Telephony Capture Service

User Manual

Version 1.0.1



**Document Modification History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Editor** | **Date** | **Modification** |
| 1.0.0 | R Monk | 2017-01-03 | Original Release |
| 1.0.1 | R Monk | 2017-01-25 | Review Modifications  Context Diagram |

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 Reference 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 TCS Environments 7

3.1 Common Containers 7

3.2 Development Environment 7

3.3 QA Environment 8

3.4 Production Environment 9

4 Use Cases 10

4.1 Launching The TCS For The First Time 10

4.1.1 Mainstream TCS 10

4.1.2 TMS Simulator 10

4.1.3 PBX Simulator 10

4.1.4 Mangle 11

4.2 TCS Health Monitoring 11

4.2.1 Kitematic 11

4.2.2 RabbitMQ Management Console 11

4.2.3 Docker Logs 12

4.3 Changing TCS Version 12

4.4 Postgres Management 13

4.4.1 Switch Postgres Containers 13

4.4.2 Rollback Recovery 14

4.4.3 Standby Recovery 14

5 Command Line Tools 16

# Introduction

## Purpose

The purpose of this document is to provide 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 server database, it also includes instructions for how to restore the database from a backup set.

This document does not include:

* Instructions to carry out 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 Software Requirements Document](TCS Software Requirements Document.docx)

## Reference and Related Documents

* [TCS Software Requirements Document](TCS Software Requirements Document.docx)
* [TCS Developer Manual](TCS Developer Manual.docx)

## Open Issues

* Add datastores for rabbitmq, pg1 / pg2, barman, and Jenkins to the various diagrams.
* The discussion on the 3 TCS operating environments possibly should be moved to the TCS Developer Manual.

# 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 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 essential. |

## Preparing the Environment

### Software Platform

The assumed platform is the following:

* Ubuntu 16.04.1
* Docker 1.13.0
* Docker-compose 1.9.0
* Kitematic 0.12.9 or later
* Git 2.7.4 or later

### Environment Variables

TCS-specific environment variables are defined in the following file:

**~/tcs/env\_PROD/env**

The syntax of this file is that of defining a typical bash environment variable. For example:

**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 scripts/project [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 function in the .bashrc file:

**tcsproj () { cd ~/tcs; source scripts/project; }**

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

**$ tcsproj [TCS Version]**

# TCS Environments

The TCS software supports 3 different environments: Development, QA, and Production. These environments can co-exist, although the most typical usage would be for only one environment to be active at any given time (e.g. the Development environment is typically used by off-site personnel, whereas the Production environment runs exclusively on-site). As will be seen, though, the QA and Production Environments are simultaneously active during installation of new software. This section discusses each of the environments in more detail.

## Common Containers

Practical considerations mandate that the queuing service (RabbitMQ) and the database service (Postgres) not be replicated in each environment for those situations where the environments are coincidently active. Nevertheless, it is prudent to isolate the environments to the maximum extent possible. This section discusses how this is achieved.

Each TCS environment requires 2 queues, for a total of 6 different queues. Therefore, the queuing service is configured to support 6 queues as follows. All queues are independent, are isolated from each other, and are fed from different sources.

* DEV\_TMS\_QUEUE
* DEV\_DB\_QUEUE
* QA\_TMS\_QUEUE
* QA\_DB\_QUEUE
* PROD\_TMS\_QUEUE
* PROD\_DB\_QUEUE

Similarly, Postgres instances are configured to isolate the 3 environments using the Postgres ‘database’ concept (e.g. a Postgres instance can simultaneously support multiple databases). The 3 Postgres databases are named ‘dev’, ‘qa’, and ‘prod’.

## Development Environment

The Development Environment is discussed in the TCS Developer Manual.

