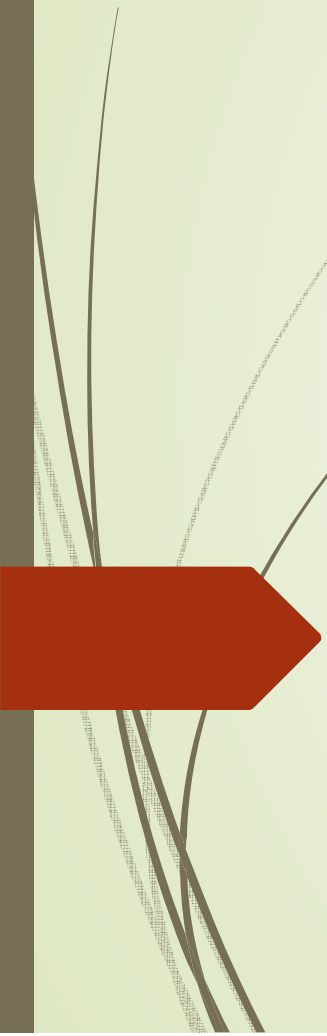


Vagrant & Docker

AMT – Labo 2 (more or less)

Authors: Bignens Julien & Brito Carvalho Bruno

Summary

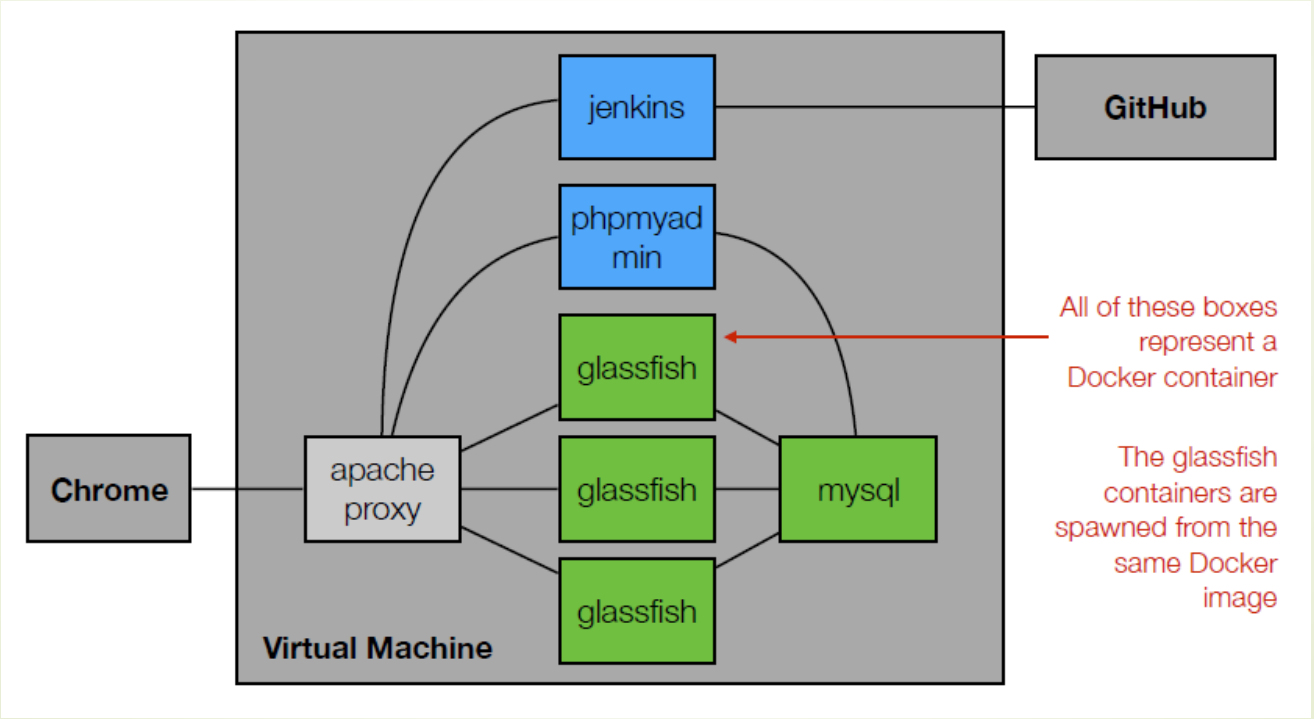
- Introduction
- Architecture
- What we actually do
- Sharing the volumes
- Some magic
- Basic example
- Vagrantfile
- Dockers
- A few tricks

