Telephony Capture Service

User Manual

Version



**Document Modification History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Editor** | **Date** | **Modification** |
| 1.0.0 | R Monk | 2017-01-01 | 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 6

2.2.4 tcsproj 6

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 Management 10

3.5.1 Switch Postgres Containers 10

3.5.2 Rollback Recovery 11

3.5.3 Standby Recovery 11

4 Useful Command Line Utilities 13

# 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

It is assumed that the user is generally familiar with Linux command line utilities and is able to navigate the Linux directory structure.

Beyond that having some further familiarity of any or all of the following is an advantage:

|  |  |
| --- | --- |
| [git](https://git-scm.com/documentation) | Used to manage TCS versions. |
| [docker](https://docs.docker.com/) | docker and a closely aligned product, docker-compose, are heavily exploited by the TCS. However, as far as this manual is concerned, docker’s capabilities have been largely captured in Linux scripts, aliases, and functions. |
| [Postgres](https://www.postgresql.org/docs/9.6/static/index.html) | The more that the user knows about Postgres the better, nevertheless, the TCS user can effectively use the TCS without Postgres knowledge. |
| [barman](http://www.pgbarman.org/) | Barman is an open source product designed to manage Postgres backups and Postgres Write Ahead Logs. Working knowledge of barman is useful, but not necessary. |

## Preparing the Environment

### Software Platform

The assumed platform is the following:

* Ubuntu 16.04.1
* Docker 1.12.5
* 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 TMS\_HOST=192.168.3.43**

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 capabilities.

### TCS Version Numbers

TCS Version numbers are of the 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.

**$ cd ~**

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

**$ tcsproj v1.0**

**$ git checkout tags/v1.0 -b v1.0 2**

**$ tcs [pg1 | pg2] 3**

1. Clones the tcs GitHub repository into the tcs folder.
2. Checks out the tcs version as it existed in version 1.0.
3. This command will create the main TCS containers and then configure them into the running state. Note that the user must select which Postgres container will be used to support the application, either pg1 or pg2.

* pbx-interface
* tms-interface
* database-interface
* pg1 or pg2
* rabbitmq
* barman

If not already locally available, the **tcs** command 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 sink for data destined for the TMS (otherwise the TMS queue would 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**

### 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

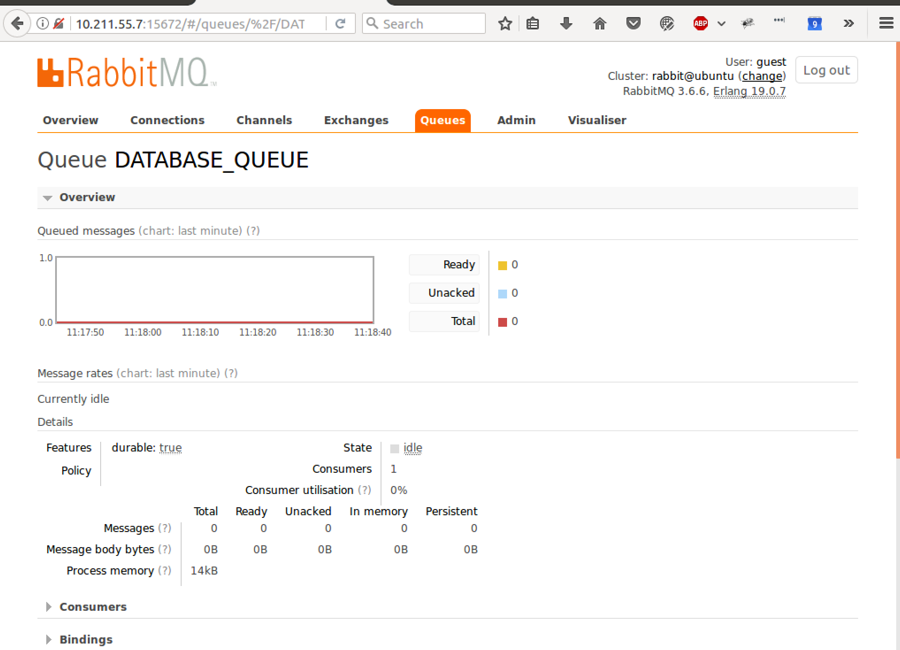
This section provides information for how to monitor and otherwise check that the TCS is performing properly.

