API References

Version _25.2.1.1, 16 July 2025

Table of Contents

1. RBFS API User Guide

1.1. RBFS REST APIs

This document contains all the information about RBFS REST API services, their purposes and how to use these APIs. This documentation goes hand-in-hand with the RBFS OpenAPI document which provides information about all RBFS REST API endpoints. It is recommended to refer to this document in conjunction with the RBFS OpenAPI document.

1.1.1. Introduction to RBFS REST APIs

RBFS REST APIs allow customers and partners to programmatically access information from the RBFS software components. RBFS REST APIs enable users to manage and automate many of their tasks by accessing and consuming the RBFS data simply and securely.

RBFS REST API architecture supports containerized deployments with a centralized configuration and management. It also enables the configuration and management of distinct daemons and services.

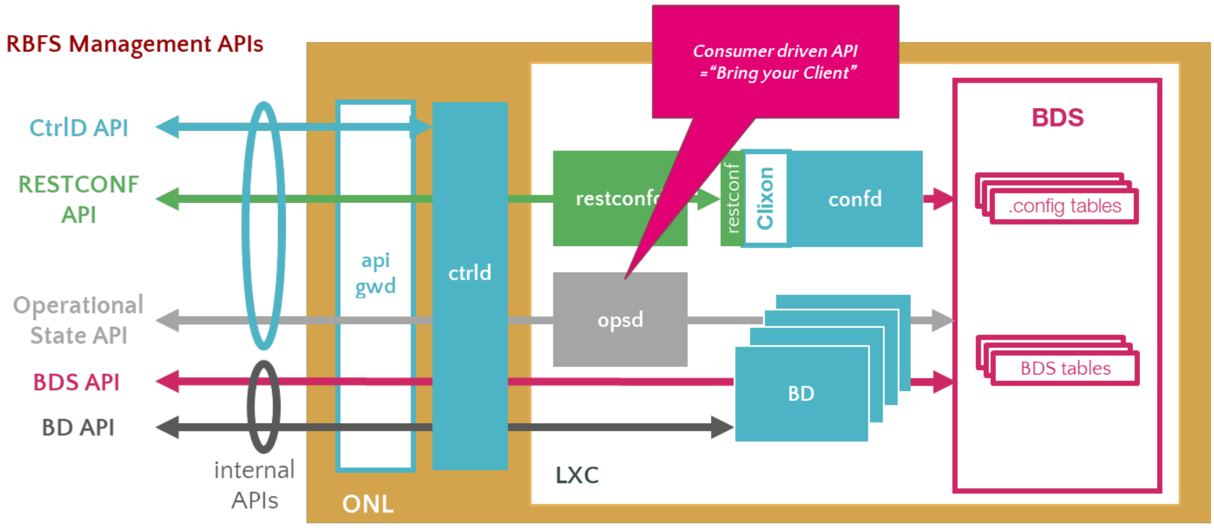
RBFS APIs adhere to the Representational State Transfer (REST) principles and allow consumers to securely connect to the RBFS device, obtain information (read) and run actions or operations to apply changes. RBFS REST APIs use JavaScript Object Notation (JSON) for the exchange of information. The OpenAPI Specification format, which is a broadly accepted industry standard for describing REST APIs, is used to describe, consume, and visualize RBFS REST APIs.

1.1.2. Understanding RBFS APIs

RBFS, a disaggregated Broadband Network Gateway, provides many APIs to make all communications possible with various RBFS software components, the host operating system (ONL), and the hardware platform. RBFS consists of several independent microservices including the API Gateway daemon (ApiGwD) and Control daemon (CtrlD). Both of these microservices, known as daemons, play crucial roles in managing RBFS instances.

API gateway functions only if management plane security is enabled. All API requests from external clients are routed to the API Gateway. After successful authentication by the API Gateway, these requests are forwarded to the Control daemon. CtrlD, which is aware of the state and port information of all daemons that reside in the RBFS container, can forward requests to the respective daemons.

> If security is disabled, API Gateway does not function and, then CtrlD becomes the first entry point for external requests.



The illustration presents how the RBFS REST APIs communication happens with various underlying software components. RBFS microservices, which perform various functions, are containerized in an open Linux container. RBFS APIs are generally categorized into Public Management APIs and Internal APIs. Public Management APIs include CtrlD API, RESTCONF API, and Operational state API. These APIs are used for managing and automating many of the network administration tasks.

> The use of internal APIs such Brick Daemon APIs and BDS APIs are not supported.

The illustration also presents the API Gateway daemon (ApiGwD) and Control daemon (CtrlD), which are deployed on the host operating system (open network Linux) along with the RBFS container. The API Gateway acts as an entry point that provides a secure channel for all REST APIs by authenticating all requests. After successful authentication by the API Gateway, CtrlD (Control Daemon) passes all the requests to the respective daemons (that reside in the RBFS container) which are responsible for performing certain tasks.

RBFS APIs

CtrlD API: CtrlD runs on the host OS (ONL) and acts as a proxy to the other APIs including high-level APIs for the RBFS configurations. The CtrlD API, implemented by the CtrlD, performs various tasks such as starting the container and rebooting the device. In case of a software upgrade, this API is used to trigger the upgrade.

API Gateway: You can deploy the API Gateway Daemon (ApiGwD) on the host OS (that is ONL) to secure the RBFS management plane. The API Gateway authenticates all API requests using JSON web tokens. The API Gateway Daemon acts as the TLS endpoint for the hardware platform and it converts external access token into an internal RtBrick token /SEC/. Finally, it forwards the requests to the CtrlD.

The API Gateway also enforces an API throttler that provides a mechanism called API throttle quotas to protect the RBFS system resources from being exhausted with too many requests by a single client system that uses the RBFS APIs very extensively.

Operational State APIs: The Operational State API, provided by the Operational State Daemon (opsd), allows accessing system states such as routing protocol states, interface states, subscriber states and resource utilization. The Operational State Daemon takes care of examining the operational state of a switch and runs actions to diagnose and troubleshoot the problems. The operational state is ephemeral and state data is lost when the switch reboots.

RESTCONF APIs: With RESTCONF, you can manage all the configurations in RBFS.

Prometheus APIs: Prometheus APIs help to retrieve metrics from various RBFS components for visibility.

ℹ RFC and draft compliance mentioned in the document are partial except as specified.

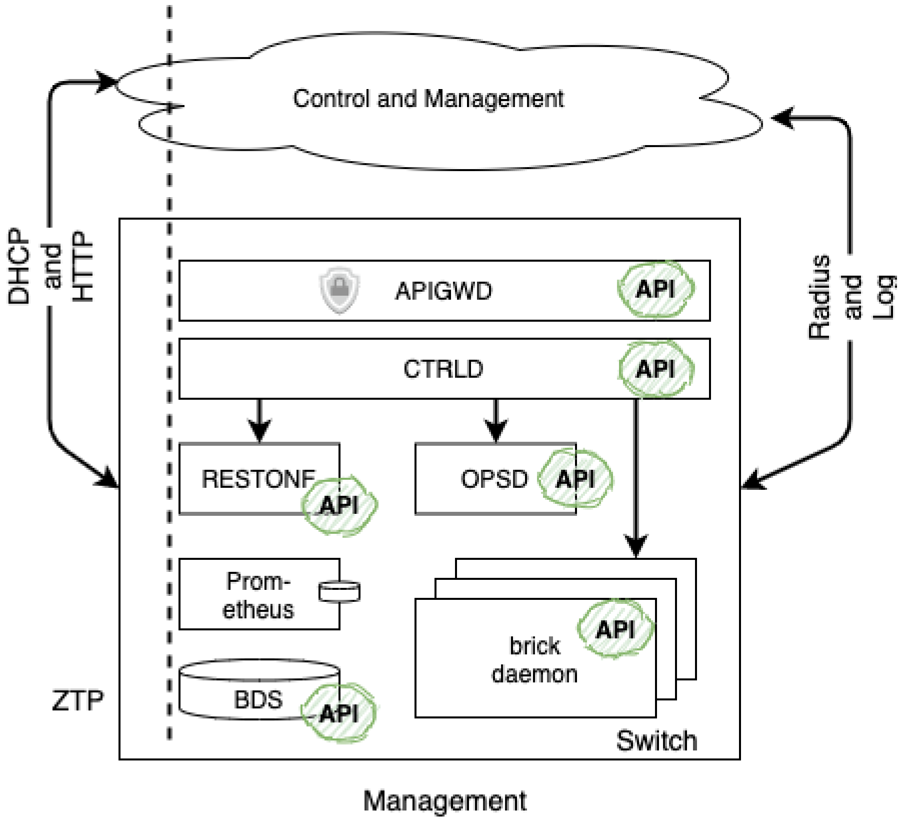
1.2. Controller Daemon

This chapter provides information about the Controller Daemon (CtrlD) which is a major software component in the RBFS ecosystem. It serves as a proxy to various APIs, including high-level APIs for RBFS configurations.