Introduction

- This presentation is about our lab on Vagrant and Docker
- The objective was to build an entire architecture of continuous integration and then deploy our first application on it
- Using two technologies:
  - Vagrant
  - Docker
- Repo : https://github.com/bbcnt/AMT_Vagrecker (private ATM)

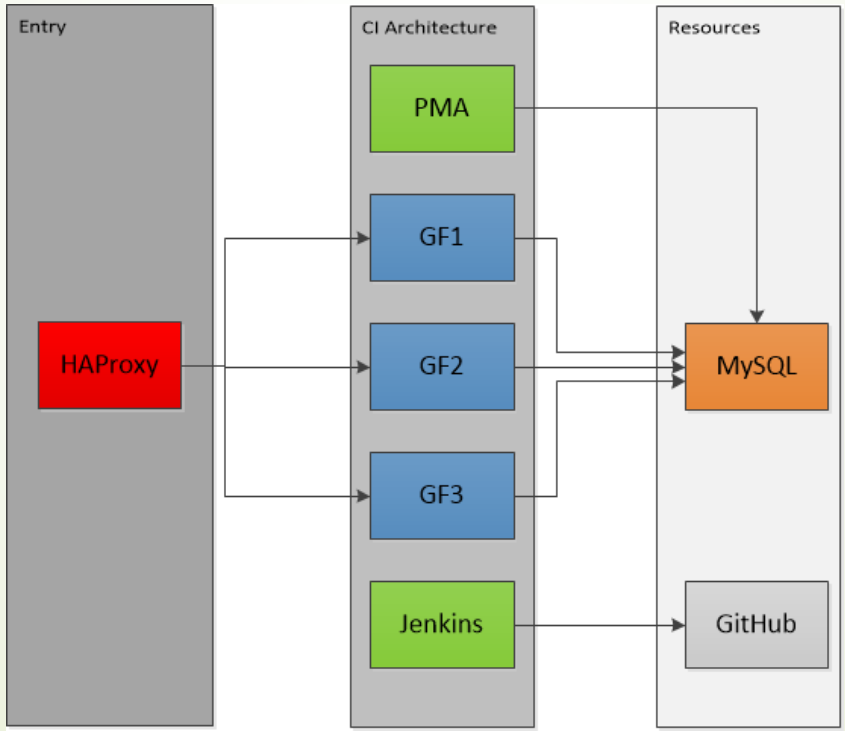
Architecture – The base

- This is the wanted result:



Architecture – Our version

- Let's take a look at the architecture that was built:



Architecture - Differences

- What's different ?
  - We replaced Apache by HAProxy
  - Why?
    - Because it was easier, and also, because it's a real Load Balancer
  - One problem remains though
    - We can't access the other services (Jenkins, PHPMyAdmin) with the same IP.
    - Example:
      - Jenkins: http://localhost:9001
      - PHPMyAdmin: http://localhost:9004
      - Web Application: http://localhost:9002
    - We make use of Docker's NAT system (mapping IP and Ports)

What we actually do

- Generate a .war file from the Maven project with Jenkins (the Maven project is pulled from GitHub)
- Deploy this project on a glassfish instance
- Link everything so the glassfish server can work with the MySQL database.
- Add a few tools, like PHPMyAdmin, to be able to work efficiently with MySQL.
- Add a few other instances of Glassfish (replicates of the first one).
- Put in some load balancing and we are done.

Sharing the volumes

- We use the sharing of volumes offered by Docker to make our stuff work.
- By sharing the autodeploy directory of gf1 with Jenkins and the other gf
  - (gf meaning glassfish, not girlfriend)

Containers Name	Shared volumes	Links	Volumes From
mysql	-	-	-
phpmyadmin	-	mysql	-
gf1	/opt/glassfish/glassfish4/glassfish/domains/domain1/autodeploy/	mysql	-
gf2	-	mysql	gf1
gf3	-	mysql	gf1
jenkins	-	gf1	gf1
haproxy	-	gf1 / gf2 / gf3	-

- We can make some magic work.

