Database schema initialization

1. Introduction

Database initialization is performed in 2 parts:

- schema+users creation
- adding some data inside the database (does not require root access to the database)

This document emphasize about the first part, but the second one could be implemented in the very same step.

Below are the motivations:

- Prevent accidental schema upgrades
- Provide administrator more detail and debugging information so they know how a schema upgrade is progressing
    - Add init method for xrootd to wait for schema upgrade to proceed
- Separation of xrootd and mariadb for development

Technical requirements:

- xrootd must communicate with mariadb via sockets. TCP connections are not performant in past testing.
- mariadb connections via sockets are authenticated
- mariadb needs to start before xrootd
- 1 xrootd instance must be paired with 1 mariadb instance. This pairing should run as 1 per node.

2. Strategies

2.1. From inside the Pod running mysql

Reminder: Both the Qserv worker and Czar Pods embed a mysql container, so running entrypoint script in any of their container will allow access to mysql file socket.

blocked URL

2.1.1. From an initContainer

Pros:

Network safe: Can use local mysql socket

User friendliness for k8s admin: +++++ (the initContainer from the database pod initialize the database schema)

For xrootd add in schema upgrade flag file (emptyDir) to signal to xrootd that schema upgrading is occurring. xrootd startup script pauses if that flag is there.

Cons:

Need to start/stop mysqld inside the initContainer

mysql and entrypoint run be in the same initContainer

xrootd will fail until schema update and start/top mysqld finishes.

xrootd is stateless application so can be run as a deployment

Example:

Current Qserv setup, works since ~2 years

Also used for creating qserv-ingest schema

2.1.2. From a sidecar container

Pros:

Network safe: Can use local mysql socket

User friendliness for k8s admin: ++++

entrypoint container do not need to embed mysql

For Qserv worker the sidecar container can be xrootd or cmsd, but it is not a good separation of concern then. And it would not be trivial for the

Cons:

Need to run `sleep infinity` at the end of the sidecar container command

Example:

From the k8s official book: MongoDB is initialized inside a sidecar container which finishes with an infinite loop: http://eddiejackson.net/azure/Kubernetes_book.pdf#%5B%7B%22num%22%3A547%2C%22gen%22%3A0%7D%2C%7B%[…]%22%3A%22XYZ%22%7D%2Cnull%2C125.91296%2Cnull%5D

2.1.3. From the database container

This behavior is the one used by two mainstream products: **MongoDB** helm chart and **Vitess** operator.

Question: does mariadb provide a feature to create custom schema at startup (ping Fritz Mueller)?

Pros:

Network safe: Can use local mysql socket

User friendliness for k8s admin: +++

Cons:

mysql and entrypoint run be in the same initContainer

Example:

1. **Vitess** initializes the schemas from inside its mysql pods:

   ```
   kubectl describe pods example-vttablet-zone1-2469782763-bfadd780
   ...
   mysqld:
       Container ID:
       Image:         vitess/lite:latest
       Image ID:
       Port:          3306/TCP
       Host Port:     0/TCP
       Command:
         /vt/bin/mysqlctld
       Args:
         --db-config-dba-uname=vt_dba
         --db_charset=utf8
         --init_db_sql_file=/vt/secrets/db-init-script/init_db.sql
         --logtostderr=true
         --mysql_socket=/vt/socket/mysql.sock
         --socket_file=/vt/socket/mysqlctl.sock
         --tablet_uid=2469782763
         --wait_time=2h0m0s
   ...
   ```

   Look at the option `--init_db_sql_file` above. It seems {Vitess} team has developed a `mysqlctld` process to start and init its mysql pods, this looks a bit like Qserv `entrypoint` but it is running from inside the MySQL container
2. **MongoDB** helm chart initialize the clustered database in the same pod that start mongodb (see setup.sh script): https://github.com/bitnami/charts/blob/master/bitnami/mongodb/templates/replicaset/scripts-configmap.yaml,

2.2. From outside the Pod running mysql

WARNING: this strategy will require an **additional network 'root@%' access to mysql**, this is not a recommended for security. The security consideration is that another pod could be scheduled to read data from localhost. Note they would need to have the authentication credentials too to access data.

This strategy could apply for replication controller/replication database, but not for Czar and Worker which both embed their mysql container.

blocked URL

2.2.1. From an initContainer

Pros:

emtrypoint container does not need to embed mysql

Cons:

Network unsafe: Require 'root@%' access to mysql (`%` will be required because the pod running the container will not be registered inside DNS if its main service is not started, because of readinessProbe)

User friendliness for k8s admin: ++, difficult to understand which service initialize which database without having a good knowledge of the project

Not useful for Czar and Worker as long as they embed a mysql container?

Example:

None yet

2.2.2. From a Container

Pros:

entrypoint container does not need to embed mysql

seperate init for xrootd

Cons:

Network unsafe: Require `root@%` access to mysql (`%` will be required because the pod running the container will not be registered inside DNS if its main service is not started, because of readinessProbe)

User friendliness for k8s admin: +, difficult to understand which service initialize which database without having a good knowledge of the project

Not useful for Czar and Worker as long as they embed a mysql container?

Example:

None yet

Remark: `root@%` access to mysql can be removed after database initialization in order to reduce risk, but this may end-up in a complex database upgrade procedure (add this access at the beginning of each upgrade, and then remove it)