MySQL operations in Docker

Giuseppe Maxia
QA Architect
VMware, Inc
About me

Who's this guy?

- Giuseppe Maxia, a.k.a. "The Data Charmer"
- QA Architect at VMware
- 25+ years development and DB experience
- Long timer MySQL community member.
- Oracle ACE Director
- Blog: http://datacharmer.blogspot.com
- Twitter: @datacharmer
Disclaimer
Better be clear about this

› This is community work.

› Non affiliation:
  • I don't work for Oracle. All I say here, good or bad, is my opinion.

› Not talking for my company:
  • All I say is my own stuff. My company does not influence or censor what I say here.
Preamble
How this tutorial evolved
Preamble
How this tutorial evolved

- The aim of this tutorial is to show what you can do with MySQL on Docker
The aim of this tutorial is to show what you can do with MySQL on Docker

It wanted to be also advice on best practices
Preamble

How this tutorial evolved

- The aim of this tutorial is to show what you can do with MySQL on Docker
- It wanted to be also advice on best practices
- I researched and tested over several months
The aim of this tutorial is to show what you can do with MySQL on Docker.

It wanted to be also advice on best practices.

I researched and tested over several months.

I decided to stick to the *how to*, but without advice.
The aim of this tutorial is to show what you can do with MySQL on Docker

It wanted to be also advice on best practices

I researched and tested over several months

I decided to stick to the how to, but without advice.

You will see why at the end.
Agenda

- Differences between containers and VMs
- Advantages of containers only deployments
- Immutable architecture vs build and update
- Deploying a single container
- Storage and customisation
- Networking and replication
- Security
- Orchestrators and database containers
- Cloud and containers (and VM with containers)
- MySQL group replication
- On-the-fly custom containers
- Making your own containers
Standalone app - standalone server
Sandboxes - apps on shared resources
Virtual machines
Containers
Containers
What is a container?

Software distribution blocks

- Virtualization system
- NOT a virtual machine
- Works close to the host operating system
- Can only run in the same O.S. as the host
- Less secure than VM
- Extremely fast to deploy
- Almost as efficient as a standalone server
Mutable architecture
How we work with virtual machines
Mutable architecture

How we work with virtual machines
Mutable architecture
How we work with virtual machines
Mutable architecture

How we work with virtual machines

Hypervisor

Operating system

Puppet / Chef

HW
Mutable architecture

How we work with virtual machines

- VM
- Hypervisor
- Guest O.S.
- Operating system
- Puppet / Chef
- HW
- Slow load
Mutable architecture

How we work with virtual machines

- Hypervisor
- Operating system
- VM
- Guest O.S.
- Libraries
- Puppet / Chef
- HW
Mutable architecture

How we work with virtual machines

- Hypervisor
- Operating system
- Guest O.S.
- Libraries
- Service

HW

Puppet / Chef
Mutable architecture

How we work with virtual machines

- Hypervisor
- Operating system
- VM
- Guest O.S.
- Libraries
- Service
- Updates

HW
Mutable architecture

How we work with virtual machines

- Hypervisor
- Operating system
- VM
- Guest O.S.
- libraries
- Puppet / Chef
- slow load
- updates
Mutable architecture

How we work with virtual machines

- Hypervisor
- Operating system
- VM
- Guest O.S.
- Libraries
- Puppet / Chef
- Service
- Updates

slow load
Immutable architecture
How we work with containers
Immutable architecture
How we work with containers

Puppet / Chef

HW
Immutable architecture

How we work with containers
Immutable architecture
How we work with containers

service
lib layer
Kernel layer
container
operating system

Puppet / Chef

instant load

HW
Immutable architecture
How we work with containers
Immutable architecture
How we work with containers

Puppet / Chef

service
lib layer
Kernel layer
container

operating system

instant load

HW
From containers to micro-services

Micro-services look like containers, but not quite

Micro-services:
- no O.S.
- no shell
- no extra features
- easy to compose
- communicate
Two approaches to micro-systems
Two approaches to micro-systems

how micro-systems should work
Two approaches to micro-systems

how micro-systems should work

how containers (with Docker) work
Virtual Machines vs. Containers

Deployment: speed!
Virtual Machines vs. Containers

Deployment: speed!