Sharing the volumes

- So, sharing this repo on gf1:
  - /opt/glassfish/glassfish4/glassfish/domains/domain1/autodeploy/
- Will create the same directory on those who make use of this volume.
- It will be very useful for deploying the application.

```
#Gf1 is sharing the autodeploy repertory, once a war file is put in this directory, it is automatically deployed by GF.
d.run "gf1", image: "heig/glassfish", args: "-p 4081:4848 -p 4082:8080 --link mysql:mysql -v /opt/glassfish/glassfish4/glassfish/domains/domain1/autodeploy/"
#And here, we use link the autoeploy repo of gf1, meaning, all the other GF servers will be updated once we update the first one.
d.run "gf2", image: "heig/glassfish", args: "-p 4083:4848 -p 4084:8080 --link mysql:mysql --volumes-from gf1"
d.run "gf3", image: "heig/glassfish", args: "-p 4085:4848 -p 4086:8080 --link mysql:mysql --volumes-from gf1"
```

Some Magic

- Glassfish offers a pretty cool thing, inside a domain, there is an "autodeploy" directory. Any war/jar/ear file inside will be deployed !
- So for Jenkins, it's easy, once it creates a war, it only need to copy it.
- Where?
  - Remember the shared volume?
  - By sharing the volumes from gf1 with the other gf instances and Jenkins, we only have to do this after the build on Jenkins:
  - cp $JENKINS_HOME/jobs/AMT_REST/workspace/target/AMT_REST-1.0-SNAPSHOT.war /opt/glassfish/glassfish4/glassfish/domains/domain1/autodeploy/AMT_REST.war
- And… that's all ;=)

Basic example

```
brito_000@BBC-LENOVO /F/VMs/AMT_Vagrant
$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Clearing any previously set forwarded ports...
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
    default: Adapter 2: hostonly
==> default: Forwarding ports...
    default: 7070 => 9001 (adapter 1)
    default: 9090 => 9002 (adapter 1)
    default: 3306 => 9003 (adapter 1)
    default: 5050 => 9004 (adapter 1)
    default: 4081 => 9005 (adapter 1)
    default: 4082 => 9006 (adapter 1)
    default: 4084 => 9007 (adapter 1)
    default: 4086 => 9008 (adapter 1)
    default: 4087 => 9009 (adapter 1)
    default: 22 => 2222 (adapter 1)
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
    default: SSH address: 127.0.0.1:2222
    default: SSH username: vagrant
    default: SSH auth method: private key
    default: Warning: Connection timeout. Retrying...
==> default: Machine booted and ready!
==> default: Checking for guest additions in VM...
==> default: Configuring and enabling network interfaces...
==> default: Mounting shared folders...
    default: /vagrant => F:/VMs/AMT_Vagrant
==> default: Machine already provisioned. Run `vagrant provision` or use the `--provision`
==> default: to force provisioning. Provisioners marked to run always will still run.
```

