RTCMS - Admin guide

Table of Contents

1. RTCMS - Admin and description

1.1. Directory structure of RTCMS installation

Directory rtcms-wrapper where rtcms-wrapper was installed have the following structure:

```
-- rtcms-wrapper
   |-- bin
   |   |-- cryptoParametes.sh
   |   |-- crypto-property-cli-1.3-runner.jar
   |   |-- rtcms.java.status
   |   |-- rtcms.pid
   |   |-- rtcms.sh
   |   |-- rtcms.status
   |   |-- showlog.sh
   |   `-- wrapper
   |-- config
   |   |-- application.properties
   |   |-- cTokenKey.p12
   |   `-- rtcms.conf
   |-- lib
   |   |-- libwrapper.so
   |   `-- wrapper.jar
   |-- logs
   |   |-- rtcms-log.txt
   |   |-- rtcms-log.txt.1
   |   `-- wrapper.log
   |-- quarkus-app
```

Directory	Description
rtcms-wrapper	RTCMS installation home
bin	executable scripts
config	config and key stores files for application
lib	java wrapper system library and specific jar
logs	rtcms logs files. Rotate after restart
quarkus-app	rtcms quarkus application files

RTCMS running in quarkus environment - Quarkusr.

What is Quarkus?

Traditional Java stacks were engineered for monolithic applications with long startup times and large memory requirements in a world where the cloud, containers, and Kubernetes did not exist. Java frameworks needed to evolve to meet the needs of this new world.

Quarkus was created to enable Java developers to create applications for a modern, cloud-native world. Quarkus is a Kubernetes-native Java framework tailored for GraalVM and HotSpot, crafted from best-of-breed Java libraries and standards. The goal is to make Java the leading platform in Kubernetes and serverless environments while offering developers a framework to address a wider range of distributed application architectures.

1.2. Start/Stop RTCMS

```
 rtcms.sh [status | start | stop | restart | console]
 Example:
  rtcms.sh status
..
 Result is:
   rtCMS vcard processor (not installed) is running: PID:236444, Wrapper:STARTED,
Java:STARTED
```

In this installation RTCMS not used container. Java running as linux process with tanuki software service wrapper. For details Java Service Wrapper.

1.3. showlog.sh

Command ***showlog.sh*** display *rtcms-log.txt* log file. It display last 40 lines and refresh every 15 seconds. Exit with ^C.

```
 Example:
  showlog.sh
 ^C
```

1.4. Get log and config files from server to local desktop

```
  scp rtcms@172.30.94.233:/home/rtcms/rtcms-wrapper/config/application.properties .
  scp rtcms@172.30.94.233:/home/rtcms/rtcms-wrapper/logs/rtcms-log.txt .
```

1.5. Service health check

```
curl http://172.30.94.233:8443/tokenizer/api/testDS
```

Response:

```
    {
        "status": "UP",
        "checks": [
            {
                "name": "Database connections health check",
                "status": "UP",
                "data": {
                    "<default>": "UP",
                    "ccms-camel": "UP",
                    "queue": "UP"
                }
            }
        ]
    }
```

This rest service should be used to verify the availability of **RTCMS application**.

- "status": "**UP**" show that rtcms and all db connection are available.
- <**default**> is the local RTCMS Oracle DB
- **ccms-camel** is CCMS db for GCARD_REP replication
- **queue** - is backup RTCMS database.

This service can be used for monitoring and failover from the network environment.

1.6. Swagger API

The Swagger API is available on port 9002. The Vcard API is accessed on port 8443.

Swagger application URL:

```
http://localhost:9002/q/swagger-ui/
```

It is necessary to provide access through the firewall for access to Swagger application or port forwarding.

Access npe3 port forwarding:

```
ssh rtcms@172.30.94.233  -L 9002:172.30.94.233:9002
http://localhost:9002/q/swagger-ui/
```

Separately in the **apidoc** directory, a swagger yaml, html and pdf file with a swagger description is attached.

