CTRLD Implementation Guide

Version 20.5.1-rc0, 25 May 2020

Registered Address	Support	Sales
26, Kingston Terrace, Princeton, New Jersey 08540, United States		
		+91 80 4850 5445
http://www.rtbrick.com	support@rtbrick.com	sales@rtbrick.com

Table of Contents

1. Overview . . . 3
   1.1. The CTRLD Binary . . . 3
   1.2. The Configuration File . . . 4
   1.3. The CTRLD Logs . . . 5
   1.4. The CTRLD API . . . 5
2. CTRLD and Management . . . 7
   2.1. Container Management . . . 7
   2.2. Image Management . . . 8
      2.2.1. Image Folder . . . 9
      2.2.2. Image Download . . . 9
   2.3. Container and Element Management . . . 9
   2.4. Jobs and Callbacks . . . 10
   2.5. Pub Sub . . . 10

1. Overview

This document gives an inside to the Container management of CTRLD and the interaction between the different systems and CTRLD.

The CTRLD (control daemon) is the single point of entry to an RtBrick router running the RBFS software. CTRLD primarily uses REST to control the router. CTRLD is also responsible for gathering data from the router and forwarding this information to other systems.

This role for CTRLD is shown in Figure 1.

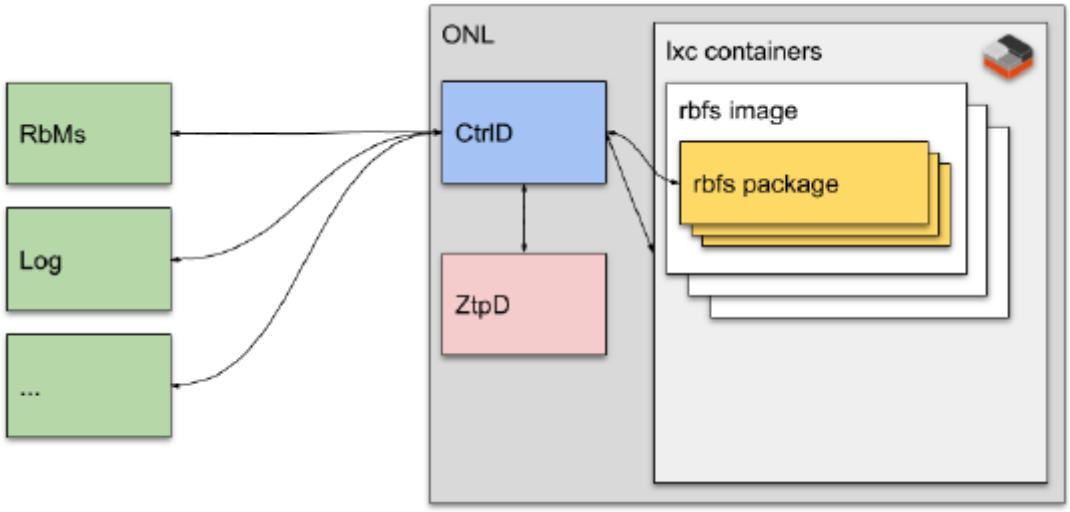

Figure 1. Overview of CTRLD

The CTRLD plays these different roles because CTRLD is the entry point for all of the different components interacting within or outside of a box. So, CTRLD can be thought of in the following way:

- CTRLD is the controller of a router box running RBFS
- CTRLD is the controller of an element within the box, such as LXC containers
- CTRLD is the gateway to an RBFS image and package

You can run multiple instances of the CTRLD on a given box.

1.1. The CTRLD Binary

In a production environment, the ctrld binary starts with default parameters. This service is called rtbrick-ctrld. The ctrld -h command shows these default parameters, highlighted in red in the listing below.

```
$ ctrld -h
Usage of ctrld:
    -addr string
        HTTP network address (default ":19091")
    -config string
        Configuration for the ctrld (default
"/etc/rtbrick/ctrld/config.json")
    -lxccache string lxc Image Cache folder (default "/var/cache/rtbrick")
    -servefromfs
        Serves from filesystem, is only used for development
    -version
        Returns the software version
```

The command ctrld -version displays the installed version of the daemoin. The version should be tagged correctly in the repository.