Basic example

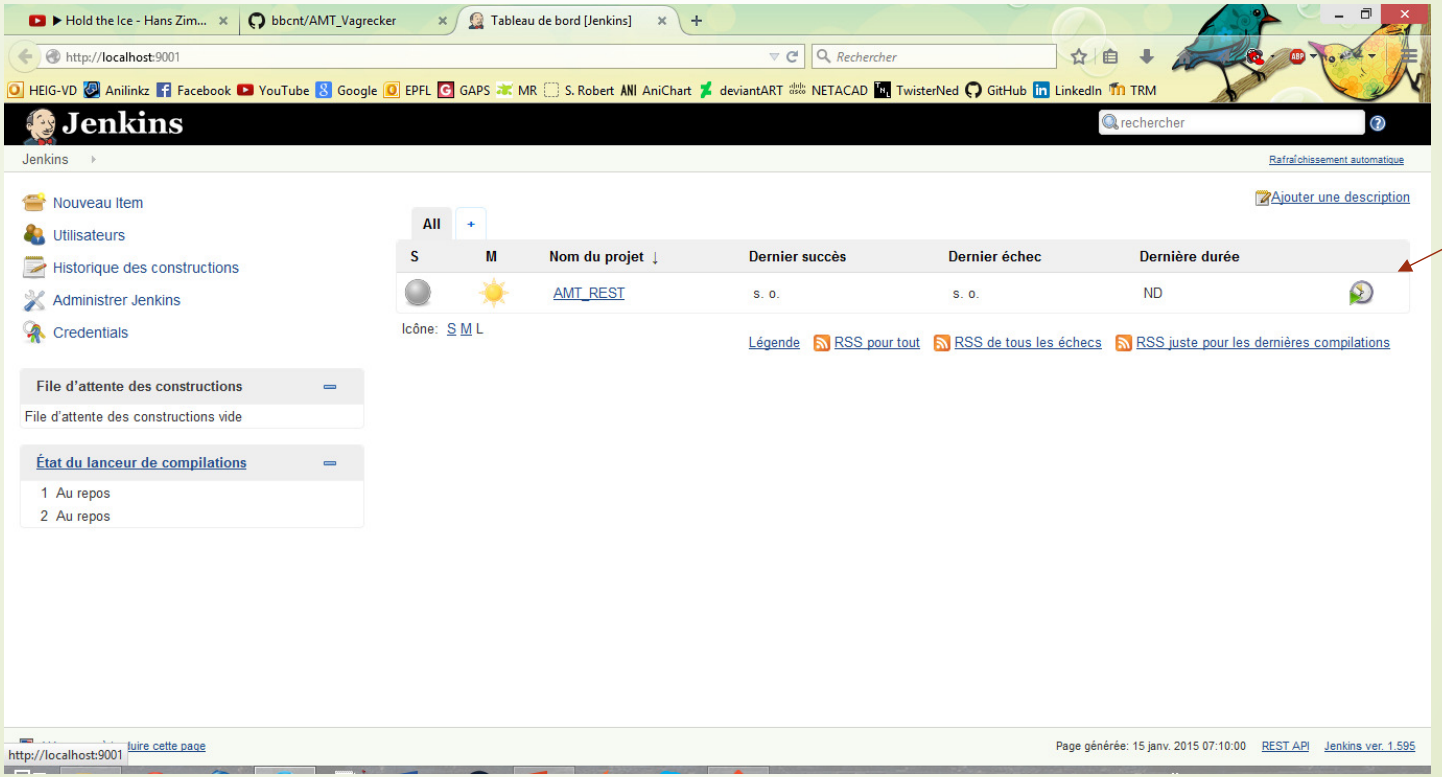
- We will now see how to make the AMT REST project work.
- First, we need to open a client in the Vagrant root directory
  - Where you'll find the Vagrantfile
- Start by typing:
  - vagrant up
  - vagrant provision (if not the first time)
- Then go to Jenkins : http://localhost:9001, click on the build button and wait.
- Access the app on http://localhost:9002/AMT_REST