## QA Environment

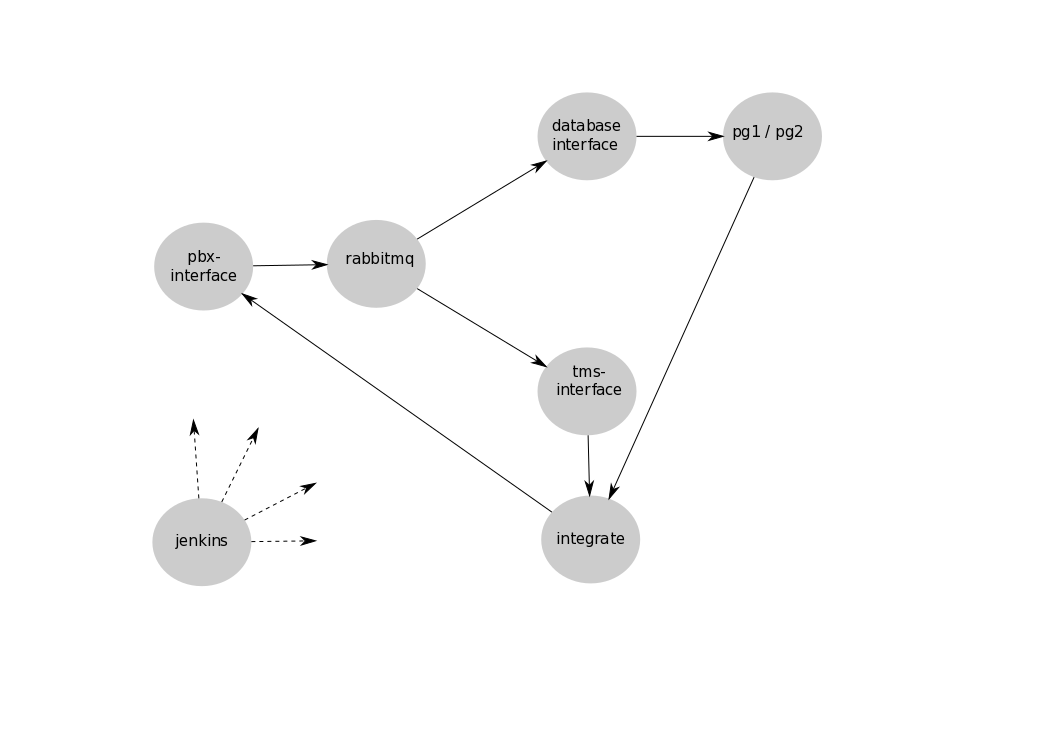
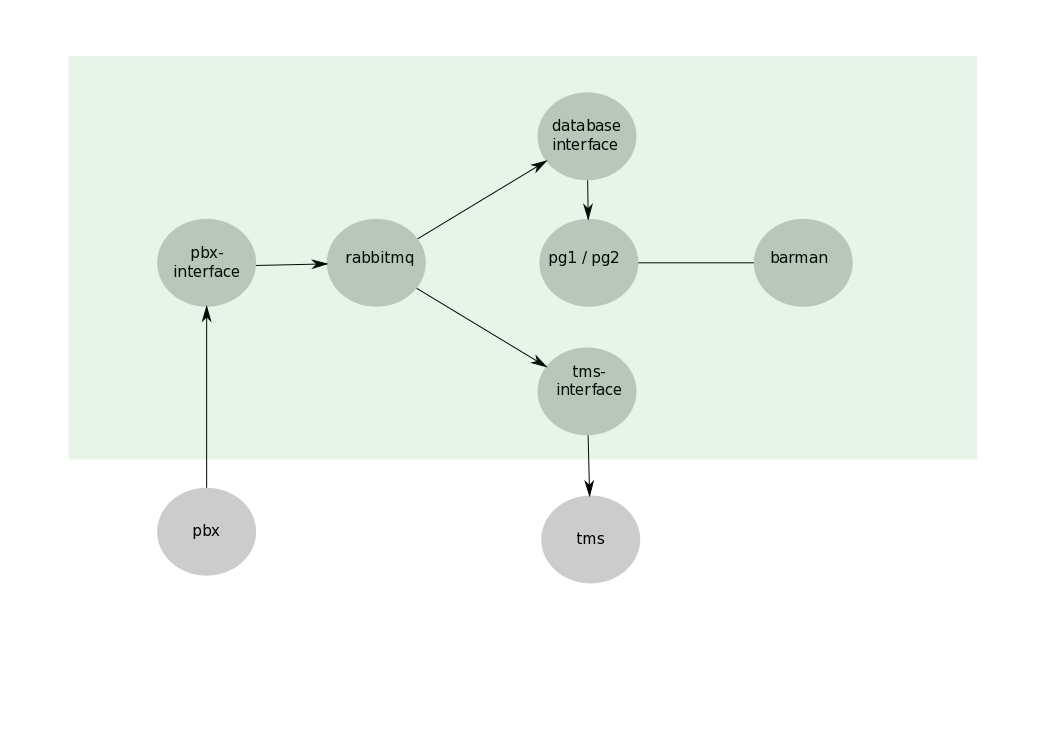