The CTRLD configuration is in the JSON file at /etc/rtbrick/ctrld/config.json. You can cat the file to display the contents, as shown below:

```
$ cat /etc/rtbrick/ctrld/config.json
{
    "rbms_enable": true,
    "rbms_host": "http://192.168.200.45",
    "rbms_authorization_header": "Basic YWRtaW46YWRtaW4=",
"rbms_heart_beat_interval": 600
}
```

1.2. The Configuration File

The CTRLD configuration file is a flat JSON file. The properties are described in Table 1.

Table 1. CTRLD Configuration File Properties

Property	Description	Default Value
rbms_enable	To enable all RBMS outgoing messages	false
rbms_host	RBMS base url e.g.: http://192.168.202.44:9009	
rbms_authorization_header	RBMS Authorization Header is set to all calls which are outgoing to RBMS	nil
rbms_heart_beat_interval	RBMS heartbeat Interval in seconds. Nil leads in no heartbeat.	nil

Property	Description	Default Value
The calls to rbms are made via a retry handler. After an unsuccessful attempt there will be done a retry call, the time between the attempt is exponential and based on the attempt number and limited by the provided minimum and maximum durations.		
rbms_retry_wait_min	Min wait time in seconds	2
rbms_retry_wait_max	Max wait time in seconds	300
rbms_retry_max	Max retries	10
The calls to the callbacks are made via a retry handler. After an unsuccessful attempt there will be done a retry call, the time between the attempt is exponential and based on the attempt number and limited by the provided minimum and maximum durations.		
callback_retry_wait_min	Min wait time in seconds	2
callback_retry_wait_max	Max wait time in seconds	300
callback_retry_max	Max retries	10
Graylog Configuration		
graylog_enable	To enable all Graylog outgoing messages	false
graylog_url	Graylog url e.g. http://127.0.0.1:12201/gelf	
graylog_heart_beat_interval	Graylog heartbeat Interval in seconds. Nil leads in no heartbeat.	nil

1.3. The CTRLD Logs

The log files for CTRLD are stored at /var/log/rtbrick-ctrld.log, and are rotated with logrotate. The log rotation configuration can be found at /etc/logrotate.d/rtbrick-ctrld.

1.4. The CTRLD API

CTRLD is built with the Domain Driven Design (DDD) Principles in mind. The model is split into modules which, in DDD, are called Bounded Contexts. Also, the CTRLD API is divided in such modules.

The CTRLD API is a rest API that leverages the Richardson Maturity Model level 2.

You can find an aways actual API overview within each running CTRLD instance at:

```
http://<hostname>:<port>/public/openapi/
```

The CTRLD API has been redesigned when we ported it to the golang programming language. However, to provide some extended backward compatibility, there is a module called “AntiCorruptionLayer” to address this problem.

These older APIs might be deleted soon, so use them with caution.

Table 1 describes the API tags used to group the APIs by their modules.

Table 2. REST API Tag Descriptions

API Tag	Description
anti_corruption_layer	These APIs are all deprecated. They exist only for older systems and backward compatibility.
client	These are not the APIs that CTRLD provides, but the APIs that a client has to provide to use the callback function of CTRLD.
ctrld/config	Configure CTRLD
ctrld/containers	Handle LXC containers (start, stop, delete, list)
ctrld/elements	Handle elements (start, stop, delete, upgrade, config)
ctrld/rbfs	Handle calls which came from the RBFS
ctrld/images	Handle all requests regarding RBFS images. (download, delete, list)
ctrld/jobs	Get information about asynchronous tasks.
ctrld/info	General info about CTRLD, like version, image and so on.
ctrld/events	For the publish/subscribe sub model, register for an event, and stay informed about events.
ctrld/system	Communication with the underlying host system.
rbfs	Communication with an RBFS element such as Proxy, File Handling, and so on.

