Telephony Capture Service

User Manual

Version 1.0.0



**Document Modification History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Editor** | **Date** | **Modification** |
| 1.0.0 | R Monk | 2016-01-01 | Original Release |
|  |  |  |  |
|  |  |  |  |

Table of Contents

1 Introduction 4

1.1 Purpose 4

1.2 Scope 4

1.3 Document Location 4

1.4 Document Status 4

1.5 Acronyms Definitions 4

1.6 References and Related Documents 4

1.7 Open Issues 4

2 Preliminaries 5

2.1 User Characteristics 5

2.2 Preparing the Environment 5

2.2.1 Software Platform 5

2.2.2 Environment Variables 5

2.2.3 TCS Version Numbers 5

2.2.4 tcsproj 5

3 Use Cases 7

3.1 Launching The TCS For The First Time 7

3.1.1 Mainstream TCS 7

3.1.2 TMS Simulator 7

3.1.3 PBX Simulator 7

3.1.4 Mangle 8

3.2 TCS Health Monitoring 8

3.2.1 Kitematic 8

3.2.2 RabbitMQ Management Console 9

3.2.3 Docker Logs 9

3.3 Rolling Back to a Previous TCS Version 9

3.4 TCS Software Upgrades 10

3.5 Postgres Container Management 10

3.5.1 Switch Postgres Container States 10

3.5.2 Point-In-Time Recovery 11

# Introduction

## Purpose

The purpose of this document is to provide specific instructions for how to carry out the various TCS use cases.

## Scope

This manual details how to start, stop, upgrade, and monitor the performance of the TCS. And because the TCS has the further responsibility to install and otherwise exploit a new database, it also includes instructions for how to restore the database from a backup set.

This document does not, however, include instructions for how to carry out any of the following:

* The various admin management functions associated with running a Linux installation.
* Instructions for installing git, docker, docker-compose, Kitematic.
* Instructions for how to modify the TCS database version, the queuing service version, nor the custom TCS software itself. There is a separate manual for such material, namely the TCS Development Manual.

## Document Location

This document is found in the ‘docs’ folder of the TCS GitHub repository:

<https://github.com/ccbcadmin/telephony-capture-service.git>

## Document Status

This document must be kept current and released concurrently with each software release.

## Acronyms Definitions

The reader is referred to the TCS SRD.

## References and Related Documents

The reader is referred to the TCS SRD.

## Open Issues

1. Database restore.

# Preliminaries

## User Characteristics

The following user capabilities are assumed in this document:

* Generally familiar with the Linux command line capabilities and able to navigate the Linux directory structure.
* It is helpful, but not necessary, to have a general understanding of [git](https://git-scm.com/documentation).
* Again, it is helpful, but not necessary to be familiar with [docker](https://docs.docker.com/) (docker is heavily exploited by the TCS, however, as far as this manual is concerned, docker’s capabilities have been captured in Linux scripts, aliases, and functions).

## Preparing the Environment

### Software Platform

The assumed platform is the following:

* Ubuntu 16.04.1
* Docker 1.2.3
* Docker-compose 1.9.0
* Kitematic 0.12.9 or later
* Git 2.7.4 or later

### Environment Variables

A number of TCS-specific environment variables must be exported in the following file:

**~/.tsc.bash**

As this file contains confidential information, it is not maintained in GitHub. Environment variables are defined using standard export syntax. For example:

**export BACKUP\_PURGE\_PERIOD\_UNITS=days**

The SRD contains detailed descriptions of the various variables. Failure to set all the environment variables to appropriate values will certainly result in the loss of one or more TCS features.

### TCS Version Numbers

TCS Version numbers are of the following form: vx.y, where v is a literal ‘v’ and both x and y are non-negative integers (e.g “v3.2” meaning version 3.2).

### tcsproj

All TCS activity is launched from the folder ~/tcs. Once there the following command must be executed:

**$ source .project.bash [tcs\_version]**

Where tcs\_version is optionally provided; if omitted, TCS Version remains unmodified.

In order to expedite this process, it is recommended to define the following bash function in the .bashrc file:

**tcsproj () { cd ~/TCS; source .project.bash; }**

Thereafter, the user can conveniently prepare for TCS commanding with the following:

**$ tcsproj [TCS Version]**

# Use Cases

## Launching The TCS For The First Time

### Mainstream TCS

It is assumed that the TCS environment variables have been configured correctly – see Section 2). In the following, it is assumed that the first install is to begin with a version 1.0 (although in reality this may not be so).