Start a VM:

it takes minutes
Virtual Machines vs. Containers

Deployment: speed!

Start a VM: it takes minutes

Start a container. It takes less than a second
Virtual Machines vs. Containers

Usage: speed!
Virtual Machines vs. Containers

Usage: speed!

Use a VM:
it runs on virtualised hardware
Virtual Machines vs. Containers

Usage: speed!

Use a VM:
it runs on virtualised hardware

Use a container.
It runs on bare metal
(shares host OS kernel space)
Requirements

What you need

‣ (Preferred) Linux (any flavour) + Docker 1.12
‣ or
‣ OSX + Docker 1.12 for Mac
‣ Windows + Docker 1.12 for Windows
Before you start
You need to install Docker. Yeah

- **On Linux**
  - `apt-get install docker`
- **or**
  - `yum install docker`
- **or**
  - `curl -sSL https://get.docker.com/ | sh`

- **On Mac/Windows**
  - `get Docker for Mac/Windows`
First step

You need to download the image

$ docker pull mysql/mysql-server
Using default tag: latest
latest: Pulling from mysql/mysql-server
5e7ce0e805ba: Pull complete
4b9736773855: Pull complete
4fb9c55b1f85: Pull complete
19452ca711c2: Pull complete
0c660f33db44: Pull complete
b8e96405c8c7: Pull complete
9bdbb574fa66: Pull complete
24c2343e048d: Pull complete
Digest:
sha256:da256:da24eddecb99cbeba979d7da8170991278ca59838401a7c7bf17c66307acc641c
Status: Downloaded newer image for mysql/mysql-server:latest
Listing images

Making sure you have the server

$ docker images

<table>
<thead>
<tr>
<th>REPOSITORY</th>
<th>TAG</th>
<th>IMAGE ID</th>
<th>CREATED</th>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ubuntu</td>
<td>latest</td>
<td>bd3d4369aebc</td>
<td>13 days ago</td>
<td>126.6 MB</td>
</tr>
<tr>
<td>mysql/mysql-gr</td>
<td>latest</td>
<td>ab9ab0656fea</td>
<td>2 weeks ago</td>
<td>742.3 MB</td>
</tr>
<tr>
<td><strong>mysql/mysql-server</strong></td>
<td>latest</td>
<td>1b3bfc4cbb3e</td>
<td>3 weeks ago</td>
<td>369 MB</td>
</tr>
<tr>
<td>mysql/shell</td>
<td>latest</td>
<td>212642e2d5a7</td>
<td>4 months ago</td>
<td>239.1 MB</td>
</tr>
</tbody>
</table>

Get the right image

The "official" image does not come from the MySQL team
Key points
If you were looking for just a brick, you need to review your assumptions

#1 - containers are full systems

- Every container contains an executable layer
- It's a full O.S.
#2 containers are always Linux

So far, there is no way around it

- Linux flavor in the container is independent from the host
- But they have a compatible kernel
- That's why containers are faster than VMs
#3 - the O.S. in the container is minimal

Don't expect frills

- The container has only the bare minimum to run the service
You need to decide your security strategy beforehand

3 possibilities:

- You can pass a password on the command line
- You can tell the container to generate a random password (don't!)
- You can pass a password using a file (recommended)
#5 - Containers are isolated

Don't expect the server to be accessible

- A MySQL server inside a container can only be accessed by other containers
- but
- You can expose ports to the outside
#6 - Data storage can be tricky

The data directory is inside a container

- By default, the data directory is inside the container
- Containers are volatile by default
- You can use an external "volume" for persistence
# configuration is done by deploying files

Forget puppet and Chef. Your container comes ready to use.

For Virtual machines:
- modifying with automated tools is good practice
- It takes time, but this is the way the game works

For Containers:
- NEVER modify a deployed container
- Always deploy with ready made components
- It gives you instantaneous containers!
Examples
Deploying a single container

The basics, according to the documentation

- docker run ... (example)
Getting started with MySQL on docker

A Naive approach

$ docker run -d mysql/mysql-server
3289f402378e45089e9f59ba0c2c3b93d5d34335a15c4a6bfe10be2bd53e5549