1.2.1. CtrlD Overview

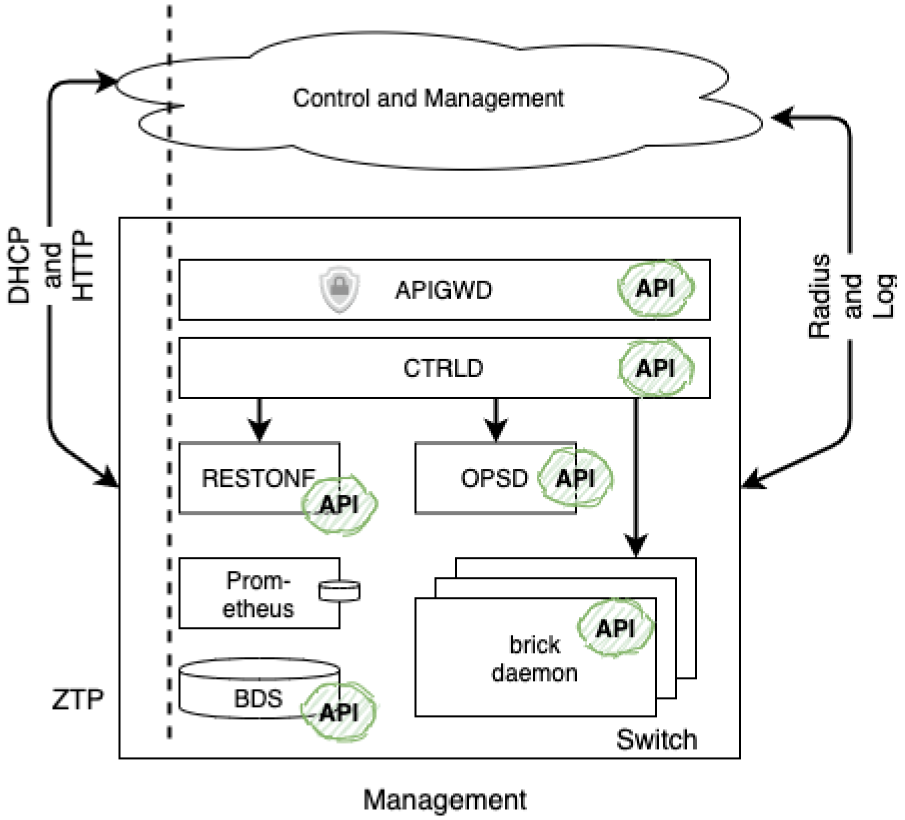
CtrlD operates on the host OS (ONL) and it is the single entry point to the router running the RBFS software. CtrlD controls and manages most of the tasks and functions in an RBFS ecosystem. You can run multiple instances of the CtrlD on an RBFS device.

CtrlD is deployed on ONL (Open Network Linux) and acts as an intermediary between the RBFS container in the ONL and external systems. The following illustration presents a high-level overview of CtrlD.



The CtrlD API, implemented by CtrlD, performs multiple tasks such as initiating the container and rebooting the device. In the event of a software upgrade, CtrlD API is used to trigger the upgrade process.

CtrlD also manages the container and controls the interaction between the different external systems such as RBMS, Graylog, and so on. CtrlD uses REST APIs to control and manage the router. CtrlD is also responsible for gathering data from the router and forwarding this information to other systems.



CtrlD plays different roles in an RBFS ecosystem. It is the gateway for all other components interacting within or outside of an RBFS device. CtrlD acts as:

- Controller for router device running RBFS.
- Controller for elements in the device, such as LXC containers.
- Gateway to RBFS images and packages.

1.2.2. CtrlD Parameters

In a production environment, the CtrlD binary starts with default parameters as the rtbrick-ctrld service. To see these default parameters, use the ctrld -h command.

```
$ 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
```

CtrlD Version

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

The CtrlD configuration can be located in the JSON file: /etc/rtbrick/ctrld/config.json. Use the cat command to display the file content.

Example:

```
$ cat /etc/rtbrick/ctrld/config.json
{
    "rbms_enable": true,
    "rbms_host": "http://198.51.100.77",
    "rbms_authorization_header": "Basic YWRtaW46YWRtaW4=",
"rbms_heartbeat_interval": 600
}
```

1.2.3. Container Management

CtrlD serves as the manager for containers. However, the RBMS, an external RBFS management system, is not aware of these containers, necessitating a systematic mapping. Understanding the correlation between RBMS, CtrlD, and Linux Containers (LXC) is important.

RBMS contains various elements, each uniquely identified by a name, representing a running RBFS instance. You can upgrade or downgrade Elements.

Each RBMS Element includes services. These services not only characterize the Element’s functionalities but also encapsulate the services running within it.

The foundational unit in the model is the 'element container', whether a single element exists on an ONL or not. The following figure illustrates the general structure of daemons and containers within the RBFS service model.

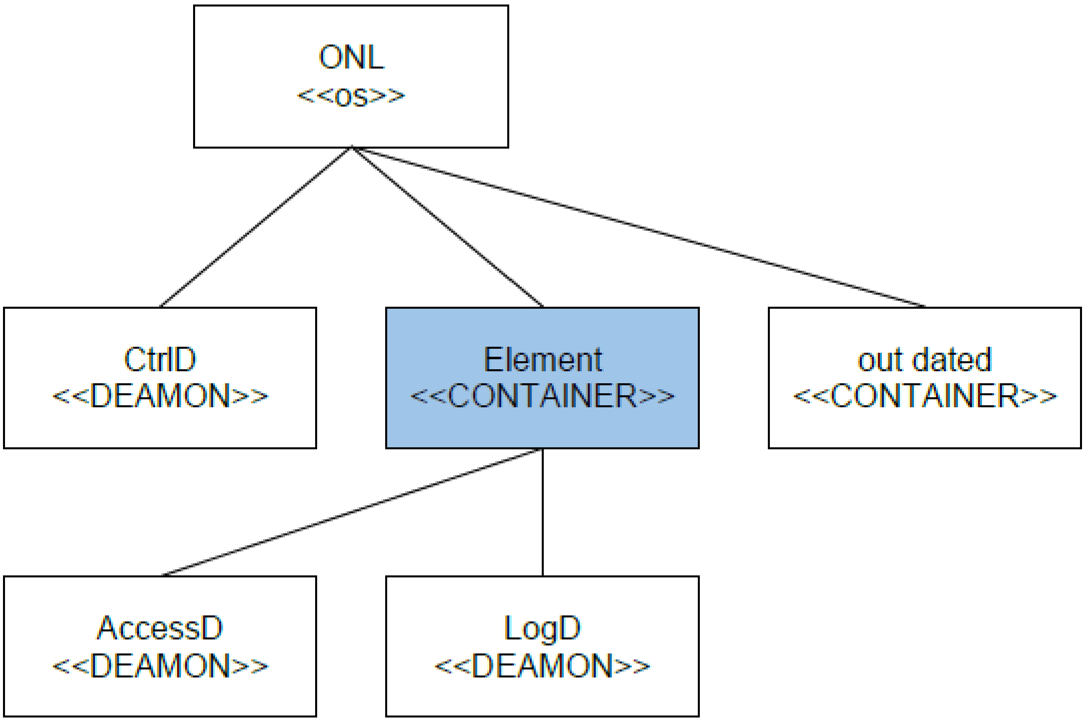


Figure 1. Service Model for RBFS

In the event of an Element undergoing upgrade or downgrade, the system automatically preserves the previous version within an outdated container. It enables the recovery of the outdated container, if the upgrade or downgrade fails. By maintaining the backup, it minimizes potential disruptions and safeguards against data loss.

