Migrate from MySQL Galera Base Solution to MySQL Group Replication

Marco Tusa

MySQL Technical Leader Percona

About me

- Open source enthusiast
- MySQL tech lead; principal architect
- Working in DB/development world over 36 years (yes, I am that old)
- Open source developer and community contributor

The Operator Team

Slava Sarzhan

Manager, Cloud Engineering

Ege Güneş

Software Engineer,

pronounced "eg-heh"

Inel Pandzic

Software engineer,

Tomislav Plavcic

Sr. QA Engineer,

Natalia Marukovich

Devops/QA

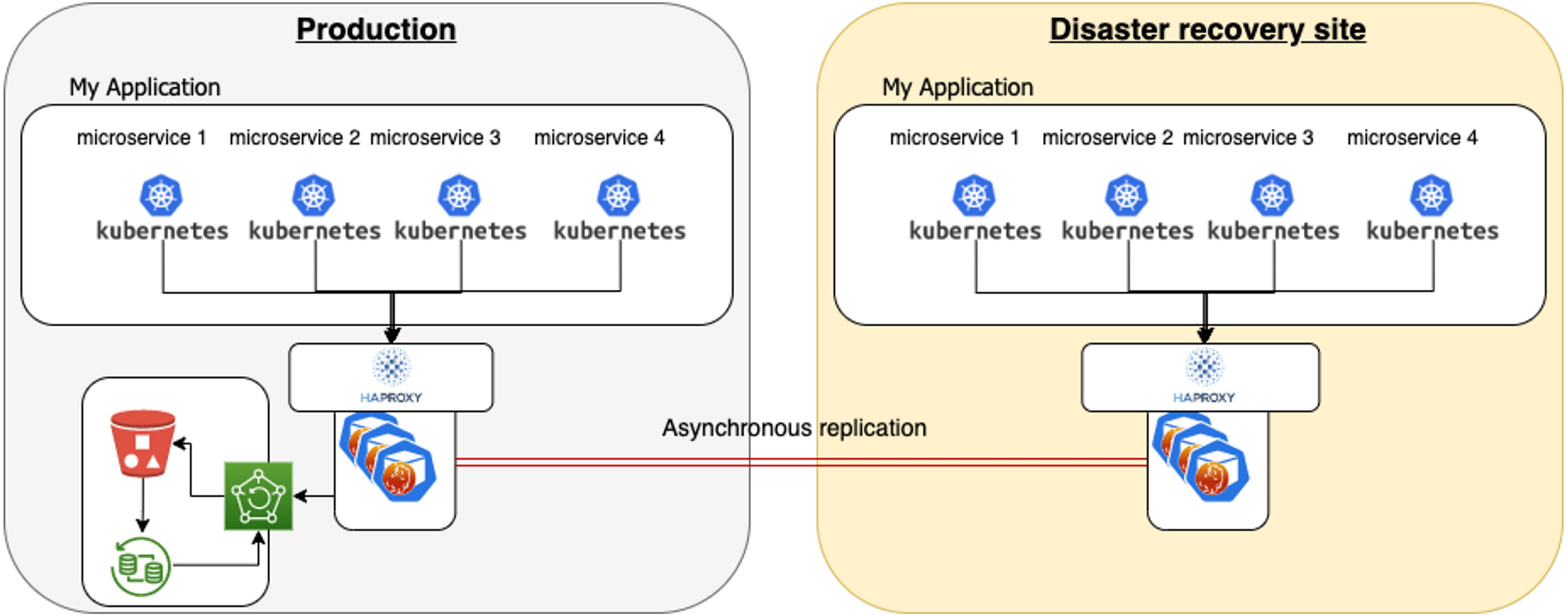
ivan.pylypenko

Devops/Eng

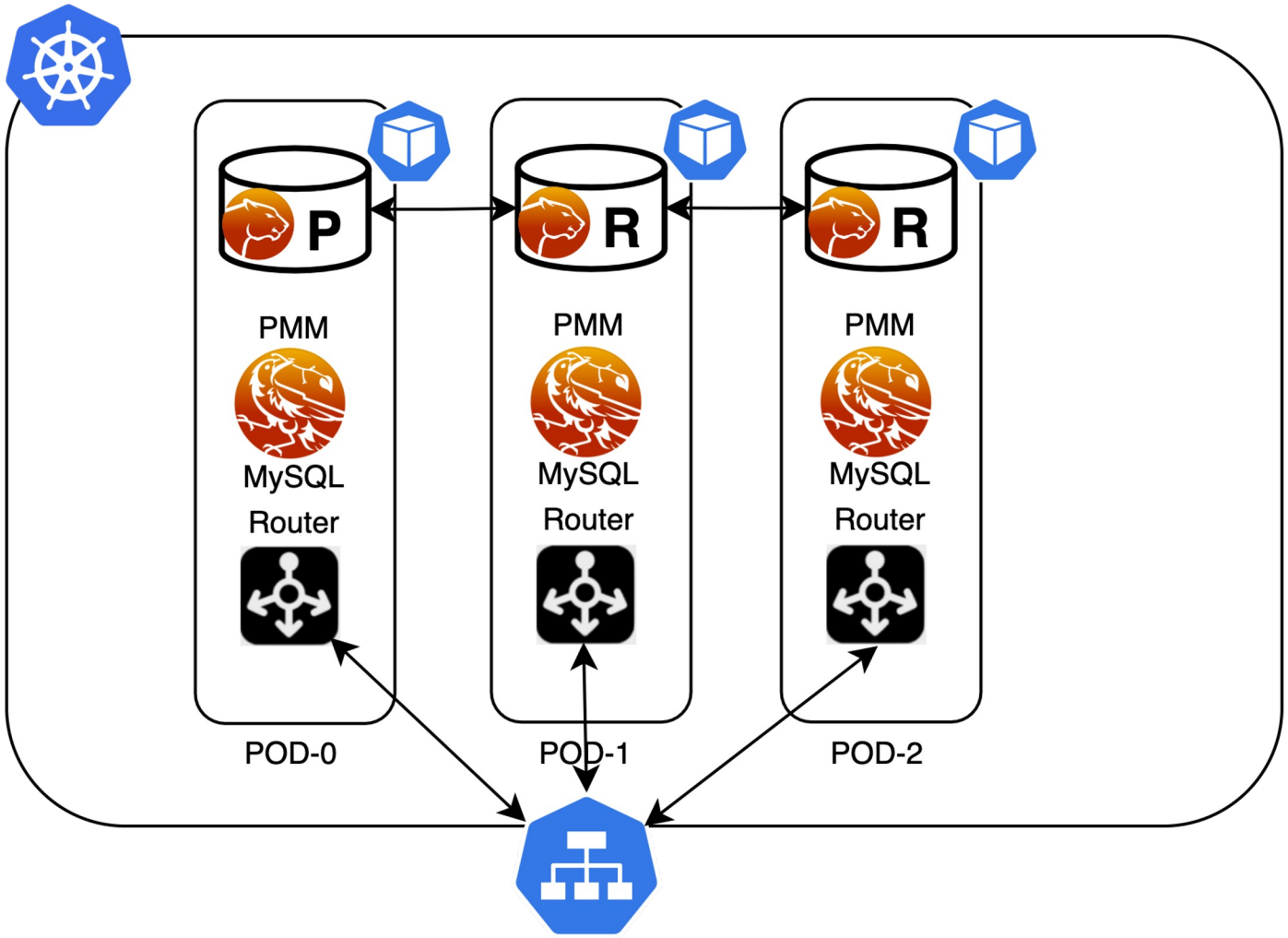
Why?