**$ cd ~**

**$ git clone https://github.com/ccbcadmin/telephony-capture-service tcs 1**

**$ cd tcs**

**$ source .project.bash 2**

**$ git checkout tags/v1.0 -b v1.0 3**

**$ tcs-up 4**

1. Clones the tcs GitHub repository into the tcs folder.
2. Defines a number of tcs-specific aliases, as well exporting the environment variables defined in the file ~/.tsc.bash (see section 2).
3. Checks out the tcs version as it existed in version 1.0.
4. Starts the main tcs containers. These are:

* pbx-interface
* tms-interface
* database-interface
* tcs-postgres
* rabbitmq
* backup-scheduler

The command **tcs-up** downloads from the Docker Hub the TCS image that corresponds to the required version number.

### TMS Simulator

During system / acceptance testing it will be necessary to have a test sink for data destined for the TMS (otherwise the queues could grow without limit, if there is an input PBX source).  This is the purpose of the TMS Simulator. It can be brought up as follows:

**$ tms-simulator-up**

### PBX Simulator

Thus far all of the containers mentioned run in the background, that is, once started they are detached from the session that started them.  This is not the case for the next two.  These run in the foreground, that is, they do not release the session until completion.

The first is the PBS Simulator and its usage is as follows:

**$ pbx-simulator source-smdr-directory**

or as an example:

**$ pbx-simulator /smdr-data/smdr-data-002**

This simulator opens a circuit to the container pbx-interface and sends SMDR messages in chronological order drawn from the data in the specified directory.

### Mangle

This tool takes as input a folder containing one or more raw SMDR files and randomizes the last 4 digits of all unknown phone numbers and records the resulting transformed records into files to be found in the output folder.

**$ mangle source-smdr-directory target-smdr-directory**

or as an example:

**$ mangle /smdr-data/smdr-data-002 /smdr-data/smdr-data-003**

## TCS Health Monitoring

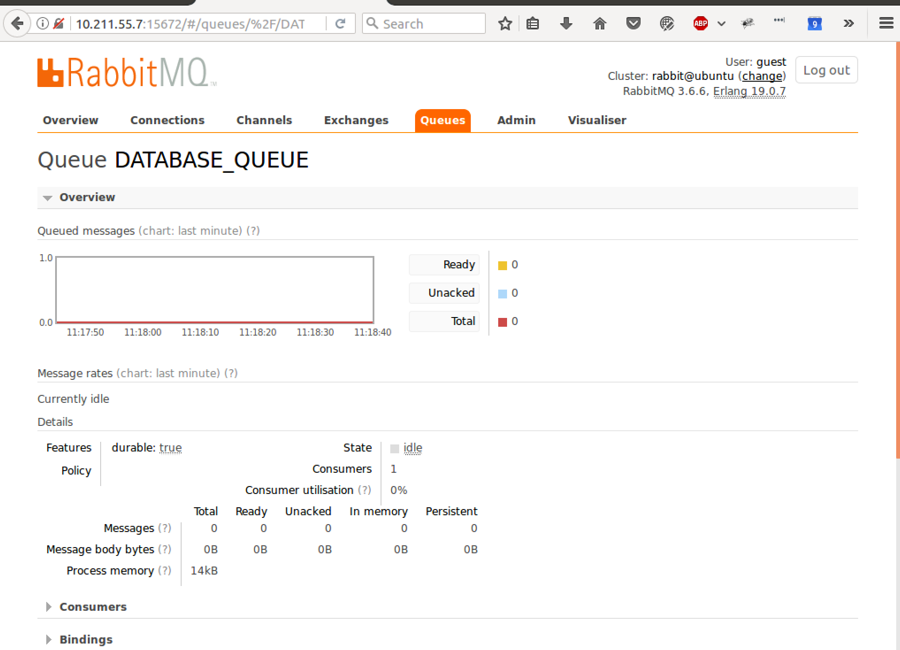
This section provides information for how to monitor and otherwise check that the TCS is performing properly.

### Kitematic

Kitematic is ideal for providing a visual confirmation that all the TCS services are running (Kitematic also provides a means to Stop, Start, and Restart individual services).

### RabbitMQ Management Console

Provides a wealth of information about the internal state of RabbitMQ. The most important one in this context is the ability to view how many messages are in the queues (nominally the queue size should be more or less 0 if everything is working properly). Note: the user name / password is ‘guest’/’guest’.



### Docker Logs

The various log files produced by the TCS containers can be retrieved by using the **$ docker logs** command. Its usage is available as follows:

**$ docker logs –help**

A particular usage is the following (all the log entries produced by the pbx-interface container prefixed with timestamps):

**$ docker logs –t pbx-interface**

## Rolling Back to a Previous TCS Version

Assume that v1.3 is currently running. The following commanding switches the TCS to version v1.2.