Figure 1: TCS QA Environment

Running TCS software in the QA Environment assures that TCS software is fit for purpose. The usual workflow is that Development announces the availability of a new software release and the customer takes delivery of the software using features that are discussed later in this document. Part of the delivery effort is to first ensure that the new software actually performs correctly in the QA environment. A few observations:

* jenkins orchestrates the activities of the other containers (jenkins can shutdown containers, restart them, swap in and out the two Postgres containers, etc.).
* A new QA-specific container, integrate, is used as a vehicle to execute a number of test routines. Although integrate is quite flexible, its responsibilities come down to some combination of the following:
  1. Send something to pbx-interface;
  2. Verify messaging coming from tms-interface; and / or
  3. Verify that the content of the database is as expected.
* jenkins monitors the output of integrate and if all tests succeed, then the new software is declared fit for purchase.

## Production Environment



The production environment ingests data from the actual PBX via pbx-interface which in turn passes the data on to two distinct RabbitMQ queues. Two other containers then draw data from the queues (one per queue); these are database-interface and tms-interface. database-interface inserts all incoming SMDR records into the active Postgres container, pg1 or pg2. Meanwhile, the tms-interface directs all data that it pulls from its queue to the TMS.

Note that the Postgres container into which the SMDR records are ingested are actually 2 such, pg1 and pg2, but only one of these are operationally active at any given time. At any given time the Standby Postgres container can be in one of three different states:

* It may not exist at all;
* It may exist but not be running;
* It may be running. In this latter case, typically a database restore was done to the standby Postgres container (such a restore would be useful when investigating some historical database anomaly).

Last but not least, the barman container provides the following services:

* In real-time, it receives and logs a database replication stream from the currently active Postgres container.
* It triggers backups of the currently active Postgres container according to user-defined scheduling needs.
* It purges backups according to the user-defined purge policy.

# 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 version 1.0 (v1.0).

**$ cd ~**

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

**$ tcsproj v1.0**

**$ stores pg1 | pg2 2**

**$ tcs 3**

1. Clones the tcs GitHub repository into the tcs folder.
2. This command will create the following common TCS containers (these containers are common to all 3 environments). Note that the user must select which Postgres container will be used to support the application. Kitematic will show the following containers.

* pg1 or pg2
* rabbitmq
* barman
* jenkins

1. Creates the following Production-specific containers named as follows by Kitematic.

* PROD-pbx-interface
* PROD-tms-interface
* PROD-database-interface

### 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**

The corresponding container name will be shown as PROD-pbx-simulator by Kitematic.

### PBX Simulator

During pre-operational phases of the project, it may be desirable to be able to send the TCS an artificial stream of SMDR messages.  This stream can be created by executing the following command:

**$ pbx-simulator**

The corresponding container name will be shown as PROD-pbx-simulator by Kitematic.

