Configuring Zero Touch Provisioning

Zero Touch Provisioning (ZTP) works as a Third Party App (TPA) in Route-Switch Processor (RSP) and Route Processor (RP). ZTP was designed to perform two different operations:

- Download and apply an initial configuration.
- Download and execute a shell script.

If the downloaded file content starts with **!! IOS XR** it is considered as a configuration file, and ZTP performs **apply_config** action on the configuration file.

If the downloaded file content starts with **#! /bin/bash**, **#! /bin/sh** or **#!/usr/bin/python** it is considered as a script file, and ZTP executes the script.

ZTP works as following:

1. XR scripts that run on boot, invoke DHCP request.
2. DHCP server returns either a user script or configuration file.
3. Download the user script or configuration file.
4. Execute the downloaded user script or apply the downloaded configuration.

Prior to Cisco IOS XR Release 6.3.1, ZTP was executed within the default network namespace and could not access the data interfaces directly. Starting with Cisco IOS XR Release 6.3.1, ZTP is executed inside the global Virtual Routing and Forwarding (VRF) network namespace with full access to all the data interfaces.

When ZTP process encounters any error, or when ZTP quits or terminates, it revert to the initial configuration that exists before starting of ZTP process.

Note

- When initiated, ZTP checks if the system start-up configuration is applied. If startup configuration is not applied, ZTP waits for 10 minutes before proceeding.
- To boot an image through ZTP, configure the ROMMON reboot mode option to 3.

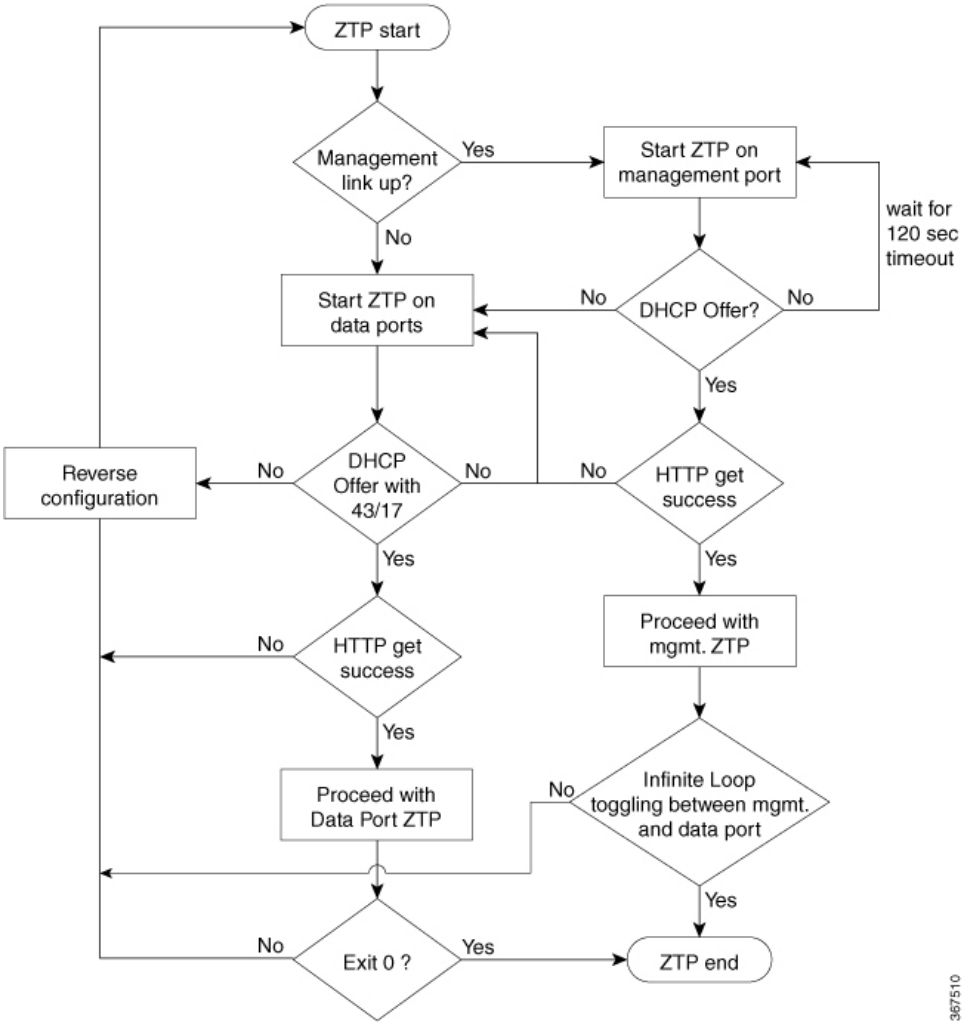
ZTP Switches between Management and Data Port

