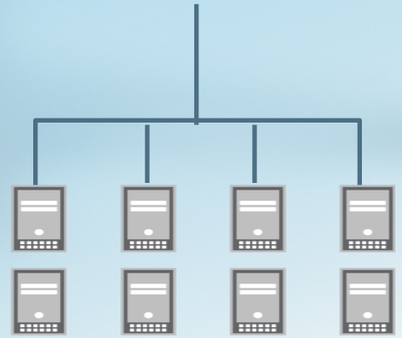
Zero Touch Provisioning

Network Provisioning

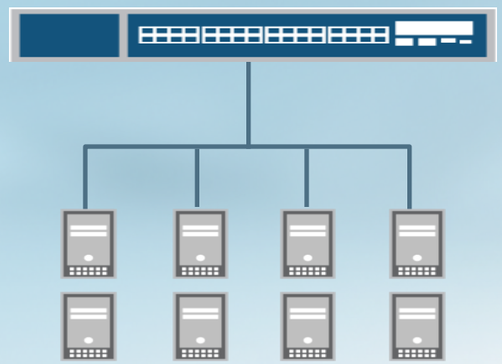
Load image and configure

EX & QFX Series Switches

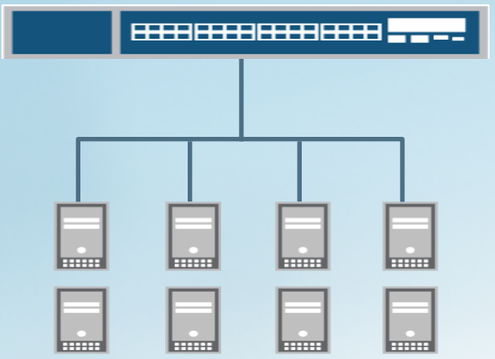
Time Consuming
Manual process requiring switches to be staged;
takes days



Error Prone
80% of network downtime
is due to manual configuration errors



Expensive
Staging is expensive
and requires trained professionals



Solution

- Zero Touch Provisioning
- Overcomes the problem of manual provisioning.
- Expects DHCP and File Servers to provide sufficient data to provision

What is ZTP? (It's Not Magic)

- The Preboot eXecution Environment (PXE) is a standardized client-server environment to boot clients from a network. Only a PXE capable Network Interface Card and small set of protocols such as DHCP and TFTP are required.

Overview [edit]

Since the beginning of computer networks, there is a persistent need for client systems able to boot a software image using appropriate configuration parameters, both retrieved at boot time from one or more networked servers. This requires a client using a set of pre-boot programs, based on industry-standard network protocols. Additionally, the initially downloaded and run Network Bootstrap Program (NBP) must be built relying on a client (device being bootstrapped via PXE) firmware layer providing a hardware-independent, standard interaction with the local network booting environment. In this case, server availability and standards compliance are key to guarantee boot process system interoperability.

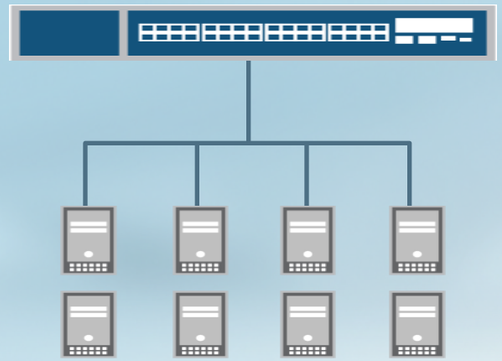
One of the first attempts in this regard was bootstrapping using TFTP standard RFC 906, published in 1984, which established the 1981 published Trivial File Transfer Protocol standard RFC 783 to be used as the standard file transfer protocol for bootstrap loading. It was followed shortly after by the Bootstrap Protocol standard RFC 951 (BOOTP), published in 1985, which allowed a disk-less client machine to discover its own IP address, the address of a TFTP server, and the name of an NBP to be loaded into memory and executed. Difficulties on BOOTP implementation among other reasons eventually led to the development of the Dynamic Host Configuration Protocol standard RFC 2131 (DHCP) published in 1997. This pioneer TFTP/BOOTP/DHCP approach felt short because at the time it was not defined the required standardized client side of the provisioning environment.