### 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:

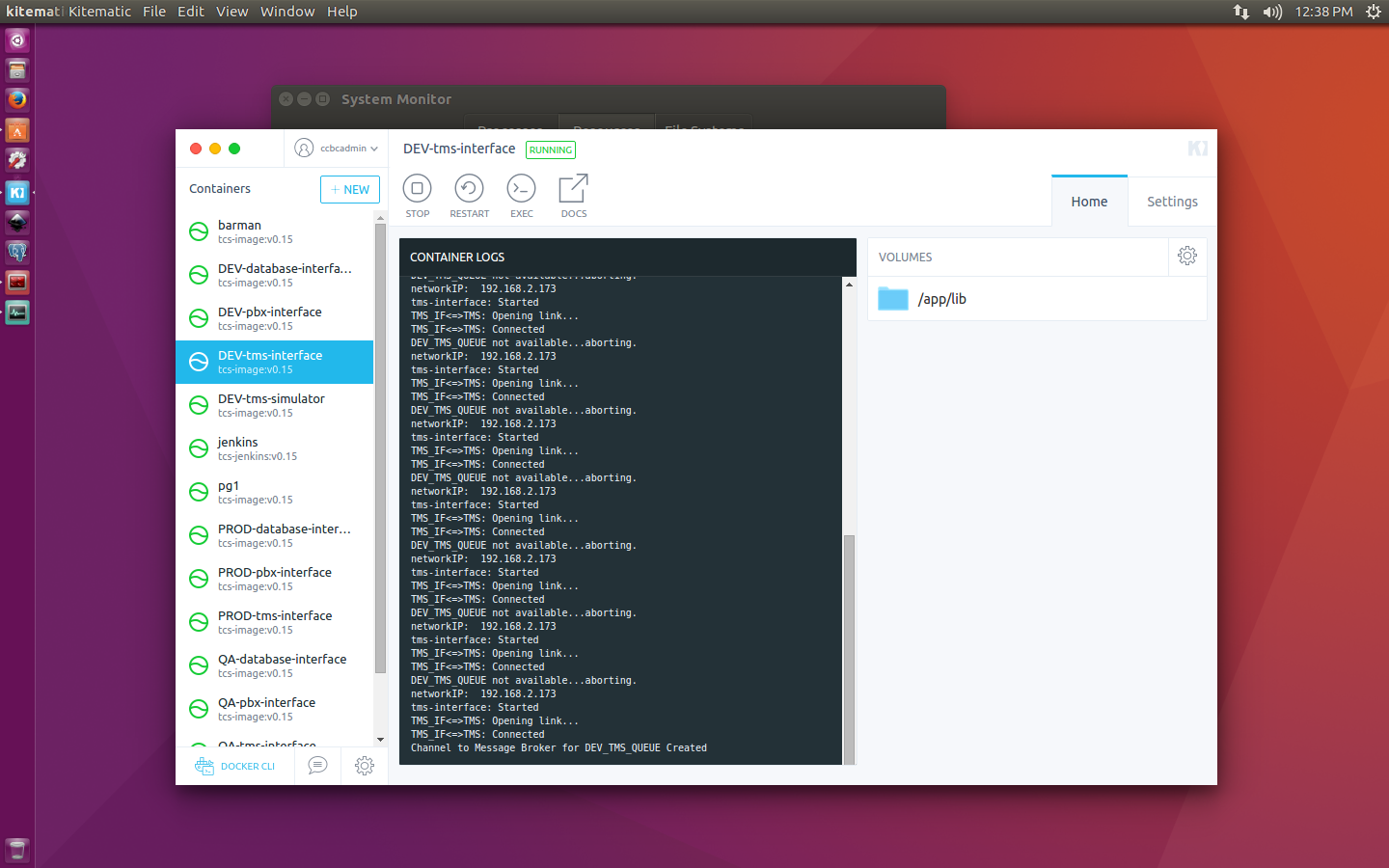
**$ mangle /smdr-data/smdr-data-002 /smdr-data/smdr-data-003**

## TCS Health Monitoring

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

### Kitematic

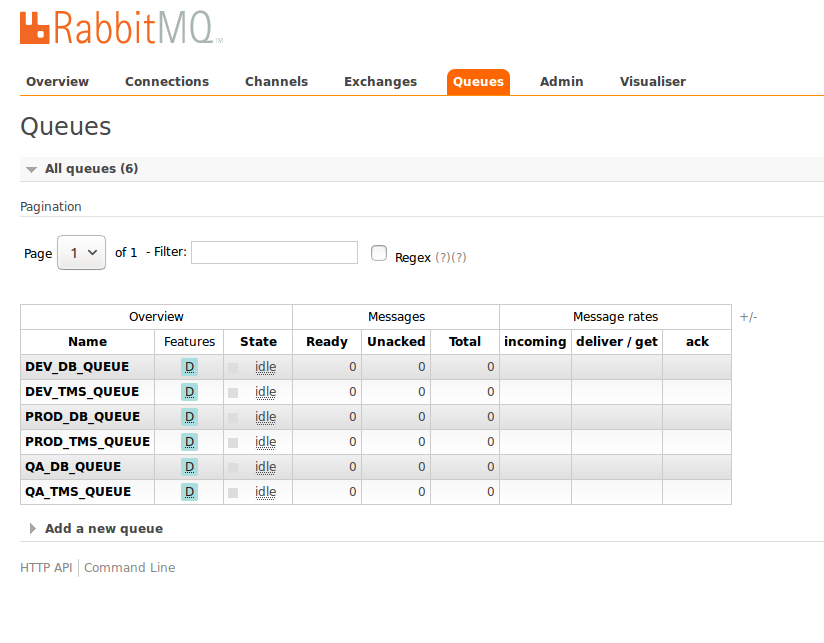
Kitematic is ideal for providing a visual confirmation that all the TCS containers are running (Kitematic also provides a means to Stop, Start, and Restart individual services).