From Cisco IOS XR Release 6.5.1, during the fresh boot of a router, auto ZTP process is initiated from the management port and switches to data port. The following events cause the ZTP process to switch between management and data port:

- When ZTP does not find an active interface.
- When ZTP does not receive DHCP response and time elapsed since dhclient started is greater than 128 seconds.
- When ZTP encounters an error.

The below flow diagram illustrates the ZTP process.

Figure 1: ZTP Process Flow Sequence



Note

- During fresh boot or manual invocation, ZTP enables IPv6 on all data port interfaces in the dataport mode.
- The auto breakout mode is not supported.
- Starting from Cisco IOS XR Release 6.5.1, auto data port is supported.

Manual ZTP Invocation

Manual Zero Touch Provisioning (ZTP) can be invoked manually via CLI commands. This manual way helps you to provision the router in stages. Ideal for testing out ZTP configuration without a reboot. If you would like to invoke a ZTP on an interfaces(data ports or management port), you don't have to bring up and configure the interface first. You can execute the **ztp initiate** command, even if the interface is down, ZTP script will bring it up and invoke dhclient. So ZTP could run over all interfaces no matter it is up or down.

Use the **ztp initiate**, **ztp breakout**, **ztp terminate**, and **ztp clean** commands to force ZTP to run over more interfaces.

- **ztp initiate**— Invokes a new ZTP session. Logs can be found in **/var/log/ztp.log**.
- **ztp terminate**—Terminates any ZTP session in progress.
- **ztp breakout**—Will peform 4x10 breakout detection.
- **ztp clean**—Removes only the ZTP state files.

  From release 6.2.3, the log file ztp.log is saved in **/var/log** folder, and a copy of log file is available at **/disk0:/ztp/ztp.log** location using a soft link. However, executing **ztp clean** clears files saved on disk and not on **/var/log** folder where current ZTP logs are saved. In order to have a log from current ZTP run, you must manually clear the ZTP log file from **/var/log/** folder.

For more information of the commands, see the ZTP command chapter in the .

This task shows the most common use case of manual ZTP invocation: invoke 4x10 breakout discovery and ZTP.

SUMMARY STEPS

1. **ztp breakout**
2. **ztp initiate dataport**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**ztp breakout** **Example:** `RP/0/RP0/CPU0:router# ztp breakout`	Tries the 4x10 breakout on 100 GE interfaces that supports breakout and are operationally down after no-shut. If the 10x10 breakout configure brings any 10GE interface operationally up, the breakout configuration is retained; if not, the breakout configuration is reverted.

Step 2	**ztp initiate dataport**
	Example:
	`RP/0/RP0/CPU0:router# ztp initiate dataport`
	`Wed Apr 22 10:52:24.417 UTC`
	`Invoke ZTP? (this may change your configuration) [confirm] [y/n] :y`
	`ZTP will now run in the background.`
	`ZTP might bring up the interfaces if they are in shutdown state.`
	`Please use "show logging" or look at /disk0:/ztp/ztp.log to check progress.`
		Invokes DHCP sessions on all data ports that are either up or could be brought up. ZTP runs in the background.