$ docker ps
CONTAINER ID        IMAGE               COMMAND
CREATED             STATUS              PORTS
NAMES

$ docker logs 3289
error: database is uninitialized and MYSQL_ROOT_PASSWORD not set
Did you forget to add -e MYSQL_ROOT_PASSWORD=... ?
Simple MySQL deployment
Here's the minimum command to deploy a MySQL container

```bash
$ docker run \
   --name mybox \
   -e MYSQL_ROOT_PASSWORD=secret \
   -d \
   mysql/mysql-server
```

06d3392543500455adcfe6cba36736739c46688de8028647d38109c0ec3cf250

# WILL RUN WITH DEFAULT VALUES
Storage and customisation
The not-so-basics, to learn step-by-step

▶ docker run ...
▶ --volume
  ▶ to define storage
▶ to change configuration instantly
▶ --env
  ▶ to pass variables
▶ [more options]
Volumes: case 1

docker create -v /data -name container1 imagename

docker run -volumes-from container1.imagename

/data did not exist in the image and it is empty
Volumes: case 2

docker create -v /data --name container1 imagename

docker run --volumes-from container1 imagename

/data existed in the image
Volumes: case 3

docker run -v /var/abc:/data -d --name mydata imagename

/var/abc in the host replaces /data in the container regardless of what was in the image before
MySQL with log-bin and server-id

To deploy a customized container we have two options:

## OPTION 1 : on the command line

```bash
docker run \
  --name mybox  \
  -e MYSQL_ROOT_PASSWORD=secret  \
  -d  \
  mysql/mysql-server --log-bin --server-id=100
```
MySQL with log-bin and server-id

To deploy a customized container we have two options:

### OPTION 2 : with a "volume" file

```bash
$ cat minimal.cnf

[mysqld]
user=mysql
log-bin=mysql-bin
server-id=100

$ docker run
  --name mybox
  -e MYSQL_ROOT_PASSWORD=secret
  -d --hostname mybox
  -v $PWD/minimal.cnf:/etc/my.cnf
mysql/mysql-server
```
Looking inside a container

It's an operating system. We can peek inside

docker exec -ti mybox bash

[root@mybox /]# mysql -p
Enter password:

[...]

Server version: 5.7.10-log MySQL Community Server (GPL)

[...]

mysql> select @@server_id;

+------------+
| @@server_id |
+------------+
|    100     |
+------------+

1 row in set (0.00 sec)
Create a DB and dedicated user

If you need a database with a user that can access it

docker run
   --name mybox \
   -e MYSQL_ROOT_PASSWORD=secret \n   -d \n   -e MYSQL_DATABASE=personnel \n   -e MYSQL_USER=pers_user \n   -e MYSQL_PASSWORD=pers_password \n   mysql/mysql-server --log-bin --server-id=100
Single container DEMO
Security

A dangerous mix of security defaults

- dangerous --env to set passwords
- useless ability of creating random passwords
- little used ability of passing passwords via file
Suggested approach by the MySQL team
This is the official advice (not good for automated operations)

$ docker run --name mybox  \
   -e MYSQL_RANDOM_ROOT_PASSWORD=yes  \
   -d mysql/mysql-server
31c8880bb7c25e44f2a413e0150c7c1ab732afe4a3b76531f9e4ea
e763b34a51

$ docker logs mybox
Initializing database
Database initialized
MySQL init process in progress...
[...]
GENERATED ROOT PASSWORD: ;3LEq^AGMUjDiN&EpYmYK*ij0KU
An alternate approach
Here you won't see a password, but you can use it

$ echo $RANDOM | sh256sum | cut -c 1-16 > pwd.txt

$ docker run --name mybox --hostname mybox \
  -v $PWD/pwd.txt:/root/pwd.txt \
  -e MYSQL_ROOT_PASSWORD=/root/pwd.txt \
  -d mysql/mysql-server

$ docker exec -ti mybox bash
[root@mybox /]# mysql -p$(cat /root/pwd.txt)

# You can also save the password in an alternate file
An even better approach

Here you won't see a password, but you can use it

$ echo $RANDOM | sh256sum | cut -c 1-16 > pwd.txt
$ cat my_safe.cnf
[client]
user=root
password=CHANGEME

$ sed -i -e "s/CHANGEME/$(cat pwd.txt)/" my_safe.cnf

$ docker run --name mybox --hostname mybox \
   -v $PWD/pwd.txt:/root/pwd.txt \
   -v $PWD/my_safe.cnf:/etc/myuser.cnf \
   -e MYSQL_ROOT_PASSWORD=/root/pwd.txt \
   -d mysql/mysql-server

$ docker exec -ti mybox mysql \
   --defaults-extra-file=/etc/myuser.cnf
Safe password DEMO
Networking and replication
Complex systems at your fingertips

- docker network
- docker run
  - --net
- automatic hostnames
Inspecting a container

Docker gives us some info on the container

docker inspect mybox

[...] large JSON object [...] 