Zero touch provisioning

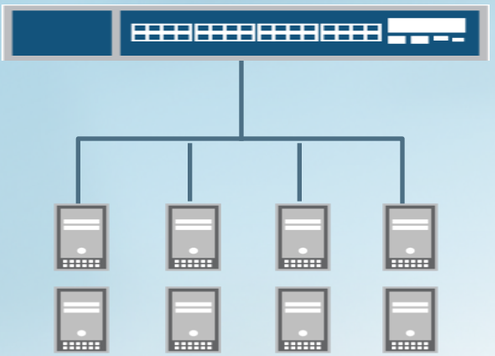
EX & QFX Series Switches

Auto Image and Configure

Auto Image and Configure



Auto Image and Configure

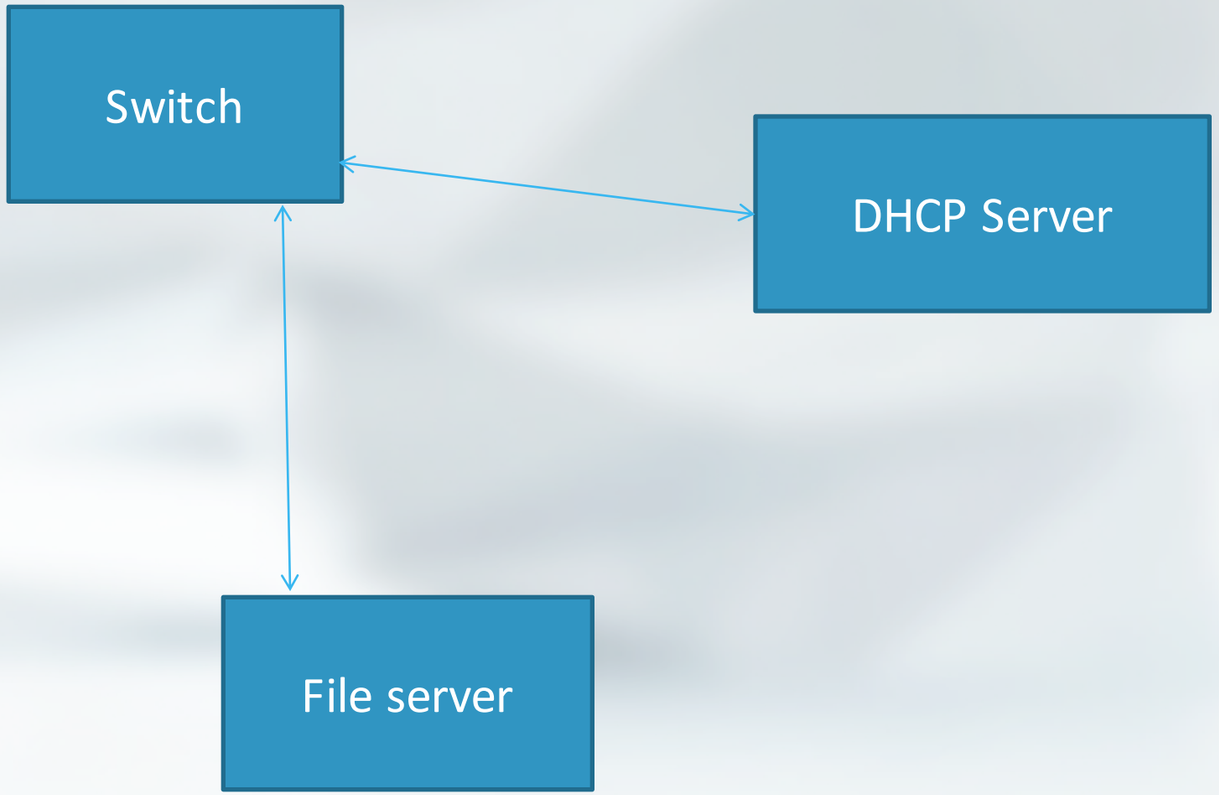


Increase Speed of Deployment & Provisioning

Reduce Configuration Errors

Reduce OPEX

Topology



Junos ZTP Topology – Components

Device:

- Power On
- Connected to Network
- Zeroized / Factory Configuration
  - “request system zeroize”

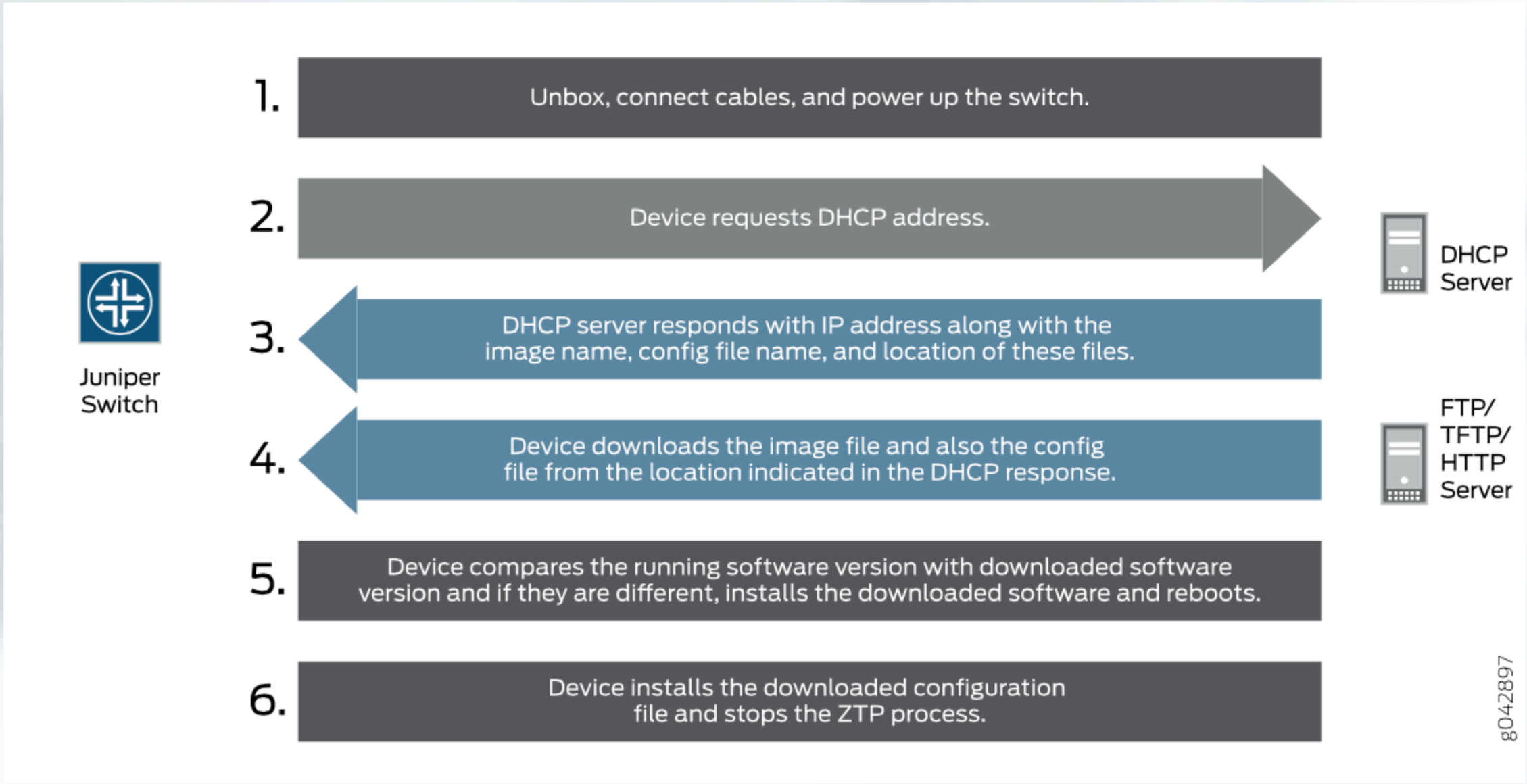
DHCP Server:

- Determines the
  - IP
  - Software Image Location
  - Configuration Location

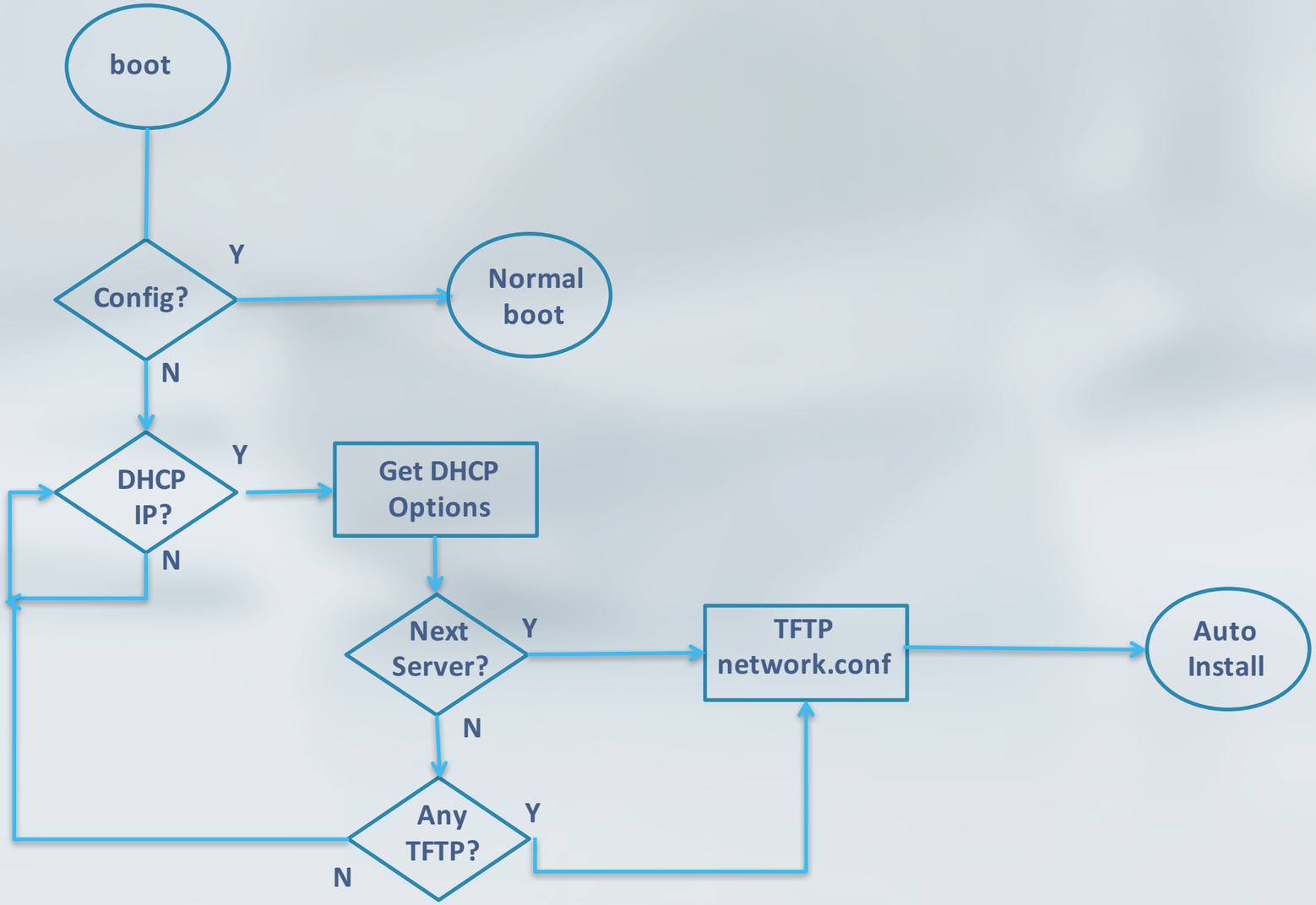
File Server:

- Serve the Requested Files

Basic ZTP Process



Junos Auto Installation (Simplified)



Device Implementation

- ZTP Process runs as part of the JDHCPD daemon.
- Is enabled when dhcp client and certain configuration is applied on device
- Will read dhcp reply and trigger ZTP cycle based on DHCP Options

Example Zeroized Config

```
vlans {
    default {
        vlan-id 1;
        l3-interface irb.0;
    }
}
chassis {
    auto-image-upgrade;
}
system {
    processes {
        dhcp-service {
            traceoptions {
                file dhcp_logfile size 10m;
                level all;
                flag all;
            }
        }
    }
}
```

```
interfaces {
    irb {
        unit 0 {
            family inet {
                dhcp {
                    vendor-id "Juniper-<model>";
                }
            }
        }
    }
 vme {
        unit 0 {
            family inet {
                dhcp {
                    vendor-id "Juniper-<model>";
                }
            }
        }
    }
}
```

ZTP DHCP Options

DHCP Option	Purpose / Description
66 or 150	File Server Address, must be IP
43 sub option 00	Software Image File Name / Path
43 sub option 01	Configuration File Name / Path
43 sub option 02	Symbolic Link vs Filename (Default Filename)
43 sub option 03	Transfer Mode (TFTP / FTP / HTTP)
43 sub option 04	Alternate Software Image File Name / Path
07	Specify Syslog Server
12	Specify Hostname of Client / Device
42	Specify Ntp Server

File Server

- File Transfer Protocols supported: FTP, HTTP, TFTP
- Although TFTP is supported, recommend that you use FTP or HTTP these transport protocols are more reliable

- Default protocol is: TFTP

ISC DHCP Server Configuration

```
option space NEW_OP;
option NEW_OP.image-file-name code 0 = text;
option NEW_OP.config-file-name code 1 = text;
option NEW_OP-encapsulation code 43 = encapsulate NEW_OP;

group {
     option tftp-server-name "17.176.31.71";
     option log-servers 17.176.31.72;
     option ntp-servers 17.176.31.73;
     option NEW_OP.image-file-name "/images/jinstall-qfx.tgz";
     option NEW_OP.transfer-mode "ftp";
     host tme-qfx3500-1 {
          hardware ethernet 88:e0:f3:71:a0:82;
          fixed-address 17.176.31.19;
          option host-name "tme-qfx3500-1";
          option NEW_OP.config-file-name “/config/tme-qfx3500-1.config";
     }
     host tme-qfx3500-1 {
          hardware ethernet f8:c0:01:c6:96:81;
          fixed-address 17.176.31.20;
          option host-name “tme-qfx3500-2";
          option NEW_OP.config-file-name “/config/tme-qfx3500-2.config";
     }
}
```

