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 Operating Environments 5

2.1 Common Containers 5

2.2 Development Environment 5

2.3 QA Environment 6

2.4 Production Environment 7

3 Docker 8

3.1 Docker Images 8

3.2 Docker Project: stores 8

3.3 Docker Project: prod 8

3.4 Docker Project: qa 9

3.5 Docker Project: dev 9

4 Software Release Workflow 10

4.1 Define the New Version Number 10

4.2 Updating the Documents 10

4.3 Coding in the Development Environment 11

4.4 Building Docker Images 11

4.5 Testing in the QA Environment 11

4.6 Fixing Problems Found in the QA Environment 11

4.7 Announcing the Software Release to the Customer 11

4.8 Fixing Problems Found in the Production Environment 11

5 Command Line Tools 12

# Introduction

Things to discuss:

* NodeJS code
* Barman
* Environments
* Postgres
* Scripts

## Purpose

The purpose does a few things:

* Explain and in some cases justifies the various TCS software design decisions.
* Provides instructions 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

# Operating Environments

The TCS software supports 3 operating 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

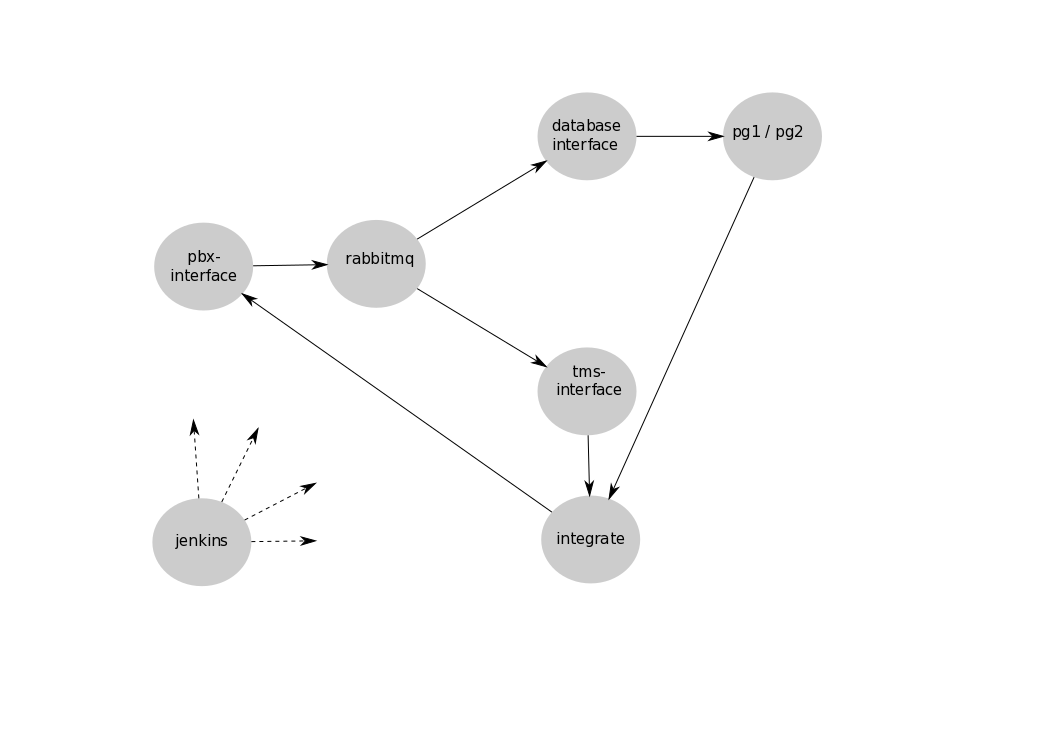
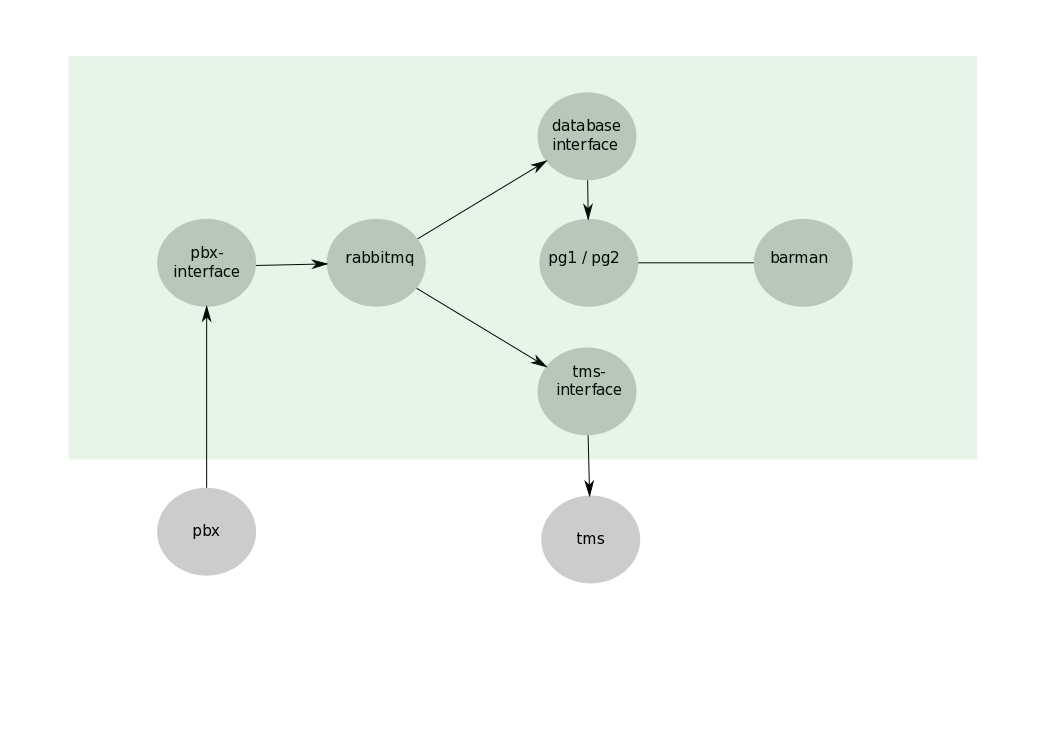


Figure : TCS QA Environment

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 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 pg1 Postgres container. Meanwhile, the tms-interface directs all data that it pulls from its queue to the TMS.