$ docker inspect \ 
   --format '{{ .NetworkSettings.IPAddress}}' mybox
172.17.0.2
How to communicate with containers

Container network

- **host**
- **network**

- **container1**
  - 172.17.0.2

- **container2**
  - 172.17.0.3

- **port1**
Container network

How to communicate with containers

- Host
- Container network
  - Container 1: 172.17.0.2
  - Container 2: 172.17.0.3
Container network

How to communicate with containers

- Host: 127.0.0.1:port2
- Container1: 172.17.0.2
- Container2: 172.17.0.3
Container network

How to communicate with containers
Container network

How to communicate with containers

- Host
- Container 1: 172.17.0.2
- Container 2: 172.17.0.3

Dedicated network
Container network
How to communicate with containers
Replication

Setting up asynchronous replication requires some effort

1. Create a dedicated network
2. Deploy the master container and slaves using the new network
3. Wait until the servers are up
4. create replication user in the master
5. run CHANGE MASTER TO in the slaves
Examples in GitHub

All the examples are available for easy usage

- [https://github.com/datacharmer/mysql-replication-samples](https://github.com/datacharmer/mysql-replication-samples)
- directory ./docker-replication
Deploying a master

Much configuration goes on

```bash
# cat my_template.cnf
[mysqld]
user     = mysql
port     = 3306
log-bin  = mysql-bin
relay-log = mysql-relay
server-id = __SERVERID__
log-error=/var/log/mysqld.log
pid-file=/var/run/mysqld/mysqld.pid
master-info-repository=table
relay-log-info-repository=table
enforce-gtid-consistency
gtid_mode=ON
```
Deploying a master

Much configuration goes on

```
sed "s/_SERVERID_/100/" < my-template.cnf \
    > $DOCKER_TMP/my_1.cnf

docker run \
    --net my_rep_net \
    --name mysql-node1 --hostname master \
    -v /opt/docker/tmp/my_1.cnf:/etc/my.cnf \
    -v /opt/docker/tmp/home_my_1.cnf:/root/home_my.cnf \
    -v /opt/docker/mysql/node_1:/var/lib/mysql \
    -e MYSQL_ROOT_PASSWORD=secret \
    -d mysql/mysql-server
```
Deploying a slave

Much configuration goes on

```bash
sed "s/_SERVERID_/200/" < my-template.cnf \\
> $DOCKER_TMP/my_2.cnf

docker run \\
  --net my_rep_net \\
  --name mysql-node2 --hostname node2 \\
  -v /opt/docker/tmp/my_2.cnf:/etc/my.cnf \\
  -v /opt/docker/tmp/home_my_2.cnf:/root/home_my.cnf \\
  -v /opt/docker/mysql/node_2:/var/lib/mysql \\
  -e MYSQL_ROOT_PASSWORD=secret \\
  -d mysql/mysql-server
```
Checking if the server is ready

This requires some creativity

```bash
function is_ready
{
    NODE=$1
    MYSQL="docker exec -it mysql-node$NODE mysql \ --defaults-file=/root/home_my.cnf "
    READY=$(eval "echo \$MYSQL -BN -e 'select 12345' | tr -d '\n' | tr -d '\r'"
    if [ "\$READY" == "12345" ];
        then
            echo OK
        fi
}

# loop through nodes
node_ready=$(is ready $NODE)
```

# 58
Setting the options file in root

After we're sure that the server is fully up

docker exec -it mysql-node$NODE \ cp /root/home_my.cnf /root/.my.cnf
Creating the replication user

Now we can use mysql client without password

```bash
MASTER="docker exec -it mysql-node1 mysql"

$MASTER -e 'create user rdocker \identified by "rdocker"'

$MASTER -e 'grant replication slave on *.* to rdocker'
```
Executing CHANGE MASTER TO in slaves

