

Deploy Router Using Bootz

With the Bootz process, you can securely and seamlessly provision network devices accurately within minutes and without any manual intervention.

Table 1: Feature History Table

Feature	Release Information	Feature Description
Provisioning Using Bootz Process	Release 7.11.1	This feature allows devices in the network to establish a secure connection with the remote Bootz server and authenticate information using a three-step validation process. This process involves validating the network device, the Bootz server, and the onboarding information thereby mitigating security risks and preventing malicious actions during remote provisioning.

Unlike the Secure ZTP process, which relies on vendor-specific definitions for bootstrapping a device, the Bootz process offers a specification that outlines data elements in a vendor-agnostic manner. It also details the necessary operations at turn-up time, integrating them into the boot process.

Also, the bootstrap request in the Bootz process includes the unique identifier or serial number for each node as opposed to the Secure ZTP process where the bootstrap request does not include serial numbers. The Bootz server returns the signed onboarding information with ownership voucher and owner certificate for the requested serial number of the device.

- Supported Bootz versions, on page 2
- Components used in the Bootz Process, on page 2
- Onboard Devices Using Bootz Workflow, on page 3
- Obtain Ownership Voucher, on page 4
- Build bootstrapping data, on page 5
- Bootz workflow commit process for hw-module commands, on page 6
- Provision Bootz Using DHCP Server, on page 6
- BootstrapStream RPC for Bootz process, on page 16
- Immutable bootstrap configurations, on page 19

• Bootz dynamic configuration, on page 22

Supported Bootz versions

This table provides the Bootz versions supported in each release. The Bootz Bootstrap server must be compatible with the respective Bootz version.

Table 2: Bootz Versions

Release	Version with File Path
Release 26.2.1	`openconfig/bootz v0.6.1`
Release 25.4.1	`openconfig/bootz v0.5.0`
Releases 24.3.1, 24.4.1, 25.1.1, 25.2.1, 25.3.1	`openconfig/bootz v0.3.1`
Release 24.2.1	`openconfig/bootz v0.1.1-0.20231106050618-8d6e2559f803`
Release 24.1.1	`openconfig/bootz v0.0.0-20230809153947-e6bc0be82dd8`
Release 7.11.1	`openconfig/bootz v0.1.0`

Components used in the Bootz Process

These components are part of the Bootz process.

- **Onboarding Device (Router):** A router is a Cisco device that you want to provision and connect to your network. Bootz is supported only on platforms that have *Hardware TAM*[1] support.
- **DHCP Server:** The DHCP server provides the URL where the Bootz process can access the bootstrapping information.
- **MASA Server:** You can generate and store the ownership voucher in the MASA server. The MASA server sends the ownership voucher to the Bootz server so that the Bootz process validates the device and establishes device ownership.
- **Bootz Bootstrap Server:** A Bootz Bootstrap server is any gRPC server used as a Bootz bootstrapping data source. For example, Google Proto. The Bootz Bootstrap server is complaint with Openconfig Bootz standards.

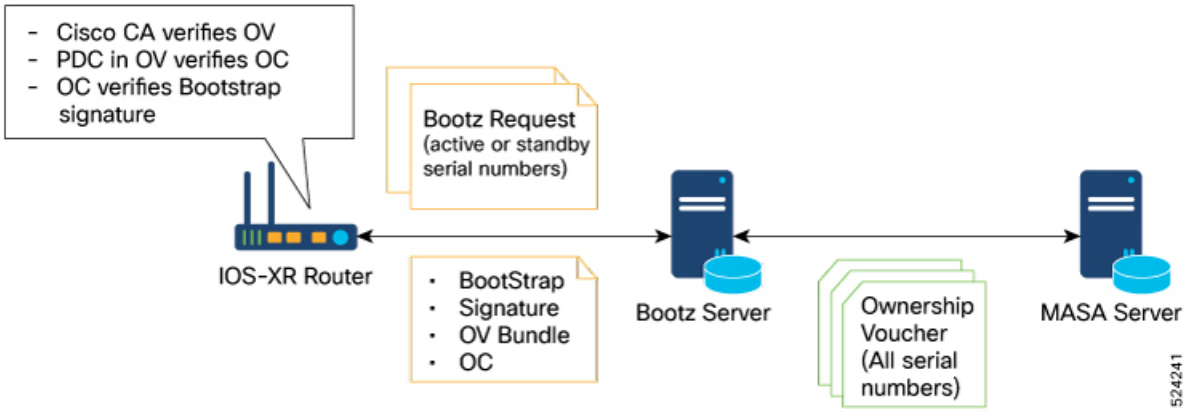
Note Bootz only supports a single name-server. As a result, when the DHCP server has more than one server address configured, Bootz fails to apply the server configuration.

[1] A secure storage device that stores the customer certificates and Cisco's internal secure data like trust anchors, SUDI certificates, secure flags, and other security information.

The Bootz server contains these artifacts:

- **Cisco IOS XR software images:** You can download Cisco images, SMU, and patches from the Cisco Support & Downloads page.
- **Bootstrapping Data:** It is a collection of data that you have created and uploaded to the Bootz server. The router obtains this data from the Bootz server during the provisioning process.

  Starting with IOS XR Release 24.4.1, the bootstrap configuration remains immutable throughout the device's operation. For more information, refer the topic Immutable bootstrap configuration.



Onboard Devices Using Bootz Workflow

The Cisco IOS XR software supports Bootz provisioning capabilities. The Bootz process uses the Google Remote Procedure Call (gRPC) protocol for fetching information from a remote server.

The Bootz workflow performs these validations to onboard the remote devices securely.

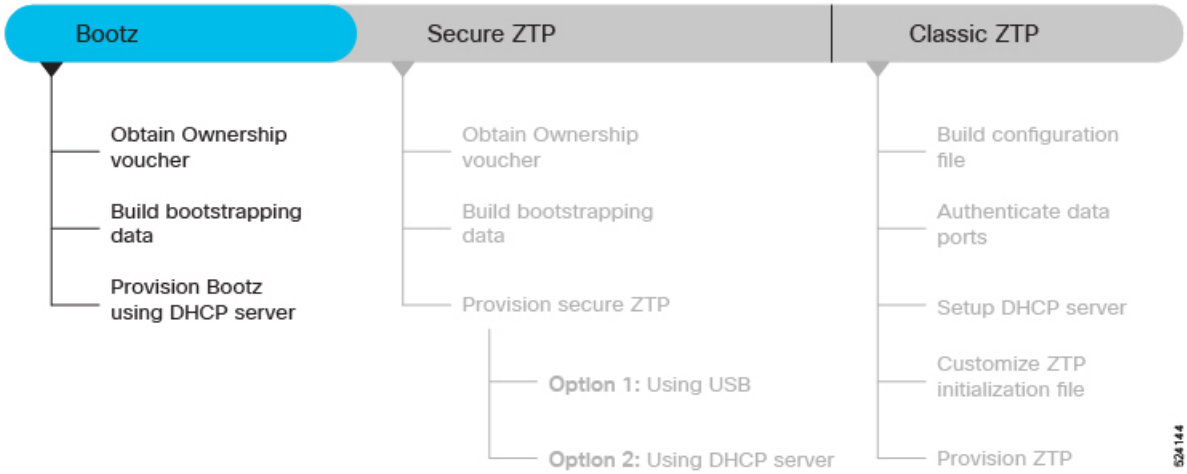
1. **Router Validation**: The Bootz server authenticates the router before providing the bootstrapping data.
2. **Server Validation**: The router in turn validates the Bootz server and ensures that the onboarding is performed for the correct network. Once it is validated, the Bootz server sends the bootstrapping data (for example, a YANG data model) or artifact to the router.
3. **Artifact Validation**: The router validates the bootstrapping data or artifacts received from the Bootz server.

