Stefan Schimanski

MariaDB Galera on a Mesos Cluster with Docker

Mar 4, 2015

Building on top of Erkan Yanar's work of packaging and running Galera in Docker containers (Github project) on Giant Swarm, I concluded that also Galera and Mesos in connection with Mesosphere's Marathon should be great fit.

Erkan convinced me that Galera's cluster join+quorum logic makes it really suitable for a dynamic environment like a Mesos cluster. The main properties compared to other much more static distributed system - hello Redis, hello Mongo – are the following:

- Galera cluster logic does not depend on the history, but only on the last state and the new state.
- Galera nodes are all equal
- Galera nodes cleanly leave a cluster on SIGTERM and are then just forgotten by the other nodes (read: do not count to the quorum anymore).

Especially the first property means that quorum calculations are done ad-hoc when necessary. The required size of a surviving partition during a network split changes when nodes leave cleanly or join the cluster. *This property is essential to scale up and down* in a Marathon like environment.

Seeding a cluster and join nodes

When a first node of a cluster comes up (or after no partition with a quorum survived), this node must be "manually" declared as the seed node of new cluster (`--wsrep-new-cluster`).

I am going to call all non-seed nodes only "nodes".

When using the containers (compare below) manually or via the Mesos Marathon `galera.json` app definition, the seed must be started first. When it is up and healthy, the *nodes* are started and join the *seed node*. **Then the seed node must be stopped.** The other nodes keep running. The reason for this is that the *seed node* will start a new cluster again when restarted (e.g. by Marathon), even though the other *nodes* still form a valid cluster. This is probably not what one wants.

Running the containers

For the *seed node* and the normal *nodes* the same Docker container is used, namely sttts/galera-mariadb-10.0-xtrabackup. As the name suggests it uses MariaDB 10.0 with the Galera extension and xtrabackup for the initial SST (in contrast to rsync). The images are based on Ubuntu 14.04 and MariaDB public packages. The same should work with Percona or MySQL packages:

```
$ docker run -d -v /data:/var/lib/mysql -p 3306 -p 8080 \
    -e XTRABACKUP_PASSWORD=abc -e MYSQL_ROOT_PASSWORD=secret \
    sttts/galera-mariadb-10.0-xtrabackup seed

$ docker run -d -v /data:/var/lib/mysql -p 3306 -p 8080 \
    sttts/galera-mariadb-10.0-xtrabackup \
    -e XTRABACKUP_PASSWORD=abc \
    node 172.17.0.81,172.17.0.97

$ docker run -d -v /data:/var/lib/mysql -p 3306 -p 8080 \
    sttts/galera-mariadb-10.0-xtrabackup \
    -e XTRABACKUP_PASSWORD=abc \
    node 172.17.0.81 --any-mysql-argument-you-like
```

Service discovery

Galera nodes have to find each other. In my setup I have deployed Consul as a DNS server for the `galera.services.dc1.consul` domain, using progrium's registrator to register Docker containers.

Moreover, registrator is started with the `-internal` parameter such that the internal container IP is published in Consul. This at least needs Consul 0.5. The network setup resembles what is described in my former post "Adventures with Weave and Docker".

The consequence of all this is that it is enough to pass `node galera.services.dc1.consul` to the container to find the existing cluster and join it. The entrypoint script of the container will resolve the domain `galera.services.dc1.consul` and pass the resulting IPs as `qcomm://` parameter to `mysqld`.

Health Checks

The containers not only launch the `mysqld` daemon on port `3306`, but also a small health check web server on port `8080`. This web server provides a web service with the following understanding of the HTTP status codes:

- 100 - the initial sync is not finished yet
- 200 - the node is in sync and writable
- 503 - the node is not in sync and not doing the initial sync.

It is easy to use this as a health check inside Marathon, compare `galera.json` below.

The health check itself is written in Go, forked from CloudFoundry and extended with the HTTP code 100 support.

The latter is needed in order to make Marathon ignore the non-healthy state of a node which needs time for its initial sync, potentially longer than the grace period that is defined in the Marathon app health check definition. For this reason, the `ignoreHttp1xx` option is activated in the Marathon app (available from Marathon >0.8.1 on). In theory (not tested due to the lack of a big databases which actually takes long to sync with xtrabackup) Marathon should leave the task running, even though the SST might take long, minutes or hours.

