Database initialization

How to install and initialize database

- Initialize database on your server
- Initialize database using Docker
- Initialize database on Kubernetes
- Creating a multimaster MariaDB replica
- Configuring database cluster

Initialize database on your server

> The purpose of this tutorial is to show how to initialize a database required for Soffid IAM installation.

Prerequisites

First of all, you sould install a database required in the Soffid IAM installation.

The supported databases are:

- MySQL
- MariaDB
- PostgreSQL
- Oracle
- Microsoft SqlServer

MySQL/MariaDB

In order to configure MySQL database you need access to the database administration tool (mysql) with superuser permissions using a TCP/IP connection. If needed, please create a user for the Soffid installation. If you don't have such a user, or don't know its password, please access MySQL as root, execute the **mysql** tool and create the user with **grant command** *(where ADMIN_USER is the user to be used during the installation process to create the soffid repository database and ADMIN_PASSWORD is the required password).*

```
create database soffid;
use soffid;
grant all privileges on *.* to ADMIN_USER@localhost identified by 'ADMIN_PASSWORD' with grant option;
```

In addition, in order to be able to manage big files, like process definitions or software add-ons, we have to modify the **max_allowed_packet** parameter on MySQL. This parameter is commonly located on the **/etc/mysql/my.cnf** file.

>

> You can find thedefault option file locations on Linux, Unix, Mac or Windows following this link.

```
[mysqld]
max_allowed_packet=128M
```

> If the version of MariaDB is 10.1.38, or newer, the recommended value for max_allowed_packet is 512M

Note: in the case, we will obtain the next '*The size of BLOB/TEXT data inserted in one transaction is greater than 10% of redo log size. Increase the redo log size using innodb_log_file_size*.' error when trying to upload an addon, we may update the default value of this mysql/mariadb parameter. This parameter is commonly allocated on the /etc/mysql/my.cnf file.

```
[mysqld]
innodb_log_file_size=256M
```

If you are installing on a Ubuntu 18.04 server, the default character set is set to utf8mb4. Using this character set can cause problems, as many index sizes will exceed the maximum key size of 767 bytes. To prevent this problem, change the following settings:

```
[mysqld]
character-set-server  = Latin1
collation-server      = Latin1_general_ci
```

Alternatively, if UTF character set is required, write the following settings:

```
[mysqld]
character-set-server  = utf8mb4
collation-server      = utf8mb4_general_ci
innodb_large_prefix   =  1
innodb_file_format    = Barracuda
innodb_file_per_table =  1
```

> Following this link you will find the steps to set up a two nodes database cluster.

Oracle

A new database instance should be created. Optionally two tablespaces should be created (SOFFID_DATA and SOFFID_INDEX) to separate soffid tables and indexes.

```
CREATE TABLESPACE SOFFID_DATA DATAFILE '/app/oracle/oradata/project/soffid_data.dbf' SIZE 200M EXTENT MANAGEMENT LOCAL AUTOALLOCATE
```

To create the tablespace is necessary to provide the full path name, its size and MANAGEMENT AUTOALLOCATE option. The autoallocate option is needed because the tables are not sized by database creation scripts. Also, the Oracle Listener must have a TCP/IP port accepting connections.

Microsoft SQLServer

You must enable the SQL Server Browser Service at startup and the authentication method have to be set to “SQL Server and Windows Authentication mode”.

In addition, you must ensure that 'READ_COMMITTED_SNAPSHOT" parameter is enabled, you can do so with the following command:

```
ALTER DATABASE [database_name] SET READ_COMMITTED_SNAPSHOT ON
```

Initialize database using Docker

> The purpose of this tutorial is to show how to initialize a database **MariaDB** required for Soffid IAM installation using Docker.

Prerequisites

1. Install docker (https://docs.docker.com/install/)

2. Create a docker network, that network allows you to connect containers to the same bridge network to communicate:

```
sudo docker network create -d bridge NETWORKNAME
```

> For the correct installation of Soffid it is recommended not to use the underline character _ in the network name.

sudo docker exec -i -t ID_CONTAINER /bin/bashMySQL/MariaDB

First step will be initialize MariaDB with Docker, in this case we attach the container to an exist network:

```
sudo docker run -d --name mariadb-service  --network=NETWORKNAME -e "MYSQL_ROOT_PASSWORD=ADMIN_PASSWORD" mariadb
```

Second, you can check the deployed containers:

```
sudo docker ps
```

Then, you must connect to the created container:

```
sudo docker exec -i -t mariadb-service  /bin/bash
```

