# Zero-Touch Provisioning (ZTP) Guide

**Version 20.9.1, 25 September 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 |

©Copyright 2020 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.

# Table of Contents

1. Introduction to ZTP . . . 3
2. ZTP in a Nutshell . . . 4
  2.1. DHCPD . . . 6
  2.2. HTTPD (Management Server) . . . 7
    2.2.1. Nginx base configuration . . . 7
    2.2.2. ZTP configuration . . . 8
  2.3. HTTPD (Management Server) . . . 10
    2.3.1. Nginx base configuration . . . 10
    2.3.2. ZTP configuration . . . 11
  2.4. CTRLD . . . 14
    2.4.1. Trigger the ZTP process . . . 15
    2.4.2. Management Server URL Discovery . . . 15
    2.4.3. Request the configurations . . . 15
    2.4.4. Business Events . . . 16
    2.4.5. Overall Process Flow . . . 18
3. References . . . 21

# 1. Introduction to ZTP

A major goal in any network is a high level of automation. This includes the automatic provisioning of switches newly installed in the network, a process known as **Z**ero-**T**ouch **P**rovisioning (**ZTP**).

A new switch comes preinstalled with the **O**pen **N**etwork **I**nstallation **E**nvironment (**ONIE**). The ONIE is an open source installation environment that acts as an enhanced boot loader utilizing facilities in a Linux/BusyBox environment. This small Linux operating system allows end-users and channel partners to install the target **N**etwork **OS** (**NOS**) as part of provisioning.

Because ONIE needs the ability to obtain the configuration and image binaries through the management interface, it requires a management LAN.

ONIE has access only to the management interface. ONIE starts a **D**ynamic **H**ost **C**onfiguration **P**rotocol (**DHCP**)-based discovery process to obtain basic configuration information, such as the management IP-Address and the URL of the image to install on the switch. Then ONIE pulls the image and boots it.

Even after the ONIE boots the image, the switch is not configured. This leads to questions about how to configure the switch. The RtBrick images come with some pre-installed daemons. The preinstalled **C**on**tr**o**l** **D**aemon (**CTRLD**) is responsible for the management of the switch, and takes over after the image is activated. This daemon is responsible for configuring the switch properly.

To do this, a hardware box needs to connect to a DHCP server and a management server through the management LAN.

The management server is responsible for providing the image binaries and the configuration of each device.

In summary, there are two major steps in the ZTP process:

- ONIE:
  - DHCP discovery
  - Image download
  - Image activation
- CTRLD
  - DHCP discovery
  - Switch configuration

# 2. ZTP in a Nutshell

This section describes the ZTP process in a Nutshell, figure 1 illustrates the process at a high level.

The process is split into two main parts:

- ONIE image discovery and Installation.
  - ONIE uses DHCP to discover the IP address along with the image download URL based on the provided DHCP options. For download, ONIE allows different ways to pull an image from the repository. In this ZTP process, HTTP is used to pull the image because ONIE conveys the serial number as the HTTP header. This serial number allows the image repository to identify the switch and select the appropriate image.
  - See the ONIE image discovery for further information (/ONIE/)
- CTRLD configuration discovery and application.
  - CTRLD sends DHCPINFORM to ask for all options needed for config discovery.
  - The configurations are downloaded from the management server (HTTPD) and applied.