Prior to Cisco IOS XR Release 24.3.1, the Bootz workflow processed the ownership voucher (OV) and onboarded devices only with an active Route Processor (RP). Starting with Cisco IOS XR Release 24.3.1, the Bootz workflow now processes OV information for both active and standby RPs, enabling the onboarding of devices with multiple standby control cards or line cards.

For more information about how the Bootz workflow works for the standby RP, see the Bootz Workflow for Standby RP.

This figure provides the Bootz workflow and the processes involved in the workflow. The sections that follow describe these processes in detail.

Figure 1: Bootz Workflow



Obtain Ownership Voucher

The ownership voucher is used to identify the owner of the device by verifying the owner certificate stored in the device.

Table 3: Feature History Table

Feature	Release Information	Feature Description
Ownership voucher request enhancement	Release 25.3.1	Introduced in this release on: Fixed Systems (8200 [ASIC: Q200, P100], 8700 [ASIC: P100, K100]); Modular Systems (8800 [LC ASIC: Q100, Q200, P100]) From this release onwards, when requesting an ownership voucher for fixed systems, you must use the chassis serial numbers in the Ownership Voucher request. Additionally, for both modular and fixed chassis systems, the serial number included in the ownership voucher request must match the SUDI subject name or the certificate on the device. This ensures accurate device identification and owner verification.

How to obtain Ownership Voucher

These steps help you obtain the ownership voucher from Cisco:

1. Contact Cisco Support.
2. Provide these information in your request to Cisco.
    - **Pinned Domain certificate (PDC):** PDC is an X.509 v3 certificate structure that uses Distinguished Encoding Rules (DER). The router uses this certificate to trust a public key infrastructure for verifying a domain certificate supplied to the router separately in the bootstrapping data. This certificate could be an end-entity certificate, including a self signed entity.

- Purchase order details with the serial numbers of the routers.

Sample Request:

```
{
   "expires-on": "2016-10-21T19:31:42Z",
   "assertion": "verified",
   "serial-number": "JADA123456789",
   "idevid-issuer": "base64encodedvalue==",
   "pinned-domain-cert": "base64endvalue==",
   "last-renewal-date": "2017-10-07T19:31:42Z"
}
```

3. Cisco generates the ownership voucher in .vcj format (Example: DCA213140YX.vcj) and sends the voucher in response to your request.

Note

- On fixed systems, use chassis serial number in the Ownership Voucher request.
- On modular systems, use Route Processor serial number in the Ownership Voucher request.

Build bootstrapping data

Bootstrapping data is the configuration data that

- the router retrieves from the Bootz server using gRPC protocols
- configures and sets up the router during its initial boot sequence
- comprises parameters such as gRPC services to be run and the management connectivity setup, and
- is immutable after the initial router boot up, from IOS XR Release 24.4.1 onwards. For more information, see Immutable bootstrap configurations.

Steps to build the bootstrapping data:

1. Create and upload the bootstrapping data to the gRPC server or Bootz bootstrap server.
2. The router sends a bootstrap request with these artifacts to the Bootz server.
   - Serial number of the control card or line card
   - Software image to download and install
   - Bootloader Password for the device
   - Certificate used to validate the bootstrap server
   - Bootstrap server configuration information such as server credentials, path information, authentication information, and certificates

   For the request message format, see the Bootstrap Request Message.

3. The Bootz server returns the listed bootstrapping data in its response to the router. The router receives these data during the provisioning process.
   - **Signed Bootstrap Response:** Each bootstrap response contains the onboarding information for:
     - A single control card or line card for active RP.
     - One or more control cards or line cards for standby RP.

     For the response message format, see the Bootstrap Response Message for a single card.
   - **Owner Certificate:** The owner certificate is installed on the router with your organization's public key. The router uses this public key in the owner certificate to verify the signature in the signed bootstrap response artifact.
   - **Ownership Voucher:** The ownership voucher is used to identify the device owner by verifying the owner certificate stored in the device. Cisco generates and supplies the ownership voucher in response to your request containing the PDC and device serial numbers. For more information, see How to obtain Ownership Voucher.
4. When the router obtains the onboarding information from the Bootz server, the router reports the bootstrapping progress to the Bootz server using the API calls.

Bootz workflow commit process for hw-module commands

The Bootz workflow now requires a two-step commit process for **hw-module** prefixed configuration:

1. Preconfiguration commit: If Bootz's *BootstrapDataResponse* message contains **hw-module** configurations, Bootz executes a pre-configuration stage to commit hw-module prefixed configuration. This stage includes a mandatory device reload prior to the actual device provisioning.
2. Configuration commit: After the device reload, perform the actual configuration commit.

