Organize the migration of a hundred databases to the cloud
Percona Live ONLINE
May 12th 2021
Meet Maxime Fouilleul
Engineering Manager for DBRE

“Make the database not a problem.”
Package and support the database catalog for BlaBlaCar application services.

Provide expertise in software engineering to help teams choose the right database for them and to ensure they use it the right way.
## The go-to marketplace for shared road travel

BlaBlaCar is a community-based marketplace allowing members to book seats in individual cars and buses alike. From carpool to buses, we have one common moto: #ZeroEmptySeats.

<table>
<thead>
<tr>
<th>Geographies</th>
<th>Position</th>
<th>Countries</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Carpool</strong></td>
<td>Leader in all our markets</td>
<td>Global (22 countries)</td>
</tr>
<tr>
<td><strong>Bus Marketplace</strong></td>
<td>Leader in Eastern Europe</td>
<td>Russia, Ukraine, Poland, Brazil</td>
</tr>
<tr>
<td></td>
<td>Early stage in Brazil</td>
<td></td>
</tr>
<tr>
<td><strong>BBC Branded Buses</strong></td>
<td>Leader in France</td>
<td>France, Germany</td>
</tr>
</tbody>
</table>
BlaBlaCar

“The go-to marketplace for shared travel”

90 million members

25 million travelers per quarter

22 countries
90 million members
bare-metal servers

1 type of hardware

3 disk profiles

fleet

CoreOS
ggn

“Distributed init system”
dgr

Container Registry

etcd

host

create

build

store

monitoring
terve

mysql-main1

mysql

PODs

rkt

Service Discovery

nerve

zookeeper

carpooling

100% Containers Powered Carpooling
Our production database infra in 2019

- MySQL: 30 Production Clusters
- Cassandra: 7 Production Clusters
- RabbitMQ: 13 Production Clusters
- Redis: 19 Production Clusters
- Elasticsearch: 6 Production Clusters
- PostgreSQL: 5 Production Clusters
- Kafka: 8 Production Clusters
- Couchbase: 5 Production Clusters
The mission

2019 sign cloud provider end of 2020 close on-premise

Consolidate the DBRE team staffing plan includes 4 SRE database enthusiastic

Migrate 100+ databases Package reliable systems, accompany the migration and decommissioning
“Fly me to the cloud”
The DBRE vision

Google Compute Engine

Google Kubernetes Engine

Google Managed Services

GCP Marketplace
The DBRE vision

Google Compute Engine
Try to avoid 👌

Google Kubernetes Engine

Google Managed Services

GCP Marketplace
The DBRE vision

- Google Compute Engine
  - Try to avoid 👍
- Google Kubernetes Engine
  - Prefer 👍
- Google Managed Services
- GCP Marketplace
The DBRE vision

Google Compute Engine
Try to avoid 👍

Google Kubernetes Engine
Prefer 👍

Google Managed Services
Do 👍

GCP Marketplace
The DBRE vision

- **Google Compute Engine**: Try to avoid 👌
- **Google Kubernetes Engine**: Prefer 👍
- **Google Managed Services**: Do 🤞
- **GCP Marketplace**: Don’t 👎
“Be transparent to ensure buy-in”
Be clear on iterations

alpha
Ready to test

beta
Ready to industrialize

GA
Ready to prod
Documentation as log

- 2020-07-20 - MariaDB - SLI/SLO - Specs and first implementation
- 2020-01-29 - MariaDB - What version to use in GCP?
- 2020-01-29 - MariaDB - Rely on disk snapshots for backups
- 2020-01-29 - MariaDB - using MaxScale as a layer 7 proxy
- 2020-01-29 - CloudSQL - New HA design supported
- 2019-12-06 - CloudSQL user management
- 2019-12-05 - CloudSQL limitations - Can't purge binary logs, let it grow, let it grow!
- 2019-12-03 - MariaDB - New features for the chart would be documented in release notes
- 2019-11-13 - MariaDB - Reboot the packaging of MariaDB in GKE
- 2019-10-25 - MariaDB - Using deported Prometheus exporter to monitor CloudSQL...
- 2019-10-03 - CloudSQL limitations - Does the lack of triggers can impact us
- 2019-10-02 - CloudSQL limitations - Replicate from MariaDB to CloudSQL is not possible.
- 2019-10-02 - CloudSQL network optimizations (bye-bye CloudSQL Proxy)
- 2019-10-01 - CloudSQL provisioning, why a terraform module?
- 2019-09-06 - CloudSQL everywhere?
- 2019-07-31 - MariaDB - Adding safe_to_bootstrap override capability to avoid getting stuck after full crash
- 2019-04-04 - MariaDB - Graceful restart - Kubernetes Probes + PDB
- 2019-01-19 - Make or Buy Study, why not CloudSQL?
- 2018-10-03 - MariaDB - Docker image + Helm chart (Alpha = stable release for testing)
- 2018-09-28 - MariaDB - Performance Benchmarks, BBC Baremetal VS GCP
Gamify the knowledge sharing process