Both 'element' and 'container' are different in RBFS. In RBFS, a container always refers to an lxc-container. However, in a white box environment, a container is denoted simply as rtbrick. To integrate them effectively into the RBMS, it is essential to assign them appropriate names for clarity and functionality.

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

This method offers several benefits:

Updating Container

- Upgrading or downgrading the containers (for example, upgrading to a higher version rbrick-v2).
- It is possible to stop the currently running container version 1 and to launch container version 2.

- It facilitates fast container updates. If an update fails, it can revert to the previous state by stopping the second container and restarting the first one.

Rename Elements

If element.config is not available, default to using the element name as the container name.

1.2.4. Images Management

The images are stored within Open Network Linux (ONL) at /var/cache/lxc/rtbrick. For each image, there is one subfolder: /var/cache/lxc/rtbrick/<image-folder>. You can identify the images based on a set of fields, as described in the following table.

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}

Image Repository

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 is 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

Image Download

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

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

1.2.5. 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 possible to rename an element. For more information, see How to rename LXD / LXC container.

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.

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 the following table.

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.

1.2.6. CtrlD API

CtrlD has been designed with Domain Driven Design (DDD) principles. The model is divided into modules, known as Bounded Contexts in DDD. Similarly, the CtrlD API is organized into these modules. The CtrlD APIs are REST APIs that adhere to level 2 of the Richardson Maturity Model.

You can find an API overview within each running CtrlD instance:

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

The CtrlD API was redesigned when ported to the Golang programming language.

To ensure extended backward compatibility, a module called AntiCorruptionLayer is introduced to address this issue.

> ℹ Some older APIs may soon be deprecated or removed, use them with caution.

The following table shows the API tags used to group the APIs by their respective modules.

REST API Tag Descriptions

API Tag	Description
anti_corruption_layer	These APIs are deprecated and are only present for older systems to ensure backward compatibility.
client	These APIs are not provided by CtrlD, but rather, they are the APIs that a client must provide to use CtrlD's callback function.
ctrld/config	Configure CtrlD.
ctrld/containers	Handle LXC containers (start, stop, delete, and list)
ctrld/elements	Handle elements (start, stop, delete, upgrade, and config)
ctrld/rbfs	Handle calls that come from the RBFS.
ctrld/images	Handle all requests regarding RBFS images. (download, delete, and list)
ctrld/jobs	Get information about asynchronous tasks.
ctrld/info	General information about CtrlD such as 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.

1.2.7. 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.

1.2.8. Pub-Sub Model

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

For example, the ZTP daemon (ZTPD) can subscribe to events, and ZTPD is informed if the event occurs in CtrlD.

1.2.9. CtrlD Logs

The log files for CtrlD are stored at /var/log/rtbrick-ctrld.log, and are rotated with logrotate. The log rotation configuration is stored at /etc/logrotate.d/rtbrick-ctrld.

1.3. Functions and Use cases of RBFS REST APIs

1.3.1. CtrlD API

The CtrlD performs various functions in the lifecycle of an RBFS container. Zero-Touch Provisioning installs the RBFS software on the hardware devices with minimal human intervention. It is the responsibility of the CtrlD to initiate, discover and download the startup configuration files as part of zero-touch provisioning (ZTP) process. The CtrlD retrieves the base URL and executes the startup configuration on the device, that is pre-installed with ONL, the host operating system.

CtrlD API acts as a proxy endpoint that defines the way the API proxy interacts with the backend services. CtrlD is a go-between, which sits in the middle, for requests from clients and the API. As the API proxy, CtrlD, resides in front of the API, enforcing policies that dictate the usage of the API.

CtrlD API can be used to reboot the switch with or without a software upgrade. As a proxy endpoint, users can access the following APIs through CtrlD:

- RESTCONF API
- Operational State API
- Prometheus API

For more information about CtrlD API use case and examples, see section *5.1. CtrlD API: Use Case and Example'*.

To view the CtrlD API Reference, navigate to RBFS APIs, and select CTRLD API Reference from the drop-down list.

1.3.2. RESTCONF API

RESTCONF is an HTTP-based REST API protocol for network management and automation. It provides a programmatic interface for accessing data defined in YANG. The YANG model describes the configuration syntax. In RBFS, the RESTCONF API provides the configuration data.

Users can use RESTCONF API to execute various configurations in RBFS. RESTCONF API can be used for the following tasks:

- Read configurations
- Replace configurations
- Partial configuration update
- Remove configurations

For more information about RESTCONF API use case and examples, see section *5.2. RESTCONF API: Use Case and Example*.

To view the RESTCONF API Reference, navigate to RBFS APIs, and select RESTCONF API Reference from the drop-down list.

1.3.3. Operational State API

The Operational State API (opsd) provides the system state information. The ospd is backward compatible and supports running older and newer versions of applications together in the network. The Backward compatibility feature is useful whenever newer RBFS releases are rolled out.

The Operational State API was implemented using the Python language that provides a collaborative system to all stakeholders including integration partners, customers, professional services and engineering to collaborate on the API endpoints.

To view the Operational State API Reference, navigate to RBFS APIs, and select Operational State API Reference from the drop-down list.

For more information about Operational State API use case and examples, see section *5.3. Operational State API: Use Case and Example*.

1.3.4. Guidelines and Limitations

When you execute configurations through management APIs, and then with the Command Line Interface at the same time, it results in conflicts when you commit the configuration through the CLI. The reason is that CtrlD directly interacts with the backend applications and these changes are not synced with the CLI.

1.4. Configure API Gateway and CtrlD Components

The section provides configuration information for various components of the API Gateway and CtrlD. Both the API Gateway and CtrlD have been installed as part of the RBFS installation on the host operating system. You must complete some additional configurations for running API Gateway and CtrlD.

1.4.1. Configure API Gateway Components

The API Gateway service is called rtbrick-apigwd and the API Gateway daemon contains some default parameters.

- To know the installed API gateway version, run the apigwd -version command.
- To display the default settings and the location of configuration files, run the apigwd -help command.

The mandatory API Gateway configuration files include:

- API Gateway configuration file: /etc/rtbrick/apigwd/config.json
- JWKS file for access token verification: /etc/rtbrick/apigwd/access_secret_jwks.json
- X509 public/private key file in the 'pem' format: /etc/rtbrick/apigwd/tls.pem

SSL/TLS Certificate

API Gateway uses Transport Layer Security (TLS), also known as Secure Sockets Layer (SSL), certificates to authenticate and secure all requests pass through API Gateway.

If there is no TLS certificate provided, API Gateway generates one certificate signed by a self-signed root CA.

You can specify a TLS certificate in any of the following ways:

- Deploy the TLS file through ZTP
- Provide the URL for TLS file downloading in the config.json file.

With this setting, API Gateway monitors the server periodically for new TLS files using the HTTP caching directives as described in the RFC 7234 to avoid unnecessary downloads.

tls.pem file

The tls.pem file, which contains the X509 certificate (public and private key), enables TLS in PEM format (as described in the RFC 7468). The tls.pem file can be changed on the file system. API Gateway automatically reloads the tls.pem file whenever it gets changed or replaced.