- Group communication system
- Binlog VS GCache
- GTID VS Sequence Number
- Flow Control
- WAN Support (no way)
- Schema changes
- OS Support

Our environment (k8s) PXC

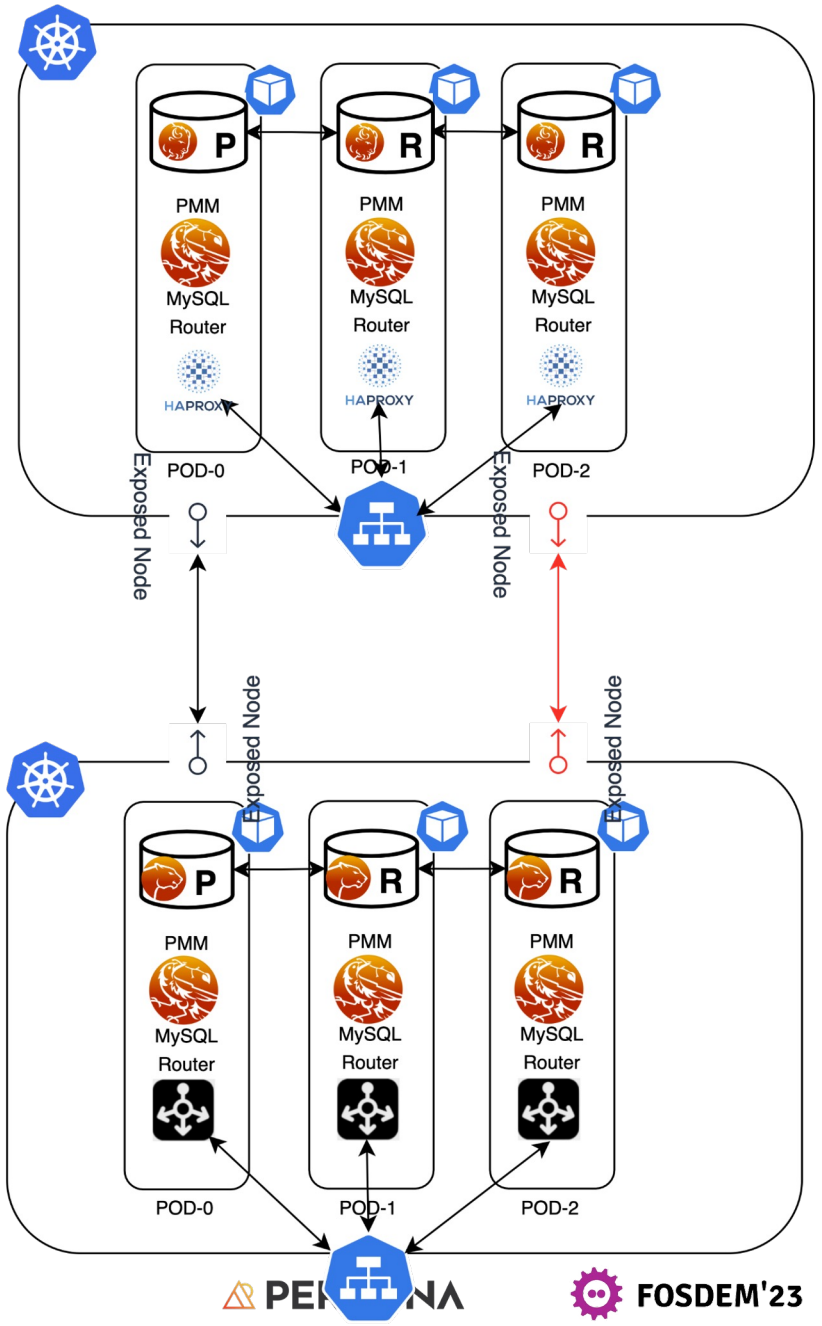


Our environment (k8s) Group Replication



What we will build

- Expose nodes directly
- Asynchronous replication
- Failover channels



Steps

1. Review k8s resources definition on Source & Replica
2. Roll-out standard GR k8s solution
3. Collect fallback information
4. Prepare for work Replica nodes
5. Do Clone or Backup
6. Rebuild Group Replication cluster
7. Add the missing nodes
8. Build Async-replica and add (manually) the auto-failover features.