Note that the pg2 container is exclusively used to run a historical version of the operational database. Its data content can be loaded (and Postgres activated) by running the utility pg-offline-recovery. Note that the container pg2 need not exist at all (GONE) or it may exist, but not be running (STOPPED).

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.

# Docker

This section provides the background and context for docker and docker-compose usage.

The TCS consists of 4 Docker projects. There is a relationship between the TCS Docker projects and TCS Operating Environments, but the correspondence is not perfect, starting with the stores project, which supports all 3 operating environments.

## Docker Images

## Docker Project: stores

The name ‘stores’ was selected as a catchall for TCS containers that own data that continues to exist independently of the containers themselves. For example, if pg1 is shut down and removed, pg1’s data continues to exist and is available for when pg1 is restarted. The ‘stateful’ nature of this data is common to all ‘stores’ containers. The containers belonging to this docker project are the following:

|  |  |
| --- | --- |
| **Stores Container** | **Data Description** |
| pg1 | Operational database |
| pg2 | Offline database |
| rabbitmq | Queue data |
| barman | Backup and WAL log files |
| jenkins | QA test configuration and test execution history |

The typical TCS user, even if aware that docker is a key technology exploited by the TCS, likely will not be aware of the ‘stores’ docker project. This is because when the tcs command is triggered, $ tcs, the ‘stores’ containers are automatically started, along with the application containers (discussed next). Similarly, shutting down the TCS, $ tcs down, also shuts down the ‘stores’ containers.

It may occasionally be useful to shut down the ‘stores’ containers independently of the application containers. This can be done as follows: $ stores-down. Nevertheless, the stores-down script is not likely to be useful in the Production operating environment, hence likely a developer-only tool.

## Docker Project: prod

As discussed, the TCS consists of 3 operating environments and these can co-exist. The docker project ‘prod’ consists of 3 containers:

* prod-pbx-interface
* prod-tms-interface
* prod-database-interface

The command $ tcsproj defaults the operating environment to Production. A subsequent $ tcs will start both the Stores and Production containers.

The command prompt displays the terminal session’s current operating environment.

Should it be the case that a terminal has been left in one of the other environments (QA or Development), then the user can easily reconfigure back to the Production environment with the command $ prod.

## Docker Project: qa

As discussed, the TCS consists of 3 operating environments and these can co-exist. The docker project ‘qa’ consists of 3 containers:

* qa-pbx-interface
* qa-tms-interface
* qa-database-interface

The command $ tcsproj defaults the operating environment to Production, however, executing $ qa reconfigures the operating environment to QA.

The command prompt displays the terminal session’s current operating environment.

## Docker Project: dev

As discussed, the TCS consists of 3 operating environments and these can co-exist. The docker project ‘dev’ consists of 3 containers:

* dev-pbx-interface
* dev-tms-interface
* dev-database-interface

The command $ tcsproj defaults the operating environment to Production, however, executing $ dev reconfigures the operating environment to Development.

The command prompt displays the terminal session’s current operating environment.

# Software Release Workflow

This section provides instructions for how to go about deploying a software release to the customer. In the following it is assumed that the developer is working within the master git branch ($ git checkout master). It is possible that the programmer

## Define the New Version Number

The version number is of the form vX.Y, where X and Y are both integers. A first decision is to decide whether the version upgrade is significant or not. If significant, then new X = 1 + X of the latest previous release and Y is set to 0.

If not significant, then X is left unchanged and Y is incremented by 1.

Then the following:

* git branch vX.Y  
  Creates a git branch with the same name as the new version number.
* git checkout vX.Y  
  Checkout the new branch (meaning that subsequent edits are applied only to the new branch).
* open .tcs.version  
  and record the new version number in the file .tcs.version.

At this minimal stage, development of a new version has begun. Note, however, it also possible that a bug fix may need to be applied during the development of the new version. That bug fix will be applied to the currently operational software, meaning that still another branch will be required, and further, this bug fix may need to be applied to the new branch just created. In such a case, a git merge is required. The subject of git merging is outside the scope of this document, but information on this topic can be found [here](https://git-scm.com/docs/git-merge).

## Updating the Documents

A number of documents are listed in the ./docs folder. It is recommended to update these document as required and in advance of any further changes. These documents need to be kept current, although not all software changes will require these documents to be modified (note: if the software release addresses only a bug fix, then it is possible that none of the documents will need to be modified). On the other hand, if the software release addresses some new major new capability, then it is likely that one or more of these documents will require updating (including this one). Certainly adding some new major capability will require that the TCS Software Requirements Document to be updated - at minimum.

A key part of a document update is to also update the document version number (note: document version numbers are not coincident with software release version numbers).

## Coding in the Development Environment

ToDo

## Building Docker Images

ToDo

## Testing in the QA Environment

ToDo

## Fixing Problems Found in the QA Environment

ToDo

## Announcing the Software Release to the Customer

ToDo

## Fixing Problems Found in the Production Environment

ToDo

# 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.