2. CTRLD and Management

This section describes CTRLD container and image management.

2.1. Container Management

The CTRLD is responsible for container management. But the RBMS is not aware of the containers. Therefore, a proper mapping to containers is needed. This section describes the correlation between the RBMS, CTRLD and LXC.

The RBMS has various elements, each identified by name. Each element describes a running RBFS instance. It is possible to upgrade and downgrade elements.

Each element in RBMS has services. That is a bit confusing, because the services not only describe the services of an element itself, but also describe the services running the element.

The root aggregate of the model is the element container, whether there is one element on an ONL or not. The general structure of daemons and containers in the RBFS service model is shown in Figure 2.

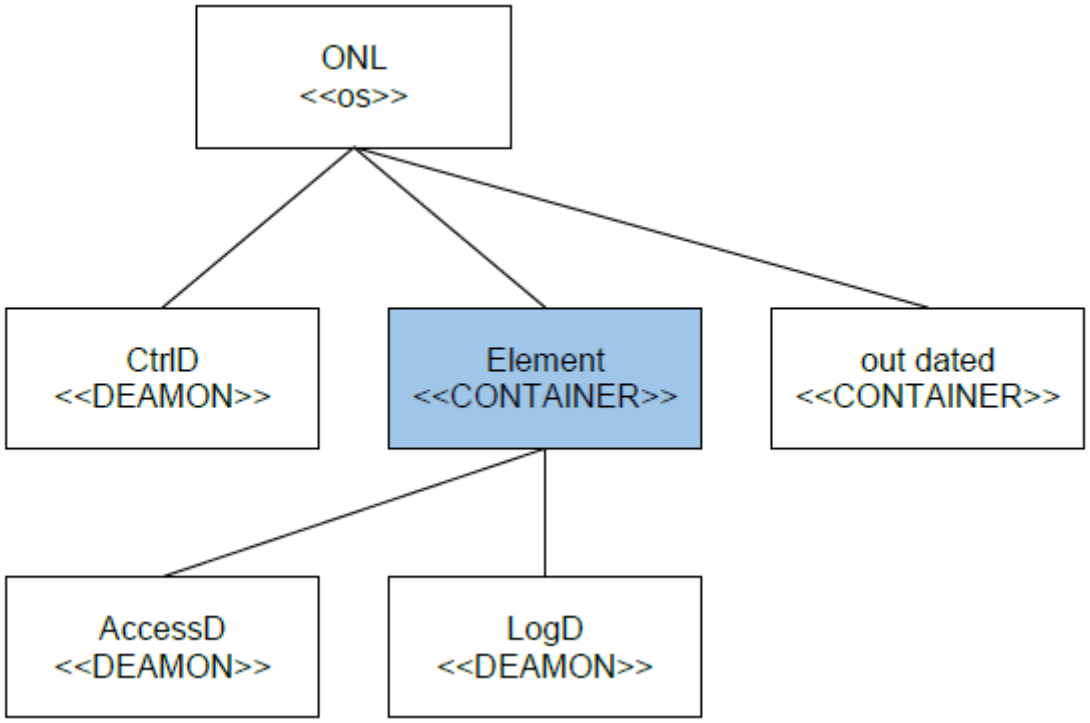

Figure 2. Service Model for RBFS

In case that an element is up- or downgraded the old container is saved as an outdated container. Therefore, it is possible to recover an outdated container, if an upgrade fails or the upgrade has errors.

It is important to understand the difference between an element and a container.

In an RDFS context, a container is always an lxc-container. The containers in a whitebox are simply called “rtbrick”. In order to make them useful for the RBMS, they need a proper name.

You configure the element-name and the pod-name of a container in the lxc-container root directory (/var/lib/lxc/rtbrick/element.confg).

This method provides many advantages:

- Container updates
  - Prepare an update of the container (for example, rtbrick-v2)
  - Stop container version 1 and start container version 2
  - This allows fast updates of containers: if the update is corrupt, stop the second container and restart the first container.
- Renaming of elements

If there is no element.config available, then the element name is the name of the container.