/etc/rtbrick/apigwd/tls.pem example

```
-----BEGIN CERTIFICATE-----
NOT A REAL KEY
-----END CERTIFICATE-----
-----BEGIN RSA PRIVATE KEY-----
NOT A REAL KEY
-----END RSA PRIVATE KEY-----
```

JSON Web Token File

JSON Web Token (RFC 7517 enables secure transmission of information between client and server as a JSON object. API Gateway validates the access token against a JSON Web Key Set (JWKS) and it allows specifying two sources for the keyset. The sources are consulted in the following order for validation:

A local file on the file system

This file can be deployed through ZTP. It is recommended to deploy a local file on the file system. If it is an empty key set file, there is a default pre-configured file on the system that is used.

Remote file URLs through the config.json

You can provide the JWKS remote file URLs through the config.json file. Whenever a token needs to be verified, a JWKS file gets downloaded.

Whenever an access token needs to be verified, the API Gateway queries the server for the current JWKS file using the HTTP caching directives (as described in RFC 7234) to avoid unnecessary downloads.

> ℹ An RFC 7234-compliant cache is used for downloading the configuration file.

JSON Web Key Set

The JSON Web Key Set contains the public keys used to verify any JSON Web Token issued by the authorization system.

The access_secret_jwks.json file can be updated on the file system.

API Gateway automatically reloads the tls.pem file whenever it gets changed or replaced.

/etc/rtbrick/apigwd/access_secret_jwks.json example

```
{
  "keys": [
    {
      "kty": "RSA",
      "e": "AQAB",
      "use": "sig",
      "kid": "access",
      "alg": "RS256",
      "n": "NOT A REAL KEY"
    }
  ]
}
```

These keys authenticate external requests coming to the API Gateway. The right key is selected by the kid (key id) attribute. With this key, the access tokens are verified and converted into an RtBrick token.

config.json File

The config.json is the configuration file of the API Gateway. API Gateway automatically reloads the file whenever it gets changed or replaced.

/etc/rtbrick/apigwd/config.json example

```
{
  "access_token_jwks_urls": [
    "http://192.168.202.56:8080/primaryJWKS",
    "http://192.168.202.56:8080/secondaryJWKS"
  ],

  "request_rate": 5,
  "request_burst": 10,
  "report_rejects_every": 10
}
```

The following table presents the various attributes and the description of the config.json file.

/etc/rtbrick/apigwd/config.json format

Name	Type	Description
access_token_jwks_urls	[]string	Allows to specify multiple JWKS remote URLs.
Remote PEM file		
pem_urls	[]string	Allows to specify multiple PEM remote URLs. Empty list disables the download.
pem_reload_time	int	Allows to specify the time after a new reload is triggered. 0 disables the download.
Request rate limit		
request_rate	float	The allowed requests per second per client.

request_burst	int	Is the maximum number of tokens that can be consumed at once, without respect to the rate.
report_rejects_every	int	Report rejects only every x seconds to avoid massive logging to a GELF endpoint.

1.4.2. Control Daemon (CtrlD)

The CtrlD service is called rtbrick-ctrld and the CtrlD contains some default parameters.

- To know the installed CtrlD version, run the ctrld -version command.
- To display the default settings and the location of configuration files, run the ctrld -help command.

The CtrlD configuration files include:

- The CtrlD configuration file: /etc/rtbrick/ctrld/config.json
- The Role-Based Access Control policy file: /etc/rtbrick/ctrld/policy.json
- The element configuration file for the container: /var/lib/lxc/<container-name>/element.config

config.json file

The config.json file can be changed using API. If the file is updated on the file system, CtrlD must be restarted.

> Changes to the config.json file will come into effect only after CtrlD gets restarted. Use the CtrlD API to apply in-service configuration changes at runtime. CtrlD updates the config.json file to get the changes applied through the API persistent.

/etc/rtbrick/ctrld/config.json example

```
{
  "element_name": "element_name",
  "pod_name": "pod_name",
  "rbms_enable": true,
  "rbms_host": "http://198.51.100.48",
  "rbms_authorization_header": "Bearer THIS IS NOT A REAL KEY",
  "rbms_heartbeat_interval": 10,
   "logging": {
    "heartbeat_interval": 60,
    "aliases": {
```

```
   "default": {
    "endpoints": [
     {
      "type": "gelf",
      "max_log_level": 5,
      "buffer_size": 500,
      "network": "http",
      "address": "http://10.200.32.49:12201/gelf"
     },
     {
      "type": "syslog",
      "max_log_level": 5,
      "buffer_size": 30,
      "network": "udp",
      "address": "10.200.32.49:516"
     }
    ]
   },
   "ztp": {
    "endpoints": [
     {
      "type": "gelf",
      "max_log_level": 4,
      "buffer_size": 20,
      "network": "http",
      "address": "http://10.200.32.49:12201/gelf"
     }
    ]
   }
  }
 },
 "auth_enabled": false
}
```

The following table presents the various attributes and descriptions for CtrlD config.json file.

/etc/rtbrick/ctrld/config.json format

Name	Type	Description
element_name	string	The name of the element (container).
pod_name	string	The pod name. Pod stands for point (zone) of deployment. A pod can contain a group of elements.
rbms_enable	bool	To enable all RBMS outgoing messages rbms_host.
rbms_host	string	RBMS base URL. For example, http://198.51.100.144:9009
rbms_authorization_header	string	RBMS Authorization Header is set to all calls which are outgoing to RBMS.

rbms_heartbeat_interval	int	RBMS heartbeat interval in seconds (0 means deactivated)
auth_enabled	bool	To enable the authorization and authentication.

<table>
<tr>
<td><b>logging</b></td>
<td>
<p>Log configuration for the host personality of the switch. The routing instances (elements) can configure the logging in the RBFS configuration, and that is forwarded, for the routing instance, to CtrlD. The alias (also known as external log server or Plugin Alias) default acts as the default alias if a specific alias is not defined.</p>
<table>
<tr><td>Name</td><td>Type</td><td>Description</td></tr>
<tr><td><b>alias</b></td><td>string</td><td>Logical name of the endpoints. For example, ztp for ztp messages.</td></tr>
</table>
<p>Each alias can have multiple endpoints. If an alias does not define any endpoint, the alias is disabled and the message is not sent and to the default alias.</p>
<table>
<tr><td>Name</td><td>Type</td><td>Description</td></tr>
<tr><td><b>type</b></td><td>string</td><td>Type could be syslog or gelf.</td></tr>
<tr><td><b>max_log_level</b></td><td>string</td><td>MaxLogLevel that will be forwarded (default "Notice: 5)"</td></tr>
<tr><td><b>network</b></td><td>string</td><td>Network get network either tcp, udp or http. Consider the support matrix: * gelf: http * syslog: udp, tcp</td></tr>
<tr><td><b>buffer_size</b></td><td>string</td><td>BufferSize that will be used for the fanout, if the buffer is full, the newer messages that arrive are thrown away.</td></tr>
<tr><td><b>address</b></td><td>string</td><td>Address where to send the message</td></tr>
<tr><td><b>formatter</b></td><td>string</td><td>The formatter that should be used. Consider the support matrix:
• gelf: none
• syslog: rfc5424</td></tr>
</table>
</td>
</tr>
</table>

policy.json file

The policy.json file is used to configure role-based access control (RBAC) for CtrlD. This file can be changed using API. If it is changed on the file system, CtrlD must be restarted.