Step .1 - Review k8s resources definition

review the cr.yaml

- Add expose on source and replica
- Add group replication specific variables
- Create on source migration user which will be (eventually used)

Both Source & Replica

```
expose:
    enabled: true
    type: LoadBalancer
```

Replica

```
configuration:
    [mysqld]
        loose_binlog_transaction_dependency_tracking = WRITESET
        loose_global group_replication_view_change_uuid = AUTOMATIC;
```

Step .2 - Roll-out standard GR k8s solution

Let us review the new entry points

```
[tusa@tusacentral88 deploy]$ kubectl get po
NAME                                           READY   STATUS    RESTARTS   AGE
percona-server-mysql-operator-7c984f7c9-lrghw   1/1     Running   0          2d23h
ps-mysql1-mysql-0                              2/2     Running   0          15m
ps-mysql1-mysql-1                              2/2     Running   1          14m
ps-mysql1-mysql-2                              2/2     Running   1          8m16s
ps-mysql1-router-7779c4797c-f6zh2              1/1     Running   0          3m44s
ps-mysql1-router-7779c4797c-f8j54              1/1     Running   0          3m44s
ps-mysql1-router-7779c4797c-nsgt8              1/1     Running   0          3m44s
```

```
[tusa@tusacentral88 deploy]$ kubectl get services
NAME                     TYPE           CLUSTER-IP       EXTERNAL-IP
ps-mysql1-mysql          ClusterIP      None             <none>
ps-mysql1-mysql-0        LoadBalancer   172.20.6.135     a1012342c911a4070b4ccb885dea5444-1575711499.eu-central-1.elb.amazonaws.com
ps-mysql1-mysql-1        LoadBalancer   172.20.46.183    a3a96dd8e22c94091a0376a76dfe7cd0-1217391740.eu-central-1.elb.amazonaws.com
ps-mysql1-mysql-2        LoadBalancer   172.20.214.88    a5ed8b292c0a9439f8f4e61fbe26c966-1131396173.eu-central-1.elb.amazonaws.com
ps-mysql1-mysql-unready  ClusterIP      None             <none>
ps-mysql1-router         LoadBalancer   172.20.156.107   afe11799c789b4eaa89fe357edc5d40a-1808116416.eu-central-1.elb.amazonaws.com
```

Step .2 - A look to the PXC one

Let us review the new entry points

```
[tusa@tusacentral88 deploy]$ kubectl get pods -n pxc
NAME                                               READY   STATUS      RESTARTS   AGE
mt-cluster-1-haproxy-0                             3/3     Running     2          46d
mt-cluster-1-haproxy-1                             3/3     Running     1          46d
mt-cluster-1-haproxy-2                             3/3     Running     1          46d
mt-cluster-1-pxc-0                                 4/4     Running     0          5d21h
mt-cluster-1-pxc-1                                 4/4     Running     0          5d21h
mt-cluster-1-pxc-2                                 4/4     Running     0          5d21h
percona-xtradb-cluster-operator-547dfcc4fb-lgs2k   1/1     Running     0          46d
```

```
NAME                              TYPE           CLUSTER-IP       EXTERNAL-IP
mt-cluster-1-haproxy              LoadBalancer   172.20.217.125   ac966f7d46c04400fb92a3603f0e2634-193113472.eu-central-1.elb.amazonaws.com
mt-cluster-1-haproxy-replicas     LoadBalancer   172.20.194.19    a5c8836b7c05b41928ca84f2beb48aee-1936458168.eu-central-1.elb.amazonaws.com
mt-cluster-1-pxc                  ClusterIP      None             <none>
mt-cluster-1-pxc-0                LoadBalancer   172.20.140.234   ad01e81ec492547b3ba9be983a698f8e-2010099597.eu-central-1.elb.amazonaws.com
mt-cluster-1-pxc-1                LoadBalancer   172.20.3.56      a0c6f369720784828aaf854cc65fe1a1-761786835.eu-central-1.elb.amazonaws.com
mt-cluster-1-pxc-2                LoadBalancer   172.20.17.202    a4a586f5340f946b796dfd9440353d2f-1378373518.eu-central-1.elb.amazonaws.com
mt-cluster-1-pxc-unready          ClusterIP      None             <none>
percona-xtradb-cluster-operator   ClusterIP      172.20.26.139    <none>
```

Step .2 What about expose?

