Telephony Capture Service

Developer Manual

Version



**Document Modification History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Editor** | **Date** | **Modification** |
| 1.0.0 | R Monk | 2017-02-01 | 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 TCS Environments 5

2.1 Common Containers 5

2.2 Development Environment 5

2.3 QA Environment 6

2.4 Production Environment 7

3 Software Release Workflow 8

4 Command Line Tools 9

# Introduction

## Purpose

The purpose does a few things:

* Explains and in some cases justifies the various TCS software design decisions.
* Provides instructions for how to do a TCS software release.

## Scope

This document does not:

* Provide the TCS software requirements. The reader can find these in the TCS Software Requirements Document (SRD).
* It is also not a user manual. That material can be found in the document TCS User 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

None

# 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

ToDo

## QA Environment

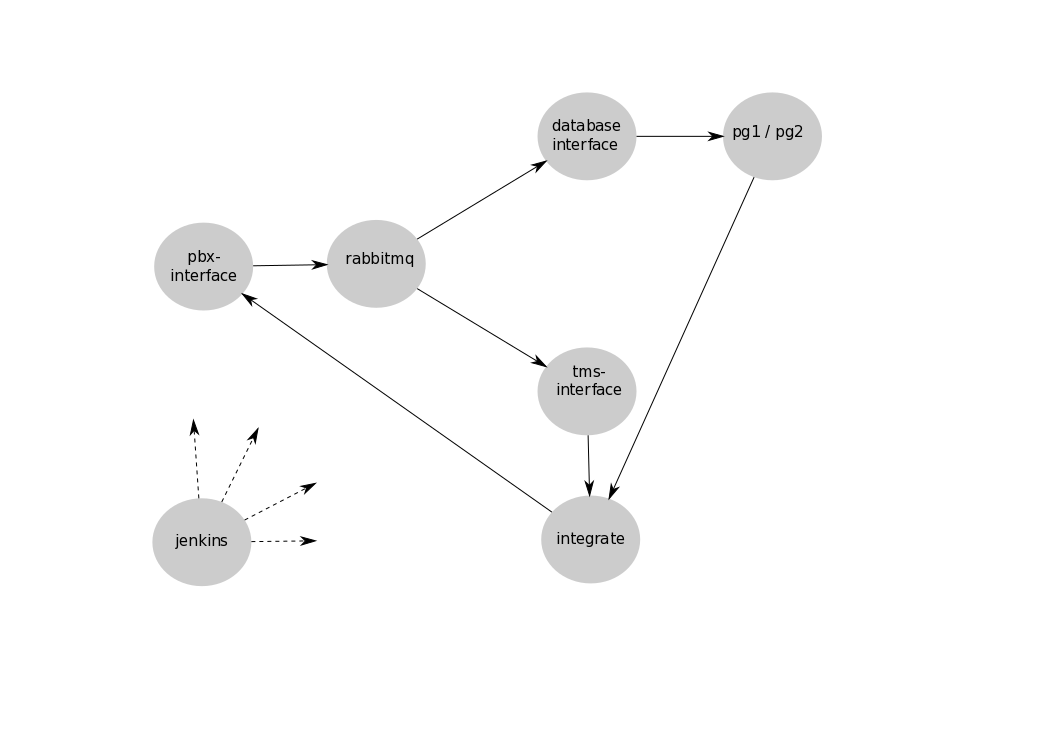
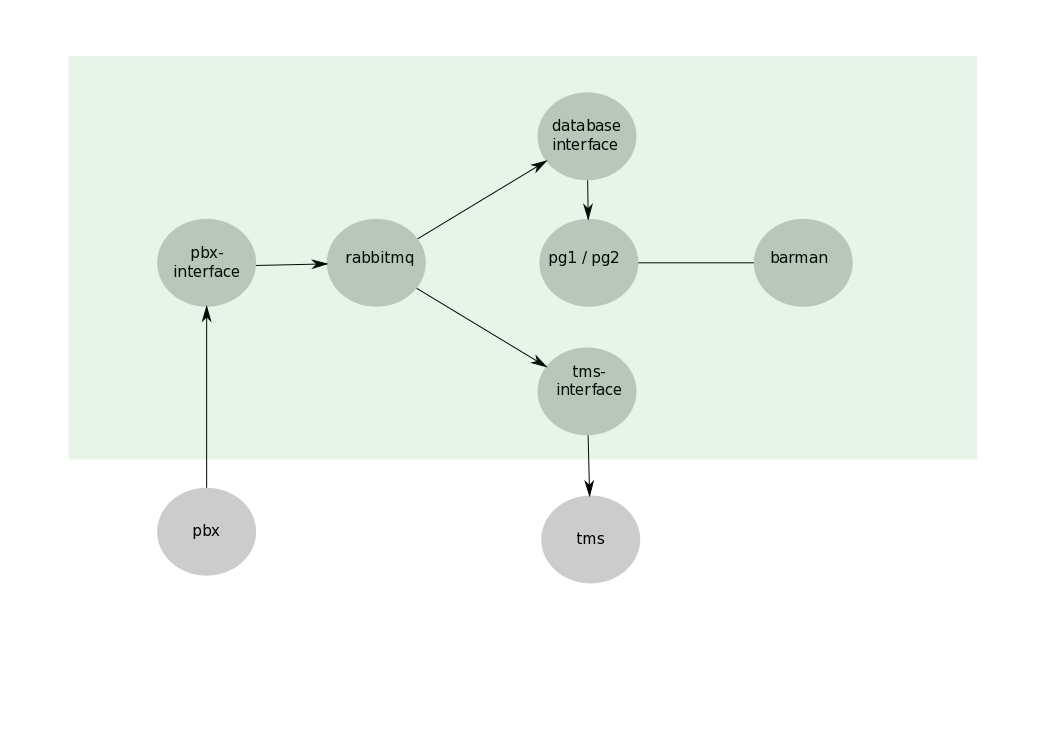


Figure : TCS QA Environment

Running TCS software in the QA Environment provides assurance that software deliveries are 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.
* 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, this other one may not exist at all, it may exist but not be running, but it could also be running in a standby state. In the 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:

* 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.

# Software Release Workflow

# 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.