In order to configure MySQL database you need access to the database administration tool (mysql) with superuser permissions using a TCP/IP connection. If needed, please create a user for the Soffid installation. If you don't have such a user, or don't know its password, please access MySQL as root, execute the **mysql** tool and create the user with **grant command** *(where ADMIN_USER is the user to be used during the installation process to create the soffid repository database and ADMIN_PASSWORD is the required password).*

Coonect to MySQL:

```
mysql -u root -p
```

Create database and grant permissions:

```
create database soffid;
use soffid;
grant all privileges on *.* to ADMIN_USER@'%' identified by 'ADMIN_PASSWORD' with grant option;
```

In addition, in order to be able to manage big files, like process definition or software addons, we have to modify **max_allowed_packet** parameter on MySQL. This parameter is commonly allocated on the /etc/mysql/my.cnf file.

```
[mysqld]
max_allowed_packet=128M
```

> If the version of MariaDB is 10.1.38, or newer, the recommended value for max_allowed_packet is 512M

Note: in the case we will obtain the next '*The size of BLOB/TEXT data inserted in one transaction is greater than 10% of redo log size. Increase the redo log size using innodb_log_file_size*.' error trying to upload an addon, we may update the default value of this mysql/mariadb parameter. This parameter is commonly allocated on the /etc/mysql/my.cnf file.

```
[mysqld]
innodb_log_file_size=256M
```

If you are installing on a Ubuntu 18.04 server, default character set is set to utf8mb4. Using this character set can cause problems, as many index sizes will exceed maximum key size of 767 bytes. To prevent this problem, change following settings:

```
[mysqld]
character-set-server  = Latin1
collation-server      = Latin1_general_ci
```

Alternatively, if UTF character set is required, write the following settings:

```
[mysqld]
character-set-server  = utf8mb4
collation-server      = utf8mb4_general_ci
innodb_large_prefix   =  1
innodb_file_format    = Barracuda
innodb_file_per_table =  1
```

Following this link you will find the steps to setup a two nodes database cluster.

Video Tutorial

MariaDB initialization using Docker

https://www.youtube.com/embed/mDJeSRbrn7w

Initialize database on Kubernetes

> The purpose of this tutorial is to show how to initialize a **MariaDB** database required for Soffid IAM installation on Kubernetes.

MySQL/MariaDB

To initialize MariaDB on Kubernetes first of all you must create a Persistent Volume. Storage in the cluster will be provisioned using Storage Classes.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: local-pv3
spec:
  capacity:
    storage: 10Gi
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: local-storage
  local:
    path: /home/ulocal/kubernetes-disk3
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - soffid123
---
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mariadb-claim3
spec:
  storageClassName: local-storage
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
```

> Path "/home/ulocal/kubernetes-disk3" must be exists.

Then you must define the MariaDB deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mariadb3
  labels:
    app: soffid
    instance: "Soffid-3"
    type: database
spec:
  strategy:
    rollingUpdate:
      maxSurge: 0
      maxUnavailable: 1
    type: RollingUpdate
  replicas: 1
  selector:
    matchLabels:
      app: soffid
      instance: "Soffid-3"
      type: database
  template:
    metadata:
      labels:
```

```
    app: soffid
    instance: "Soffid-3"
    type: database
spec:
  restartPolicy: Always
  containers:
    - name: mariadb3
      image: mariadb
      resources:
        limits:
          memory: 2Gi
        requests:
          memory: 400Mi
      args:
        - "--max-allowed-packet=175M"
        - "--innodb-log-file-size=256M"
        - "--character-set-server=utf8"
        - "--collation-server=utf8_bin"
        - "--net-read-timeout=3600"
        - "--net-write-timeout=3600"
        - "--innodb-buffer-pool-size=100M"
      ports:
        - containerPort: 3306
          name: db-port
      env:
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mariadb
              key: root_password
        - name: MYSQL_USER
          valueFrom:
            secretKeyRef:
              name: mariadb
              key: username
        - name: MYSQL_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mariadb
              key: password
```

```
            - name: MYSQL_DATABASE
              value: soffid
          volumeMounts:
          - name: mysql-persistent-storage3
            mountPath: /var/lib/mysql

      volumes:
        - name: mysql-persistent-storage3
          persistentVolumeClaim:
            claimName: mariadb-claim3
---
apiVersion: v1
kind: Service
metadata:
  name: mariadb3-service
  namespace: default
spec:
  clusterIP: None
  ports:
  - name: mariadb
    port: 3306
    protocol: TCP
    targetPort: 3306
  selector:
    app: soffid
    instance: "Soffid-3"
    type: database
  type: ClusterIP
```