/etc/rtbrick/ctrld/policy.json example

```
{
  "permissions": [
    {"sub": "system", "obj": "/*", "act": ".*" },
    {"sub": "supervisor", "obj": "/*", "act": ".*" },
    {"sub": "operator", "obj": "/*", "act": ".*"},
    {"sub": "reader", "obj": "/*", "act": "GET"},
    {"sub": "reader", "obj":
"/api/v1/rbfs/elements/{element_name}/services/{service_name}/proxy/bds/table/walk
", "act": ".*"},
    {"sub": "reader", "obj":
"/api/v1/rbfs/elements/{element_name}/services/{service_name}/proxy/bds/object/get
", "act": ".*"}
  ]
}
```

/etc/rtbrick/ctrld/policy.json format

Name	Type	Description
sub	string	Subjects means the role which has the permission. Here RegexMatch Function is used: a regular expression pattern matcher.
obj	string	Object is the REST endpoint. Here KeyMatch4 Function is used: KeyMatch4 determines whether key1 matches the pattern of key2 (similar to RESTful path), key2 can contain a * and other patterns: • "/foo/bar" matches "/foo/" • "/resource1" matches "/{resource}" • "/parent/123/child/123" matches "/parent/{id}/child/{id}" • "/parent/123/child/456" does not match "/parent/{id}/child/{id}"

Name	Type	Description
act	string	And Action is the HTTP Method. Here RegexMatch Function is used: a regular expression pattern matcher.

The rules are:

- The user with the role 'system' can access all the rest endpoints and act on them with all HTTP methods.
- The user with the role 'reader' can access all rest endpoints; but can only call the HTTP GET method.
- All authenticated users are allowed to access the proxy endpoint with all HTTP methods.

element.config file

The element.config file allows to rename the default element name. By default, the element name and the container name are the same. You can use the file to rename the default element name so that the element name and the container name are differentiated.

/var/lib/lxc/<container-name>/element.config example

```
```

/var/lib/lxc/<container-name>/element.config format

Name	Type	Description
element_name	string	Name of the element (By default, it is the container name).
pod_name	string	Name of the PoD.
ztp_enabled	bool	If enabled, the ZTP post process starts when the switch is moved to the operational state "up". It is recommended to set this to 'false'. In that case, only the initial installation or reinstallation triggers that process.

1.5. Events

RBFS REST APIs play important roles in fetching event logs. Event logs are records of events that occur in the different functional areas of the RBFS ecosystem. In RBFS, there are different types of logs. Almost every daemon or module in RBFS generates a variety of logs. All these logs, which are generated from different components, can be exported to the log management server, where you can view and analyze the real-time data.

Log events originate from the RBFS log facility and form a structured log record. If logging is disabled, then no logs are produced. For more information about RBFS Logging, see /resources/techdocs/current/loggingug/logging_intro.html[Logging User Guide].

1.5.1. Alerts

Alerts are event logs that originate from alert configurations. Alerts either report an issue or notify that an issue has been resolved. Alerts are fully under the control of a customer. Users can implement alert rules to produce an alert that triggers automated action in the management system. An alert event is also a business event when it triggers automated actions.

1.5.2. Business Events

A business event is a record of events that originate from the control daemon, irrespective of the logging configuration. Business events notify the management system about significant state changes for triggering automated actions.

Business events are recorded by CtrlD and these events are static without any changes release after release.

APIGwD and CtrlD send different GELF and Syslog messages about status changes or the progress of processes to a GELF or Syslog endpoint.

The following table presents the business event message format:

GELF message format

Name	Type	Mandatory	Description
Default Message Fields			

Name	Type	Mandatory	Description
version	String	Yes	The GELF message format version. Default value: 1.1
host	String	Yes	The hostname is assigned via DHCP to the management interface. Defaults to the management IP address if no hostname is assigned.
level	int	Yes	Message Severity. See Table-1.
timestamp	float	Yes	Unix epoch time in seconds with an optional fraction of milliseconds.
short_message	String	Yes	Problem message.
full_message	String	No	Detailed problem description.
_daemon	String	Yes	Name of the daemon.
_log_module	String	Yes	The module name identifies the component that created the log record. It allows segregating log records into different streams. Each stream can apply different processing rules and also be processed by different organizational units of the network operator.
_log_event	String	Yes	The log event identifies the log message template in the log configuration. The log event simplifies finding where in the system the log record was created. The log event should be succinct and typically conveys a unique reason code. In addition, the log event should be a reference that can be looked up in the product troubleshooting guide.

Name	Type	Mandatory	Description
_serial_number	String	Yes	The serial number of the switch. This allows tracking hardware replacements, even if the element name remains the same. Empty if not available.
_rtb_image_version	String	No	ONL Image Version that is installed on the switch that reports this message.
_origin	String	No	host or container, defines the origin of a message. This is only set for events that are ambiguous.
		ZTP Message Fields	
_config_name	String	No	Exposes the loaded configuration name. Only set when a configuration file was processed or an attempt to process the file failed (e.g., 404 Not Found response from the HTTP server while attempting to load the configuration)
_config_sha1	String	No	Exposes the SHA1 checksum of the loaded configuration. Only set when the HTTP server returns a configuration.
_operational_state	String	No	Exposes the operational state of the element.
		Request Message Fields	
_rid	String	No	Request ID, either X-Request-ID or new generated
_user_name	String	No	User name out of the access token
_user_subject	String	No	User subject out of the access token
_received_time	String	No	Time when the requested arrived
_method	String	No	HTTP method
_url	String	No	HTTP url

Name	Type	Mandatory	Description
proto	String	No	HTTP protocol
_remote_ip	String	No	HTTP remote ip address
		Service State Message Fields	
_service_name	String	No	Service name
_service_operational_state	String	No	Operational Service
_service_startup_time	Number	No	Service startup time in unix epoch time, the number of seconds elapsed since January 1, 1970 UTC.
_service_down_flap_time	Number	No	Last down flap time in unix epoch time, the number of seconds elapsed since January 1, 1970 UTC.
_service_down_flap_counter	Number	No	Last down flap time in unix epoch time, the number of seconds elapsed since January 1, 1970 UTC.
_service_restarted	String	No	Restart is set to true if service_startup_time was changed.
_service_scope	String	No	Either "host" or "container" indicates if the service is running in the host or inside the container.

Level Descriptions as in RFC 5424

Level	Name	Comment
0	**Emergency**	System is unusable
1	**Alert**	Action must be taken immediately
2	**Critical**	Critical conditions
3	**Error**	Error conditions
4	**Warning**	Warning conditions
5	**Notice**	Normal but significant condition
6	**Informational**	Informational messages
7	**Debug**	Debug-level messages

GELF sample message

```
{
  "_config_name": "ctrld",
  "_config_sha1": "f1e06ef1e53becde6f8baf2b2fafe7dc9c36f6f0",
  "_daemon": "ctrld",
  "_element_name": "leaf01",
  "_log_event": "ZTP0011I",
  "_log_module": "ztp",
  "_serial_number": "591654XK1902037",
  "host": "leaf01",
  "level": 6,
  "short_message": "ztp ctrld config set",
  "timestamp": 1588382356.000511,
  "version": "1.1"
}
```

Event Types

Instance	severity	log_module	log_event	log config	description
ztp	Notice	ztp	ZTP0011I	ctrld	ztp ctrld config set
ztp	Warn	ztp	ZTP0012W	ctrld	ztp ctrld config not provided
ztp	Alert	ztp	ZTP0013E	ctrld	ztp ctrld config not set
ztp	Notice	ztp	ZTP0021I	ctrld	ztp startup config set
ztp	Warn	ztp	ZTP0022W	ctrld	ztp startup config not provided
ztp	Alert	ztp	ZTP0023E	ctrld	ztp startup config not set
ztp	Notice	ztp	ZTP0041I	ctrld	ztp ctrld rbac config set
ztp	Warn	ztp	ZTP0042W	ctrld	ztp ctrld rbac config not provided
ztp	Alert	ztp	ZTP0043E	ctrld	ztp ctrld rbac config not set
ztp	Notice	ztp	ZTP0051I	ctrld	ztp tls config set
ztp	Warn	ztp	ZTP0052W	ctrld	ztp tls config not provided
ztp	Alert	ztp	ZTP0053E	ctrld	ztp tls config not set
ztp	Notice	ztp	ZTP0061I	ctrld	ztp accessjwks config set
ztp	Warn	ztp	ZTP0062W	ctrld	ztp accessjwks config not provided
ztp	Alert	ztp	ZTP0063E	ctrld	ztp accessjwks config not set
ztp	Notice	ztp	ZTP0071I	ctrld	ztp apigwd config set