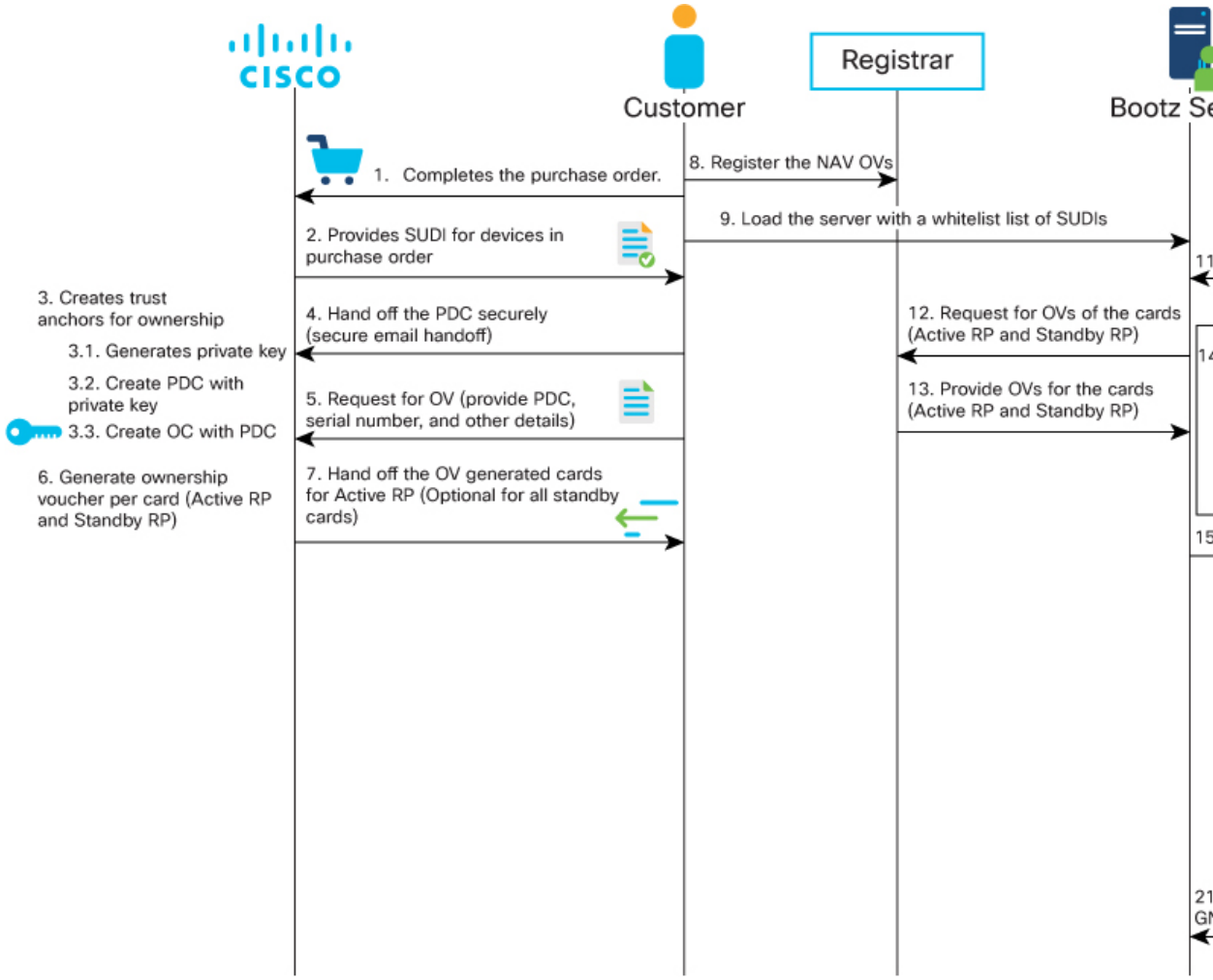
This approach ensures that changes to hardware modules are processed safely and in the correct sequence.

Provision Bootz Using DHCP Server

When you boot the device, the Bootz process initiates automatically on a device without prior configuration. During the process, the router receives the details of the configuration file from the DHCP server.

This figure illustrates the end-to-end sequence of the Bootz process:

Figure 2: End-to-end sequence of the Bootz process



Before you begin

As part of the initial setup for secure ZTP, the network administrator:

- Ensures to enable secure ZTP on the router using the **ztp secure-mode enable** command and reload the router.
- Contacts Cisco Support and follows the steps in How to obtain Ownership Voucher to obtain a voucher from Cisco.

Procedure

Step 1 Upload the listed bootstrapping data to the Bootz server. Refer to your vendor documentation as the upload procedure may vary from server to server.

- Cisco IOS XR software images

Note
Download Cisco images, SMU, and patches from the Cisco Support & Downloads page.

- Serial numbers of the routers to be onboarded
- Owner certificates
- Pinned Domain Certificate (PDC)
- Ownership vouchers

Step 2 Set up the DHCP server to provide the redirect URL to the router:

Before triggering the secure ZTP process, configure the DHCP server so that it provides the location of the IOS-XR image to the router. For information about how to configure the DHCP server, see your DHCP server documentation.

Configure these parameters in the DHCP server:

- `option-code`: Use one of these DHCP SZTP redirect option parameters in the `option-code` setting.
  - OPTION_V4_SZTP_REDIRECT (143): DHCP v4 code for IPv4.
  - OPTION_V6_SZTP_REDIRECT (136): DHCP v6 code for IPv6.
- `option-length`: Provide the option length in octets.
- `bootstrap-servers`: A list of servers. The onboarding device contact these servers for the bootstrapping data.

  `"bootz://<ip-address-or-hostname>[:<port>]<endpoint>"`

Example: `option dhcp6.bootstrap-servers code 136 = text;`

Step 3 Power on the router.

This procedure provides the high-level workflow of the Bootz process:

a. When you boot the device with an IOS-XR image, the secure ZTP process verifies if the secure ZTP mode (`secure-ztp mode`) is enabled. If not enabled, the device boots normally.

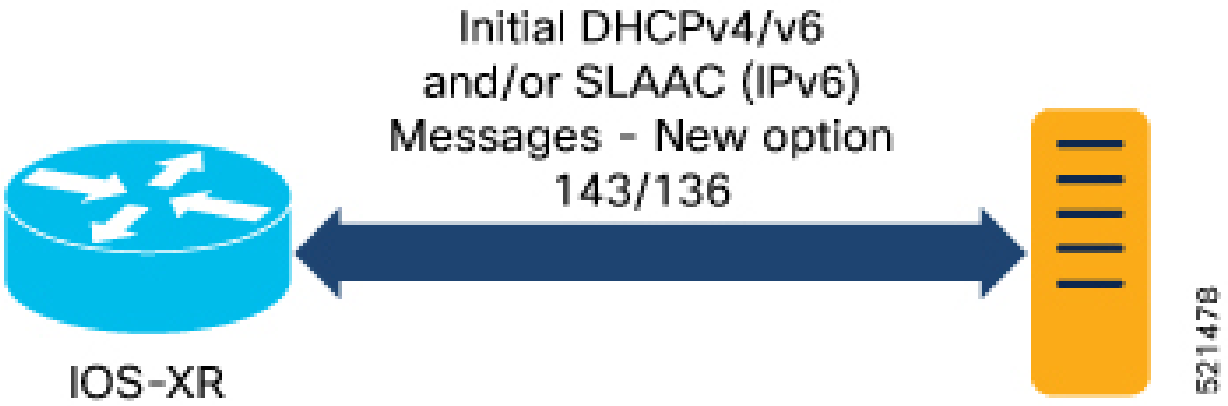
Note
When `secure-ztp mode` is enabled, the ZTP process accepts only the `secure-redirect-URL` and ignores the presence of the boot file name option from the DHCP response.

b. **DHCP discovery**:

1. The router initiates a DHCP request to the DHCP server.
2. The DHCP server responds with a DHCPv4 143 address option (for IPv4 addressing) or a DHCPv6 136 option (for IPv6 addressing).