**$ tcsproj**

**$ tcs-down 1**

**$ git checkout tags/v1.2 -b v1.2 2**

**$ tcs-up 3**

1. Force a shutdown of all containers.
2. Set back the tcs environment to use version v1.2 of the software.
3. Before executing tcs-up, TCS environment variables should be reviewed in the TCS Software Requirements Document. Required environment variables can be added and / or changed from one version to the next.

The image for the rollback version may still be available locally, in which case no pull from the Docker Hub will be required and hence the TCS processing will be restored that much faster.

## TCS Software Upgrades

Upgrading the TCS software is similar to doing a rollback, except that the new version must first be acquired from GitHub. The following assumes that the TCS is to be upgraded to version 1.4. Carry out the following:

**$ tcsproj**

**$ git pull 1**

**$ tcs-down**

**$ git checkout tags/v1.4 -b v1.4 2**

**$ tcs-up 3**

1. Retrieves the very latest TCS software from GitHub.
2. Directs git to set the TCS environment to version 1.4.
3. Before executing tcs-up, TCS environment variables should be reviewed in the TCS Software Requirements Document. Required environment variables can be added and / or changed from one version to the next.

## Postgres Container Management

The TCS runs with 2 Postgres containers, Postgres1 and Postgres2. Nominally one of these containers supports the mission operationally and the other is configured in a cold standby state. This section provides the following:

* Instructions to switch the states of these containers, so that the currently operational Postgres database is reconfigured to the standby state and visa-versa.
* Instructions to do a Point-In-Time Recovery on the standby container.

### Switch Postgres Container States - Routine

Good practice says to actually exercise available recovery processes, even if there is no pressing need to do so. The intent being that if and when the day comes that there is a genuine need for a recovery procedure to be carried out that the procedures do in fact work.

Note that the following process renders both databases unavailable for a short period, however, this is accepted in that no data loss will incur due to the buffering and retry capabilities of other TCS containers which do require access to the database.

Two procedures need to be carried out: A controlled shutdown of the operational Postgres container followed by a procedure to reconfigure the currently standby container to an operational state.

**Shutdown of the Operational Postgres Container**

* Stop the inflow to the database, do a final manual database backup, and then stop the operational Postgres container altogether.

**$ cd ~/tcs**

**$ docker stop database-interface 1**

**$ barman 2**

**$ barman backup pg1**

**$ exit**

**$ docker update –restart no postgres1 3**

**$ docker stop postgres1 4**

1. Stop the flow of data to the database.
2. Opens a shell in the barman container and do a final manual backup.
3. Ensure that this container will not be automatically restarted at boot time.
4. Shutdown the currently operational instance of Postgres.

**Starting the Standby Postgres Container**

**$ barman 1**

**$ barman recover pg1 latest /pg2\_data**

**$ exit**

**$ docker update –restart unless-stopped postgres2 3**

**$ docker start postgres2**

1. **Use barman to recover the postgres1 database to the postgres2 container.**
2. **Reconfigure the postgres2 container so that it is automatically restarted in the case of a reboot.**
3. **Start the Postgres in the PG2 container to use the nominal Postgres port 5432.**

### Point-In-Time Recovery

There may come a time when it would be useful to investigate the state of the database at some point in the past. This section provides the instructions for how to do this. Preliminary considerations are the following:

* Nominally only one of the Postgres containers is running with the other stopped. After doing a PITR, an historical version of the database is recovered into the stopped Postgres container, with a further follow-up step in which the stopped container itself is set into the running state, hence making the PITR database available.
* Both Postgres containers cannot listen on the same port (the nominal Postgres listening port is 5432). The consequence is that the Postgres container that contains the PITR database listens instead on port 5433.
* Hence, a user wishing to do a PITR database investigation, must remember to configure said tools to use port 5433 (‘Postgres-aware’ software tools invariably default the port selection to 5432).
* During the PITR investigation, normal database activities continue unawares.
* The consequence of a PITR investigation may be that the user wants to modify the operational database in some way (for example, to recover a table that should not have deleted). Such actions are doable, but information to do such things are outside the scope of this manual. The user is advised to refer to one of the many excellent Postgres references, the prime one being the online [Postgres documentation](https://www.postgresql.org/docs/9.6/static/index.html) itself.
* Once the PITR investigation and any consequential activities are complete, the PITR database container should be set back into the stopped state, which can be done from Kitematic or the ‘$ docker stop’ command.