Keep nodes exposed

ONLY

For the time of the migration

Then

Remove it

Step .2 – User for migration

Is best practice to use a special user for the migration that you will drop after it is completed.

On source:

```
 create user migration@'%' identified by 'migration_password';
 grant backup_admin on *.* to migration@'%'
```

On replica:

```
 create user migration@'%' identified by 'migration_password';
 GRANT SYSTEM_USER, REPLICATION SLAVE, CONNECTION_ADMIN, BACKUP_ADMIN,
GROUP_REPLICATION_STREAM, CLONE_ADMIN,SHUTDOWN ON *.* to migration@'%';
```

Step .3 - Collect fallback information

Collect as much information you can in case something goes wrong.

No need for a backup recreate the cluster is way faster at this stage.

Collect:

- Users & **Grants** ←
- mysql_innodb_cluster_metadata
- Group Replication variables

Step .4 - Prepare for work Replica nodes

Kubernetes has few probes that we want to silence.

For MySQL and Router

```
   readinessProbe:
     timeoutSeconds: 86400
#      timeoutSeconds: 60
   livenessProbe:
#      timeoutSeconds: 60
     timeoutSeconds: 86400
```

And in some moment we will also force operator to stay out of our way with:

```
kubectl -n mt-ps exec ps-mysql1-mysql-X -c mysql -- touch /var/lib/mysql/full-cluster-crash
kubectl -n mt-ps exec ps-mysql1-mysql-X -c mysql -- rm /var/lib/mysql/full-cluster-crash
```

Step .5 - Do Clone or Backup

Clone or backup?

If you have Same Server version:
Use clone.

- easier
- faster
- automated

If you have different Server version:
Use backup.
Operator restore example:

```
apiVersion: ps.percona.com/v1alpha1
kind: PerconaServerMySQLRestore
metadata:
  name: pxc-data-full-jan-19-2023
spec:
  clusterName: ps-mysql1
  backupSource:
    destination: s3://bucket/dir/mt-cluster-1-2023-01-19-17:59:20-full
    storage:
      s3:
        bucket: operator-testing
        credentialsSecret: mt-cluster-1-backup-s3
        region: eu-west-3
      type: s3
```

Note: Percona Server operator can use ANY MySQL compatible image

Step .5 - Do Clone or Backup - preparation

First prevent operator to act on the pod we are going to work on:

```
kubectl -n mt-ps exec ps-mysql1-mysql-0 -c mysql -- touch /var/lib/mysql/full-cluster-crash
```

Then locally connect:

```
kubectl -n mt-ps exec  --stdin --tty ps-mysql1-mysql-0 -c mysql -- /bin/bash
```

Be sure we are on the primary and if not make it primary:

```
select group_replication_set_as_primary(@@server_uuid);
```

```
select * from performance_schema.replication_group_members;
```

```
+---------------------------+--------------------------------------+--------------------------------------------+-------------+--------------+-------------+
| CHANNEL_NAME              | MEMBER_ID                            | MEMBER_HOST                                | MEMBER_PORT | MEMBER_STATE | MEMBER_ROLE |
+---------------------------+--------------------------------------+--------------------------------------------+-------------+--------------+-------------+
| group_replication_applier | 2ab96aaf-9b2b-11ed-a653-863c12047f83 | ps-mysql1-mysql-0.ps-mysql1-mysql.mt-ps |        3306 | ONLINE       | PRIMARY     |
| group_replication_applier | 4b5f04df-9b2b-11ed-a612-4299358decf6 | ps-mysql1-mysql-1.ps-mysql1-mysql.mt-ps |        3306 | ONLINE       | SECONDARY   |
| group_replication_applier | 6be72ca0-9b2b-11ed-a6fd-22a4e552c7e9 | ps-mysql1-mysql-2.ps-mysql1-mysql.mt-ps |        3306 | ONLINE       | SECONDARY   |
+---------------------------+--------------------------------------+--------------------------------------------+-------------+--------------+-------------+
```