ztp	Warn	ztp	ZTP0072W	ctrld	ztp apigwd config not provided
ztp	Alert	ztp	ZTP0073E	ctrld	ztp apigwd config not set
ztp	Notice	ztp	ZTP1000I	ctrld	ztp process finished
security	Warn	security	SEC0001W	ctrld	access forbidden
security	Warn	security	SEC0002W	ctrld	access invalid rtb token
security	Warn	security	SEC0003W	ctrld	access invalid access token
security	Warn	security	SEC0004W	ctrld	not able to download remote keys
security	Warn	security	SEC0005W	ctrld	not able to download remote pem
security	Warn	security	SEC0006W	ctrld	request rate limited (this message is also rate limited, and can be controlled in the apiwd config)
element	Notice	element	HTB0001	ctrld	heartbeat with the operational_state
element	Notice	element	STA0001	ctrld	element state change
element	Notice	element	STA0021	ctrld	service up
element	Error	element	STA0022	ctrld	service unexpected down
element	Notice	element	STA0023	ctrld	service expected down
element	Notice	element	STA0003	ctrld	ready for service
element	Notice	element	STA0031	ctrld	module new (one of the modules is newly discovered e.g. fan, SFP ..., this event will be fired after every reboot of ctrld)

element	Notice	element	STA0032	ctrld	module parameter changed (one of the parameters of the module got changed e.g. fan, SFP …)
element	Notice	element	STA0033	ctrld	module removed (one of the modules got removed e.g. fan, SFP …)
ALL	Notice	element	STA0040	all	messages could have been dropped
prometheus	?	?	?	element	messages generated by prometheus alerts

1.6. Appendix: Use case Scenario and Examples for RBFS REST APIs

1.6.1. CtrlD API: Use Cases and Examples

A single CtrlD instance can serve multiple RBFS containers. Each container forms a network element and has a unique name which is the 'element name'.

All API calls (requests) to an RBFS container contain the element name in the URL path.

BNG is used as the element name in the following examples. NOTE: *10.0.0.1* is used as the management IP address in the following examples.

Rebooting a Switch

This example shows how to reboot the switch.

Rebooting a switch is an example of an *asynchronous* operation.

The API call returns the acknowledgment immediately that the reboot request has been *accepted* (HTTP Status Code 202).

The following list shows the HTTP request to reboot switch 10.0.0.1. The URL contains no element name because the entire switch will be rebooted.

```
POST /api/v1/ctrld/system/_reboot HTTP/1.1
Host: 10.0.0.1:19091
```

Triggering a Software Upgrade

A software upgrade can be performed by executing the Zero-Touch Provisioning (ZTP) process again. The switch will be rebooted in Open Network Install Environment (ONIE) update mode and allow ONIE to discover the OS installer image and startup configuration files from the ZTP server.

Rebooting a switch is an example of *asynchronous* operation.

The API call returns a response immediately that the reboot request has been *accepted* (HTTP Status Code 202).

The following example shows the HTTP request to reboot switch 10.0.0.1. The URL contains no element name because the entire switch will be rebooted.

```
POST /api/v1/ctrld/system/_update HTTP/1.1
Host: 10.0.0.1:19091
```

Using the Proxy Endpoint

The CtrlD proxy endpoint forwards API calls to the daemons hosting the API in the RBFS container. Three daemons host publicly available APIs:

- opsd, the operational state daemon, hosts the Operational State API.
- restconfd, the RESTCONF daemon, hosts the RESTCONF API.
- Prometheus hosts the Prometheus API.

> All examples in the following sections use the CtrlD proxy endpoint.

The following table summarizes the proxy endpoint paths for the daemons: opsd, restconfd, and Prometheus and an element named *BNG*.

Daemon	Path
Operational State Daemon (opsd)	/api/v1/rbfs/elements/cnbg-1/services/opsd/proxy

Daemon	Path
RESTCONF Daemon (restcondf)	/api/v1/rbfs/elements/cnbg-1/services/restconfd/proxy
Prometheus	/api/v1/rbfs/elements/cnbg-1/services/prometheus/proxy

1.6.2. RESTCONF API: Use Cases and Examples

Reading the Current Configuration

The following request returns the complete switch configuration.

```
GET /api/v1/rbfs/elements/BNG/services/restconfd/proxy/restconf/data HTTP/1.1
Host: 10.0.0.1:19091
```

Reading the Time-series Settings

The RESTCONF Protocol RFC describes a comprehensive query syntax, which allows retrieving certain parts of the configuration.

The following API call retrieves all the time-series settings.

```
GET /api/v1/rbfs/elements/BNG/services/restconfd/proxy/restconf/data/rtbrick-
config:time-series HTTP/1.1
Host: 10.0.0.1:19091
```

```
{
    "rtbrick-config:time-series": {
        "metric": [
            {
                "name": "subscriber_sessions",
                "table-name": "local.access.subscriber.count",
                "bds-type": "object-metric",
                "prometheus-type": "gauge",
                "description": "Established subscriber sessions",
                "attribute": [
                    {
                        "attribute-name": "ipoe_established",
                        "label": [
                            {
                                "label-key": "access_type",
                                "label-value": "ipoe",
                                "label-type": "static"
                            },
                            {
                                "label-key": "ifp_name",
                                "label-value": "ifp_name",
                                "label-type": "dynamic"
                            }
                        ]
                    }
                ]
            },
            {
                "name": "total_cpu_util_percent",
                "table-name": "global.chassis_0.resource.cpu_usage",
                "bds-type": "object-metric",
                "prometheus-type": "gauge",
                "description": "Total CPU utilization in percent",
                "attribute": [
                    {
                        "attribute-name": "total_cpu",
                        "label": [
                            {
                                "label-key": "cpu",
                                "label-value": "cpu_id",
                                "label-type": "dynamic"
                            }
                        ]
                    }
                ]
            },
            {
                "name": "total_memory_free_kilobyte",
                "table-name": "global.chassis_0.resource.mem_usage",
                "bds-type": "object-metric",
                "prometheus-type": "gauge",
                "description": "Total free RAM memory in kilo bytes",
                "attribute": [
                    {
                        "attribute-name": "free_mem",
                        "label": [
                            {
                                "label-key": "memory",
                                "label-value": "mem_id",
                                "label-type": "dynamic"
                            }
                        ]
                    }
                ]
            },
            {
                "name": "total_memory_used_kilobyte",
                "table-name": "global.chassis_0.resource.mem_usage",
                "bds-type": "object-metric",
                "prometheus-type": "gauge",
                "description": "Total used RAM memory in kilo bytes",
                "attribute": [
                    {
                        "attribute-name": "used_mem",
                        "label": [
                            {
                                "label-key": "memory",
                                "label-value": "mem_id",
                                "label-type": "dynamic"
                            }
                        ]
                    }
                ]
            }
        ]
    }
}
```

RESTCONF also allows filtering for a specific time series.

```
GET /api/v1/rbfs/elements/BNG/services/restconfd/proxy/restconf/data/rtbrick-
config:time-series/metric=subscriber_sessions HTTP/1.1
Host: 10.0.0.1:19091
```

```
{
    "rtbrick-config:metric": [
        {
            "name": "subscriber_sessions",
            "table-name": "local.access.subscriber.count",
            "bds-type": "object-metric",
            "prometheus-type": "gauge",
            "description": "Established subscriber sessions",
            "attribute": [
                {
                    "attribute-name": "ipoe_established",
                    "label": [
                        {
                            "label-key": "access_type",
                            "label-value": "ipoe",
                            "label-type": "static"
                        },
                        {
                            "label-key": "ifp_name",
                            "label-value": "ifp_name",
                            "label-type": "dynamic"
                        }
                    ]
                }
            ]
        }
    ]
}
```

In addition, RESTCONF allows querying the key values. The following API call selects the names of the configured time-series metrics.

```
GET /api/v1/rbfs/elements/BNG/services/restconfd/proxy/restconf/data/rtbrick-
config:time-series/metric=/name HTTP/1.1
Host: 10.0.0.1:19091
```

```
{
    "rtbrick-config:name": [
        "subscriber_sessions",
        "total_cpu_util_percent",
        "total_memory_free_kilobyte",
        "total_memory_used_kilobyte"
    ]
}
```

Adding a New Time-series

The RESTCONF API also allows replacing and adding new configurations to an existing configuration. The data is exchanged in JSON format.