Basic example

- Vagrant provision

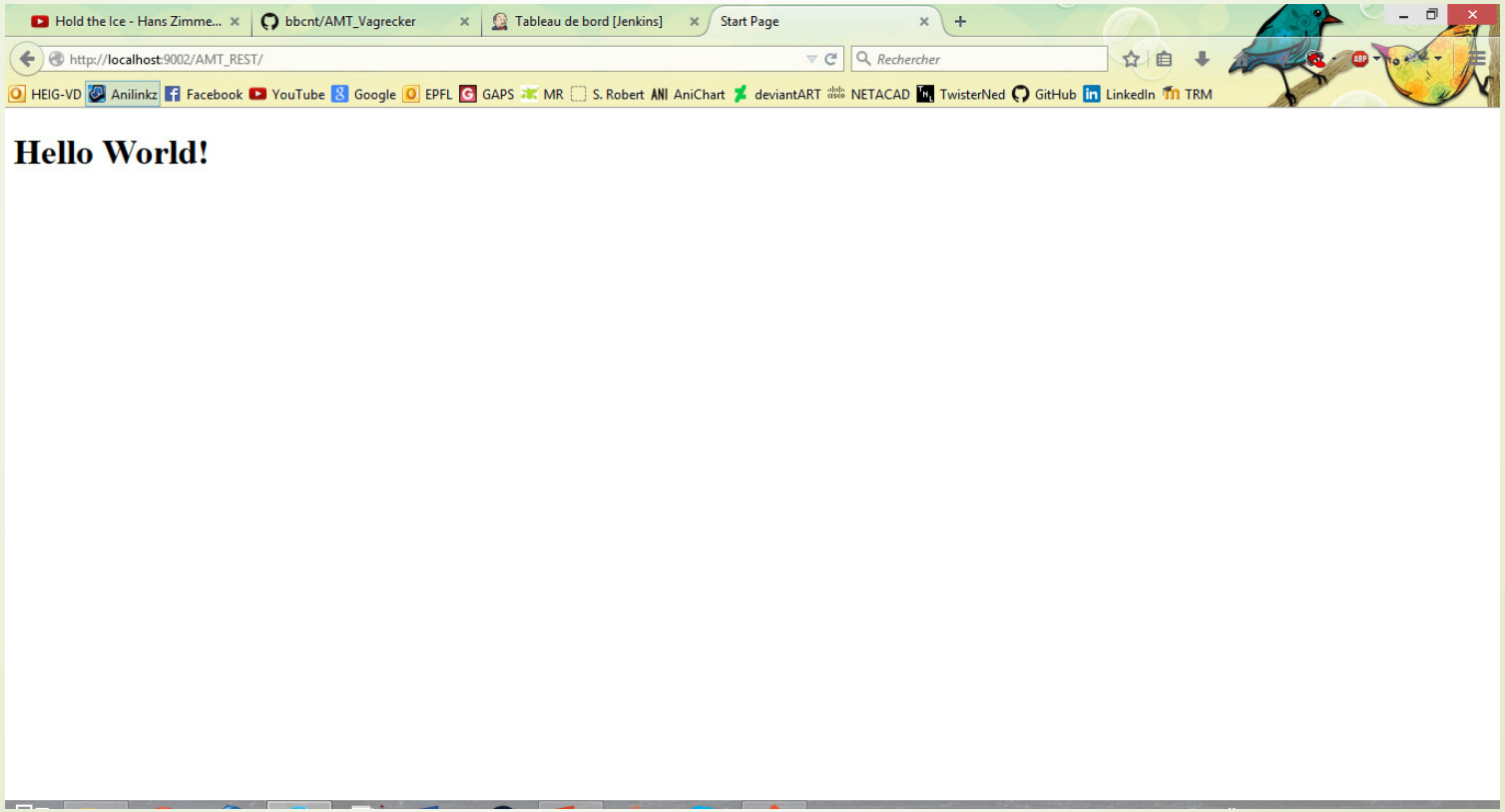
```
==> default:  ---> Using cache
==> default:  ---> 5c86f9cd4a90
==> default: Step 2 : ADD haproxy.cfg /etc/haproxy/haproxy.cfg
==> default:  ---> Using cache
==> default:  ---> ac40235408f9
==> default: Step 3 : ADD start.bash /haproxy-start
==> default:  ---> Using cache
==> default:  ---> 9868e99b932f
==> default: Step 4 : VOLUME /haproxy-override
==> default:  ---> Using cache
==> default:  ---> a4ba7f0fed33
==> default: Step 5 : WORKDIR /etc/haproxy
==> default:  ---> Using cache
==> default:  ---> 6ae25177a61f
==> default: Step 6 : CMD bash /haproxy-start
==> default:  ---> Using cache
==> default:  ---> c2ba0b3d20ea
==> default: Step 7 : EXPOSE 80
==> default:  ---> Using cache
==> default:  ---> 093c41c71629
==> default: Step 8 : EXPOSE 443
==> default:  ---> Using cache
==> default:  ---> 9c0115c59b14
==> default: Successfully built 9c0115c59b14
==> default: Starting Docker containers...
==> default: -- Container: mysql
==> default: -- Container: phpmyadmin
==> default: -- Container: gf1
==> default: -- Container: gf2
==> default: -- Container: gf3
==> default: -- Container: jenkins
==> default: -- Container: haproxy
```

