Building a TomEE cluster using Docker

1. Setup Docker
   1. Go to <https://www.docker.com/> and follow the steps to install Docker at your OS
   2. The Docker download is at:

<https://www.docker.com/products/docker-toolbox>

* 1. If you are not used to Docker, I recommend you to follow its first tutorial. You will spend half an hour to understand its general features
  2. Once you are ready, its time to build our TomEE container.

1. Build a TomEE container with Docker
   1. Go to <https://hub.docker.com/>
   2. On “Search” textbox, write “tomee” and press Enter
   3. Click in the result called “tomee official”
   4. In the next page you will find some informations about this image
   5. Open de “Docker Quickstart Terminal” (in Mac OS X, or its equivalents in others OS’s)
   6. Once the terminal is opened, check the message where says:

“docker is configured to use the default machine with IP...”

* 1. Take a note of this IP, you will need it to access the TomEE running in your container
  2. Execute this command:

docker run -it --rm -p 8080:8080 tomee:8-jre-1.7.2-webprofile

* 1. Hey! Don’t forget the “-p 8080:8080” above. You are telling the Docker to expose this port, otherwise you won’t be able to access it from your host.

1. Deploy your app at your container
   1. Stop your running TomEE (CTRL + C), choose a folder and run this command:

mkdir mytomeedocker

* 1. Put inside this folder
     1. Your <app>.war
     2. A standard server.xml from the same TomEE version
  2. Go to your this new folder and create a Dockerfile

touch Dockerfile

* 1. Open the file

open -e Dockerfile

* 1. Write those lines

FROM tomee:8-jre-1.7.2-webprofile

ADD <app>.war /usr/local/tomee/webapps/<app>.war

ADD server.xml /usr/local/tomee/conf/server.xml

* 1. Change the <app> to the very name of you .war file
  2. Save your Dockerfile and close it
  3. Check if your lines were saved:

cat Dockerfile

* 1. Edit the server.xml file and add those lines (at Engine node):

<Cluster className=*"org.apache.catalina.ha.tcp.SimpleTcpCluster"*

channelSendOptions=*"6"*>

<Manager className=*"org.apache.catalina.ha.session.BackupManager"*

expireSessionsOnShutdown=*"false"* notifyListenersOnReplication=*"true"*

mapSendOptions=*"6"* />

<!-- <Manager className="org.apache.catalina.ha.session.DeltaManager" expireSessionsOnShutdown="false"

notifyListenersOnReplication="true"/> -->

<Channel className=*"org.apache.catalina.tribes.group.GroupChannel"*>

<Membership className=*"org.apache.catalina.tribes.membership.McastService"*

address=*"228.0.0.4"* port=*"45564"* frequency=*"500"* dropTime=*"3000"* />

<Receiver className=*"org.apache.catalina.tribes.transport.nio.NioReceiver"*

address=*"auto"* port=*"5000"* selectorTimeout=*"100"* maxThreads=*"6"* />

<Sender className=*"org.apache.catalina.tribes.transport.ReplicationTransmitter"*>

<Transport

className=*"org.apache.catalina.tribes.transport.nio.PooledParallelSender"* />

</Sender>

<Interceptor

className=*"org.apache.catalina.tribes.group.interceptors.TcpFailureDetector"* />

<Interceptor

className=*"org.apache.catalina.tribes.group.interceptors.MessageDispatch15Interceptor"* />

<Interceptor

className=*"org.apache.catalina.tribes.group.interceptors.ThroughputInterceptor"* />

</Channel>

<Valve className=*"org.apache.catalina.ha.tcp.ReplicationValve"*

filter=*".\*\.gif|.\*\.js|.\*\.jpeg|.\*\.jpg|.\*\.png|.\*\.htm|.\*\.html|.\*\.css|.\*\.txt"* />

<Deployer className=*"org.apache.catalina.ha.deploy.FarmWarDeployer"*

tempDir=*"/tmp/war-temp/"* deployDir=*"/tmp/war-deploy/"* watchDir=*"/tmp/war-listen/"*

watchEnabled=*"false"* />

<ClusterListener

className=*"org.apache.catalina.ha.session.ClusterSessionListener"* />

</Cluster>

1. Building your appliance
   1. Now lets build your appliance! Run this command:

docker build -t tomee-war --build-arg WAR\_FILE=app\_test.war .

* 1. IMPORTANT: do not forget the dot (“.”) in the end of command. It won’t work if you do
  2. Wow! Now you have your own Docker image, with your own application. Let’s run it (open one Docker terminal for each command):

docker run --name host1 -p 8080:8080 tomee-war

docker run --name host2 -p 8081:8080 tomee-war

docker run --name host3 -p 8082:8080 tomee-war

* 1. We did it three times because we don’t want a cluster with just one node!
     1. Important: on the next time you start those nodes, just run “docker start host1” (or other hosts)
  2. Now you have three instances of you appliance running in cluster (you can check on their logs and confirm if they found each other).

1. Bringing a load balancer to the architecture
   1. Open a docker Terminal and run this command:

docker pull jasonwyatt/nginx-loadbalancer

* 1. It will download a docker image with nginx load balancer
  2. Go to your mytomeedocker folder and create a env.list file with this content:

TOMCAT\_1\_PORT\_8080\_TCP\_ADDR=host1

TOMCAT\_2\_PORT\_8080\_TCP\_ADDR=host2

TOMCAT\_3\_PORT\_8080\_TCP\_ADDR=host3

TOMCAT\_PATH=/app\_test

TOMCAT\_REMOTE\_PORT=8080

TOMCAT\_REMOTE\_PATH=/app\_test

\*I used “app\_test” as app name. Change to your app’s name.

* 1. Save the file and close it
  2. In the same folder, let’s run the command that will start you load balancer

docker run --name loadbalancer -p 80:80 --link host1:host1 --link host2:host2 --link host3:host3 --env-file ./env.list jasonwyatt/nginx-loadbalancer

* 1. When it shows “Starting Nginx...”, it’s done

1. Testing your architecture
   1. Open a browser and, using the IP that Docker gave to you, go to (for example):

<http://192.168.99.100/app_test/>

* 1. My app print the host that is giving the response by simply adding this line to index.jsp:

<%="Host: " + InetAddress.getLocalHost().getHostAddress()%>

* 1. If everything is working fine, it will show you one of your nodes IP. To check if your cluster is fine, stop that node and refresh the page. The value printed will change!