Mesos Marathon

To make Galera work as a Marathon application the two roles *node* and *seed* must be combined into an application group.

When starting a cluster for the first time, the seed has to be started first while the `instances` value for the normal *nodes* is zero.

If you use `curl` to start Marathon apps, this will look similar to this:

```
$ curl -i -X PUT -d @galera.json -H "Content-Type: Application/json" master1:5080/v2/gro
HTTP/1.1 100 Continue

HTTP/1.1 200 OK
Expires: 0
Content-Type: application/json
Server: Jetty(8.y.z-SNAPSHOT)
Pragma: no-cache
Cache-Control: no-cache, no-store, must-revalidate
Content-Length: 92

{"version":"2015-03-04T16:48:02.301Z","deploymentId":"df4d885e-dde8-4021-9ba6-78b0273745
```

When the *seed* task comes up and is green the other nodes can be scaled up, e.g. via Marathon's web interface.

Marathon with 3 nodes

After the first normal *node* is healthy, the *seed* can **and should** be suspended.

Congratulation, your cluster is up!

You can scale up and down as you like. For the usual reasons it's good to have an odd number of nodes up to survive network splits without loosing the quorum on both sides.

Here is the `galera.json`, tested with Marathon 0.8.1rc2:

```
{
    "id": "/galera",
    "groups": [{
        "id": "/galera/test",
        "apps": [{
            "id": "seed",
            "container": {
                "type": "DOCKER",
                "docker": {
                    "image": "sttts/galera-mariadb-10.0-xtrabackup",
                    "network": "BRIDGE",
                    "portMappings": [
                        { "containerPort": 3306, "hostPort": "0", "protocol": "tcp" },
                        { "containerPort": 8080, "hostPort": "0", "protocol": "tcp" }
                    ]
                },
                "volumes": []
            },
            "env": {
                "SERVICE_NAME": "galera",
                "SERVICE_TAGS": "seed",
                "XTRABACKUP_PASSWORD": "3240fd7as9f8798",
                "MYSQL_ROOT_PASSWORD": "secret"
            },
            "args": [
                "seed"
            ],
            "cpus": 0.5,
            "mem": 512,
            "instances": 1,
            "maxLaunchDelaySeconds": 10,
            "healthChecks": [{
                "protocol": "HTTP",
                "portIndex": 1,
                "gracePeriodSeconds": 30,
                "intervalSeconds": 5,
                "maxConsecutiveFailures": 6,
                "ignoreHttp1xx": true
            }],
            "upgradeStrategy": {
                "minimumHealthCapacity": 0
            }
        }, {
            "id": "node",
            "container": {
                "type": "DOCKER",
                "docker": {
                    "image": "sttts/galera-mariadb-10.0-xtrabackup",
                    "network": "BRIDGE",
                    "portMappings": [
                        { "containerPort": 3306, "hostPort": "0", "protocol": "tcp" },
                        { "containerPort": 8080, "hostPort": "0", "protocol": "tcp" }
                    ]
                },
                "volumes": []
            },
            "env": {
                "SERVICE_NAME": "galera",
                "XTRABACKUP_PASSWORD": "3240fd7as9f8798",
                "MYSQL_ROOT_PASSWORD": "secret"
            },
            "args": [
                "node",
```

```
                "galera.service.dc1.consul"
            ],
            "cpus": 1,
            "mem": 512,
            "instances": 0,
            "maxLaunchDelaySeconds": 10,
            "healthChecks": [{
                "protocol": "HTTP",
                "portIndex": 1,
                "gracePeriodSeconds": 30,
                "intervalSeconds": 5,
                "maxConsecutiveFailures": 6,
                "ignoreHttp1xx": true
            }],
            "upgradeStrategy": {
                "minimumHealthCapacity": 0.8,
                "maximumOverCapacity": 0.2
            },
            "dependencies": ["seed"]
        }]
    }]
}
```

Storage

Intentionally I did not care about the storage yet in the Marathon app definition above. Not even a volume is used to map a directory from the host. For a production setup of course one has to solve this.