Authentication on Data Ports

On fresh boot, ZTP process is initiated from management ports and may switch to data ports. To validate the connection with DHCP server, authentication is performed on data ports through DHCP option 43 for IPv4 and option 17 for IPv6. These DHCP options are defined in option space and are included within **dhcpd.conf** and **dhcpd6.conf** configuration files. You must provide following parameters for authentication while defining option space:

- Authentication code—The authentication code is either 0 or 1; where 0 indicates that authentication is not required, and 1 indicates that MD5 checksum is required.
- Client identifier—The client identifier must be 'exr-config'.
- MD5 checksum—This is chassis serial number. It can be obtained using **echo -n $SERIALNUMBER | md5sum | awk '{print $1}'**.

Here is the sample **dhcpd.conf** configuration. In the example below, the option space called **VendorInfo** is defined with three parameters for authentication:

```
class "vendor-classes" {
   match option vendor-class-identifier;
}

option space VendorInfo;
option VendorInfo.clientId code 1 = string;
option VendorInfo.authCode code 2 = unsigned integer 8;
option VendorInfo.md5sum code 3 = string
option vendor-specific code 43 = encapsulate VendorInfo;
subnet 10.65.2.0 netmask 255.255.255.0 {
  option subnet-mask 255.255.255.0;
  option routers 10.65.2.1;
  range 10.65.2.1 10.65.2.200;
```

```
}
host xrv9k-1-mgmt {
   hardware ethernet 00:50:60:45:67:01;
   fixed-address 10.65.2.39;
   vendor-option-space VendorInfo;
   option VendorInfo.clientId "exr-config";
   option VendorInfo.authCode 1;
   option VendorInfo.md5sum "aedf5c457c36390c664f5942ac1ae3829";
   option bootfile-name "http://10.65.2.1:8800/admin-cmd.sh";
}
```

Here is the sample **dhcpd6.conf** configuration file. In the example below, the option space called **VendorInfo** is defined that has code width 2 and length width 2 (as per dhcp standard for IPv6) with three parameters for authentication:

```
log-facility local7;
option dhcp6.name-servers 2001:1451:c632:1::1;
option dhcp6.domain-search "cisco.com";
dhcpv6-lease-file-name "/var/lib/dhcpd/dhcpd6.leases";
option dhcp6.info-refresh-time 21600;
option dhcp6.bootfile-url code 59 = string;
option dhcp6.user-class code 15 = string;
option space CISCO-EXR-CONFIG code width 2 length width 2;
option CISCO-EXR-CONFIG.client-identifier code 1 = string;
option CISCO-EXR-CONFIG.authCode code 2 = integer 8;
option CISCO-EXR-CONFIG.md5sum code 3 = string;
option vsio.CISCO-EXR-CONFIG code 9 = encapsulate CISCO-EXR-CONFIG;
subnet6 2001:1451:c632:1::/64{
 range6 2001:1451:c632:1::2 2001:1451:c632:1::9;
 #host NCS5501-2 {
      #host-identifier option dhcp6.client-id
00:02:00:00:00:09:46:4f:43:32:30:35:31:52:30:57:34:00;
       option CISCO-EXR-CONFIG.client-identifier "exr-config";
       option CISCO-EXR-CONFIG.authCode 1;
       #invalid md5
       #option CISCO-EXR-CONFIG.md5sum "90fd845ac82c77f834d57a034658d0f1";
       #valid md5
       option CISCO-EXR-CONFIG.md5sum "90fd845ac82c77f834d57a034658d0f0";
       if option dhcp6.user-class = 00:04:69:50:58:45 {
       option dhcp6.bootfile-url "http://[2001:1851:c632:1::1]/NCS5501-2/image.iso";
       }
       else {
        #option dhcp6.bootfile-url
"http://[2001:1851:c632:1::1]/NCS5501-2/ncs5500-mini-x.iso.sh";
        option dhcp6.bootfile-url "http://[2001:1851:c632:1::1]/NCS5501-2/ztp.cfg";
       }
 #}
}
```