Vendor Specific options required for Auto image upgrade

Syslog and NTP servers

image that need to be installed. Supports symbolic link also

Configuration file for Auto configuration

MAC to IP Mapping and Host name for the system

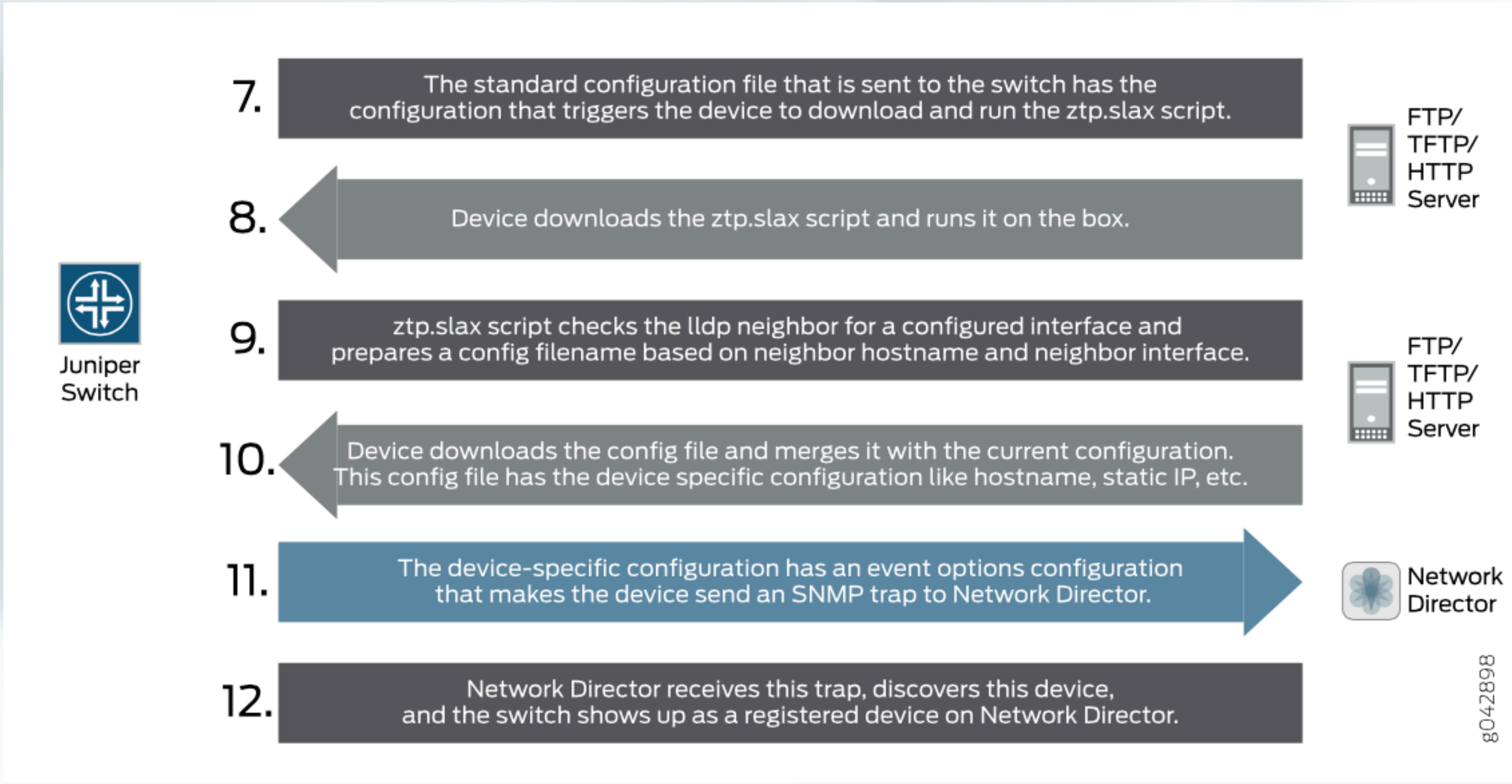
DNSMASQ DHCP/TFTP Server Configuration

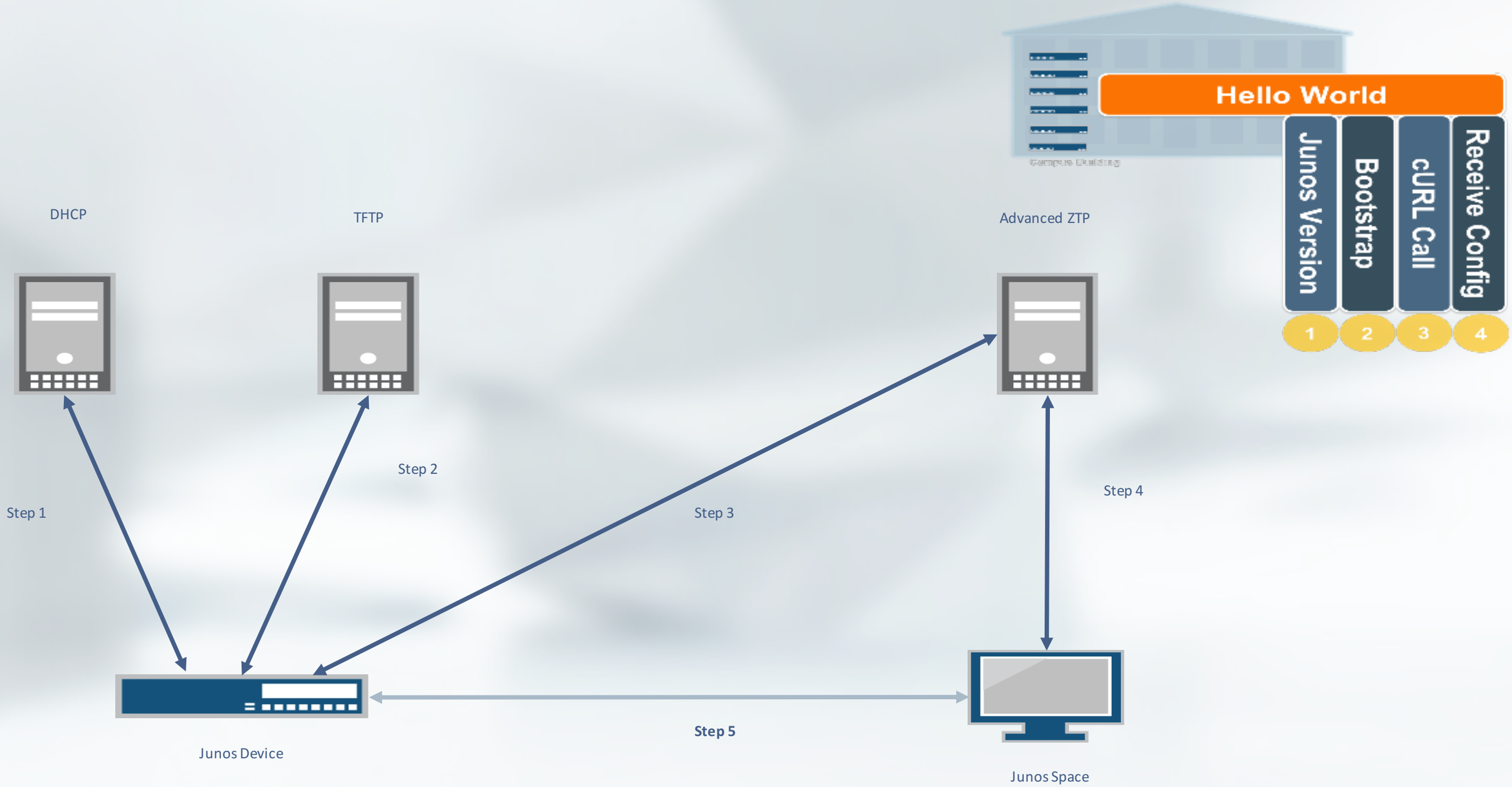
```
dhcp-leasefile=/var/lib/misc/dnsmasq.leases
log-facility=/var/lib/misc/dnsmasq.log
log-dhcp
port=0
enable-tftp
tftp-root=/tftpboot
dhcp-option=150,10.250.30.10
dhcp-vendorclass=set:ex2200-clients,Juniper-ex2200-24t # Set tags based on Vendor Class
dhcp-host=set:hostname,ma:ca:dd:re:ss # set tags based on mac
dhcp-option=tag:ex4200,encap:43,00,"image-file-name"
dhcp-option=tag:ex4200,encap:43,01,"config-file-name"
dhcp-option=tag:ex4200,encap:43,02,"image-file-type"
dhcp-option=tag:ex4200,encap:43,03,"transfer-mode"
dhcp-option=tag:ex4200,150,10.10.10.10 # file server
dhcp-range=tag:ex4200,10.10.10.2,10.10.10.100,255.255.255.0,1500m
```