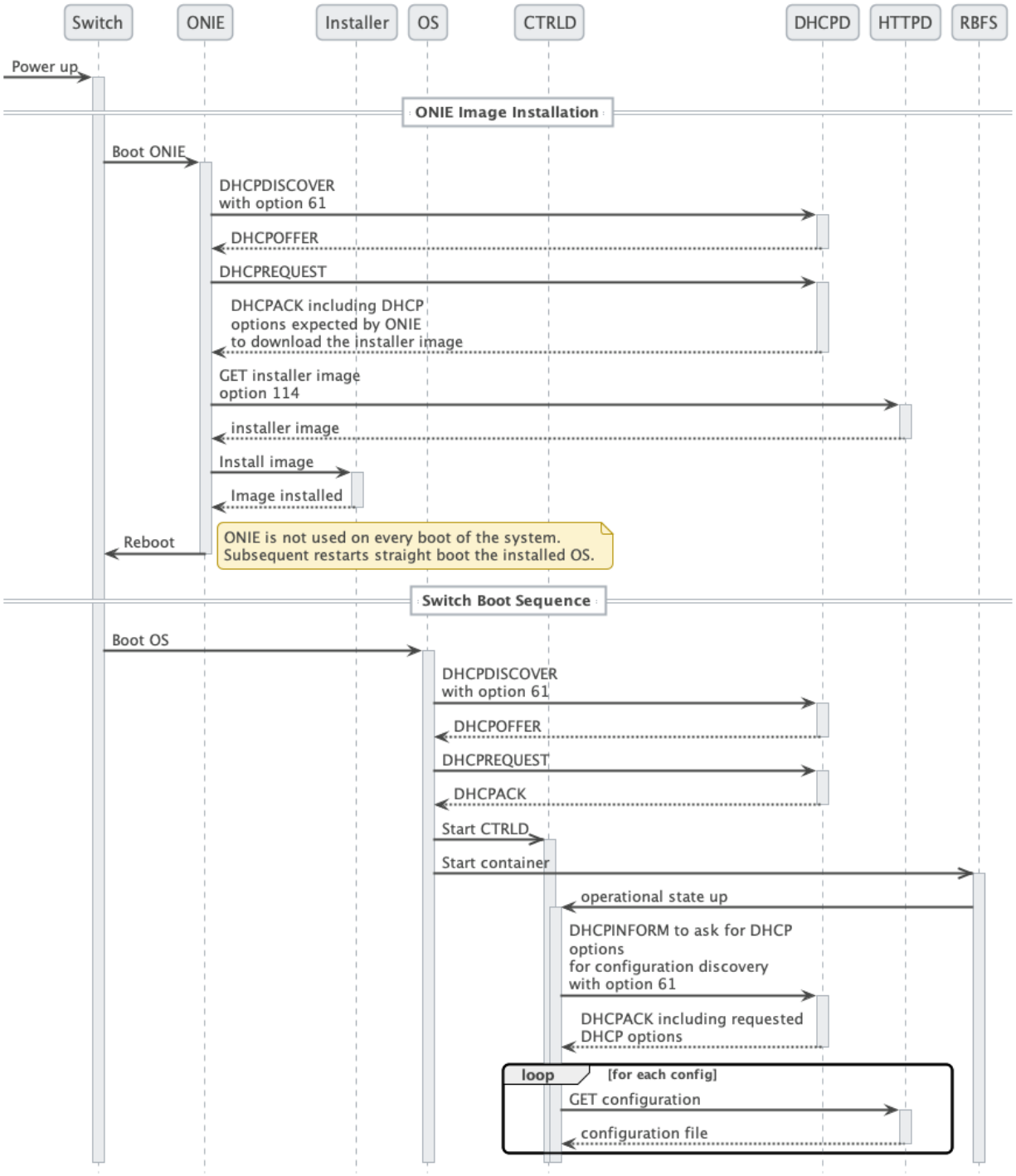


*Figure 1. The ZTP Process*

Figure 2. depicts the relationship between the fabric, the DHCP server and the Management Server.