Basic example



Click Here

Basic example



Vagrantfile

- Mapping des ports

```
#These are the port forwarding that are configured
config.vm.network "forwarded_port", guest: 7070, host: 9001 # Jenkins
config.vm.network "forwarded_port", guest: 9090, host: 9002 # Reverse proxy (entrypoint for users)
config.vm.network "forwarded_port", guest: 3306, host: 9003 # MySQL Server (pretty useless as is)
config.vm.network "forwarded_port", guest: 5050, host: 9004 # PHPMyAdmin (use localhost:9004/phpmyadmin)
config.vm.network "forwarded_port", guest: 4081, host: 9005 # GF1 config (4848)
config.vm.network "forwarded_port", guest: 4082, host: 9006 # GF1 web (8080)
config.vm.network "forwarded_port", guest: 4084, host: 9007 # GF2 web (8080)
config.vm.network "forwarded_port", guest: 4086, host: 9008 # GF3 web (8080)
config.vm.network "forwarded_port", guest: 4087, host: 9009 # HAProxy (stats on localhost:9009/haproxy login: admin, admin)
```

Vagrantfile

```
config.vm.provision "docker" do |d|

    #MySQL Server
    d.build_image "/vagrant/docker/mysql", args: "-t heig/mysql"
    d.run "mysql", image: "heig/mysql", args: "-p 3306:3306"

    #PHPMyAdmin (works with the MySQL server with the link)
    d.build_image "/vagrant/docker/phpmyadmin", args: "-t heig/phpmyadmin"
    d.run "phpmyadmin", image: "heig/phpmyadmin", args: "-p 5050:80 --link mysql:mysql"

    #Glassfish Servers
    d.build_image "/vagrant/docker/glassfish", args: "-t heig/glassfish"

    #Gf1 is sharing the autodeploy repertory, once a war file is put in this directory, it is automatically deployed by GF.
    d.run "gf1", image: "heig/glassfish", args: "-p 4081:4848 -p 4082:8080 --link mysql:mysql -v /opt/glassfish/glassfish4/glassfish/domains/domain1/autodeploy/"
    #And here, we use link the autoeploy repo of gf1, meaning, all the other GF servers will be updated once we update the first one.
    d.run "gf2", image: "heig/glassfish", args: "-p 4083:4848 -p 4084:8080 --link mysql:mysql --volumes-from gf1"
    d.run "gf3", image: "heig/glassfish", args: "-p 4085:4848 -p 4086:8080 --link mysql:mysql --volumes-from gf1"

    #Jenkins Server

    #We create out .war file from the project, then we put it directly into gf1 autodeploy directory. We don't need to use
    #asadmin or anything else, it will be deployed automatically
    d.build_image "/vagrant/docker/jenkins", args: "-t heig/jenkins"
    d.run "jenkins", image: "heig/jenkins", args: "-p 7070:8080 --link gf1:gf1 --volumes-from gf1"

    #Reverse Proxy (technically, the user only needs to connect to localhost:9002 and he will access one of the 3 GF instances)

    d.build_image "/vagrant/docker/haproxy", args: "-t heig/haproxy"
    d.run "haproxy", image: "heig/haproxy", args: "-p 9090:80 -p 4087:8080 --link gf1:gf1 --link gf2:gf2 --link gf3:gf3"


end
```

Docker - MySQL

```
FROM dockerfile/ubuntu

RUN groupadd mysql
RUN useradd mysql -g mysql -u 4242

RUN apt-get update \
 && apt-get install -y mysql-server \
 && rm -rf /var/lib/mysql/mysql \
 && rm -rf /var/lib/apt/lists/* # 20140918

ADD start /start
ADD example_config.sql /tmp/example_config.sql
RUN chmod 755 /start

EXPOSE 3306

VOLUME ["/var/lib/mysql"]
VOLUME ["/run/mysqld"]

CMD ["/start"]
```

Docker - MySQL

- We create users that can remotely connect:

```
mysql -uroot -e "CREATE USER 'bruno'@'localhost' IDENTIFIED BY '123456';"
mysql -uroot -e "CREATE USER 'bruno'@'%' IDENTIFIED BY '123456';"
mysql -uroot -e "GRANT ALL PRIVILEGES ON * . * TO 'bruno'@'localhost';"
mysql -uroot -e "GRANT ALL PRIVILEGES ON * . * TO 'root'@'%';"
mysql -uroot -e "GRANT ALL PRIVILEGES ON * . * TO 'bruno'@'%';
```