Also here we can use mysql client without password

```
SLAVE="docker exec -it mysql-node$SLAVE_NODE mysql 

$SLAVE -e "change master to \
    master_host='mysql-node1', \
    master_port=$MASTER_PORT, MASTER_AUTO_POSITION=1"

$SLAVE -e 'START SLAVE user="rdocker" password="rdocker" '
```
Replication DEMO
Orchestrators

The way of containers is not (yet) the way of MySQL

- Docker orchestrators are easy to use
- But not for asynchronous database systems
Cloud and containers

Also a wasteful mix of virtualising systems

- Docker is best used on bare metal
- Using containers with VMs is a waste of performance
- Docker on Mac or Windows does not work always as expected
Virtual machines + containers
MySQL Group Replication

Synchronous database systems can be easy to set up

- Docker MySQL and Group Replication
- Works well when the system is fast
- It can have nasty side failures (when not used on Linux).
Using group replication

Get the image

$ docker pull mysql/mysql-gr

Using default tag: latest

latest: Pulling from mysql/mysql-gr
deed2f14de27: Pull complete
87e819187015: Pull complete
383377483443: Pull complete
ed703ed668c5: Pull complete
3a0716e5cd17: Pull complete
e6af449de9a: Pull complete
45b8d9ccdc34: Pull complete
[...]

Digest:
sha256:0d690f3bd27075250be546ef91278c1b855fda2a0ab1bb2
231153eaddb9e70485

Status: Downloaded newer image for mysql/mysql-gr:latest
Using group replication

Starting nodes

docker network create group1
SEEDS=( 'node2:6606,node3:6606'
        'node1:6606,node3:6606'
        'node1:6606,node2:6606' )
for N in 0 1 2
do
    NODE=$((N+1))
docker run -d  --name=node$NODE  --net=group1  \
    --hostname=node$NODE  \
    -e MYSQL_ROOT_PASSWORD=aTestPwd  \
    -e MYSQL_REPLICATION_USERNAME=rpl_user  \
    -e MYSQL_REPLICATION_PASSWORD=rpl_pass  \
    mysql/mysql-gr  \
    --group_replication_group_seeds=${SEEDS[N]}  \
    --server-id=$NODE
done
Using group replication

Checking nodes

```sql
select MEMBER_HOST, MEMBER_STATE from performance_schema.replication_group_members
```

1. row
MEMBER_HOST: node1
MEMBER_STATE: ONLINE

2. row
MEMBER_HOST: node2
MEMBER_STATE: ONLINE

3. row
MEMBER_HOST: node3
MEMBER_STATE: ONLINE
Using group replication

Checking replication - create a table in each node

-- node1
create table test.t1(id int not null primary key)

-- node2
create table test.t2(id int not null primary key)

-- node3
create table test.t3(id int not null primary key)
Using group replication

Checking replication - checking tables in each node

-- node1 (repeat for node2 and node3)
show tables from test
+----------------+
| Tables_in_test |
|----------------+
| t1            |
| t2            |
| t3            |
+----------------+
Group Replication DEMO
On-the-fly custom containers

Your choices are limited

- The "official" MySQL image => Debian
- MySQL image from Oracle => OracleLinux
- Percona server image => CentOS
- None offers 5.0 or 5.1

- What if you want MySQL 5.7 or 5.1 on CentOS, Debian, or Ubuntu?
mysql-docker-minimal

A community project, offering building blocks for your containers

- https://github.com/datacharmer/mysql-docker-minimal
- There are reduced images for MySQL from 5.0 to 8.0
- And customized images for Ubuntu, CentOS, Debian.
Reviewing volumes: case 2

docker create -v /data -name container1 imagename

docker run --volumes-from container1 imagename

/data existed in the image
Containers as application carriers

docker create -v /app -name container1 my_app
docker run -volumes-from container1 deploy_image
mysql-docker-minimal

Build in two steps

# Creates a volume, containing MySQL binaries

```
$ docker create \
   --name my_bin \
   -v /opt/mysql \
   datacharmer/mysql-minimal-5.7
```