### 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’.

ToDo: change defaults.



### 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**

## Changing TCS Version

Assume that v1.3 is currently running. The following commanding switches the TCS to version v1.4.

*Before proceeding, the TCS environment variables should be reviewed. Environment variables can be added and / or modified from one version to the next.*

**$ tcsproj v1.4 1**

**$ tcs down 2**

**$ stores down 3**

**$ stores pg1|pg2 4**

**$ tcs 5**

1. Change the version. This command triggers the following actions:
   1. A Git pull request to GitHub to ensure that the local Git repository is up-to-date;
   2. A Git checkout to branch v1.4.
   3. A pull request to Docker Hub to ensure that version v1.4 of the TCS image is available locally.
   4. The new version number is recorded locally.
2. Shutdown the Production-specific containers.
3. Shutdown all the Stores containers.
4. Restart the Stores containers (this time using version v1.4 software). Usually the Postgres container selection, pg1 or pg2, will not have changed.
5. Restart the Production-specific containers (this time using version v1.5 software).

## 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 standby container.
* 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 to the standby can be used to investigate the state of the database at some previous point in time without disturbing the TCS.

Before proceeding, the reader is advised to review the following table that summarizes valid Postgres container states.

|  |  |  |
| --- | --- | --- |
| **TCS State** | **Docker State** | **Listening Port** |
| active | running | 5432 |
| standby | running | 5433 |
| standby | stopped | - |
| standby | gone | - |

A few comments are in order:

* It is completely valid for a standby Postgres container to not be running in the sense of a Docker container running. In fact, a standby Postgres container may not even exist (gone) in the Docker sense.
* If a Postgres container is standby / running, such a container is likely being used to investigate some historical state of the database. A Postgres container that is in this state does not interfere or otherwise impact the TCS application.
* The subject of the Listening Port is discussed subsequently.

### 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 provides the switch:

**$ pg-switch**

The following summarizes the consequences 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 database area using the latest backup.
* The standby container is then reconfigured to be the active container.
* The source of database changes is restored.

### Rollback Recovery

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

Before proceeding, the user will need the following:

* The user will need to be aware of which Postgres container’s backups are available. The list of available backup\_ids is provided by executing the following command:

**$ list-backups**

The barman backup\_id must be the latest backup taken BEFORE the target time.

* If the rollback is to be up to a specific time, then the target time is required and is expressed in ISO format, e.g. “2016-12-15T12:00:00”. Target time is optional.

Armed with this information, the user can now proceed with the recovery:

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

* Note: The recovery will be to the active container, which may or may not be the source of the selected 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 standby Postgres container. Preliminary considerations are the following:

* 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 been 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 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. Please refer to the previous section for instructions on how to identify the appropriate backup\_id and target time (the latter is optional).

The user can now proceed with the recovery as follows:

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

Final notes:

* The recovery will be to the current standby container, which may or may not be the source of the selected backup.
* The standby Postgres container’s database is not backed up.

# Command Line Tools

The following command line tools are not essential to run or administer the TCS, nevertheless, they will likely prove to be useful.

**$ barman-exec**

Opens an interactive shell to the barman container. This allows full access to the barman command line interpreter.

**$ pg1-exec**

Opens an interactive shell to the pg1 container.

**$ pg2-exec**

Opens an interactive shell to the pg2 container.

**$ psql1**Attempts to open an interactive [psql](http://postgresguide.com/utilities/psql.html) terminal to the pg1 container. If successful, the greeting indicates whether pg1 is the active or standby container.

If not successful in opening an interactive terminal, the command will terminate as follows:

1. If the pg1 container exists: pg1 STOPPED
2. If the pg1 container does not exist: pg1 GONE

**$ psql2**

See the discussion for the $ psql1 command.

**$ stores pg1 | pg2 | down**

If pg1 or pg2 is selected starts all of the stores rabbitmq, jenkins, barman, the selected Postgres container. If down is selected, then all the stores containers are stopped and removed.

**$ tcs [down]**

This command allows the user to either start or shutdown environment-specific containers. See also **$ tcsenv**.

**$ tcsenv [dev | qa | prod ]**

If no environment is provided, then the current environment is displayed. If a valid environment selection is provided, then the environment is switched to the user’s selection. Once changed, then subsequent use of the $ tcs command will apply to that environment. Given that the default environment is ‘prod’, there should be limited need for this command.