Video

- Zero Touch Provisioning / Junos OS for EX and QFX Series Switches

ZTP Process Extended





Links

- Example ztp.slax

SRX USB Boot

- The SRX Series USB Auto install feature simplifies the upgrading of Junos OS images, when there is no console access to an SRX Series device that is located at a remote site. This feature allows you to upgrade the Junos OS image with the minimum configuration effort by just inserting a USB flash drive into the USB port of the SRX Series device and performing a few simple steps. This feature can also be used to reformat the boot device and recover the SRX Series device after boot media corruption.
- Before you begin the installation, ensure that the following prerequisites are met:

  The Junos OS upgrade image and autoinstall.conf files must be copied to the USB device. The instructions on how to copy the image

Questions

- What happens when no client goes to bound state?
- What happens when no client has sufficient ZTP options?
- What happens when file server failed to respond to ZTP request?
- What happens when file downloaded is corrupted?
- What happens when config file is errenous?

- Device restarts dhcp client state on all configured interfaces

Troubleshooting Commands

- show dhcp client binding interface vme detail
- show dhcp client statistics
- request dhcp client renew interface <>
- request dhcp client renew all
- show dhcp client binding
- show dhcp client binding brief
- show dhcp client binding interface <>
- clear dhcp binding all
- clear dhcp binding interface <>

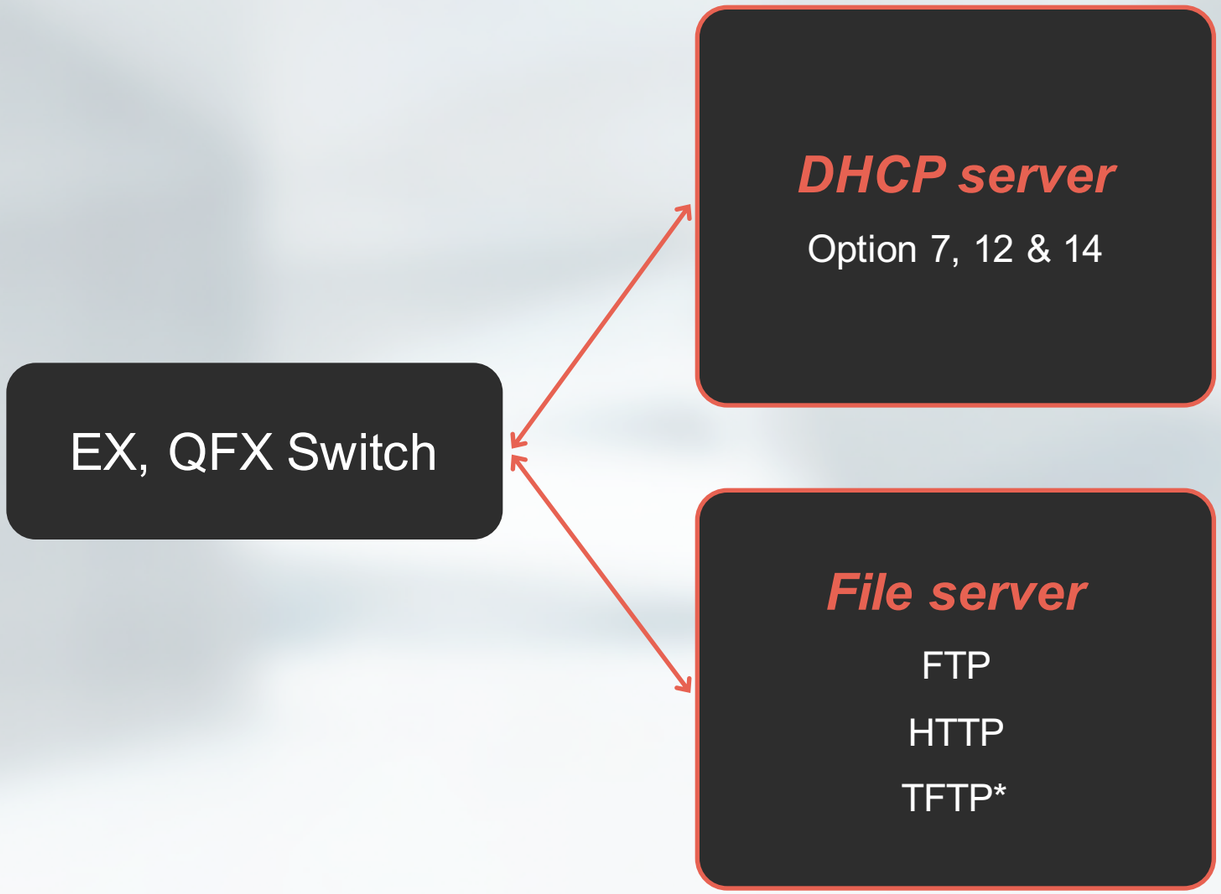
Further Troubleshooting

- DHCP discovery / reply packets are reaching the server?
  - tcpdump –i <interface> udp
- Check dhcp_log file on device
- File Server Reachable?
  - netstat -rn
  - Ping