Note
URLs to access bootstrap servers for further configuration are listed in options 136 and 143.

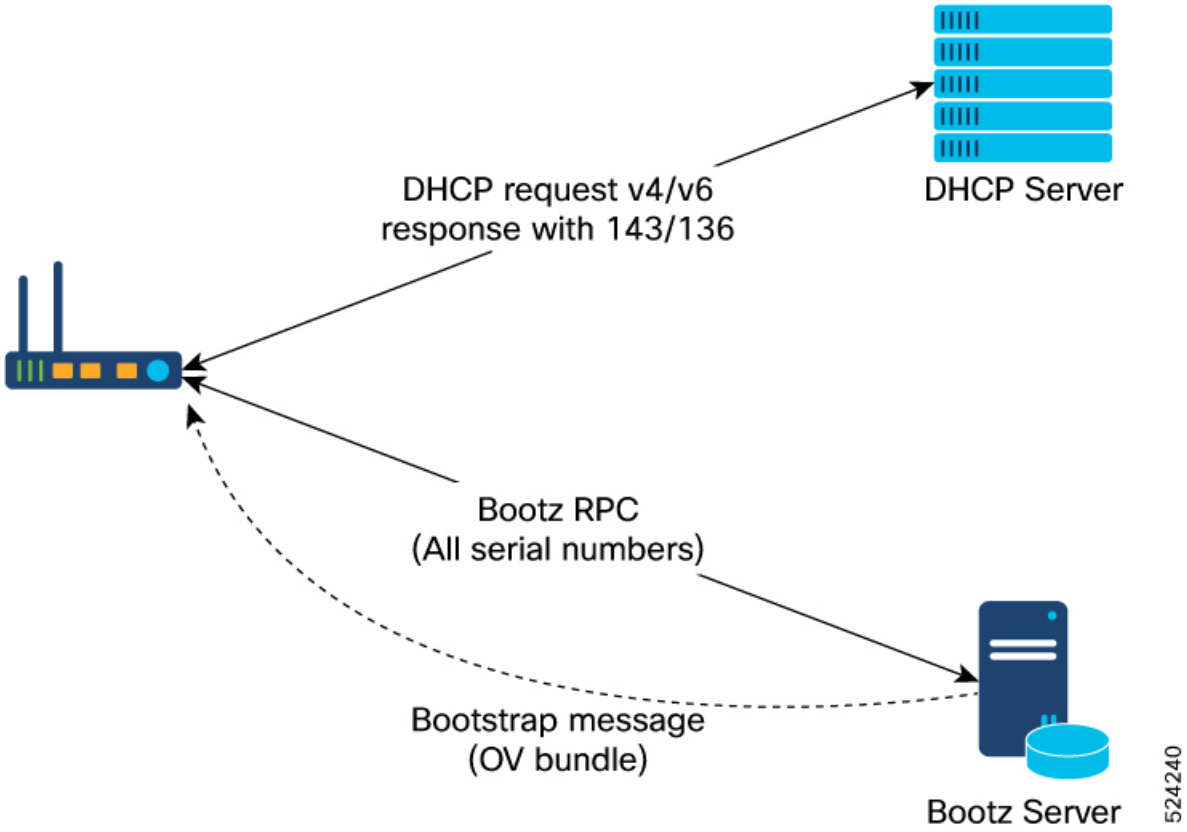
Figure 3: DHCP discovery



c. **Router and Bootz server validation**:

1. After receiving the URL from the DHCP server, the router initiates a gRPC connection to the Bootz server. The Bootz server IP address is obtained from the DHCP response.
2. The Bootz server authenticates the router before it provides the bootstrapping data.
3. After the Bootz server authenticates the router or the onboarding device, the router validates the Bootz server to ensure that the onboarding is performed for the correct network.

   After validating the Bootz server, the router sends the serial number for each control card or line card and other artifacts in its bootstrap request.
4. After its validation, the Bootz server sends the required artifacts along with the bootstrap response data to the router or the onboarding device.



d. Ownership Voucher verification:

The router receives the bootstrap response data that contains owner certificate, ownership voucher for each serial number, and the details of the image upgrade, if any.

Bootstrap response data includes the following:

- Image path
- Image version
- Trust anchor
- Boot configuration
- GNSI artifacts

These artifacts come from the Bootz server as a bootstrap response gRPC message. The router verifies the ownership voucher by validating its signature to one of its preconfigured trust anchors and downloads the image. When the router obtains the onboarding information, it reports the bootstrapping progress to the Bootz server.

If the router has sent the bootstrap request with serial numbers for multiple control cards or line cards, the Bootz server may optionally create an *OV bundle*[2] (.tar file) that contains the ownership voucher for each serial number and return the bootstrap response message with this OV bundle to the router.

[2] The OV bundle contains the ownership voucher for each serial number.

If the Bootz server cannot create an OV bundle, follow the procedure in How the Router Obtains and Processes the OV Information to know how the router obtains the OV information for multiple serial numbers.

e. Artifact Validation:

The router validates the artifacts received from the Bootz server as follows:

1. The device extracts the `pinned-domain-cert` node, an X.509 certificate from the ownership voucher to verify the owner certificate.
2. The device authenticates the owner certificate by performing the X.509 certificate path verification process on the trusted certificate.
3. Finally, the device verifies whether the artifact is signed by the validated owner certificate.

f. Provision the device:

1. The device first processes the boot image information.
2. Executes the script and then onboards the artifacts received from the Bootz server.
3. From 24.4.1 onwards, the device reports the status of the GNSI artifacts to the Bootz server.

g. After the onboarding process is completed, the network device is operational.