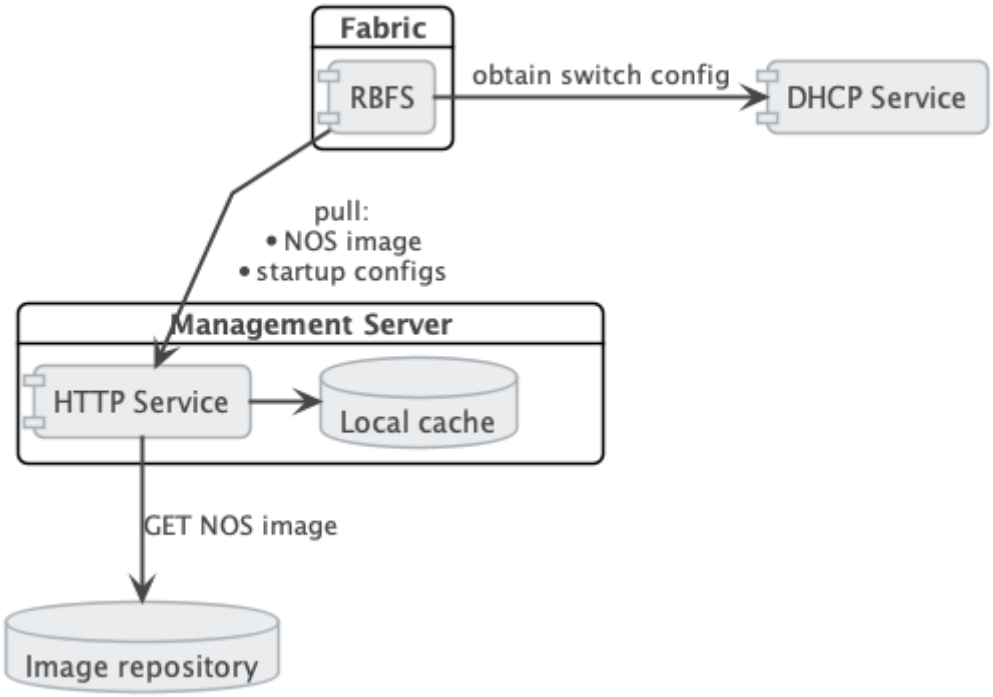


*Figure 2. The Management Server Architecture*

## 2.1. DHCPD

Because of its low set of requirements, the default DHCP server shipped with ubuntu, isc-dhcp, is used to run the DHCP service.

The following code shows an example configuration of a DHCP server and hardware box (**dhcp.conf**).

*dhcp.conf*

```
authoritative;
default-lease-time 600;
max-lease-time 72----

# This is only needed if the version is lower than 4.4
option loader-pathprefix code 210 = text;


subnet 10.0.0.0 netmask 255.255.255.0 {
  range 10.0.0.200 10.0.0.250;
  option routers 10.0.0.138;
  option domain-name-servers 10.0.0.210;
  option domain-name "local";
  host LEAF01 {
    # Identify client by MAC address.
    hardware ethernet 48:65:ee:11:da:85;
    fixed-address 10.0.0.250;
    option host-name LEAF01;
    # Set DHCP option 114 (default-url) to set the installer image URL.
    # ONIE loads the installer image from the specified URL.
    option default-url "http://managementserver/ztp/image";
    # Set DHCP option 210 (path prefix) to set the configuration base URL.
    # CTRLD loads all configuration files from this base URL.
    option loader-pathprefix "http://managementserver";
  }
}
```

Most of the used options are already predefined in the ISC-DHCP server please find a Reference under /ISCKB/ , the loader-pathprefix is defined since DHCP 4.4, so if you use an older one please define it as described above.

## 2.2. HTTPD (Management Server)

The HTTP service is responsible to provide the NOS installer and the configuration files. Therefore, an open-source HTTP Server (nginx) is used. Nginx is configured to read the ONIE_SERIAL_NUMBER HTTP header and maps the serial number to the NOS installer image download path, and maps the serial number to the configuration files.

This section describes the configuration of the nginx server.

### 2.2.1. Nginx base configuration

The nginx.conf file shows the basic configuration.

