jboss.as.server] (ServerService Thread Pool -- 34) WFLYSR
webapp_web.1.wby0b04t7bap@moby    | 23:14:16,076 INFO  [org.jboss.as] (Controller Boot Thread) WFLYSRV0060: Http mana
webapp_web.1.wby0b04t7bap@moby    | 23:14:16,077 INFO  [org.jboss.as] (Controller Boot Thread) WFLYSRV0051: Admin cor
webapp_web.1.wby0b04t7bap@moby    | 23:14:16,077 INFO  [org.jboss.as] (Controller Boot Thread) WFLYSRV0025: WildFly F
```

Asegúrese de esperar a que aparezca la última instrucción de registro.

Aplicación de acceso

Ahora que los servidores WildFly y Couchbase han sido configurados, vamos a acceder a la aplicación. Es necesario especificar la dirección IP del host donde WildFly se está ejecutando (`localhost` en nuestro caso).

El punto final se puede acceder en este caso como:

```
Curl -v http: // localhost: 8080 / airlines / resources / airline
```

La salida se muestra como:

Ejemplo 1. Obtener todos los recursos de salida

```
*   Trying ::1...
* Connected to localhost (::1) port 8080 (#0)
> GET /airlines/resources/airline HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.43.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Connection: keep-alive
< X-Powered-By: Undertow/1
< Server: WildFly/10
< Content-Type: application/octet-stream
< Content-Length: 1402
< Date: Wed, 01 Feb 2017 03:04:41 GMT
<
* Connection #0 to host localhost left intact
[{"travel-sample":{"country":"United States","iata":"Q5","callsign":"MILE-AIR","name":"40-Mile Air","icao":"MLA","id"
```

Esto muestra 10 aerolíneas desde el `travel-sample` balde.

El conjunto completo de comandos se muestra en https://github.com/docker/labs/blob/master/developer-tools/java/chapters/ch05-compose.adoc#access-application .

Detener la aplicación

If you only want to stop the application temporarily while keeping any networks that were created as part of this application, the recommended way is to set the amount of service replicas to 0.

```
docker service scale webapp_db=0 webapp_web=0
```

This is especially useful if the stack contains volumes and you want to keep the data. It allows you to simply start the stack again with setting the replicas to a number higher than 0.

Remove Application Completely

Shutdown the application using `docker stack rm webapp`:

```
Removing service webapp_db
Removing service webapp_web
Removing network webapp_default
```

This stops the container in each service and removes the services. It also deletes any networks that were created as part of this application.