ZTP Bootscript

If you want to hard code a script to be executed every boot, configure the following.

```
conf t
    ztp bootscript /disk0:/myscript
commit
```

The above configuration will wait for the first data-plane interface to be configured and then wait an additional minute for the management interface to be configured with an IP address, to ensure that we have connectivity in the third party namespace for applications to use. If the delay is not desired, use:

```
conf t
    ztp bootscript preip /disk0:/myscript
commit
```

Note When the above command is first configured, you will be prompted if you wish to invoke it now. The prompt helps with testing.

This is the example content of **/disk0:/myscript**:

```
#!/bin/bash
exec &> /dev/console # send logs to console
source /pkg/bin/ztp_helper.sh

# If we want to only run one time:
xrcmd "show running" | grep -q myhostname
if [[ $? -eq 0 ]]; then
    echo Already configured
fi

# Set the hostname
cat >/tmp/config <<%%
!! XR config example
hostname myhostname
%%
xrapply /tmp/config

#
# Force an invoke of ZTP again. If there was a username normally it would not run. This
forces it.
# Kill off ztp if it is running already and suppress errors to the console when ztp runs
below and
# cleans up xrcmd that invokes it. ztp will continue to run however.
#
xrcmd "ztp terminate noprompt" 2>/dev/null
xrcmd "ztp initiate noprompt" 2>/dev/null
```

ZTP Utilities

ZTP includes a set of shell utilities that can be sourced within the user script. **ztp_helper.sh** is a shell script that can be sourced by the user script. **ztp_helper.sh** provides simple utilities to access some XR functionalities. Following are the bash functions that can be invoked:

- **xrcmd**—Used to run a single XR exec command:

  ```
  xrcmd "show running"
  ```

- **xrapply**—Applies the block of configuration, specified in a file:

  ```
  cat >/tmp/config <<%%
  !! XR config example
  ```

```
hostname node1-mgmt-via-xrapply
%%
xrapply /tmp/config
```

- **xrapply_with_reason**—Used to apply a block of XR configuration along with a reason for logging purpose:

```
cat >/tmp/config <<%%
!! XR config example
hostname node1-mgmt-via-xrapply
%%
xrapply_with_reason "this is a system upgrade" /tmp/config
```

- **xrapply_string**—Used to apply a block of XR configuration in one line:

```
xrapply_string "hostname foo\ninterface GigabitEthernet0/0/0/0\nipv4 address 1.2.3.44
255.255.255.0\n"
```

- **xrapply_string_with_reason**—Used to apply a block of XR configuration in one line along with a reason for logging purposes:

```
xrapply_string_with_reason ”system renamed again" "hostname venus\n interface
TenGigE0/0/0/0\n ipv4 address 172.30.0.144/24\n”
```

- **xrreplace**—Used to apply XR configuration replace in XR namespace via a file.

```
cat rtr.cfg <<%%
!! XR config example
hostname node1-mgmt-via-xrreplace
%%
xrreplace rtr.cfg
```

- **admincmd**—Used to run an admin CLI command in XR namespace. Logs can be found in **/disk0:/ztp/ztp_admincmd.log**

```
admincmd running [show platform]

ztp-user connected from 192.0168.0.1 using console on host
sysadmin-vm:0_RP0# show platform | nomore
Tue Jan 30 23:12:30.757 UTC
Location Card Type HW State SW State Config State
------------------------------------------------------------------------------
0/RP0 NCS-5501 OPERATIONAL OPERATIONAL NSHUT
0/FT0 NCS-1RU-FAN-FW OPERATIONAL N/A NSHUT
0/FT1 NCS-1RU-FAN-FW OPERATIONAL N/A NSHUT
0/PM0 NCS-1100W-ACFW OPERATIONAL N/A NSHUT
0/PM1 NCS-1100W-ACFW OPERATIONAL N/A NSHUT
```