*/etc/nginx/nginx.conf*

```
user  nginx;
worker_processes  1;

error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;
# Load the javascript module which is used in the ztp
# specific configuration
load_module modules/ngx_http_js_module.so;

events {
    worker_connections  1024;
}


http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;
    # ztp logformat with ONIE-SERIAL-NUMBER header
    # CTRLD sends additional headers other than the serial number
    # that could also be used
    log_format  ztp   '$remote_addr - $remote_user [$time_local] '
                      '[onie=$http_onie_serial_number] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';
    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';


    access_log  /var/log/nginx/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    keepalive_timeout  65;

    #gzip  on;

    include /etc/nginx/conf.d/*.conf;
}
```

## 2.2.2. ZTP configuration

The rtb_ztp.js file shows the javascript module for mapping the ONIE-SERIAL-NUMBER header to the configuration files.

Therefore, for each serial number a <serial_number>.json file is read. This files contains the files which have to be delivered for the specific device.

*/etc/nginx/conf.d/rtb_ztp.js*

```
var fs = require("fs");
var configFolder = "/usr/share/nginx/html/configs/";

function open_db(r,serial) {
    var data, map;
    var file = configFolder+serial+".json";
    try {
        data = fs.readFileSync(file);
    } catch (e) {
        r.warn("not able to find " + file)
        throw Error("open_db: " + e);
    }
    try {
        map = JSON.parse(data);
    } catch (e) {
        r.error("not able to parse " + file)
        throw Error("open_db: " + e);
    }
    return map;
}

function resolve(r, what) {
    try {
        var map = open_db(r, r.headersIn["onie-serial-number"]);
        return map[what];
    } catch (e) {
        return "not found";
    }
}

function resolveCTRLD(r) {
    return resolve(r,"ctrld")
}

function resolveCTRLDRBAC(r) {
    return resolve(r,"ctrldrbac")
}

function resolveElement(r) {
    return resolve(r,"element")
}

function resolveStartup(r) {
    return resolve(r,"startup")
}

function resolveImage(r) {
    return resolve(r,"image")
}
```

## 2.3. HTTPD (Management Server)

The HTTP service is responsible to provide the NOS installer and the configuration files. Therefore, an open-source HTTP Server (nginx) is used. Nginx is configured to read the ONIE_SERIAL_NUMBER HTTP header and maps the serial number to the NOS installer image download path, and maps the serial number to the configuration files.

This section describes the configuration of the nginx server.

### 2.3.1. Nginx base configuration

The nginx.conf file shows the basic configuration.

*/etc/nginx/nginx.conf*

```
user  nginx;
worker_processes  1;

error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;
# Load the javascript module which is used in the ztp
# specific configuration
load_module modules/ngx_http_js_module.so;

events {
    worker_connections  1024;
}


http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;
    # ztp logformat with ONIE-SERIAL-NUMBER header
    # CTRLD sends additional headers other than the serial number
    # that could also be used
    log_format  ztp   '$remote_addr - $remote_user [$time_local] '
                      '[onie=$http_onie_serial_number] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';
    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';


    access_log  /var/log/nginx/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    keepalive_timeout  65;

    #gzip  on;

    include /etc/nginx/conf.d/*.conf;
}
```

## 2.3.2. ZTP configuration

The rtb_ztp.js file shows the javascript module for mapping the ONIE-SERIAL-NUMBER header to the configuration files.

Therefore, for each serial number a <serial_number>.json file is read. This files contains the files which have to be delivered for the specific device.

*/etc/nginx/conf.d/rtb_ztp.js*

```
var fs = require("fs");
var configFolder = "/usr/share/nginx/html/configs/";

function open_db(r,serial) {
    var data, map;
    var file = configFolder+serial+".json";
    try {
        data = fs.readFileSync(file);
    } catch (e) {
        r.warn("not able to find " + file)
        throw Error("open_db: " + e);
    }
    try {
        map = JSON.parse(data);
    } catch (e) {
        r.error("not able to parse " + file)
        throw Error("open_db: " + e);
    }
    return map;
}

function resolve(r, what) {
    try {
        var map = open_db(r, r.headersIn["onie-serial-number"]);
        return map[what];
    } catch (e) {
        return "not found";
    }
}

function resolveCTRLD(r) {
    return resolve(r,"ctrld")
}

function resolveCTRLDRBAC(r) {
    return resolve(r,"ctrldrbac")
}

function resolveElement(r) {
    return resolve(r,"element")
}

function resolveStartup(r) {
    return resolve(r,"startup")
}

function resolveImage(r) {
    return resolve(r,"image")
}
```