The easiest solution for the moment (without flocker or Quobyte's offering) is to use volumes on the host where the set of hosts is known. This allows to use Marathon constraints like this to run at most one instance on each of these hosts:

```
"constraints": [
    ["hostname", "UNIQUE"],
    ["hostname", "LIKE", "host1|host2|host3"]
]
```

Then fixed volumes from the host can be used without conflict of multiple Galera tasks:

```
"volumes": [{
    "containerPath": "/var/lib/mysql",
    "hostPath": "/data/galera",
    "mode": "RW"
},
```

Of course, in the long run this is not really what we want. Both, flocker and Quobyte's cluster file system look promising and will be subject of a follow-up post probably. Quobyte explicitly claims that it is suitable for databases, something I would not say about glusterfs or cephfs if you are interested in performance.

In this context of performance note the value `0` of `innodb_flush_log_at_trx_commit` in galera.cnf. Because Galera guarantees synchronous query replication, the file system sync is not important anymore because on a hardware crash the other nodes will still have the data. Hence, the storage speed is less important than in a classical non-Galera setup.

Another interesting solution to watch getting developed in the storage arena is Mesos' native persistence data support. Twitter itself is actively working on getting MySQL ready with support for this, a talk about Mysos is submitted for the MesosCon 2015 in Seattle this summer. I am looking forward to see how Galera can make use of this as well.

Just a moment ago Docker announced the take-over of SocketPlane. As storage is as important as network in a dynamic cluster world, let's see what happens in this area with all those projects coming up. Exciting times!

Recovery

MariaDB Galera can be bootstrapped on Marathon as lined out above. Scaling works fine, fault tollerance is good – as long as one part of the cluster has quorum and survives.

In contrast, it is not clear how to handle a real split brain without surviving cluster. Marathon will detect that all nodes become unhealthy and will eventually restart them. But this won't help.

In a normal more static setup, the DBA would find out which node had the latest state and would start this creating a new cluster, then let all other nodes join again. This is a manual step of course and does not fit well into the Mesos world.

Another immediate idea in the outlined setup would be to start the *seed* node again. But this does not have the latest data because it was not running anymore.

The only way to get the cluster up and running again would be to login into the node with the latest data (hopefully) and run this command:

```
SET GLOBAL wsrep_provider_options='pc.bootstrap=true';
```

As Marathon restarts nodes when they are unhealthy for some time, you must be fast enough to do this and to let Marathon notice that this node is healthy again. When you managed to do this, Marathon will eventually restart all other node and the cluster will self-heal from this point on.

To find out which node has the most advanced data, you can look at the logs. You will find something like:

```
Docker startscript:  Get the GTID positon
150304 18:55:20 [Note] WSREP: wsrep_start_position var submitted: '4b83bbe6-28bb-11e4-a8
```

The last commited transaction on this node was 2. As described the node with the highest number must be used for bootstrapping and therefore recovering the cluster. Compare Percona's recovery howto for more information.

Outlook

Recovery is not really satisfactory. It is not clear how to recover a cluster which is down completely in a "Mesos-like" way. One has to access the tasks directly and moreover the time to do this is limited because Marathon restarts the tasks. Some more clever ideas are necessary here.

The storage topic is quite in the move right now in the Docker and in the Mesos world. For now the static host volume kind of works, but of course much more dynamic solutions would be great.

Because not many MySQL client libraries support Galera clusters directly, it would make sense to include a haproxy (or some other MySQL supporting proxy) in front of the Galera cluster. Using the health check on port `8080`, which is used by Marathon to decide whether a node is healthy, can also easily be used by this proxy to only redirect queries to PRIMARY and non-read-only nodes. This will probably be the next step – added later to this post – to get this setup into a production environment of an app based on Galera.

Finally it should be highlighted that Galera's cluster logic really feels like a great fit with the dynamic nature of a Mesos cluster. This is in contrast to a lot of other services which come from the static pre-Docker world. So I completely agree with Erkan that Galera does a lot of things right.

Stefan Schimanski
Stefan Schimanski
stefan.schimanski@gmail.com
sttts
the1stein