Step .5 - Do Clone or Backup

Declare the possible entry points for the clone, take this from the previous list:

```
kubectl get services -n pxc
```

You will have something like this:

```
SET GLOBAL clone_valid_donor_list = 'a0c6f369720784828aaf854cc65fe1a1-761786835.eu-central-1.elb.amazonaws.com:3306';
```

Now is time to clone:

```
stop group_replication;
```

```
set global super_read_only=0;
```

```
CLONE INSTANCE FROM 'migration'@'a0c6f369720784828aaf854cc65fe1a1-761786835.eu-central-1.elb.amazonaws.com':3306 IDENTIFIED BY 'migration_password';
```

Step .5 - Do Clone or Backup - Checks & restore

Once clone is done and instance restarted check the execution:

```
SELECT * FROM performance_schema.clone_status\G
*************************** 1. row ***************************
           ID: 1
          PID: 0
        STATE: Completed
   BEGIN_TIME: 2023-01-23 15:16:57.760
     END_TIME: 2023-01-23 15:26:13.864
       SOURCE: a0c6f369720784828aaf854cc65fe1a1-761786835.eu-central-
1.elb.amazonaws.com:3306
  DESTINATION: LOCAL INSTANCE
     ERROR_NO: 0
ERROR_MESSAGE:
  BINLOG_FILE: binlog.000160
BINLOG_POSITION: 40236
 GTID_EXECUTED: d5788564-7644-11ed-8106-9aa868b6508f:1-18021860  <---
```

Also restore the Grants as previously saved.

Step .6 - Rebuild Group Replication cluster

We will rebuild manually to have better control, using dba.createCluster() had some issue in this context.

Run the commands:

```
Set sql_log_bin=0;
 set global group_replication_group_name = "82f62567-9b2b-11ed-8857-863c12047f83";
 set global group_replication_start_on_boot       =off;
 set global group_replication_group_seeds='ps-mysql1-mysql-0.ps-mysql1-mysql.mt-ps:33061,ps-mysql1-mysql-1.ps-mysql1-mysql.mt-ps:33061,ps-mysql1-mysql-2.ps-mysql1-mysql.mt-ps:33061';
 set global group_replication_bootstrap_group = off;
 set global group_replication_local_address = 'ps-mysql1-mysql-2.ps-mysql1-mysql.mt-ps:33061';
 CHANGE MASTER TO MASTER_USER='replication', MASTER_PASSWORD='replication_password' FOR CHANNEL 'group_replication_recovery';
Set sql_log_bin=1
```

Let us start it:

```
SET GLOBAL group_replication_bootstrap_group=ON;
START GROUP_REPLICATION;
SET GLOBAL group_replication_bootstrap_group=OFF;
```