Example sample.json file:

*/usr/share/nginx/html/configs/sample.json*

```
{
  "description": "192.168.202.116",
  "ctrld": "ctrld.json",
  "ctrldrbac": "ctrldrbac.json",
  "startup": "sample_startup.json",
  "element": "sample_element.json",
  "image": "http://pkg.rtbrick.net/_/images/latest/rtbrick-onl-
installer/rtbrick-onl-installer-accessleaf-qmx-20.4.0-
g8daily.20200415051734+Bmaster.C059a09ea"
}
```

The default.conf file describes the ztp server configuration. Be aware of the equals match for the locations (location = /ztp/config/ctrld), otherwise the url for *ctrld* would also match for *ctrldrbac* which would result in a miss configuration.

*/etc/nginx/conf.d/default.conf*

```
js_include conf.d/rtb_ztp.js;

js_set $ctrld resolveCTRLD;
js_set $ctrldrbac resolveCTRLDRBAC;
js_set $element resolveElement;
js_set $startup resolveStartup;
js_set $image resolveImage;

server {
    listen       80;
    server_name  localhost;

    root   /usr/share/nginx/html;
    #charset koi8-r;
    error_log  /var/log/nginx/ztp.error.log  warn;
    access_log  /var/log/nginx/ztp.access.log  ztp;

    location = /ztp/config/startup {
        try_files /configs/$startup =404;
        error_page 405 =200 $uri;
    }

    location = /ztp/config/ctrld {
        try_files /configs/$ctrld =404;
        error_page 405 =200 $uri;
    }

    location = /ztp/config/ctrldrbac {
        try_files /configs/$ctrldrbac =404;
        error_page 405 =200 $uri;
    }

    location = /ztp/config/element {
        try_files /configs/$element =404;
        error_page 405 =200 $uri;
    }
```

```
    location = /ztp/image {
        return 302 $image;
    }

    #endpoint for upload and download
    location /ztp/files {
        rewrite /ztp/(.*) /$1  break;
        proxy_pass http://127.0.0.1:8080;
    }

    #legacy endpoint
    location /ztp/message {
        return 204;
    }

    location / {
        index  index.html index.htm;
    }
    #error_page  404              /404.html;

    # redirect server error pages to the static page /50x.html
    #
    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /usr/share/nginx/html;
    }
}
```

The upload endpoint forwards the requests to the uploadservice, a golang service which provides the ability to upload files to the configuration server.

RtBrick provides a running solution entirely as a self-contained docker container.

## 2.4. CTRLD

CTRLD acts as a post-ZTP demon, it runs after the image is activated. This demon is responsible for configuring the switch properly.

There are various configurations which CTRLD can load from a management server and apply it to the system.

- **CTRLD config**: the base configuration for CTRLD. There the RBMS and Graylog can be specified, but also the authentication and authorization mechanism can be controlled.
- **Element config**: Each LXC container can be configured for CTRLD. So the element config is the link between the element name and the container. By default, the element name is equal to the container name, but to configure differently, this can be specified also in the element configuration. Also the ZTP process for the element can be disabled via this configuration.
- **CTRLD rbac policy**: the **R**ole **B**ased **A**ccess **C**ontrol (**RBAC**) policy of CTRLD is

specified in this configuration file.

- **Startup Config**: The RBFS switch configuration.

### 2.4.1. Trigger the ZTP process

The ZTP process in CTRLD is triggered for a specific container (LXC) on the switch. This can be triggered in the following ways.

- By the switch (RBFS lxc container) itself by sending the operational state up to CTRLD.
- By sending a REST request to trigger the ZTP process to CTRLD (/api/v1/ctrld/ztp/_run).

If the ZTP Process for the specified container is enabled (element.config), the process starts.