> The Content-Type header must be set to application/yang-data+json.

YANG is a modeling language that is used to describe the configurations. The RESTCONF Open API definition is generated from the YANG models and contains a reference to the respective YANG model.

The following API request adds a new default_bgp_prefixes_count time-series to the RBFS configuration.

```
PUT /api/v1/rbfs/elements/BNG/services/restconfd/proxy/restconf/data/rtbrick-
config:time-series/metric=default_bgp_prefixes_count HTTP/1.1
Host: 10.0.0.1:19091
Content-Type: application/yang-data+json
Content-Length: 4315

{
    "rtbrick-config:metric": [
        {
            "name": "default_bgp_prefixes_count",
            "table-name": "default.bgp.peer",
            "bds-type": "object-metric",
            "prometheus-type": "gauge",
            "description": "BGP peerings default instance",
            "attribute": [
                {
                    "attribute-name": "ipv4_unicast_update_rcvd_cnt",
                    "label": [
                        {
                            "label-key": "afi",
                            "label-value": "ipv4",
                            "label-type": "static"
                        },
                        {
                            "label-key": "direction",
                            "label-value": "in",
                            "label-type": "static"
                        },
                        {
                            "label-key": "peer",
                            "label-value": "peer_ipv4_address",
                            "label-type": "dynamic"
                        },
                        {
                            "label-key": "safi",
                            "label-value": "unicast",
                            "label-type": "static"
                        }
                    ]
                },
                {
                    "attribute-name": "ipv4_unicast_update_sent_cnt",
                    "label": [
                        {
                            "label-key": "afi",
                            "label-value": "ipv4",
                            "label-type": "static"
                        },
                        {
                            "label-key": "direction",
                            "label-value": "out",
                            "label-type": "static"
                        },
                        {
                            "label-key": "peer",
                            "label-value": "peer_ipv4_address",
                            "label-type": "dynamic"
                        },
                        {
                            "label-key": "safi",
                            "label-value": "unicast",
                            "label-type": "static"
                        }
                    ]
                },
                {
                    "attribute-name": "ipv6_unicast_update_rcvd_cnt",
                    "label": [
                        {
                            "label-key": "afi",
                            "label-value": "ipv6",
                            "label-type": "static"
                        },
                        {
                            "label-key": "direction",
                            "label-value": "in",
                            "label-type": "static"
                        },
                        {
                            "label-key": "peer",
                            "label-value": "peer_ipv4_address",
                            "label-type": "dynamic"
                        },
                        {
                            "label-key": "safi",
                            "label-value": "unicast",
                            "label-type": "static"
                        }
                    ]
                },
                {
                    "attribute-name": "ipv6_unicast_update_sent_cnt",
                    "label": [
                        {
                            "label-key": "afi",
                            "label-value": "ipv6",
                            "label-type": "static"
                        },
                        {
                            "label-key": "direction",
                            "label-value": "out",
                            "label-type": "static"
                        },
                        {
                            "label-key": "peer",
                            "label-value": "peer_ipv4_address",
                            "label-type": "dynamic"
                        },
                        {
                            "label-key": "safi",
                            "label-value": "unicast",
                            "label-type": "static"
                        }
                    ]
                }
            ]
        }
    ]
}
```

The API returns a 201 Created response if a new metric was added and returns 204 No Content if an existing metric got updated.

Removing a Time-series

The following API request removes the default_bgp_prefixes_count time series from the switch configuration.

```
DELETE /api/v1/rbfs/elements/BNG/services/restconfd/proxy/restconf/data/rtbrick-
config:time-series/metric=default_bgp_prefixes_count HTTP/1.1
Host: 10.0.0.1:19091
```

The API returns a 204 No Content response if the delete operation succeeded. Any API call to remove a configuration that does not exist results in a 409 Conflict error response.

Example Response:

```
{
    "ietf-restconf:errors": {
        "error": {
            "error-type": "application",
            "error-tag": "data-missing",
            "error-severity": "error",
            "error-message": "Data does not exist; cannot delete resource"
        }
    }
}
```

1.6.3. Operational State API: Use Cases and Examples

The operational state API provides access to operational state data, including routing protocols, subscriber, and system state information. The operational state is a runtime information and will be reset after each reboot.

> ℹ The operational state API is accessed through the CtrlD proxy endpoint.

> 💡 Use Prometheus for operational state monitoring and metric sampling.

Querying Subscriber Sessions

The following example shows how to read up to five (limit=5) active subscriber sessions on port *ifp-0/0/0* (ifp_name=ifp-0/0/0).

```
GET /api/v1/rbfs/elements/BNG/services/opsd/proxy/subscribers?ifp_name=ifp-
0/0/0&limit=5 HTTP/1.1
Host: 10.0.0.1:19091
```

The API call uses the CtrlD proxy endpoint to invoke the Operational State API, the element name is *BNG* and the service name is *opsd*.

The following example shows five IPoE subscriber sessions.

```
[
    {
        "subscriber_id": 216454257090494480,
        "subscriber_id_str": "216454257090494480",
        "subscriber_state": "ESTABLISHED",
        "subscriber_user_name": "02:00:00:00:00:06@ipoe",
        "access_type": "IPoE",
        "accounting_session_id": "216454257090494480:1695654885",
        "ifp_name": "ifp-0/0/0",
        "outer_vlan": 128,
        "inner_vlan": 6,
        "client_mac": "02:00:00:00:00:06",
        "agent_remote_id": "DEU.RTBRICK.6",
        "agent_circuit_id": "0.0.0.0/0.0.0.0 eth 0:6"
    },
    {
        "subscriber_id": 216454257090494481,
        "subscriber_id_str": "216454257090494481",
        "subscriber_state": "ESTABLISHED",
        "subscriber_user_name": "02:00:00:00:00:07@ipoe",
        "access_type": "IPoE",
        "accounting_session_id": "216454257090494481:1695654885",
        "ifp_name": "ifp-0/0/0",
        "outer_vlan": 128,
        "inner_vlan": 7,
        "client_mac": "02:00:00:00:00:07",
        "agent_remote_id": "DEU.RTBRICK.7",
        "agent_circuit_id": "0.0.0.0/0.0.0.0 eth 0:7"
    },
    {
        "subscriber_id": 216454257090494482,
        "subscriber_id_str": "216454257090494482",
        "subscriber_state": "ESTABLISHED",
        "subscriber_user_name": "02:00:00:00:00:08@ipoe",
        "access_type": "IPoE",
        "accounting_session_id": "216454257090494482:1695654885",
        "ifp_name": "ifp-0/0/0",
        "outer_vlan": 128,
        "inner_vlan": 8,
        "client_mac": "02:00:00:00:00:08",
        "agent_remote_id": "DEU.RTBRICK.8",
        "agent_circuit_id": "0.0.0.0/0.0.0.0 eth 0:8"
    },
    {
        "subscriber_id": 216454257090494483,
        "subscriber_id_str": "216454257090494483",
        "subscriber_state": "ESTABLISHED",
        "subscriber_user_name": "02:00:00:00:00:09@ipoe",
        "access_type": "IPoE",
        "accounting_session_id": "216454257090494483:1695654885",
        "ifp_name": "ifp-0/0/0",
        "outer_vlan": 128,
        "inner_vlan": 9,
        "client_mac": "02:00:00:00:00:09",
        "agent_remote_id": "DEU.RTBRICK.9",
        "agent_circuit_id": "0.0.0.0/0.0.0.0 eth 0:9"
    },
    {
        "subscriber_id": 216454257090494484,
        "subscriber_id_str": "216454257090494484",
        "subscriber_state": "ESTABLISHED",
        "subscriber_user_name": "02:00:00:00:00:0a@ipoe",
        "access_type": "IPoE",
        "accounting_session_id": "216454257090494484:1695654885",
        "ifp_name": "ifp-0/0/0",
        "outer_vlan": 128,
        "inner_vlan": 10,
        "client_mac": "02:00:00:00:00:0a",
        "agent_remote_id": "DEU.RTBRICK.10",
        "agent_circuit_id": "0.0.0.0/0.0.0.0 eth 0:10"
    }
]
```

Each subscriber session has a unique subscriber ID and the subscriber ID is an unsigned 64-bit integer.