# Uses that volume with your choice O.S.

```
$ docker run -ti \
   --volumes-from my_bin \
   --name mybox \
   datacharmer/my-ubuntu bash
```
Custom containers DEMO
MySQL Shell with Docker

Instead of installing MySQL shell …

- Use a Docker image
- Shell ready to use
- No side effects on the O.S.
Testing the MySQL document store

Easy and painless

docker run --name mybox --net docnet \
  -e MYSQL_ROOT_PASSWORD=secret -d \
  -v $HOME/data:/data \
  mysql/mysql-server \
  --plugin-load=mysqlx:mysqlx.so

docker exec -ti mybox mysql -psecret \
  -e 'source /data/world_x-db/world_x.sql'

docker run --name myshell --rm \
  --net docnet -ti mysql/shell \
  -h mybox -u root -psecret world_x
Make your own Docker images
Not a difficult task

- Choose where to start from (bare image, or already made MySQL image)
- Add your own features
- repack
Creating containers DEMO
How I tested MySQL 8

On a Mac, having only Linux binaries

- I was traveling
- Without a Linux box
- With only my Mac and a Linux tarball
Cross-OS testing - part 1

setting the environment

# I extracted the tarball in a directory on my Mac

mkdir ~/opt/linux
cd ~/opt/linux
tar -xzf mysql-8.0.0-dmr-linux-glibc2.12-x86_64.tar.gz
mv mysql-8.0.0-dmr-linux-glibc2.12-x86_64 8.0.0
Cross-OS testing - part 2

setting the environment

# I copied the latest MySQL Sandbox next to the tarball

cd ~/opt linux
tar -xzf ~/Downloads/MySQL-Sandbox-3-2-00.tar.gz
Cross-OS testing - part 3

Something went wrong

# I tried running a simple container, but I found a problem

docker run -ti \ 
  -v $HOME/opt/linux/::/work \ 
  -v $HOME/data/::/data \ 
  --name mybox /datacharmer/my-centos bash

cd /work/MySQL-Sandbox-3.2.00

  . env.sh

make_sandbox /work/8.0.0 -- --no_show

... Missing library libnuma.so.1
Volumes to the rescue

# Percona server image has the needed library

docker run \
  -v $HOME/opt/linux/usr_transfer:/usr_transfer \
  --name mybox -e MYSQL_ROOT_PASSWORD=secret \
  -d percona/percona-server
docker exec -ti mybox \
  bash -c 'cp /usr/lib64/libnuma.so.1.1 /usr_transfer'

ls -l ~/opt/linux/usr_transfer
docker stop mybox
docker rm -v mybox
Cross-OS testing - Part 5

Now we can use the volume

docker run -ti \
  -v $HOME/opt/linux/::/work \
  -v $HOME/data/::/data \
  -v $HOME/opt/linux/usr_transfer/libnuma.so.1:/usr/lib64/libnuma.so.1 \
  --name mybox /datacharmer/my-centos bash
Cross - OS testing demo
Summing up
A few controversial thoughts to take home
Summing up
A few controversial thoughts to take home

- Containers are a promising technology
Summing up
A few controversial thoughts to take home

- Containers are a promising technology
- Current trend is aiming at micro-services
Summing up
A few controversial thoughts to take home

- Containers are a promising technology
- Current trend is aiming at micro-services
- Most current offerings are a mix with cloud
Summing up
A few controversial thoughts to take home

- Containers are a promising technology
- Current trend is aiming at micro-services
- Most current offerings are a mix with cloud
- MySQL (and databases) are not well integrated with containers
Summing up

A few controversial thoughts to take home

‣ Containers are a promising technology
‣ Current trend is aiming at micro-services
‣ Most current offerings are a mix with cloud
‣ MySQL (and databases) are not well integrated with containers
‣ MySQL on Docker:
Summing up
A few controversial thoughts to take home

› Containers are a promising technology
› Current trend is aiming at micro-services
› Most current offerings are a mix with cloud
› MySQL (and databases) are not well integrated with containers
› MySQL on Docker:
  • is wonderful for development and testing
Summing up
A few controversial thoughts to take home

- Containers are a promising technology
- Current trend is aiming at micro-services
- Most current offerings are a mix with cloud
- MySQL (and databases) are not well integrated with containers
- MySQL on Docker:
  - is wonderful for development and testing
  - You need extra care to use it in production