Video

- Use Junos Space Network Director – To Facilitate Zero Touch Provisioning

Build: Zero Touch Provisioning



- Minimal skill required by onsite deployment team
- Ensure Consistent deployment in line with company policies
- Reduce Data Center Build out from days to minutes
- Network Engineers not tied up bootstrapping boxes
- Enable creative solutions with SLAX scripts
- No Vendor lock in

Bootstrapping with ZTP

- ZTP can be leveraged to bootstrap an automation process
- The device can load a configuration allowing access to Ansible/Puppet/Chef
  - Example…Defining a user and enabling
- The device can register itself with Junos Space
- The device can receive and on box script, directing it to take further action
  - Example, query a web server, sending the devices lldp neighbors…
  - The web server then takes the information and generates a custom configuration for that device

Compatibility

All EX, QFX, and Some SRX Series Devices

EX2200 v12.2R1 and above
EX2200-C v12.2R1 and above
EX3200 v12.2R1 and above
EX3300 v12.5R5 and above
EX4200 v12.2R1 and above
EX4500 v12.2R1 and above
EX4550 v12.2R1 and above

QFX3500 v12.3X50-D10 and above
QFX3600 v12.3X50-D10 and above
QFX5100 v13.2X51-D15 and above

Unofficially
Some SRX Series Devices

Junos Space – Network Director v13.1P5 and above

Server Provisioning Tools

Puppet – Chef – Ansible

Servers?

Server Provisioning? I thought we were working with Junos Devices?

- The new DevOps is NetOps
  - Server Provisioning tools define states, not processes or scripts
    - Example:
      - Ensure package nginx is latest version
  - Generally defined in a domain specific language like ruby or yaml
  - Defining the state that the device should be in
    - Should have a route from a -> b…etc.
  - Faster implementation, lower failure rates, shortens times between fixes
  - Brings the networking environment closer to the concept of continuous delivery

DevOps for NetOps

HISTORY

Evolution / Revolution

- Server Virtualization and Cloud
- History over +7 years
- Open-Source Community

manually configured

ad-hoc bash
Perl scripting

puppet, chef
salt, ansible,
other IT
frameworks

infra.apps
built on IT
frameworks
(Hubot, Boxen)

physical,
virtual, cloud
orchestration

paradigm pivot-point!

Puppet

http://puppetlabs.com/

Compatible with:

EX 4300 v14.1X53-D10 and above

EX 4600 v13.2X51-D23 and above

QFX 5100 v 14.1X53-D10 and above

Puppet: IT Automation Software for Admins

What is Puppet?

- Puppet is a configuration management system
  - It allows you to define the state of your IT infrastructure
  - Then automatically enforces the correct state

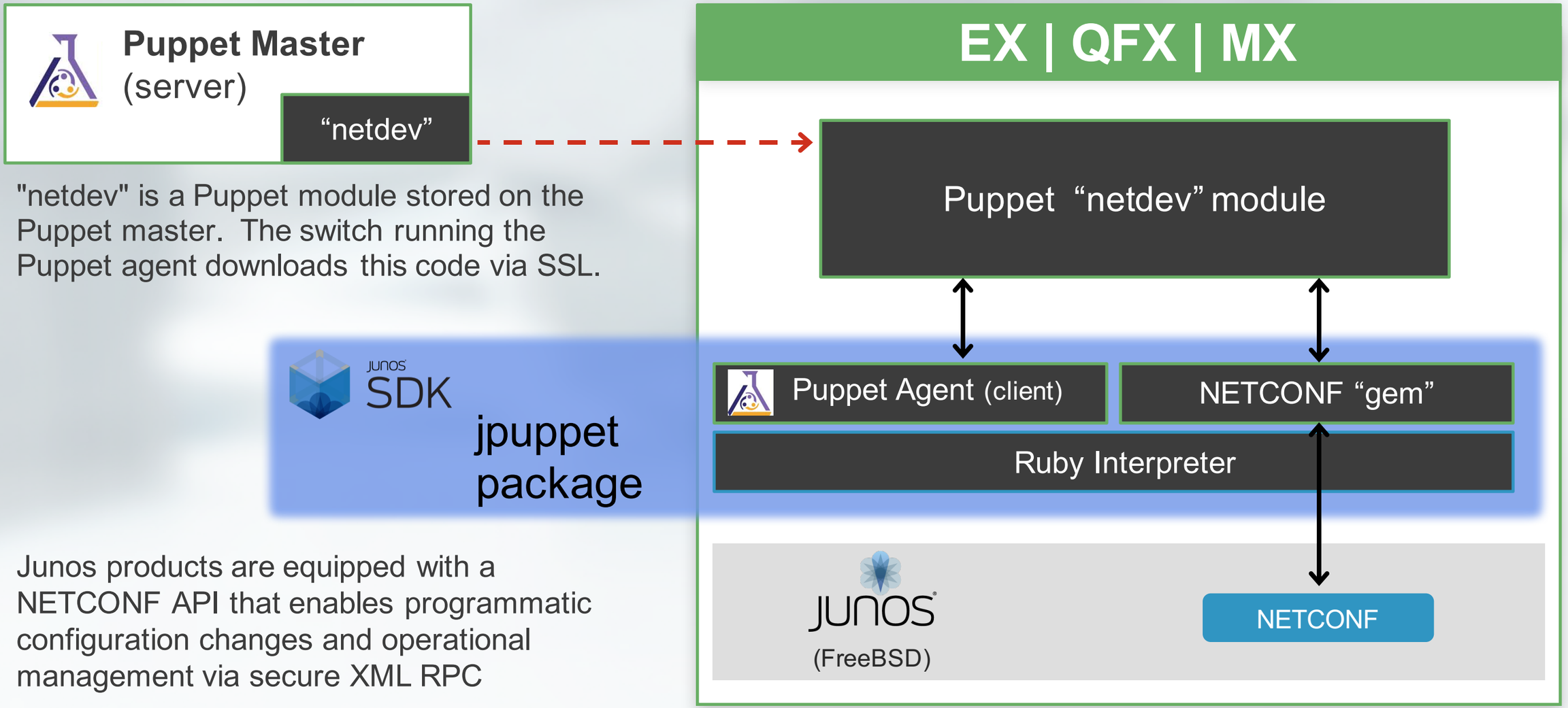
How Puppet helps?