The subscriber_id holds the numeric subscriber ID value, while subscriber_id_str contains a string representation of the subscriber ID.

> Some tools and programming libraries comply I-JSON standard. This standard defines numeric values as double-precision floating-point numbers. As a result, subscriber ID values will be rounded and may confuse. It is recommended to read the subscriber ID from the subscriber_id_str in such environments.

Terminating a Subscriber Session

The following API call terminates the subscriber session for the subscriber with the subscriber ID: 216454257090494484.

```
DELETE
/api/v1/rbfs/elements/BNG/services/opsd/proxy/subscribers/216454257090494484
HTTP/1.1
Host: 10.0.0.1:19091
```

The switch returns a 202 Accepted status code to acknowledge that the session is going to be terminated.

1.6.4. Prometheus: Use Cases and Examples

Accessing the Federation Endpoint

The Prometheus federation endpoint returns all metrics collected by Prometheus in the Prometheus Exposition Format.

```
GET
/api/v1/rbfs/elements/BNG/services/prometheus/proxy/federate?match%5B%5D=%7Bjob%3D
%22bds%22%7D HTTP/1.1
Host: 10.0.0.1:19091
```

> The match condition is required to select all BDS metrics that is metrics collected from brick daemons and the brick data store (BDS).

1.7. Related Documentation

/ONIE/	The ONIE documentation outlines the DHCP options supported for image discovery. https://opencomputeproject.github.io/onie/design-spec/discovery.html
/ZTP/	The /resources/techdocs/current/tools/rbfs-ztp.html[Zero-Touch Provisioning Guide] outlines the current configuration discovery process.
/SEC/	The /resources/techdocs/current/secmgmt/secmgmt_intro.html[Securing the Management Plane Guide] Secure the Management Plane guide gives a detailed insight on this topic.
/RADIUS/	The /resources/techdocs/current/radiusservices/radiusservices_intro.html[RADIUS Services Guide] provides an overview of the supported RADIUS attributes including a reference to the RFC that defines the message attribute.
/CTRLD/	The CTRLD API reference describes all CTRLD REST API endpoints in detail. To view the CtrlD API Reference, navigate to RBFS APIs, and select CTRLD API Reference from the drop-down list.
/RESTCONF/	The RESTCONF API reference describes all configuration API endpoints. To view the RESTCONF API Reference, navigate to RBFS APIs, and select RESTCONF API Reference from the drop-down list.
/GELF/	The Graylog Extended Log Format (GELF) is a log format, this document outlines the fundamentals. To obtain this document, contact your customer support team.

2. RBFS APIs

```
    <link rel="stylesheet" type="text/css" href="./_attachments/swagger-ui.css">
    <link rel="stylesheet" type="text/css" href="./_attachments/rtbrick-swagger.css">
    <div id="swagger-ui"></div>
    <script src="./_attachments/swagger-ui-bundle.js"></script>
    <script src="./_attachments/swagger-ui-standalone-preset.js"></script>
    <script>
    window.onload = function () {
      const DisableTryItOutPlugin = function() {
        return {
          statePlugins: {
            spec: {
              wrapSelectors: {
                allowTryItOutFor: () => () => false
              }
            }
          }
        }
      }

      // Begin Swagger UI call region
      const ui = SwaggerUIBundle({
        urls: [
          { "url": "./_attachments/rbfs/openapi_ctrld.yaml", "name": "CTRLD API Reference" },
          { "url": "./_attachments/rbfs/rtbrick-config_restconf_swagger.json", "name": "RESTCONF API Reference" },
          { "url": "./_attachments/rbfs/swagger_opsd.yaml", "name": "Operational State API Reference" },
          { "url": "./_attachments/rbfs/swagger_bds.yaml", "name": "BDS API Reference" },
        ],
        dom_id: '#swagger-ui',
        deepLinking: true,
        docExpansion:"none",
        presets: [
          SwaggerUIBundle.presets.apis,
          SwaggerUIStandalonePreset
        ],
        plugins: [
          SwaggerUIBundle.plugins.DownloadUrl,
          DisableTryItOutPlugin
        ],
        layout: "StandaloneLayout"
      })
      // End Swagger UI call region

      window.ui = ui
    }
    </script>
```

3. RBMS APIs

```
  <link rel="stylesheet" type="text/css" href="./_attachments/swagger-ui.css">
  <link rel="stylesheet" type="text/css" href="./_attachments/rtbrick-swagger.css">

  <div id="swagger-ui"></div>

  <script src="./_attachments/swagger-ui-bundle.js"> </script>
  <script src="./_attachments/swagger-ui-standalone-preset.js"> </script>
  <script>
    window.onload = function () {
      const DisableTryItOutPlugin = function() {
        return {
          statePlugins: {
            spec: {
              wrapSelectors: {
                allowTryItOutFor: () => () => false
              }
            }
          }
        }
      }

      // Begin Swagger UI call region
      const ui = SwaggerUIBundle({
        urls: [
          { "url": "./_attachments/rbms/swagger_ztp_mgmt.yaml", "name": "ZTP
Management Server API Reference" },
          { "url": "./_attachments/rbms/swagger_leitstand_template_engine.yaml",
"name": "Template Engine API Reference" },
          { "url": "./_attachments/rbms/metric.yaml", "name": "Resource Inventory
Metric API" },
          { "url": "./_attachments/rbms/meta.yaml", "name": "Resource Matadata API"
},
          { "url": "./_attachments/rbms/jobs.yaml", "name": "Job Management API" },
          { "url": "./_attachments/rbms/image.yaml", "name": "Resource Inventory
Image API" },
          { "url": "./_attachments/rbms/group.yaml", "name": "Resource Inventory
Element Group API" },
          { "url": "./_attachments/rbms/facility.yaml", "name": "Resource Inventory
Rack and Facility API" },
          { "url": "./_attachments/rbms/element.yaml", "name": "Resource Inventory
Element API" },
          { "url": "./_attachments/rbms/dns.yaml", "name": "DNS Zone API" },
          { "url": "./_attachments/rbms/commons.yaml", "name": "commons" },
        ],
        dom_id: '#swagger-ui',
        deepLinking: true,
        docExpansion:"none",
        presets: [
          SwaggerUIBundle.presets.apis,
          SwaggerUIStandalonePreset
        ],
        plugins: [
          SwaggerUIBundle.plugins.DownloadUrl,
          DisableTryItOutPlugin
        ],
        layout: "StandaloneLayout"
      })
      // End Swagger UI call region

      window.ui = ui
    }
  </script>
```

Registered Address	Support	Sales
40268, Dolerita Avenue Fremont CA 94539		
+1-650-351-2251		+91 80 4850 5445
http://www.rtbrick.com	support@rtbrick.com	sales@rtbrick.com

©Copyright 2024 RtBrick, Inc. All rights reserved. The information contained herein is subject to change without notice. The trademarks, logos and service marks ("Marks") displayed in this documentation are the property of RtBrick in the United States and other countries. Use of the Marks are subject to RtBrick’s Term of Use Policy, available at https://www.rtbrick.com/privacy. Use of marks belonging to other parties is for informational purposes only.