- And we add the PHPMyAdmin script in the DB

```
#Adding config file (some tables are needed for PHPMyAdmin)
mysql -uroot < "/tmp/example_config.sql"
```

Docker - PHPMyAdmin

- Dockerfile too big, and not very useful. We install:
  - PHP
  - Apache
  - PHPMyAdmin
- Then interesting part is :

```
/* User used to manipulate with storage */
$cfg['Servers'][$i]['controlhost'] = '';
$cfg['Servers'][$i]['controluser'] = 'root';
$cfg['Servers'][$i]['controlpass'] = '123456';
```

```
/*
 * First server
 */
$i++;
/* Authentication type */
$cfg['Servers'][$i]['auth_type'] = 'cookie';
/* Server parameters */
$cfg['Servers'][$i]['host'] = 'mysql';
$cfg['Servers'][$i]['connect_type'] = 'tcp';
$cfg['Servers'][$i]['port'] = '3306';
$cfg['Servers'][$i]['compress'] = false;
/* Select mysql if your server does not have mysqli */
$cfg['Servers'][$i]['extension'] = 'mysqli';
$cfg['Servers'][$i]['AllowNoPassword'] = false;
```

Docker - Glassfish

- Again, too big to put here, but what we do:
  - We install Glassfish
  - We create the jdbc resource and pool linked to the MySQL container
  - We copy the mysql connector driver
  - And we start it.
- RUN \
  - ./asadmin start-domain && \
  - ./asadmin --user admin --passwordfile pwdfile enable-secure-admin && \
  - ./asadmin --user admin --passwordfile pwdfile create-jdbc-connection-pool --restype=javax.sql.XADataSource --datasourceclassname=com.mysql.jdbc.jdbc2.optional.MysqlXADataSource --property user=$DB_TECHNICAL_USER:password=$DB_TECHNICAL_USER_PASSWORD:serverName=mysql:portNumber=3306:databaseName=$DB_NAME $JDBC_CONNECTION_POOL_NAME && \
  - ./asadmin --user admin --passwordfile pwdfile create-jdbc-resource --connectionpoolid $JDBC_CONNECTION_POOL_NAME $JDBC_JNDI_NAME && \
  - ./asadmin stop-domain

Docker - Jenkins

- What we do:
  - We install Java and Maven (to be able to create Maven projects)
  - We get Jenkins to work (install and setting environment)
  - We add the config files where we want and need them (we'll see those in a moment)
  - We also add the needed plugins (mostly for github support)
  - And that's all
- We also need to save a few config files (xml).

Docker - HAProxy

```
FROM dockerfile/ubuntu
# Install Haproxy.
RUN \
sed -i 's/^# \(.*-backports\s\)/\1/g' /etc/apt/sources.list && \
apt-get update && \
apt-get install -y haproxy && \
sed -i 's/^ENABLED=.*/ENABLED=1/' /etc/default/haproxy && \
rm -rf /var/lib/apt/lists/*
# Add files.
ADD haproxy.cfg /etc/haproxy/haproxy.cfg
ADD start.bash /haproxy-start
# Define mountable directories.
VOLUME ["/haproxy-override"]
# Define working directory.
WORKDIR /etc/haproxy
# Define default command.
CMD ["bash", "/haproxy-start"]
# Expose ports.
EXPOSE 80
EXPOSE 443
```

Docker - HAProxy

- And setting HAProxy
- Remember this:

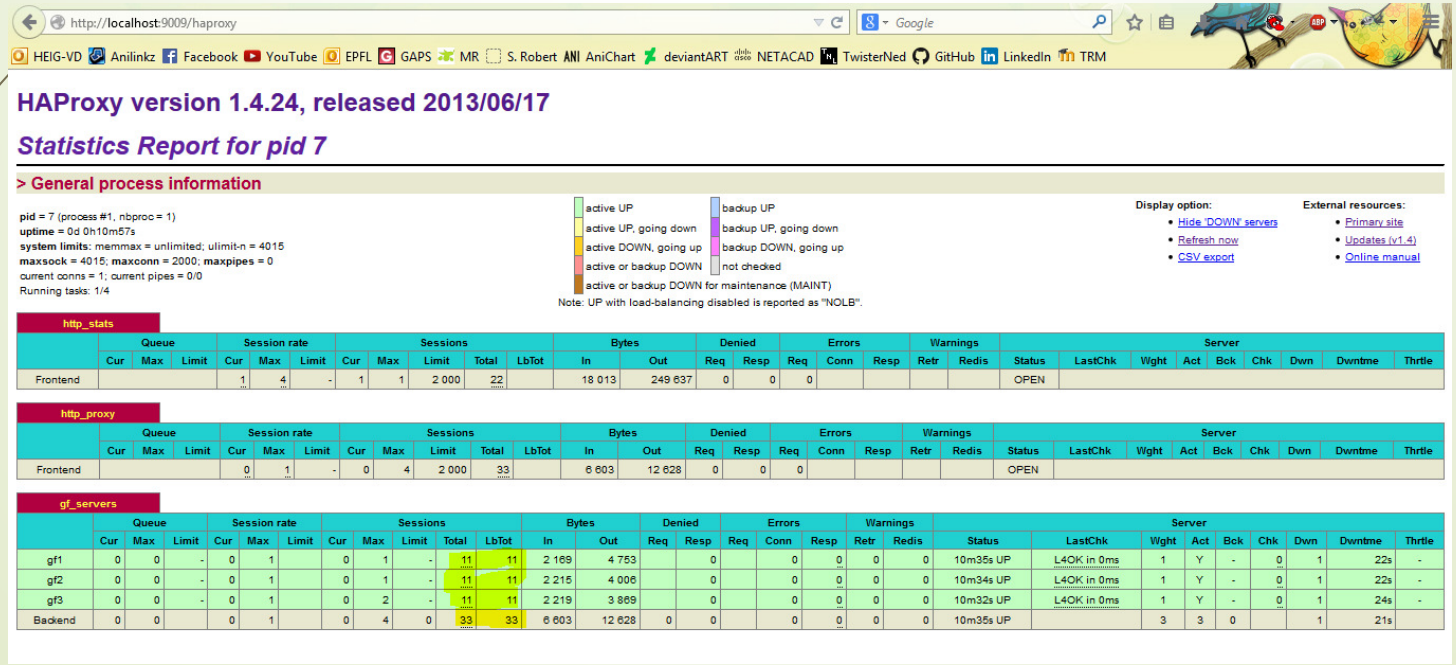
```
...
frontend http_stats
    bind :8080
    stats enable
    stats uri     /haproxy
    stats realm   Haproxy\ Statistics
    stats auth    admin:admin

frontend http_proxy
    bind :80
    default_backend  gf_servers

backend gf_servers
    server gf1 gf1:8080/AMT_REST check
    server gf2 gf2:8080/AMT_REST check
    server gf3 gf3:8080/AMT_REST check
```

```
root@9b270c07454b:/etc/haproxy # cat /etc/hosts
172.17.0.8      9b270c07454b
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.4      gf1
172.17.0.5      gf2
172.17.0.6      gf3
```

- We use the address given by the Vagrant VM.

Docker - HAProxy

- And that's all, we can now work with the API.
- Other cool thing, stats with HAProxy:



A few tricks

- A few cool things we noted.
  - Connecting to a docker:
    - sudo docker exec -i -t container_name bash
  - Download of big files (max of GitHub is 50 Mo), here jdk 8u25
    - curl -v -j -k -L -H "Cookie: oraclelicense=accept-securebackup-cookie" http://download.oracle.com/otn-pub/java/jdk/8u25-b17/jdk-8u25-linux-x64.rpm > jdk-8u25-linux-x64.rpm
    - wget --no-check-certificate --no-cookies --header "Cookie: oraclelicense=accept-securebackup-cookie" http://download.oracle.com/otn-pub/java/jdk/8u25-b17/jdk-8u25-linux-x64.rpm
  - Otherwise it won't work in a script ;)

Conclusion

- We can't show you everything, but as you can see, not so hard.
- Took a lot of time to actually configure the dockers, more than setting them
- We also tried to make this a light as possible
  - By downloading files inside the dockers and not keeping them in the host
- Vagrant and Docker were supposed to be concurrent
  - In the end, they work pretty well together!