Whether you're managing just a few servers or thousands of physical and virtual machines, Puppet automates tasks that admins often do manually, freeing up time and mental space so admins can work on the projects that deliver greater business value.

Puppet: IT Automation Software for Admins

- How Puppet works
  - Puppet language is declarative (based on ruby), rather than procedural, meaning you tell Puppet what results you want, rather than how to get there
  - Its language is clear, simple and concise, easy for pretty much anyone to read and understand
  - This clarity facilitates easier collaboration between admins and colleagues on other teams, such as software developers, testers, and network administrators

Configure: Puppet



"netdev" is a Puppet module stored on the Puppet master. The switch running the Puppet agent downloads this code via SSL.

Junos products are equipped with a NETCONF API that enables programmatic configuration changes and operational management via secure XML RPC

Chef

https://www.getchef.com/chef/

Compatible with:

EX 4300 v14.1X53-D10 and above

EX 4600 v13.2X51-D25 and above

QFX 5100 v14.1X53-D10 and above

Chef: IT Automation for Speed and Awesomeness

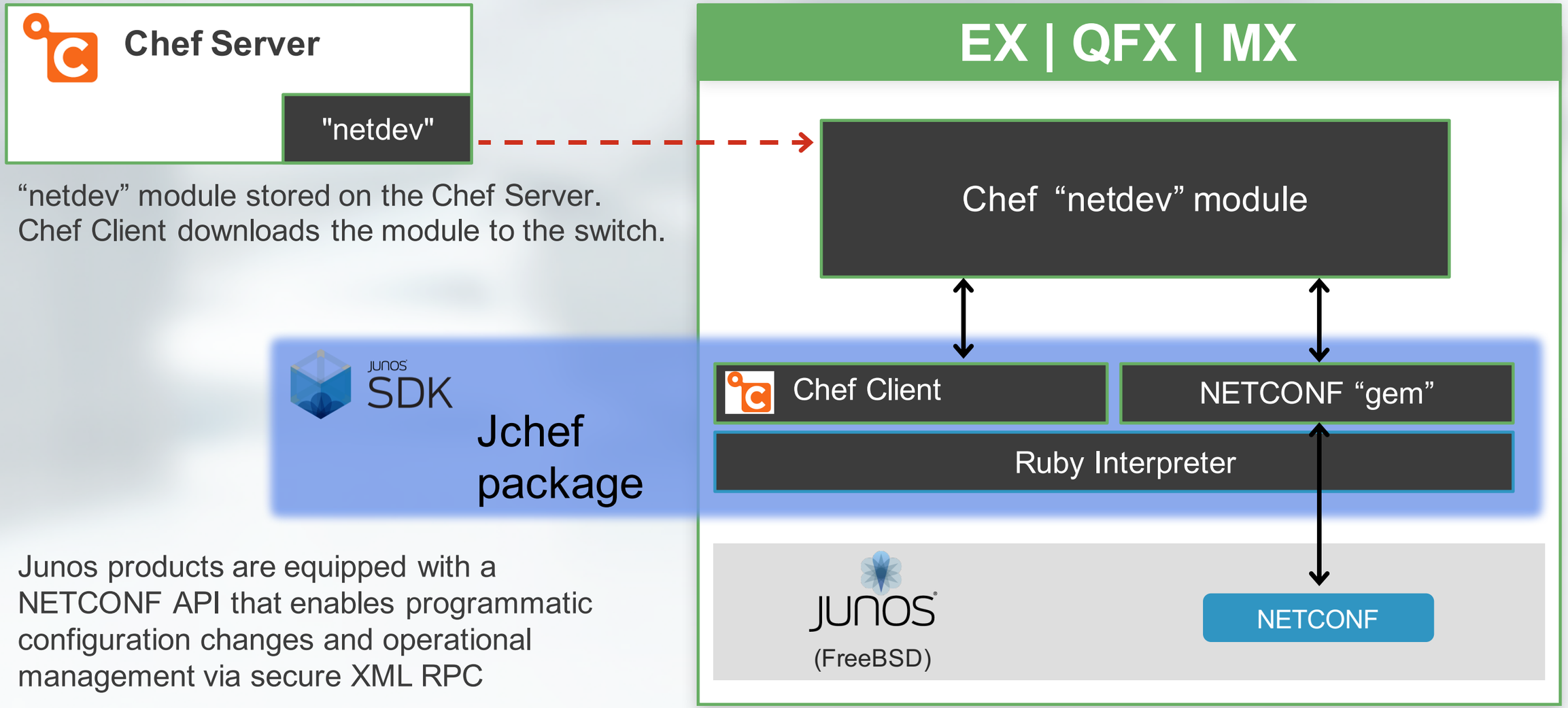
What is Chef?

- Chef turns infrastructure into code. With Chef, you can automate how you build, deploy, and manage your infrastructure. Your infrastructure becomes as version-able, testable, and repeatable as application code.

What are recipes?

Chef relies on reusable definitions known as recipes to automate infrastructure tasks. Examples of recipes are instructions for configuring web servers, databases and load balancers. Together, recipes describe what your infrastructure consists of and how each part of your infrastructure should be deployed, configured and managed.

Configure: Chef



"netdev" module stored on the Chef Server. Chef Client downloads the module to the switch.

Junos products are equipped with a NETCONF API that enables programmatic configuration changes and operational management via secure XML RPC