Bootz Workflow for Standby RP

Table 4: Feature History Table

Feature	Release Information	Feature Description
Bootz Workflow for Standby RP	Release 24.3.1	

Feature	Release Information	Feature Description
		Introduced in this release on: Fixed Systems (8200 [ASIC: P100], 8700 [ASIC: P100])(select variants only*); Modular Systems (8800 [LC ASIC: P100])(select variants only*) This feature enables the Bootz workflow to achieve full-system onboarding for devices with both active and standby Route Processors (RPs). In earlier releases, the Bootz workflow only supported onboarding for devices with an active RP. With this enhancement, the Bootz workflow can now detect faulty or tampered standby cards that are inserted dynamically during or after the active RP Bootz process. It does this by verifying the ownership voucher (OV) of the other cards during the initial Bootz process for the active RP. Faulty cards can be shut down to prevent security threats during remote provisioning, ensuring smooth network operation. This feature allows Bootz workflow to validate the standby RP as part of the active RP Bootz process. This feature introduces the **ztp bootz-server** command. This feature modifies the `Cisco-IOS-XR-ztp-cfg.yang`. (see GitHub, YANG Data Models Navigator) *This feature is supported on: • 8212-48FH-M • 8711-32FH-M • 88-LC1-36EH • 88-LC1-12TH24FH-E • 88-LC1-52Y8H-EM

Overview

The Bootz workflow now processes the ownership voucher (OV) for multiple control cards or line cards detected on the standby RP, either before the Bootz process for an active RP starts, during its progress, or after its completion.

With this new feature, the Bootz workflow:

- Allows dynamic insertion or replacement of standby control cards and line cards.
- Processes the ownership voucher (OV) bundle that the Bootz server sends to the router for the standby control cards or line cards.
- Performs the ownership verification of all standby control cards and line cards.

This feature allows you to configure the ZTP Bootz server with the **ztp bootz-server** command to store the server and vendor information received during the initial Bootz process for the active RP.

The router uses this configuration to communicate with the Bootz server and obtain the OV bundle (.tar file) for dynamically inserted or replaced standby control cards or line cards.

Prerequisites

- Configure the Bootz server to return the bootstrap data response message for all the serial numbers of the cards on the device with either the OV bundle or individual ownership voucher for each card.
- Include the **ztp bootz-server** configuration in the server's onboarding information or the vendor configuration information. This configuration is received from the Bootz server during the initial GetBootstrapDataRequest exchange for the active RP.
- Ensure that the routers running the Bootz client can process the OV bundle.

Restrictions

The dynamic Bootz workflow for the standby RP is triggered only if these conditions are met:

- Secure ZTP is enabled on the device.
- The Bootz process for the active RP is completed or not in progress.
- The Bootz server configuration from the initial Bootz process for active RP is available for dynamically inserted standby cards.

Use Cases

These use cases describe different scenarios where the standby RP cards are detected and the OV information is processed accordingly.

Use Case 1 - Standby Card Detected Before Bootz Process for Active RP Starts

If both the active RP and standby RP are detected during the initial boot process before the router communicates with the Bootz server:

- The router sends a bootstrap request to the Bootz server, including the serial numbers for both the active RP and standby RP.

- The Bootz server responds with the OV information for both the active RP and standby RP in its bootstrap response.
- If the response message for the initial bootstrap data request (for active RP) does not include an OV bundle for the standby RP, an additional bootstrap data request is triggered to fetch the OV information for the standby RP.

If there is no OV bundle in the bootstrap response for the initial bootstrap data request (for active RP),

Use Case 2 - Standby Card Detected During Bootz Process for Active RP

If the standby card is detected while the Bootz process for the active RP is in progress, the Bootz process for the standby RP is automatically triggered after the active RP's Bootz process completes.

In this scenario, the router uses the server information received during the initial Bootz process for the active RP, which you have configured using the **ztp bootz-server** command, to:

- Communicate with the Bootz server.
- Send the serial numbers for the dynamically inserted cards in its bootstrap request for standby RP to the Bootz server.
- Obtain the OV bundle (.tar file) from the Bootz server and process the OV for each card with a matching serial number. For more information about the OV bundle, see How the router obtains and processes the OV information.

The Bootz server, in turn, sends the OV information for the standby RP in its bootstrap response.

Use Case 3 - Standby Card Detected After Bootz Process for Active RP Completes

If the standby RP is detected after the initial Bootz process for active RP is completed, the Bootz process is triggered again automatically. The Bootz process is re-triggered if one of the following events occur:

- When a new card is inserted.
- When an existing card is replaced with another card.

Store the server information obtained during the initial Bootz process for active RP. As the secure ZTP workflow is not re-triggered for dynamically inserted standby control cards or line cards, the router uses this stored server information to communicate with the Bootz server and obtain the ownership vouchers for the newly inserted standby control cards or line cards.

Configuration to store server information obtained from the active RP Bootz process:

```
RP/0/RP0/CPU0:ios# config
RP/0/RP0/CPU0:ios(config)#ztp bootz-server ip 1.1.1.1 port 5000 trust-anchor
/misc/disk1/ta.cert
RP/0/RP0/CPU0:ios(config)# commit
```

Once the dynamic Bootz workflow for standby RP is triggered, the router communicates with the Bootz server using the server and vendor configuration information specified in the **ztp bootz-server** command. The Bootz server then sends the OV information for the dynamically inserted standby cards in its bootstrap response.

How the Router Obtains and Processes the OV Information

The router uses one of these methods to process the OV information that it has obtained from the Bootz server.

- **If the Bootz server is configured to send an OV bundle:**

1. The Bootz server sends the OV bundle as a single tar file in the bootstrap response to the router for the RPs.
2. The router running the Bootz client processes the tar file to verify individual ownership voucher for each serial number.
3. The router loads the owner certificate and the ownership voucher on each card with a serial number matching the serial number included in the ownership voucher.

- **If the Bootz server is configured to send individual ownership vouchers rather than an OV bundle, the router:**
  1. Communicates with the Bootz server using the server information from the bootstrap response for active RP.
  2. Sends a new bootstrap request for the standby RP
  3. Obtains the ownership voucher information for the standby RP.

BootstrapStream RPC for Bootz process

Initially, Cisco IOS XR devices relied exclusively on the unary GetBootstrapData RPC for the bootz process. This limited security and flexibility, as TLS 1.3 could only be used on platforms supporting trusted platform module (TPM) or elliptic curve cryptography (ECC)-based SUDI certificates. As a result, secure bootstrapping on many platforms was limited to TLS 1.2, and authentication flexibility were narrowly constrained by hardware-dependent certificate support.

Table 5: Feature History Table

Feature	Release Information	Feature Description
BootstrapStream RPC for Bootz Process	Release 26.2.1	Introduced in this release on: Centralized Systems (8400 [ASIC: K100])(select variants only*) BootstrapStream RPC enhances the security, flexibility, and future-readiness of Bootz process by enabling TLS 1.3-based device bootstrapping in Cisco IOS XR regardless of the hardware limitations in SUDI certificate. With the BootstrapStream RPC, backward compatibility with the existing Bootz infrastructure is preserved. This feature introduces these changes: CLI: A new command, **show ztp information bootz tls**, has been added.