2. RTCMS build and deploy from source

2.1. Build and copy RTCMS distributive

Go to project source directory: `cd vcard/implementation` or checkout from GIT

```
 quarkus build --clean --no-tests
 cd build
 zip quarkus-app.zip -r quarkus-app/*
 #
 # copy it to rtcms-test ( 172.30.94.233 )
 scp quarkus-app.zip rtcms@172.30.94.233:/home/rtcms/rtcms-wrapper/quarkus-app-
v1.48.zip
```

Optional if keystore or config file deploy:

```
    copy CaSys config file if necessary
    scp ../Installation/rtcms-wrapper/config/application.properties
rtcms@172.30.94.233:/home/rtcms/rtcms-wrapper/config/
    copy CaSys keystore file
    scp ../Installation/keystore/cTokenKey.p12 rtcms@172.30.94.233:/home/rtcms/rtcms-
wrapper/config/
```

2.1.1. Connect to RTMS test server and install new version

```
 ssh rtcms@172.30.94.233
 cd rtcms-wrapper
 rtcms.sh stop
 showlog.sh
 rm -rf quarkus-app
 unzip quarkus-app-v1.48.zip
 rtcms.sh start
 showlog.sh
```

2.1.2. Build Docker container (if RTCMS running in Docker container)

```
 quarkus build --clean --no-tests
 docker build -f src/main/docker/Dockerfile.jvm -t vcard .
 docker stop vcard
 docker rm vcard
 docker run -d -i --name vcard --hostname vcard \
      -p 9443:8443 -p 8002:9002 --network appnet  vcard
```

3. Production installation

Server name	IP addres	Discription
RTCMSP	172.30.94.231	Primery Server
RTCMSB	172.30.94.232	Backup Server

```
Linux pwd: P@55RTcm5!
Password not used for administration of RTCMS.
```

3.1. Initial installation to Production servers

3.1.1. Create ssh keys

Create in rtcms1 and rtcms2 and exchange it to login without ssh password.

```
  ssh rtcms@rtcms1
  ssh-keygen -t rsa -b 2048
  ssh-copy-id rtcms@rtcms2
  ssh-keygen -t rsa -b 2048
  ssh-copy-id rtcms@rtcms1
```

3.1.2. Initial copy and configs from test server

```
# Login to rtcms-test and copy public keys
ssh-keygen -t rsa -b 2048
ssh-copy-id rtcms@rtcms1
ssh-copy-id rtcms@rtcms2

# Copy installation files and envirmont to rtcms1 and rtcms2
scp .bash_profile rtcms@172.30.94.231://home/rtcms
scp .bash_profile rtcms@172.30.94.232://home/rtcms
scp jdk-17.0.9.tgz rtcms@172.30.94.231://home/rtcms
scp jdk-17.0.9.tgz rtcms@172.30.94.231://home/rtcms
scp rtcms-wrapper.tgz rtcms@172.30.94.231://home/rtcms
scp rtcms-wrapper.tgz rtcms@172.30.94.232://home/rtcms
scp .deltaspike rtcms@172.30.94.231://home/rtcms
scp .deltaspike rtcms@172.30.94.232://home/rtcms
scp .bash_profile rtcms@172.30.94.231://home/rtcms
scp .bash_profile rtcms@172.30.94.232://home/rtcms
```

3.1.3. Install to rtcms1

```
# Login to 172.30.94.231 and extract programs (jdk and rtcms):
   ssh rtcms@172.30.94.231
   tar xzvf jdk-17.0.9.tgz
   tar xzvf rtcms-wrapper.tgz
```

3.2. Install new version

After installation and test in rtcms-test server, the new version of RTCMS is copied and installed on the Production and Backup servers. Instalation of test server is described in section 2.1.1

The version for Primary and Backup servers is identical. The difference is only in the configuration in **appication.properties**. Running showlog.sh is to verify that the RTCMS service has started correctly.

```
#
# Copy distribution to RTCMSP server from test server
#
ssh rtcms@172.30.94.233
cd rtcms-wrapper
scp quarkus-app-v1.48.zip rtcms@172.30.94.231:/home/rtcms/rtcms-wrapper
#
# Login to RTCMSP and install new version
#
ssh rtcms@172.30.94.231
cd rtcms-wrapper
rtcms.sh stop
rm -rf quarkus-app
unzip quarkus-app-v1.48.zip
rtcms.sh start
showlog.sh
```