Finally you must create resources in a cluster:

```
kubectl apply -f mariadb-pv-file.yaml
kubectl apply -f mariadb-deployment-file.yaml
```

Video Tutorial

MariaDB initialization in Kubernetes

https://www.youtube.com/embed/_F6p8JlurXs?rel=0

Creating a multimaster MariaDB replica

This topic will cover the process to create a two node Maria DB cluster. The cluster will be configured to allow Soffid console to use either database node, which in turn will replicate data changes to the other one.

Node 1 action	Node 2 action
Create and setup a Maria DB in node 1.	
Configure Maria DB to generate binary log files. Add the following lines to /etc/mysql/my.cnf: *server-id = **1*** *log-bin* *binlog-format=row* *expire_logs_days = 15* *max_binlog_size = 1000M* *replicate-ignore-table = soffid.SC_SEQUENCE* *slave-skip-errors = 1032,1053,1062*	
Restart MariaDB: `service mysql restart`	
	Create and setup a Maria DB in node 2.
	Configure Maria DB to generate binary log files. Add the following lines to /etc/mysql/my.conf: *server-id = **2*** *log-bin* *binlog-format=row* *expire_logs_days = 15* *max_binlog_size = 1000M* *replicate-ignore-table = soffid.SC_SEQUENCE* *slave-skip-errors = 1032,1053,1062*

<table>
<tr><th>Node 1 action</th><th>Node 2 action</th></tr>
<tr><td></td><td>Restart MariaDB:
<code>service mysql restart</code></td></tr>
<tr><td>Dump current database contents:
mysqldump soffid -u soffid -p >soffid.data</td><td>Load current database contents
mysql -u soffid -p < soffid.data</td></tr>
<tr><td></td><td>Create a user for node 1 to fetch data from node 2. From mysql, execute:
grant replication slave on *.* to replication_user@<NODE1-IP>

set password for replication_user@1<NODE1-IP> = password('<NODE1-PASS>')</td></tr>
<tr><td>Create a user for node 2 to fetch data from node 1. From mysql, execute:
grant replication slave on *.* to replication_user@<NODE2-IP>

set password for replication_user@1<NODE2-IP> = password('<NODE2-PASS>')</td><td></td></tr>
<tr><td>Query current binary log position:
MariaDB [(none)]> show master status;

The result should look like this:
<table><tr><th>File</th><th>Position</th><th>Binlog_Do_DB</th><th>Binlog_Ignore_DB</th></tr><tr><td>mysqld-bin.000030</td><td>68175</td><td></td><td></td></tr></table>
The got values will be used on node 2 to start replica process.</td><td></td></tr>
</table>

<table>
<tr><th>Node 1 action</th><th>Node 2 action</th></tr>
<tr><td></td><td>Start replication from node 1 to node 2. From mysql, execute the following sentence, replacing proper values:
CHANGE MASTER TO
MASTER_HOST='<NODE1-IP>',
MASTER_USER='replication_user',
MASTER_PASSWORD='<NODE2-PASS>',
MASTER_PORT=3306,
MASTER_LOG_FILE='<NODE1-FILE>' , /** i.e. mysql-bin.000030 **/
MASTER_LOG_POS=<NODE1-POSITION>, /** i.e. 68175 **/
MASTER_CONNECT_RETRY=10;</td></tr>
<tr><td></td><td>Verify replica is working right, by executing
SHOW SLAVE STATUS \G

Check following lines:
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
Seconds_Behind_Master: 0</td></tr>
<tr><td></td><td>Query current binary log position:
MariaDB [(none)]> show master status;

The result should look like this:<table><tr><th>File</th><th>Position</th><th>Binlog_Do_DB</th><th>Binlog_Ignore_DB</th></tr><tr><td>mysqld-bin.000060</td><td>98325</td><td></td><td></td></tr></table>The got values will be used on node 1 to start replica process.</td></tr>
<tr><td>Now, start replication from node 2 to node 1. From mysql, execute the following sentence, replacing proper values:
CHANGE MASTER TO
MASTER_HOST='<NODE2-IP>',
MASTER_USER='replication_user',
MASTER_PASSWORD='<NODE1-PASS>',
MASTER_PORT=3306,
MASTER_LOG_FILE='<NODE2-FILE>', /** i.e. mysql-bin.000060 **/
MASTER_LOG_POS=<NODE2-POSITION>, /** i.e. 98325 **/
MASTER_CONNECT_RETRY=10;</td><td></td></tr>
</table>