Cisco IOS XR Release 26.2.1 introduces a new BootstrapStream RPC offering secure, bidirectional streaming for device bootstrapping through Bootz. Devices can retrieve bootstrap data over TLS 1.3 without requiring a SUDI certificate during the initial TLS handshake. Mutual authentication is managed using a nonce-based challenge–response mechanism, where the device signs a server-provided nonce with its SUDI private key.

BootstrapStream RPC addresses hardware limitations that previously restricted some platforms to TLS 1.2 for SUDI certificates. When BootstrapStream RPC is not implemented on the Bootz server, the device automatically fall back to legacy GetBootstrapData RPC, maintaining compatibility. BootstrapStream RPC also supports status reporting using the same streaming mechanism previously handled by the legacy ReportStatus RPC.

A new command, **show ztp information bootz tls**, has been introduced for displaying TLS capabilities and security context used by the device during the Bootz bootstrapping process.

Benefits of BootstrapStream RPC

The BootstrapStream RPC feature provides the following key benefits for the Bootz workflow in IOS XR:

- Enables TLS 1.3 bootstrapping across platforms, even without hardware or SUDI certificate support for TLS 1.3 during the TLS handshake.
- Removes dependency on TLS-handshake–based SUDI authentication, shifting device authentication to a nonce-based challenge–response mechanism within the RPC session.
- Improves security by supporting modern cryptographic practices avoiding deprecated RSA-PKCS#1.
- Maintains backward compatibility by automatically falling back to legacy GetBootstrapData RPC when the BootstrapStream RPC is not implemented, ensuring interoperability with existing Bootz servers.

- Requires no user configuration or operational changes. Operates transparently during first boot, preserving the existing ZTP user experience.
- Extends functionality to secure status reporting using the same streaming mechanism with ReportStatus operations.

Limitations of BootstrapStream RPC

The following limitations apply when you use the BootstrapStream RPC feature:

- If the Bootz server implements BootstrapStream and still wants to use the legacy RPC, you must explicitly return `UnImplemented` status code for the BootstrapStream RPC. If the server does not implement BootstrapStream, gRPC returns `Unimplemented` by default.
- While BootstrapStream enables TLS 1.3 transport, all devices still rely on the RSA-based SUDI and PKCS#1-based signature generation for the nonce verification workflow.
- The nonce-based challenge–response mechanism introduces extra message exchanges compared to the unary GetBootstrapData RPC, slightly increasing protocol complexity.
- BootstrapStream is limited to initial bootstrapping and status reporting, and cannot be reused for general configuration or operational data exchange.

How BootstrapStream RPC operations work

Summary

BootstrapStream RPC streamlines the initial configuration and secure ownership verification of Cisco IOS XR devices during their first boot. It ensures only authenticated devices are provisioned, using a series of validated, authenticated exchanges with a Bootz server.

The key components involved in the process are:

- Cisco IOS XR device: Boots in a factory-default state and initiates provisioning.
- Bootz server: Provides bootstrap data and manages authentication challenges during device onboarding.
- SUDI certificate: Ensures device identity and security through cryptographic validation.

Workflow

These stages describe the BootstrapStream RPC workflow:

1. The device boots in a factory-default state and initiates secure ZTP.
2. The device retrieves the Bootz server URI from the DHCP response using Option 143 for IPv4 or Option 136 for IPv6.
3. The device establishes a gRPC connection to the Bootz server using TLS 1.3, without presenting a client certificate during the TLS handshake.
4. The device starts the BootstrapStream RPC by sending a bootstrap request with chassis and control card information along with the device's SUDI certificate.

5. The Bootz server validates the SUDI certificate using the Cisco CA bundle.

   Upon successful validation, the server sends a nonce challenge to the device.

6. The device signs the received nonce using the SUDI private key via the TAM.

   The signed nonce (RSA-PKCS#1 with SHA-256) is sent back to the server.

7. The Bootz server verifies the signed nonce to authenticate the device.

   Once validated, the server sends the bootstrap response, including bootstrap data, ownership voucher and certificate, and a signed response.

8. The device verifies the ownership voucher, ownership certificate, and response signature.

   After successful verification, the device applies the immutable boot configuration and transitions to normal device provisioning.

Result

The BootstrapStream RPC workflow securely provisions Cisco IOS XR devices, automating their configuration while validating device ownership. This enables seamless, secure onboarding into the network for newly deployed devices.

Immutable bootstrap configurations

Immutable bootstrap configuration is the tamper-proof boot configuration of the router that is not affected by subsequent dynamic configuration changes by gRPC Network Management Interface (gNMI).

Table 6: Feature History Table

Feature	Release Information	Feature Description
Immutable bootstrap configurations	Release 25.4.1	Introduced in this release on: Fixed Systems (8700 [ASIC: K100])(select variants only*); *This feature is supported on Cisco 8711-48Z-M routers.

Feature	Release Information	Feature Description
Immutable bootstrap configurations	Release 24.4.1	Introduced in this release on: Fixed Systems (8200 [ASIC: P100], 8700 [ASIC: P100, K100])(select variants only*); Modular Systems (8800 [LC ASIC: P100])(select variants only*) You can now ensure your router always boots up in a known good state. This feature prevents unauthorized changes to the router's boot configuration, ensuring that it remains static and tamper-proof. This maintains the integrity and security of the router from the initial boot stage and throughout its entire operation. *This feature is supported on: • 8212-48FH-M • 8711-32FH-M • 8712-MOD-M • 88-LC1-36EH • 88-LC1-12TH24FH-E • 88-LC1-52Y8H-EM

Benefits of immutable bootstrap configurations

Immutable bootstrap configurations significantly benefit network device operation and management.

- Security and stability: Since the immutable bootstrap configuration remains unchanged after the initial setup, it ensures that the device's foundational settings are secure and stable. This reduces the risk of unauthorized changes or accidental misconfigurations.
- Consistent boot up: By maintaining a static configuration, the device can consistently boot into a known good state. This consistency is crucial for reliable network operations and troubleshooting.

Key features of immutable bootstrap configurations

These are the key features of immutable bootstrap configuration:

- Upon successful Bootz completion, the boot configuration file is stored as backup in the router. The router preserves this backup file across reloads. The boot configuration commit appears in the router's commit history as the user **local** in the output of the **show configuration commit list** command.
- The router does not allow gNMI to override the boot configurations.
- The router does not allow configuration changes for security credentials managed by gRPC Network Security Interface (gNSI) to be overwritten by gNMI or boot configuration. This means that gNSI owns the configurations for security credentials, and you must invoke gNSI to mutate this store.
- The router allows configuration changes through Command Line Interface (CLI) to take effect on the boot configuration. However if you perform a Route Processor Fail Over (RPFO), gNMI **Set** operation, or restart the Extensible Manageability Services Daemon (EMSD) process using the **process restart**

emsd command, the boot configuration is restored. The EMSD process is the process that initializes and manages the gRPC server and services on the router.