Level 1
Actions are basic tasks that should be mastered by each team member.

<table>
<thead>
<tr>
<th>Level</th>
<th>Action codename</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1</td>
<td>BasicHealthcheck</td>
</tr>
<tr>
<td>Level 1</td>
<td>RolloutMinorChanges</td>
</tr>
<tr>
<td>Level 1</td>
<td>Connect&amp;Read</td>
</tr>
<tr>
<td>Level 1</td>
<td>ManageAccess</td>
</tr>
<tr>
<td>Level 1</td>
<td>PrepareClusterBootstrap</td>
</tr>
<tr>
<td>Owner</td>
<td>ActiveOwnership</td>
</tr>
<tr>
<td>Owner</td>
<td>ModifyDataset</td>
</tr>
<tr>
<td>Owner</td>
<td>RecoverDataloss</td>
</tr>
<tr>
<td>Owner</td>
<td>Advisory</td>
</tr>
</tbody>
</table>

Level "Owner"
Actions allow the component to be actively supported, they should be mastered by at least 2 members.
Implementations
and migration paths
Kickoff of the $\alpha$lpha release
First production cluster in GCP
Last production cluster decommissioned from on-premise
Key elements for the implementations

- Leverage Kubernetes Statefulset
- master **affinity** in the PodSpec
- prefer **Google Persistent disks** over Local SSD
- leverage Persistent disk **Snapshots**
- promote **distributed ownership**
- use **Terraform** for Google managed services
MySQL
A production MySQL service in 2019

- READS
- WRITES
- ASYNC

APP

HAProxy

MariaDB Galera Cluster

Async

Async
Gather requirements

- Will your database be migrated or abandoned?
- What is the tolerated downtime for the migration?
- Can we migrate the reads separately from the writes?
1. The application can tolerate several minutes of downtime.
2. Writes can be stopped during the dataset migration.
3. The need is fairly lightweight.

Google CloudSQL

MariaDB in Kubernetes
What DBRE is packaging for CloudSQL?

**Terraform Module**
To ease and standardize the usages

**Kubernetes Operator**
To setup a Prometheus exporter
What DBRE is packaging for CloudSQL?

$ git commit -m "[prod-1][dbre] Create cloudsql-test"
$ git push origin prod-dbre-cloudsql-test
CloudSQL Migration Path

Initial stage

- APP
- READS
- WRITES
- HAProxy
- on-premise
CloudSQL Migration Path

Move the application

---

```yaml
apiVersion: v1
kind: Service
metadata:
  name: cloudsql-demo
  namespace: demo
spec:
  type: ClusterIP
  clusterIP: None
  ports:
    - protocol: TCP
      port: 3306
      targetPort: 3306
      name: mysql
---

kind: Endpoints
apiVersion: v1
metadata:
  name: cloudsql-demo
  namespace: demo
subsets:
  - addresses:
      - ip: <database-ip-address>
    ports:
      - port: 3306
```
CloudSQL Migration Path
Stop the application and switch endpoint
CloudSQL Migration Path

Restart the application
MariaDB in-house packaging via a Helm Chart

- A **StatefulSet** with Galera enabled...or not
- A **Deployment** running **ProxySQL**
- **Prometheus** exporter sidecars to export metrics
- A bunch of **Jobs** that manipulate disk snapshots
- A **Deployment** running an **SLI Prober**
- Services, RBAC, and PDB...
A production MySQL service in 2021
With asynchronous replicas
MariaDB packaging tips
Dynamically find Galera seeds