2.2. Image Management

The images are saved on the ONL under /var/cache/lxc/rtbrick.

There is only one subfolder for each image: /var/cache/lxc/rtbrick/<image-folder>

The image is identified by a series of fields, described in Table 2.

Table 3. Image Identification Fields and Descriptions

Field	Description
organization	Organization that issued the image as reverse domain name (e.g. net.rtbrick).
category	Category which can be used to describe the purpose of the image. (e.g. customer-production)
platform	Describes the Hardware Platform.
vendor_name	Vendor of the platform
model_name	Model of the platform
image_type	Image type (for example, LXC)
image_name	Image name (for example, rbfs)
element_role	Element role the image was built for (for example, LEAF).
image_version	Image revision to be activated {major}.{minor}.{patch}-{prerelease}

2.2.1. Image Folder

The image folder contains the following files:

- A metadata.YAML which identifies the image. There can also be additional attributes in the file, but the attributes to identify an image have to be in the file. An example of the RtBrick properties are shown below.

```
rtbrick_properties:
    organization: net.rtbrick
    category: customer-production
    platform:
        vendor_name: virtual/tofino
        model_name: virtual
    image_type: LXC
    role: LEAF
    image_name: rtbrick-rbfs
    image_version: 19.13.4-master
```

- A subfolder named rootfs
- The config.tpl file. This file is used to create the configuration file with the respective data in the template. You can use the following syntax to add a property from the dictionary provided by ctrld. Therefore, lxc.rootfs.path = dir:{{index . "rootfs”}} results in lxc.rootfs.path = dir:/var/lib/lxc/mega/rootfs

2.2.2. Image Download

CTRLD provides functionality to download images from a repository, therefore the url to the image is provided by the caller.

Optionally also the checksum algorithm and the value can be provided, after downloading the image, the checksum will be verified.

2.3. Container and Element Management

LXC Containers are identified as elements if they have a metadata.YAML with the fields described above. These LXC containers can also be revised containers, which are created when an upgrade of a container takes place. The revised element is named using the element name and a timestamp: revised-{element-name}-{timestamp}.

It is not yet possible to rename an element. See How to rename LXD / LXC container for more details.

A template engine to update the LXC configuration template is used for the container. Each container has the files in the /var/lib/lxc/{container-name} folder, as shown in Table 3.

Table 4. Files in the Container Folder

File	Description
config.tpl	Template for lxc configuration This file comes directly out of the image, and is stored in this folder for renaming the container. Because a rename recreates the config file.
config.data	Data which was used to fill the templates (config, hostconfig). This file is saved by ctrld, it is used by rename the container, because a rename recreates the configfile.
metadata.yaml	Information about the image the container was built from. And a lot more information.

The status of an image can be CACHED or ACTIVE, as described in Table 4.

Table 5. Image Status States and Meaning

Status	Description
CACHED	This image is on the ONL
ACTIVE	This image is on the ONL and is the image used for the actual container instance.

2.4. Jobs and Callbacks

The Jobs API is needed for asynchronous API calls. Asynchronous API calls can be used with a callback, so that the caller is informed when the job is finished, or can be used with a polling mechanism. The Job API polling asks if the job is finished. This is sometimes easier to implement, especially for scripts like robot.

The callback mechanism uses a retry handler. The retry handler performs automatic retries under the following conditions:

- If an error is returned by the client (such as a connection error), then the retry is invoked after a waiting period
- If a 500-range response code is received (except for 501 “not implemented”), then the retry is invoked after a waiting period.
- For a 501 response code and all other possibilities, the response is returned and it is up to the caller to interpret the reply.

2.5. Pub Sub

CTRLD uses a publisher and subscriber model. This model is needed for features not implemented directly in CTRLD, such as ZTP.

So, for example, the ZTP daemon (ZTPD) can subscribe to events, and ZTPD is

informed if the event occurs in CTRLD.