Step .6 - Rebuild Group Replication cluster

Once the Primary is up let us adopt the cluster:

```
mysqlsh --conect-timeout=600000 --uri 'root:root_password@ps-mysql1-mysql-0.ps-mysql1-mysql.mt-ps:3306'
ps-mysql1-mysql-0.ps-mysql1-mysql.mt-ps:3306 JS >dba.createCluster('psmysql1', {"adoptFromGr": true, "force": true})
ps-mysql1-mysql-0.ps-mysql1-mysql.mt-ps:3306 JS > c=dba.getCluster('psmysql1')
```

```
MySQL  ps-mysql1-mysql-0.ps-mysql1-mysql.mt-ps:3306 ssl  JS >
c.status()
{
    "clusterName": "psmysql1",
    "defaultReplicaSet": {
        "name": "default",
        "primary": "ps-mysql1-mysql-0.ps-mysql1-mysql.mt-ps:3306",
        "ssl": "REQUIRED",
        "status": "OK_NO_TOLERANCE",
        "statusText": "Cluster is NOT tolerant to any failures.",
        "topology": {
            "ps-mysql1-mysql-0.ps-mysql1-mysql.mt-ps:3306": {
                "address": "ps-mysql1-mysql-0.ps-mysql1-mysql.mt-
ps:3306",
                "memberRole": "PRIMARY",
                "mode": "R/W",
                "readReplicas": {},
                "replicationLag": "applier_queue_applied",
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.29"
            }
        },
        "topologyMode": "Single-Primary"
    },
    "groupInformationSourceMember": "ps-mysql1-mysql-0.ps-mysql1-
mysql.mt-ps:3306"
}
```

Step .7 - Add the missing nodes

First on both nodes: `kubectl -n mt-ps exec ps-mysql1-mysql-1 -c mysql -- touch /var/lib/mysql/full-cluster-crash`

Let us now add the other nodes one a time:

```
c.addInstance('ps-mysql1-mysql-1.ps-mysql1-mysql.mt-ps:3306',{recoveryMethod: 'auto'})
+---------------------------+--------------------------------------+--------------------------------------+-------------+--------------+-------------+
| CHANNEL_NAME              | MEMBER_ID                            | MEMBER_HOST                          | MEMBER_PORT | MEMBER_STATE | MEMBER_ROLE |
+---------------------------+--------------------------------------+--------------------------------------+-------------+--------------+-------------+
| group_replication_applier | 0f6807b3-9b41-11ed-a658-eea5306ec7fa | ps-mysql1-mysql-1.ps-mysql1-mysql.mt-ps |        3306 | ONLINE       | SECONDARY   |
| group_replication_applier | 2ab96aaf-9b2b-11ed-a653-863c12047f83 | ps-mysql1-mysql-0.ps-mysql1-mysql.mt-ps |        3306 | ONLINE       | PRIMARY     |
+---------------------------+--------------------------------------+--------------------------------------+-------------+--------------+-------------+
```

```
c.addInstance('ps-mysql1-mysql-2.ps-mysql1-mysql.mt-ps:3306',{recoveryMethod: 'auto'})
+---------------------------+--------------------------------------+--------------------------------------+-------------+--------------+-------------+
| CHANNEL_NAME              | MEMBER_ID                            | MEMBER_HOST                          | MEMBER_PORT | MEMBER_STATE | MEMBER_ROLE |
+---------------------------+--------------------------------------+--------------------------------------+-------------+--------------+-------------+
| group_replication_applier | 0f6807b3-9b41-11ed-a658-eea5306ec7fa | ps-mysql1-mysql-1.ps-mysql1-mysql.mt-ps |        3306 | ONLINE       | SECONDARY   |
| group_replication_applier | 2ab96aaf-9b2b-11ed-a653-863c12047f83 | ps-mysql1-mysql-0.ps-mysql1-mysql.mt-ps |        3306 | ONLINE       | PRIMARY     |
| group_replication_applier | 5c1eb757-9b41-11ed-a753-3abd7c163f77 | ps-mysql1-mysql-2.ps-mysql1-mysql.mt-ps |        3306 | ONLINE       | SECONDARY   |
+---------------------------+--------------------------------------+--------------------------------------+-------------+--------------+-------------+
```

```
 MySQL  ps-mysql1-mysql-0.ps-mysql1-mysql.mt-ps:3306 ssl  JS > c.status()
{
    "clusterName": "psmysql1",
    "defaultReplicaSet": {
        "name": "default",
        "primary": "ps-mysql1-mysql-0.ps-mysql1-mysql.mt-ps:3306",
        "ssl": "REQUIRED",
        "status": "OK",
        "statusText": "Cluster is ONLINE and can tolerate up to ONE failure.",
        "topology": {
            "ps-mysql1-mysql-0.ps-mysql1-mysql.mt-ps:3306": {...},
            "ps-mysql1-mysql-1.ps-mysql1-mysql.mt-ps:3306": {...},
            "ps-mysql1-mysql-2.ps-mysql1-mysql.mt-ps:3306": {...}
        },
        "topologyMode": "Single-Primary"
    },
    "groupInformationSourceMember": "ps-mysql1-mysql-0.ps-mysql1-mysql.mt-ps:3306"
}
```