- The gNOI.Bootz service manages boot configuration store completely. If you need to modify the boot configuration you must call the **Set** RPC through the **gNOI bootconfig** service.

Limitations of immutable bootstrap configuration

These are the limitations of immutable boostrap configuration:

- When you modify any of the parameters of the boot configuration using CLI, the router applies the changes. To revert the boot configuration to the original state, perform any one of the these operations:
  - RPFO
  - gNMI **Set** operation, or
  - restart of the EMSD process using the **process restart emsd** command.
- Due to the absence of a vendor-neutral boot configuration data set, the immutable bootstrap configuration may differ depending on the vendor of the network device.
- Boot configurations cannot be altered via gNMI set operations. This restriction is vital for safeguarding critical settings like hardware-module configurations and user credentials from accidental removal.

Verify immutable bootstrap configuration on the router

Use this procedure to view and verify the immutable bootstrap configurations.

Before you begin

Ensure the router is booted up and provisioned with the Bootz process.

Procedure

Step 1 Enter the **show configuration commit list** command to view the list of configurations commited.

Example:

```
Router# sh configuration commit list 3
SNo. Label/ID              User      Line               Client      Time Stamp
~~~~ ~~~~~~~~              ~~~~      ~~~~               ~~~~~~      ~~~~~~~~~~
1    1000000011            local     ems                YANG Frame  Thu Nov 14 12:26:26 2024
2    1000000010            cisco     con0_RP0_CPU0      CLI         Thu Nov 14 12:23:14 2024
3    1000000009            local     ems                YANG Frame  Thu Nov 14 05:55:50 2024
```

The router commits the bootstrap configuration as User **local**.

From the output of **show configuration commit list**, note the specific **Label/ID** listed for the User **local**.

Step 2 Enter the **show configuration commit changes** command with the **Label/ID** noted in the previous step, to view the list of bootstrap configurations.

Example:

```
Router# sh configuration commit changes 1000000011
!! Building configuration...
!! IOS XR Configuration 24.4.1
hostname cisco
username cisco
 group root-lr
 group cisco-support
 secret 10
$6$8J3HfJlo8Ah0f...$gG5juuKsunhYwcH1rRRdq2UHPk/VUxVGjVHRLAhrUFlyOoiBRjQFVdDhgYlfuOJDRmAfLv1H4AaOZX8xHrBp20
!
grpc
 no-tls
!
interface MgmtEth0/RP0/CPU0/0
 ipv4 address dhcp
!
router static
 address-family ipv4 unicast
 0.0.0.0/0 MgmtEth0/RP0/CPU0/0 192.168.122.1
 !
!
ssh server v2
end
```

Bootz dynamic configuration

Bootz dynamic configuration is a set of boot configuration fields that

- applies initial device settings that can be changed after bootstrap
- supports automation and user-driven updates via standard methods like CLI or gNMI, and
- differs from immutable configurations that remain locked for device security and operational stability.

Table 7: Feature History Table

Feature Name	Release Information	Feature Description
Bootz dynamic configuration	Release 26.3.1	Introduced in this release on: Fixed Systems (8200 [ASIC: Q100, Q200, P100], 8700 [ASIC: P100, K100], 8010 [ASIC: A100]); Centralized Systems (8600 [ASIC: Q200], 8400 [ASIC: K100]); Modular Systems (8800 [LC ASIC: Q100, Q200, P100, K100]) You can now set initial device configuration at first boot that remains flexible for future changes. This feature allows you to modify or overwrite dynamic boot-time settings through standard CLI or gNMI Set RPC calls without requiring a factory reset.

Dynamic versus immutable Bootz configurations

From IOS XR 26.3.1, Bootz defines two categories of configuration at device bootstrap, each with specific characteristics:

- **Immutable configuration:**

  Boot-time vendor and OpenConfig settings that are preserved across device reloads and cannot be changed by subsequent configuration actions.

- **Dynamic configuration:**

  Boot-time settings that are mutable after bootstrap, supporting post-boot automation and user-driven updates.

Bootz processes configurations in two phases during device startup: it first applies and preserves immutable Cisco IOS XR native and OpenConfig data, then applies the dynamic settings as regular gNMI operations. Bootz dynamic configurations are applied using fields within the BootConfig message defined in Bootz.proto.

The following table compares essential attributes of dynamic and immutable configuration:

Table 8: Comparison of dynamic and immutable configuration attributes

Attribute	Dynamic configuration	Immutable configuration
Applied at bootstrap	Yes	Yes
Mutable after device boot	Yes	No
Preserved across device reload or RP failover	No	Yes
Configuration update requires factory reset	No	Yes

Note Use dynamic configuration for settings that need to change after initial boot. Use immutable config for parameters that must remain locked for device security and operational stability.

Initial device deployment with dynamic and immutable configuration

When a new Cisco router boots, BootZ delivers both `vendor_config` and `dynamic_vendor_config` fields.

- The `vendor_config` is immutable and sets security and identity.
- The `dynamic_vendor_config` is mutable and provides BGP neighbors and interface addresses needed for deployment. After boot, automation updates BGP neighbors with gNMI `Set`, demonstrating that dynamic configuration can change without reapplying the bootstrap.

Guidelines for Bootz dynamic configuration

Configuration category selection

Reserve dynamic configuration for parameters that require future modification or automation-driven updates. Use immutable configuration fields for device settings that must not change after bootstrap, such as security policies or device identity.

Dynamic configuration consumption and persistence

Provide dynamic configuration fields only during the initial Bootz bootstrap event in the BootstrapData message.

- Do not specify dynamic config with post-boot gNOI SetBootConfig calls, as the router rejects them.
- Dynamic configuration is applied only if the bootconfig.txt file is present, which signals the device is in first-time bootstrap state.
- After initial application, dynamic config options are not consumed again, even during device restarts or route processor failovers.

Error handling and rollback

Rely on the EMSD process to enforce atomicity of bootstrap configuration. If dynamic configuration application fails, EMSD rolls back both dynamic and immutable configuration settings to protect device integrity.

- Always validate configuration data before bootstrap to lower the risk of error-triggered rollback.
- Configuration failures in dynamic or immutable sections trigger rollback and error reporting to external systems by Bootz status messages and EMSD logs.
- Operational teams should monitor EMSD artifact status files for each bootstrapping component to verify results and troubleshoot failures.

Security considerations for dynamic configuration

Apply operational security policies and validations to all dynamic configuration fields as you would with immutable configuration.

- Validate and approve all configuration content, both dynamic and immutable, before assembling BootConfig messages for device provisioning.
- Design automation and operational workflows to detect and alert on unexpected configuration artifacts or status changes during or after bootstrap.
- Bootz dynamic configuration does not introduce new external attack surfaces, but misconfiguration or using unvalidated dynamic settings may create operational risks.

How Bootz dynamic configuration works

Bootz dynamic configuration enables Cisco routers to process both dynamic and immutable configuration during device bootstrap. It facilitates rapid onboarding with flexible initial settings and protecting critical persistent parameters.

Summary

The key components involved in the process are:

- Bootz client: Retrieves and stages both immutable and dynamic bootstrap artifacts on the device.
- EMSd: Applies configuration artifacts, reports status, manages rollback on error, and updates status records.
- ZTP (Zero Touch Provisioning): Initiates automated bootstrap by communicating with infrastructure at initial device boot.

Bootz dynamic configuration ensures configuration artifacts are applied in sequence with robust error handling, guaranteeing that both immutable and dynamic configurations are properly delivered and that operational status remains traceable.

Workflow

These stages describe how Bootz dynamic and immutable configuration are processed during device bootstrap:

1. On device startup, ZTP and the Bootz client initiate bootstrap by acquiring initial configuration and credentials from the Bootz server.
   - ZTP triggers the Bootz flow using DHCP, obtaining bootstrap data (BSD) from the network.
   - The Bootz client fetches the BSD and validates all artifacts for authenticity.

   All bootstrap artifacts are validated before configuration is applied.

   The device is prepared to stage configuration files.
2. The Bootz client extracts, validates, and stages artifacts on local storage.
   - Artifacts include vendor_config, oc_config, dynamic_vendor_config, dynamic_oc_config, credentials, and policies.
   - Artifacts are classified as immutable (unchangeable) or dynamic (modifiable after the initial boot).

   The artifacts become available for EMSd to apply to the device configuration system.

   All required files are staged for EMSd processing.
3. EMSd loads artifact files and applies configurations in two steps: immutable artifacts first, followed by dynamic artifacts.
   - Immutable configuration (vendor_config, oc_config) is applied first using a gNMI Set request with priority 1.
   - Upon successful immutable application, dynamic configuration (dynamic_vendor_config, dynamic_oc_config) is applied using a separate gNMI Set request with priority 100.

   EMSd maintains the proper application order for consistency and security.

   Immutable settings are locked until device reset; dynamic settings become part of the running configuration.
4. EMSd validates each configuration application, manages rollbacks, and records status for each artifact type.
   - EMSd detects errors when configuring artifacts and initiates rollback for both immutable and dynamic configurations if required.

- Status for each artifact is written, such as for dynamic configuration, in `/misc/config/ems/gnsi` and system logs.

When...	And...	Then...	And...
A configuration artifact fails to apply	The device is in bootstrap phase	EMSd rolls back all applied artifacts (immutable and dynamic)	An error status is sent to the Bootz client and logged
All artifacts apply successfully	The device reaches operational state	EMSd marks all artifacts as OK in status files	External management services are enabled

Status files are referenced for troubleshooting and provide per-artifact reporting.

The device completes bootstrap with validated immutable and dynamic configuration in effect.

5. After bootstrap and across device restarts, only immutable configuration is enforced - dynamic configuration is not reapplied. So user updates persist after the first boot.
   - Dynamic configuration from bootconfig.txt is used only during first boot. For subsequent restarts, immutable configuration from bootconfig_merged.txt is applied.
   - Operators can change dynamic elements using gNMI Set or CLI.

   This protects operator changes from being overwritten after bootstrap and enforces critical configuration until reset.

   The device supports further dynamic configuration after deployment but ensures immutable parameters are preserved until factory reset.

Result

When the process completes, the router is securely bootstrapped with immutable and dynamic configuration - enabling reliable onboarding and ongoing operational flexibility. All status is recorded and visible for troubleshooting and compliance tracking.

Verify Bootz dynamic configuration

Verify Bootz dynamic configuration is successfully applied during device bootstrap and remains mutable.

This task helps network operators verify successful BootZ bootstrapping and diagnose configuration or artifact failures using device logs and status files.

Before you begin

- Ensure the router is booting from a factory or unconfigured state, or is being bootstrapped using Bootz.
- Confirm that the Bootz server, DHCP, and required ZTP infrastructure are reachable and properly configured.
- Make sure you can access the device shell or management interface to inspect logs and configuration files.

Follow these steps to verify and troubleshoot Bootz dynamic configuration.

Procedure

Step 1 Review ZTP logs to check bootstrap status.

Example:

```
Router# tail -f /var/log/ztp.log
```

Check for messages indicating successful completion or errors in bootstrapping, such as **Bootz OK**, **Completed**, or **Error**.

Table 9: ZTP log review outcomes

If...	Then...
Completion messages and no errors appear	Go to the next step to verify artifact status.
Error or rollback messages appear	Review logs in detail and check EMSd status for artifact failures.
Log file is missing or unreadable	Verify bootstrapping was properly initiated and confirm device access.

ZTP log review provides initial confirmation of Bootz bootstrapping success or failure.

Step 2 Compare running configuration output with the initial dynamic configuration to confirm correct application.

Example:

```
Router# show running-config
```

Table 10: Dynamic configuration verification outcomes

If...	Then...
Dynamic configuration entries are present	Dynamic configuration applied successfully and continue to next step.
Dynamic configuration entries are missing	Troubleshoot artifact status and check application logs.
Unexpected configuration present	Review original bootstrap artifacts for discrepancies.

Running configuration output indicates if dynamic configuration was applied at bootstrap.

Step 3 Change a value set by dynamic configuration using the CLI or gNMI Set and verify the change appears in running config.

Example:

```
gnmi-cli set dynamic-config-path value new-value
```

Immutable config cannot be changed; dynamic config should be updatable.

Table 11: Mutability outcomes

If...	Then...
Change to dynamic configuration is accepted and reflected	Dynamic configuration mutability is verified.
Change is rejected	Confirm the configuration path is dynamic and review error messages.
Immutable configuration path is changed	Change should be rejected by the device.

Mutability test confirms that dynamic configuration can be changed, while immutable configuration cannot.

Step 4 Run the **show logging** command to look for rollback, commit failures and bootconfig application failure messages.

Example:

```
Router# show logging
```

Error logs help isolate failures in configuration application, supporting troubleshooting efforts.

If verification succeeds, the router operates with Bootz status as OK, immutable and dynamic configuration artifacts present, dynamic configuration is mutable, and errors during bootstrap result in rollback and clear error reporting.

What to do next

After you finish verification and troubleshooting:

- Monitor device state for further changes and ensure that operational workflows can update dynamic config as needed.
- Document troubleshooting outcomes if errors occurred and escalate to Cisco support if bootstrap or persistent errors remain.