wsrep_cluster_address

```bash
init_galera_config.sh: |-
  #!/bin/bash
  set -ex

  # 1. Get seeds
  TOKEN=$(cat /var/run/secrets/kubernetes.io/serviceaccount/token)

  EP_JSON=$(curl -sSk
  -X GET $
  -H "Authorization: Bearer ${{TOKEN}}"$
  -H "Content-Type: application/json"
  -H "Accept: application/json"
  -X GET ${{API_ENDPOINT}}/endpoints/{{ .Release.Name }}/headless

  if [ "$EP_JSON" | jq -r .kind ] == "Endpoints" ]; then
    if [ "$EP_JSON" | jq -r .subsets ] != "null" ]; then
      # Endpoint = joining cluster
      SEEDS="$EP_JSON | jq -r ".subsets[0].addresses[].ip" | paste -sd, -"
    else
      # No endpoint = create a cluster
      SEEDS=""
    fi
  else
    exit 1
  fi
```

We use a Kubernetes **Headless Service** to get available (ready) endpoints. If we find endpoints we join a cluster. If we don’t find any endpoint we bootstrap a cluster.
Having accurate **Liveness** and **Readiness**

Simple ping to report the MySQL is live or not

```bash
liveness_probe.sh: |-n
#!/bin/bash

# If mysql ping or SST in progress
# Use TCP instead of Unix socket to be usable from side cars.
mysqladmin -h 127.0.0.1 -u monitoring ping 2>/dev/null || [ -d { .Values.config.mysqld.datadir }/sst ]

readiness Probe.sh: |-n
#!/bin/bash

# wsrep_local_state vs wsrep_local_state_comment
# 1 = Joining
# 2 = Donor/Desynced # Ready as we use non-blocking SST (xtrabackup/mariabackup).
# 3 = Joined
# 4 = Synced

wsrep_local_state=$(mysql -u monitoring -BN -e "SHOW GLOBAL STATUS LIKE 'wsrep_local_state'" | awk '{ print $2 }')
if [ -z $wsrep_local_state ]; then
  exit 1
fi
if [ $wsrep_local_state == 1 ] || [ $wsrep_local_state == 3 ]; then
  exit 1
fi
```

Prevent killing a node doing an SST (Galera full resync)

Only nodes **Synced** and **Donor** are considered “ready”
Having fun with the **Persistent disk Snapshots**

**MariaDB**

- **On-demand copy**
  - Restore a backup in a minute

- **Daily Copy**
  - To expose fresh dataset for BI Joes

- **Extend a Galera Clusters**
  - Avoid SST

- **Snapshot Validator**
  - To ensure we can restore backups
Having a SLI Prober to implement SLO
MariaDB migration path
MariaDB Migration Path

Initial stage
MariaDB Migration Path
Move the application
MariaDB Migration Path

Setup the database in GCP (with replication)

APP

WRITES

READS

Service (ClusterIP)

Service (ClusterIP)

ProxySQL

read-only

Asynchronous Replication

on-premise
MariaDB Migration Path
Open a Beta endpoint for reads

Asynchronous Replication
MariaDB Migration Path
D-Day: Set read-only

Asynchronous Replication
MariaDB Migration Path

D-Day: Reverse the replication stream

APP

Service (ClusterIP)

Service (ClusterIP)

Service (ClusterIP)

ProxySQL

read-only

read-only

read-only

kubernetes

on-premise

Asynchronous Replication
MariaDB Migration Path

D-Day: Enable writes in GCP

APP

<table>
<thead>
<tr>
<th>BETA READS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service (ClusterIP)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WRITES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service (ClusterIP)</td>
</tr>
</tbody>
</table>

READS

ProxySQL

1. read-only

2. Asynchronous Replication

3. read-only

kubernetes

on-premise
MariaDB Migration Path

D-Day: Change the endpoints

Asynchronous Replication
MariaDB Migration Path

Cleaning and decommissioning

Asynchronous Replication
Elasticsearch
Elasticsearch Migration Path
Implement double-writes in indexer
Elasticsearch Migration Path

Get missing data from on-premise

POST /_reindex (with op_type: create)
Elasticsearch Migration Path

Move the application

kubernetes

APP
INDEXER
SEARCH
INDEX

on-premise

HAProxy

INDEXER
That’s only two use cases...
Solutions chosen in GCP

- Cassandra → Helm Chart
- RabbitMQ → Helm Chart
- Redis → MemoryStore
- PostgreSQL → CloudSQL
- Kafka → K8S Operator
- Couchbase → MemoryStore
“Thanks!"