Step .8 - Build Async-replica

Asynchronous replica needs to be manually built

- Conflict with existing InnoDB Cluster
- Conflict with Operator operations

We are working to make it smoother and fully automated as PXC solution has

Step .8 - Build Async-replica

Said that we use standard CHANGE REPLICATION command to manage it (on all pods):

```
STOP GROUP_REPLICATION;

CHANGE REPLICATION SOURCE to master_user='migration',
master_password='migration_password',
master_host='<PXC node>',
master_auto_position=1,
master_retry_count=6,
source_connection_auto_failover=0,
master_connect_retry=10
for channel "dc1_to_dc2";

START GROUP_REPLICATION;
```

On

```
*************************** 1. row
Slave_IO_State: Waiting for source to send event
Master_Host: ad01e81ec492547b3ba9be983a698f8e-2010099597.eu-central-1.elb.amazonaws.com
Master_User: migration
Master_Port: 3306
Connect_Retry: 10
Master_Log_File: binlog.000156
Read_Master_Log_Pos: 42502
Relay_Log_File: ps-mysql1-mysql-0-relay-bin-dc1_to_dc2.000002
Relay_Log_Pos: 3389
Relay_Master_Log_File: binlog.000156
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
```

Step .8 - Build Async-failover

Last step is to set the asynchronous failover nodes (on Primary):

```
SELECT asynchronous_connection_failover_add_source('dc1_to_dc2', 'a8e6371cdbfc944baa5e10a6cece90a5-940431880.eu-central-1.elb.amazonaws.com', 3306, null, 100);

SELECT asynchronous_connection_failover_add_source('dc1_to_dc2', 'a0c6f369720784828aaf854cc65fe1a1-761786835.eu-central-1.elb.amazonaws.com', 3306, null, 80);

SELECT asynchronous_connection_failover_add_source('dc1_to_dc2', 'a4a586f5340f946b796dfd9440353d2f-1378373518.eu-central-1.elb.amazonaws.com', 3306, null, 50);
```

Activate the failover on each node (Only on Primary to start replication):

```
STOP SLAVE FOR CHANNEL 'dc1_to_dc2';CHANGE REPLICATION SOURCE to source_connection_auto_failover=1 for channel "dc1_to_dc2";START SLAVE FOR CHANNEL 'dc1_to_dc2';
```

Congratulation

You are ready to TEST

BEFORE using it for PRODUCTION

Conclusions

- We want to make the whole process (or most of it) fully automated
- Operator for Percona Server (GR solution) is in tech preview
- PXC is still the Primary solution
- Proxies: Router is the default now, HAProxy will be included in the solution soon

Still a long way to go

Refences

https://docs.percona.com/percona-operator-for-mysql/ps/index.html

https://github.com/percona/percona-server-mysql-operator

Q & A

Percona is a world-class open source database software, support, and services company focused on helping you scale and innovate with speed as you grow.

83M+
Software Downloads
TTM as of November 2022

100K+
Blog Views Per Month

800+
Customers

40%
YoY MRR Growth
As of Q4 2021

Trusted by...

More than a third of the Fortune 50

4 of the top 6 Retailers

3 of the top 5 Healthcare Companies

9 of the top 10 Tech Companies

6 of the top 10 Gaming Companies

4 of the top 5 Manufacturing Companies