4. RTCMS Database schema

4.1. Connect with sql/plus to DB:

```
# connect to test DB
sqlplus rtcms/rtcms-test@//localhost:1521/RTCMST
#
# Connect to Production
ssh rtcms@172.30.94.231
sqlplus COM_RTCMS[RTCMS]/'COM_Rtcms1p!'@//localhost:1521/rtcmsp
#
# Connect to Backup
```

```
ssh rtcms@172.30.94.232
sqlplus COM_RTCMS[RTCMS]/'COM_Rtcms2b!'@//localhost:1521/rtcmsb
```

4.2. Create Oracle DB RTCMS user script

```
    -- DROP USER --
    drop user RTCMS CASCADE;
    -- USER SQL
    CREATE USER "RTCMS" IDENTIFIED BY "rtcms-test"
    DEFAULT TABLESPACE "USERS"
    TEMPORARY TABLESPACE "TEMP";
    -- QUOTAS
    ALTER USER "RTCMS" QUOTA UNLIMITED ON "USERS";
    -- ROLES
    GRANT "CONNECT" TO "RTCMS" WITH ADMIN OPTION;
    GRANT "RESOURCE" TO "RTCMS" WITH ADMIN OPTION;
    --ALTER USER "RTCMS" DEFAULT ROLE "CONNECT","RESOURCE";
    grant AQ_ADMINISTRATOR_ROLE to rtcms WITH ADMIN OPTION;
    -- SYSTEM PRIVILEGES
    GRANT CREATE VIEW TO "RTCMS" WITH ADMIN OPTION;
    grant SELECT ON SYS.DBA_RECYCLEBIN TO "RTCMS";
    GRANT EXECUTE ON dbms_aq TO rtcms;
    GRANT EXECUTE on DBMS_AQADM to rtcms ;
    GRANT EXECUTE on DBMS_AQJMS to rtcms ;
    GRANT EXECUTE ON dbms_aqin TO rtcms;
```

4.3. Main tables and RTCMS service reservation

Table 1. RTCMS DB маин tables containing PAN information:

Tablename	Notes
BIN_TABLE	Table **replicated** between Primary and Backup site
TOKEN_REGISTRATION	contains details about the registered tokens. Table **replicated** between Primary and Backup site
GCARD_REP	Table for replication to CCMS DB
TOKEN_QTAB	Advance Queue table for JMS messages with localD DB changes

Any change to BIN_TABLE and TOKEN_REGISTRATION resulting from an RTCMS API call is recorded as a JMS message in TOKEN_QTAB. *Backup* server read change messages from TOKEN_QTAB and applies them to its local RTCMS DB schema. Primary RTCMS server has similar behavior, reading changes from primary.

Both servers are configured to generate local change messages and apply generated messages from other server.

```
# Read changed record from remote queue
vcrd.casys.replicte.jms.runconsumer=true

# Generate changed record to local queue for other server
vcrd.casys.replicte.jms.runauditlog=true
```

4.4. Synchronization with CCMS

Each of the Primary and Backup servers periodically copy locally created vcard requests from local GCARD_REP to CCMS DB in table with the same name - GCARD_REP. After copying them, the GCARD_LOAD() procedure is started.

Records older than 30 days are deleted once a day.

The synchronization and deletion periods are set in **application.properties**.

```
# synchronization interval with CaSys CMS in sec
vcard.casys.ccms.synch=30s

# execute GCARD_LOAD()
vcard.casys.ccms.procExec=true

# run every day at 17h and 15min DELETE GCARD_REP old record < 30 days
vcard.casys.ccms.schedule=0+15+17+*+*+?
vcard.casys.ccms.delgcard=DELETE GCARD_REP where GCARD_REP.CREATE_DATE <
```

```
(CURRENT_TIMESTAMP - 30)
```

5. Token generation and protection of card information.

Table 2. Tables containing PAN information:

Tablename	Column	Protection
TOKEN_REGISTRATION	HSM_TOKEN	Encrypt PAN with AES256
	BANK_TOKEN	HMac-SHA256 (BANK_ID+BANK_TOKEN)
GCARD_REP	NCRD	Encrypt PAN with AES256

The PAN and HMac encryption keys are located in the cTokenKey.p12 keystore.