### 2.4.2. Management Server URL Discovery

CTRLD has to discover the management server URL in order to download the configuration from the management server. Therefore, a management interface is defined which allows to send an DHCPINFORM request to the DHCP server.

The request contains the **DHCP option 61** that conveys the client identifier. The client identifier is either omitted or contains the serial number. The serial number is gathered from the onie file system information file /lib/platform-config/current/onl/onie-info.json. If that does not result in a valuable result the following command is executed dmidecode -s system-serial-number (see /RFC2131/ and /RFC2132/ for further information).

There are at least two DHCP options requested, **DHCP option 54** that conveys the IP address of the DHCP server (see /RFC2132/ for further information), and **DHCP option 210** that conveys the path prefix for all configuration files (see /RFC5071/ for further information).

If the DHCP option 210 is not returned, CTRLD attempts to read the configurations from the IP address of the ZTP server. Otherwise, CTRLD attempts to read the configurations from the base URL specified in DHCP option 210.

### 2.4.3. Request the configurations

The request to the Management server contains the following HTTP headers:

- ONIE-SERIAL-NUMBER: This serial number is either the onie serial number or empty string.
- CONTAINER-NAME: Container that triggered the ZTP process.
- ELEMENT-NAME: Element name that triggered the ZTP process.

- HOST-NAME: Host name of the device that triggered the ZTP process.

All this information can be used to select the right configurations for the container. This also allows the use of ZTP Configuration Process for virtual environments.

The requested URL:

- CTRLD Config: <management server url>/ztp/config/ctrld
- Element Config: <management server url>/ztp/config/element
- CTRLD rbac policy: <management server url>/ztp/config/ctrldrbac
- Startup Config: <management server url>/ztp/config/startup If one of the files is not found the process still goes forward.

## 2.4.4. Business Events

During the ZTP Process log messages are sent to the configured ztp graylog endpoint.

*Table 1. GELF message format*

| Name | Type | Mandatory | Description |
|---|---|---|---|
| **version** | String | Yes | The GELF message format version. Default value: 1.1 |
| **host** | String | Yes | The hostname 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 second with 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. |

| Name | Type | Mandatory | Description |
|---|---|---|---|
| **_log_event** | String | Yes | The log event identifies the log message template in the log configuration. The log event simplifies to find 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. |
| **_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. |
| **_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 returned a configuration. |