### 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 [pg1 | pg2] 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 the tcs command, the 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 retrieved 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 [pg1 | pg2] 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, TCS environment variables should be reviewed. Required environment variables can be added and / or changed from one version to the next.

## Postgres Management

The TCS runs with 2 Postgres containers, pg1 and pg2. This section provides the following:

* Instructions to switch the active Postgres container to the other.
* Instructions to do a Point-In-Time Recovery to the active Postgres container.
* Instructions to do a Point-In-Time Recovery to the standby Postgres container. Such a database recovery can be used, for example, to investigate the state the database at some previous point in time without disturbing the TCS.

### Switch Postgres Containers

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 acceptable in that no data loss will incur due to the buffering and retry capabilities of the other TCS containers.

The following command does the switch:

**$ pg-switch**

The following bullets summarize the actions of this command:

* A final full backup is taken of the active database.
* The source of database changes is suspended.
* The active Postgres container is removed.
* A backup recovery procedure is carried out into the standby Postgres atabase area using the latest backup. This container is then reconfigured to be active.
* The source of database changes is restored.

### Rollback Recovery

Rollback recovery allows a point-in-time recovery to the currently active Postgres container. Once the recovery is complete, normal TCS operations are restored. Some measure of data loss is likely.

Before proceeding, the user will need the following information:

* There will be a history of both pg1 and pg2 backups, so the user will need to be aware of which Postgres container’s backups to use for the rollback. The list of available backup\_ids is provided by executing the following command:

**$ list-backups**

* If the rollback is to be up to a specific time, then they will need the target time expressed in ISO format, e.g. “2016-12-15T12:00:00”. Target time is optional.
* The barman Backup Id for the latest backup taken BEFORE the target time.

The user can now proceed with the recovery as follows:

**$ pg-rollback-recovery pg1|pg2 backup\_id [target\_time]**

Note: The recovery will be to the currently active Postgres container, which may or may not be the source of the backup.

### Standby Recovery

There may come a time when it would be useful to investigate the state of the database at some point in the past. Standby recovery is very similar to a rollback recovery except that the target of the recovery is to the Postgres container that is not currently supporting the application. This section provides the instructions for how to do this. Preliminary considerations are the following:

* A Standby Recovery is applied to the standby Postgres container, hence making an historical database accessible.
* The two Postgres containers cannot listen on the same port simultaneously (the nominal Postgres listening port is 5432). The consequence is that the standby Postgres container listens instead on port 5433. To access the standby container, the user must remember to configure tools to use port 5433 (‘Postgres-aware’ tools invariably default the port selection to 5432).
* The consequence of an investigation of the state of the standby database could be, for example, a need to recover a table that should not have deleted. Such actions are quite doable, nevertheless, the details are outside the scope of this manual. The reader is advised to refer to one of the many excellent Postgres references, starting with the excellent online [Postgres documentation](https://www.postgresql.org/docs/9.6/static/index.html) itself.

Carrying out a standby recovery is very similar to a rollback recovery:

* There likely will be a history of both pg1 and pg2 backups, so the user will need to be aware of which Postgres container’s backups to use for the point-in-time recovery.
* The target time for the recovery, which is expressed in ISO format, e.g. “2016-12-15T12:00:00”. This parameter is optional.
* The barman Backup Id for the latest backup taken BEFORE the target time. This can be found by executing the following command:

**$ list-backups**

The user can now proceed with the recovery as follows:

**$ pg-standby-recovery pg1|pg2 backup\_id [target\_time]**

A final note: the standby Postgres container’s database is not backed up.

# Useful Command Line Utilities

$ psql1

$ psql2