application.properties *the keys and encryption type example:*

```
vcard.casys.keystore=${user.dir}/Installation/keystore/cTokenKey.p12
vcard.casys.keystore.secret=$ENC(74ec7520478d1c861855b9d2879e5a461)

### HMAC-SHA512 Key - Bank token is hash of bank_id + encrypted pan
vcard.casys.keystore.hmackey=mtoken512

### AES key - encrypt PAN
vcard.casys.keystore.tokenkey=tokenkey256
vcard.casys.keystore.cparams=AES/ECB/PKCS5Padding

### DESEDE 192 bit key. If 3des used for pan encryption
#vcard.casys.keystore.tokenkey=tokenkey
#vcard.casys.keystore.cparams=TripleDES/ECB/PKCS5Padding
```

5.1. Защита на CVV2

CVV2 is not stored. It is calculated by the HSM on each request.

HSM config parameters in **application.properties***:*

```
 #-- HSM parameters --
vcard.casys.hsm.use=true
vcard.casys.hsm.ip=172.30.94.110
vcard.casys.hsm.port=1500
vcard.casys.hsm.timeout=5000
```

6. Encrypt parameters

6.1. Master password и master salt

A hash of ***Master password*** is used to encrypt parameter values in the **application.properties** config file. It is known only to the administrator who created it.

The hash of ***Master salt*** is used to encrypt the hash of ***Master password***, to store on disk along with the hashed hash of the *Master salt*.

Master salt is set in the applications and that’s it "the secret of the application". Can be a constant or dynamically determined during application execution. Initially it may be not known to the executable program. Such Master salt, for example, is a hash of the IP address of the VM, the user name under which it works and the application etc.

In the directory ~/.deltaspike is the file *master.hash* containing the master password and master salt. For more details see the link: DeltaSpike Crypto Mechanism.

Master password and ***Master salt*** are already generated in the current installation.

Major secrets are:

```
Master password is: CaSys_j2ee01
Master(application) salt is: rtcmsSystem
```

As a general rule, encrypted configuration parameters value is appear surrounded by "$ENC(...)". You can compute this values using the CLI command.

6.2. Encrypt parameters with cryptoParametes.sh

The cryptoParametes.sh command is used to generate master secrets and encrypt value of parameters.

An example of generate encrypted value of datasource password rtcms-test

```
cryptoParametes.sh encrypt -p -s
Enter text: ********
Enter master salt: ********
INFO  Encrypted secret is ecb6f2a4ad759d6a173bfbaf554a11c6
```

Example to setting encrypted password for datasource in **application.properties**.

```
quarkus.datasource.password=$ENC(ecb6f2a4ad759d6a173bfbaf554a11c6)
```

6.3. Details of command cryptoParametes.sh

Example display subcommands

```
cryptoParametes.sh -h

INFO  The application is starting. Version 1.0
Usage: cryptoParameters [-hV] Commands
  -h, --help      Show this help message and exit.
  -V, --version   Print version information and exit.

Commands:
  encode   Create and hash master password and master salt
  encrypt  Encrypt a secret with a master password
```

Example display subcommand parameters

```
cryptoParametes.sh encode -h

INFO  The application is starting. Version 1.0
Usage: cryptoParameters encode [-f] -p -s
Create and hash master password and master salt
  -f, --overwrite        Force overwrite
* -p, --masterPassword   create yourMasterPassword.
* -s, --masterSalt       create Application masterSalt.

cryptoParametes.sh encrypt -h
INFO  The application is starting. Version 1.0
Missing required options: '--masterSalt', '-plaintext'
Usage: cryptoParameters encrypt -p -s
Encrypt a secret with a master password
* -p, -plaintext     encrypt a secret with a previously stored masterPassword
* -s, --masterSalt   Application masterSalt.
```

6.3.1. Create master.hash file with Master password and Master salt

Example: Create master.hash file with hashed secrets:

```
cryptoParameters encode -p -s
.Enter master password: ********
.Enter master salt: ********
```

*Contents of ~/**.deltaspike/master.hash** contains the Master password encrypted hash and the Master salt hash*

`0a51281e6568a178df0f67238bf5f810220fd54c4e7a29cd2ae45fdf0c688162=
68997211f4bf661825d8284c2432c98b87ba22e487fec91be4ca6dffa7beb1b8d9831199a776ae23ff59d4fe0aa749b
56a9c774b6085f6fb0ffd2e6ad6ac3bc9b1247e7965526809481c38026f6b2ec7`