- **xrapply_with_extra_auth**—Used to apply XR configuration that requires authentication, in XR namespace via a file. The **xrapply_with_extra_auth** API is used when configurations that require additional authentication to be applied such as alias, flex groups.

```
cat >/tmp/config <<%%
!! XR config example
alias exec alarms show alarms brief system active
alias exec version run cat /etc/show_version.txt
%%
```

```
xrapply_with_extra_auth >/tmp/config
```

- **xrreplace_with_extra_auth**—Used to apply XR configuration replace in XR namespace via a file The **xrreplace_with_extra_auth** API is used when configurations that require additional authentication to be applied such as alias, flex groups

```
cat >/tmp/config <<%%
!! XR config example
alias exec alarms show alarms brief system active
alias exec version run cat /etc/show_version.txt
%%
xrreplace_with_extra_auth >/tmp/config
```

Examples

ZTP logs its operation on the flash file system in the directory **/disk0:/ztp/**. ZTP logs all the transaction with the DHCP server and all the state transition. Prior executions of ZTP are also logged in **/disk0:/ztp/old_logs/**.

The following example displays the execution of a simple configuration script downloaded from a data interface using the command **ztp initiate interface Ten 0/0/0/0 verbose**, this script will unshut all the interfaces of the system and configure a load interval of 30 seconds on all of them.

```
#!/bin/bash
################################################################################
# *** Be careful this is powerful and can potentially destroy your system ***
#                *** !!! Use at your own risk !!! ***
#
# Script file should be saved on the backend HTTP server
################################################################################

source ztp_helper.sh
config_file="/tmp/config.txt"
interfaces=$(xrcmd "show interfaces brief")

function activate_all_if(){
  arInt=($(echo $interfaces | grep -oE '(Te|Fo|Hu)[0-9]*/[0-9]*/[0-9]*/[0-9]*'))
  for int in ${arInt[*]}; do
    echo -ne "interface $int\n no shutdown\n load-interval 30\n" >> $config_file
  done
  xrapply_with_reason "Initial ZTP configuration" $config_file
}

### Script entry point
if [ -f $config_file ]; then
  /bin/rm -f $config_file
else
  /bin/touch $config_file
fi
activate_all_if;
exit 0
```

The following example displays the ZTP logging output:

```
Oct 11 11:05:38 172.30.0.54 ztp-script: Hello from ncs-5001-c !!!
Oct 11 11:05:40 172.30.0.54 ztp-script: current=6.1.1, desired=6.1.1
Oct 11 11:05:40 172.30.0.54 ztp-script: Version match, proceeding to configuration
Oct 11 11:05:41 172.30.0.54 ztp-script: Starting autoprovision process...
```

```
Oct 11 11:05:42 172.30.0.54 ztp-script: ### XR K9SEC INSTALL ###
Oct 11 11:05:44 172.30.0.54 ztp-script: ### Downloading complete ###
Oct 11 11:05:55 172.30.0.54 ztp-script: Waiting for k9sec package to be activated
Oct 11 11:06:01 172.30.0.54 ztp-script: ### XR K9SEC INSTALL COMPLETE ###
Oct 11 11:06:03 172.30.0.54 ztp-script: ### Installing midnight commander ###
Oct 11 11:06:04 172.30.0.54 ztp-script: ### Downloading system configuration ###
Oct 11 11:06:05 172.30.0.54 ztp-script: ### Downloading system configuration complete ###
Oct 11 11:06:06 172.30.0.54 ztp-script: ### Applying initial system configuration ###
Oct 11 11:06:11 172.30.0.54 ztp-script: !!! Checking for errors !!!
Oct 11 11:06:14 172.30.0.54 ztp-script: ### Applying system configuration complete ###
Oct 11 11:06:15 172.30.0.54 ztp-script: Autoprovision complete...
```