Node 1 action	Node 2 action
Verify replica is working right, by executing *SHOW SLAVE STATUS \G* Check following lines: Slave_IO_Running: Yes Slave_SQL_Running: Yes Seconds_Behind_Master: 0	
Now, create and start SC_SEQUENCE table in node 1. This sequence will generate values 1, 11, 21, 31, 41, and so on: *CREATE TABLE `SC_SEQUENCE` (* *`SEQ_NEXT` bigint(20) NOT NULL,* *`SEQ_CACHE` bigint(20) NOT NULL,* *`SEQ_INCREMENT` bigint(20) NOT NULL* *);* *INSERT INTO SC_SEQUENCE VALUES (1, 100, 10);*	
	Now, create and start SC_SEQUENCE table in node 2. This sequence will generate values 2, 12, 22, 32, 42, and so on:: *CREATE TABLE `SC_SEQUENCE` (* *`SEQ_NEXT` bigint(20) NOT NULL,* *`SEQ_CACHE` bigint(20) NOT NULL,* *`SEQ_INCREMENT` bigint(20) NOT NULL* *);* *INSERT INTO SC_SEQUENCE VALUES (2, 100, 10);*

Now, configure the Console to use the following jdbc URL:

jdbc:mariadb:sequential://mariadb-host-1,mariadb-host-2/soffid

Configuring database cluster

Once the database replica is setup, it's important to guarantee transactianality rules. To achive it, one database instance must be acting as the master and other as the slave.

Using corosync and pacemaker, you can configure a floating IP address that will mark which one is the active one at each moment.

Node 1	Node 2
Install Corosync and Pacemaker. It is recommended to use **apt** or **yum** because these programs will handle dependencies for you, making the process much easier.	Install Corosync and Pacemaker.
Cluster nodes need a key in order to authenticate the packages sent between them by corosync. *sudo corosync-keygen* Once the key has been generated, copy it to the other nodes: *sudo scp /etc/corosync/authkey <user>@<other-cluster-node>:/home/<user>*	
	Once the key has been copied, move the copied key from the */home/<user>* route to */etc/corosync/authkey*
Now we need to tell Corosync which IP to use to communicate with other nodes in the cluster. Open */etc/corosync/corosync.conf* and edit the bindnetaddr field. Set the right IP and save the file. We need to do this in every node in the cluster, although you can use the same file if you set the right name in your hosts file.	
	Configure Corosync with the right IP binding as done in node 1.
Configure the */etc/default/corosync* file to enable Corosync changing START to yes "START=yes". Then we can start Corosync using *sudo service corosync start*.	
	Enable Corosync and start it as in node 1.

Node 1	Node 2
Allow the nodes a few seconds to start, then you can monitor the cluster nodes using sudo crm_mon. The result should be similar to this: ========== *Last updated: Mon Mar 31 14:05:23 2015* *Stack: corosync* *Current DC: yourDC - partition with quorum* *Version: 1.x.x-yourversion* *2 Nodes configured, 2 expected votes* *0 Resources configured.* ========== *Online: [node1 node2]*	
	Check the nodes with sudo crm_mon
Corosync is ready, now we will tell Pacemaker which resources we want it to handle in HA. These will be the database and a virtual IP (VIP) we will use to address the cluster. Add the VIP to the node, and then use this to create the resource: *sudo crm configure primitive FAILOVER-ADDR ocf:heartbeat:IPaddr2 params ip="your.virtual.IP" nic="your.network.device" op monitor interval="10s" meta is-managed="true"* You can check the result using *sudo crm status*, which should look something like: *Last updated: Wed Jan 18 10:21:12 2017 Last change: Tue Jan 17 13:08:25 2017 by hacluster via crmd on nodename* *Stack: corosync* *Current DC: nodename(version 1.1.14-70404b0) - partition with quorum* *2 nodes and 2 resources configured* *Online: [node1 node2]* *Full list of resources:* *Resource Group: my_cluster* *FAILOVER-ADDR (ocf::heartbeat:IPaddr2): Started node2*	
Now we will add the database. We will use: *sudo crm configure primitive FAILOVER-MARIADB lsb::mysql op monitor interval=15s*	