*Table 2. 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"
}
```

*Table 3. Event Types*

| severity | log_module | log_event | description |
|---|---|---|---|
| Info | ztp | ZTP0011I | ztp ctrld config set |
| Warn | ztp | ZTP0012W | ztp ctrld config not provided |
| Alert | ztp | ZTP0013E | ztp ctrld config not set |
| Info | ztp | ZTP0021I | ztp startup config set |
| Warn | ztp | ZTP0022W | ztp startup config not provided |
| Alert | ztp | ZTP0023E | ztp startup config not set |
| Info | ztp | ZTP0031I | ztp element config set |
| Warn | ztp | ZTP0032W | ztp element config not provided |
| Alert | ztp | ZTP0033E | ztp element config not set |
| Info | ztp | ZTP0041I | ztp ctrld rbac config set |
| Warn | ztp | ZTP0042W | ztp ctrld rbac config not provided |
| Alert | ztp | ZTP0043E | ztp ctrld rbac config not set |

## 2.4.5. Overall Process Flow

The 2 figures below show the CTRLD ZTP process flow.

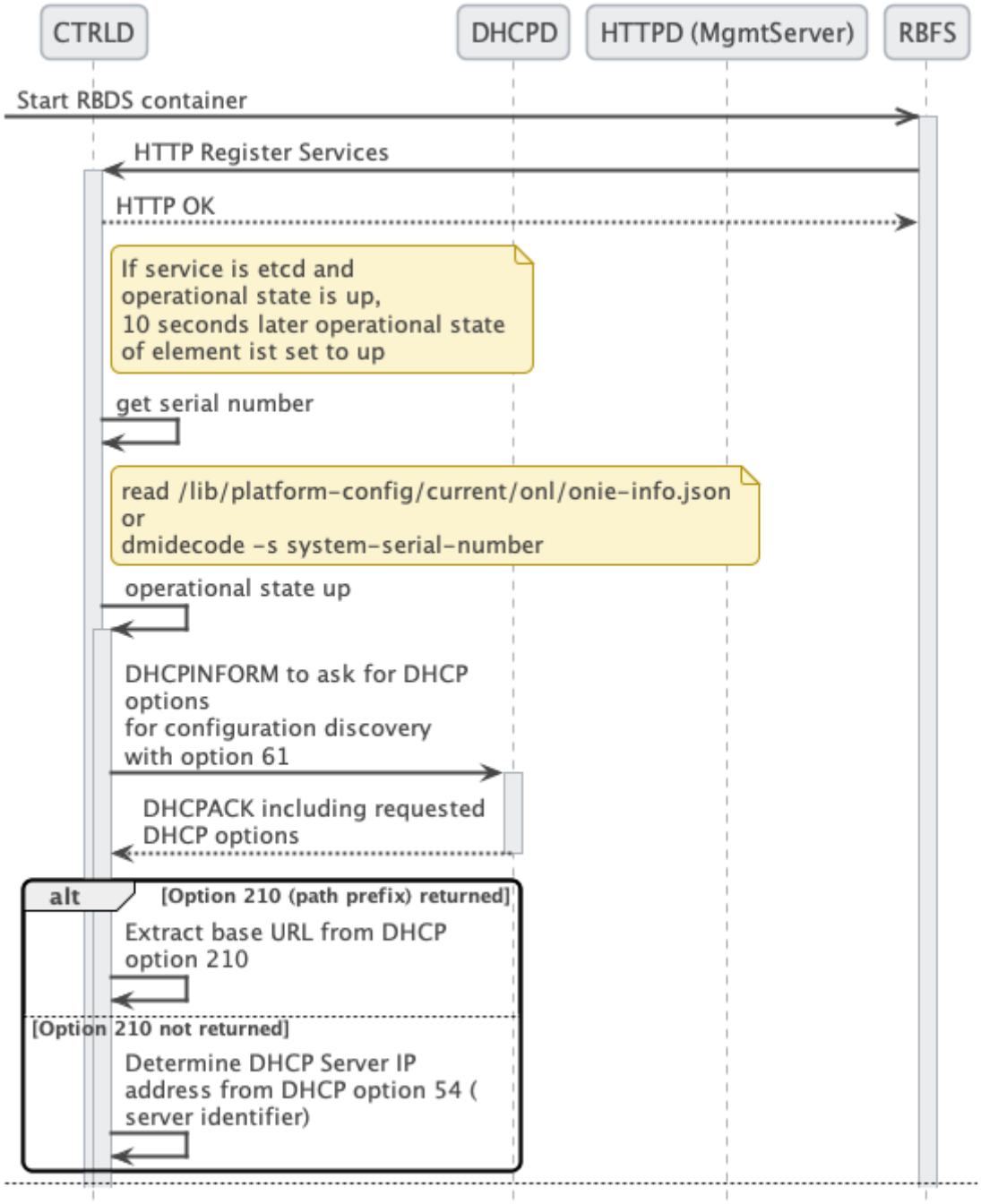


*Figure 3. CTRLD ZTP process flow (Part 1/2)*

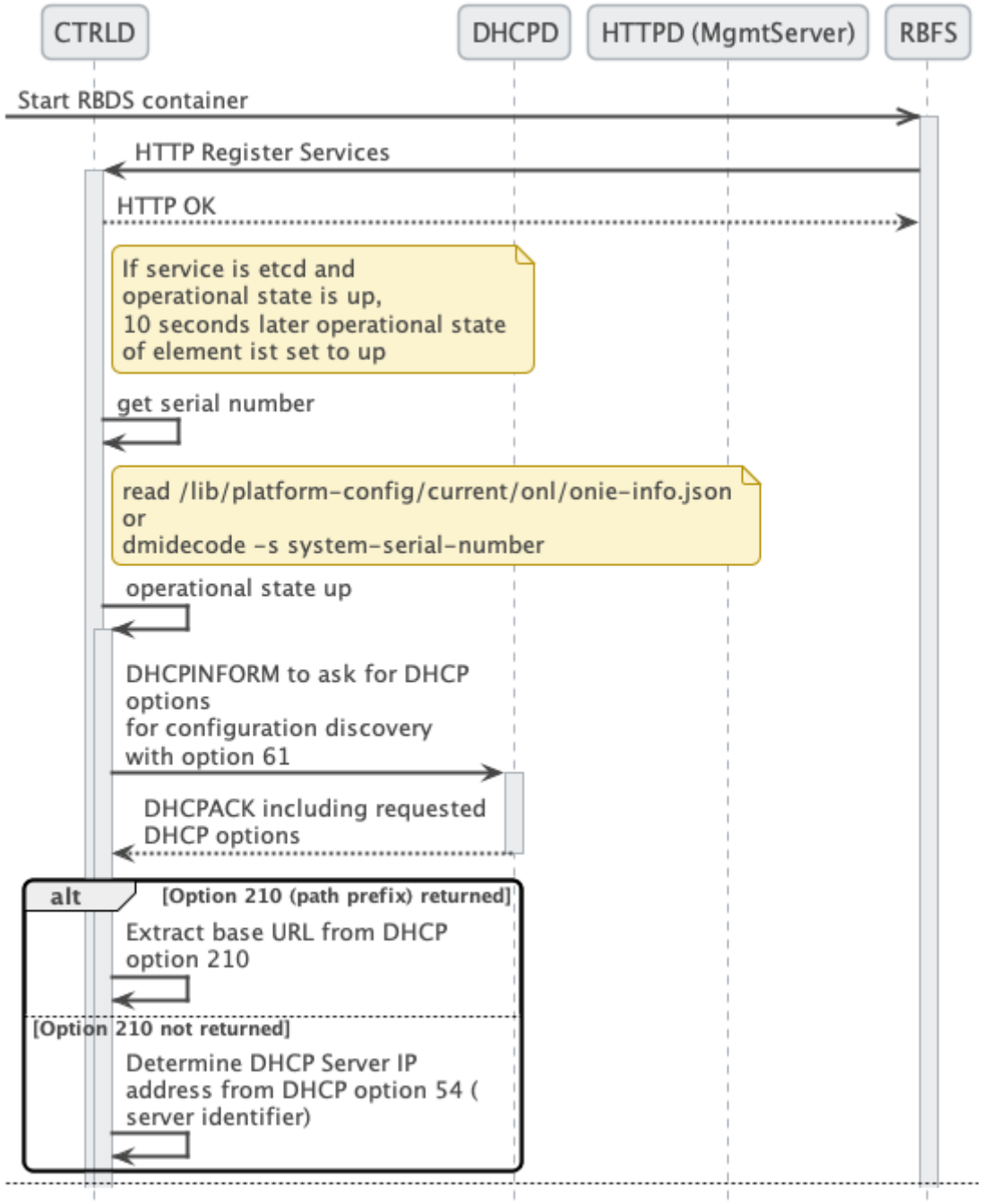


*Figure 4. CTRLD ZTP process flow (Part 2/2)*

# 3. References

*Table 4. References*

| | |
|---|---|
| /ONIE/ | Open Network Installation Environment Image Discovery |
| /RFC2131/ | RFC2131 - Dynamic Host Configuration Protocol |
| /RFC2132/ | RFC2132 - DHCP Options and BOOTP Vendor Extensions https://tools.ietf.org/html/rfc2132 |
| /RFC5071/ | RFC5071 - Dynamic Host Configuration Protocol Options Used by PXELINUX |
| /ISCKB/ | ISC Default DHCP Options |