Ansible

http://www.ansible.com/

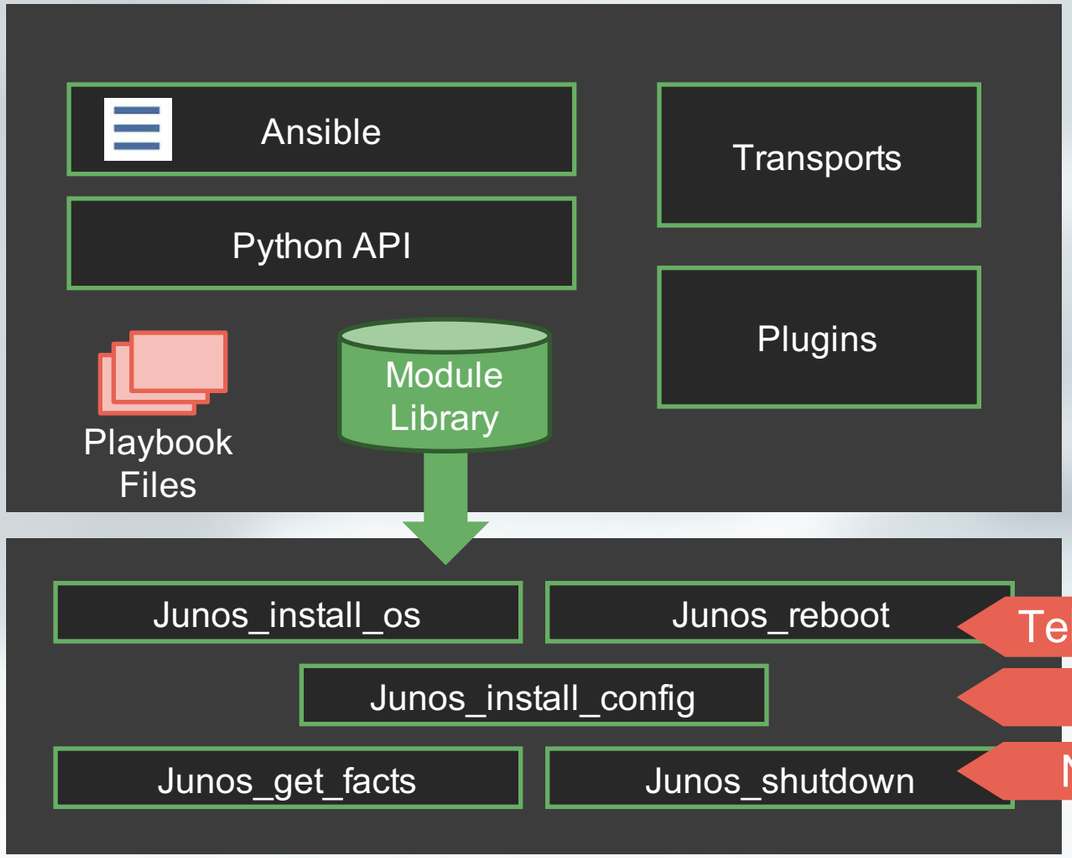
Compatible with:
All Junos Platforms

Ansible – Simple IT Orchestration

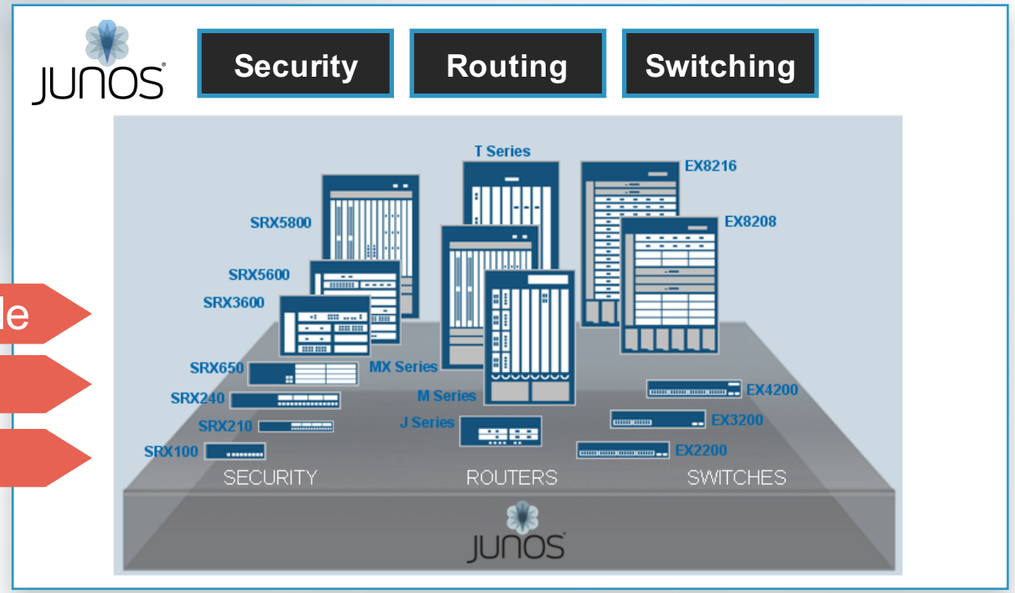
- Ansible is a simple IT automation engine that automates provisioning, configuration management, application deployment, and intra-service orchestration
- Designed for multi-tier deployments, Ansible models your IT infrastructure by describing how all of your systems inter-relate, rather than just managing one system at a time
- It uses no agents and no additional custom security infrastructure
- It uses a very simple language (YAML, in the form of Ansible Playbooks) that allow you to describe your automation jobs in a way that approaches plain English

Build: Ansible

IT Automation Framework



- Agentless and simple approach
- Does not require coding skills
- Work flow Engine
- Ansible can be used for Network/Compute/Storage



Deploy Playbook

deploy.yml

```
---
### ------------------------------------------------------------------
### deploy.yml
### ------------------------------------------------------------------
###
###   1. check reachability via NETCONF
###   2. upgrade Junos OS if necessary; reboot & wait for restart
###   3. template build device specific Junos configuration
###   4. load/override the Junos configuration file
###
### ------------------------------------------------------------------

### ------------------------------------------------------------------
###    First ensure the hosts are reachable via the NETCONF protocol
### ------------------------------------------------------------------

- include: junos/nc_ready.yml timeout=1

### ------------------------------------------------------------------
### Install Junos OS on devices that need it
### ------------------------------------------------------------------

- include: junos/install_os.yml

### ------------------------------------------------------------------
### Now generate the target specific junos.conf initial config
### and install it on the target
### ------------------------------------------------------------------

- include: config/make.yml
- include: config/install.yml

```

Ansible-junos-stdlib

All Junos devices

- junos_get_facts — Retrieve device-specific information from the host
- junos_install_config — Modify the configuration of a device running Junos OS
- junos_install_os — Install a Junos OS software package
- junos_shutdown — Shut down or reboot a device running Junos OS
- junos_zeroize — Remove all configuration information on the Routing Engines and reset all key values on a device

Juniper Networks provides support for using Ansible to deploy devices running the Junos operating system (Junos OS). The Juniper Networks Ansible library, which is hosted on the Ansible Galaxy website under the role junos, enables you to use Ansible to perform specific operational and configuration tasks on devices running Junos OS, including installing and upgrading Junos OS, deploying specific devices in the network, loading configuration changes, retrieving information, and resetting, rebooting, or shutting down managed devices.

Playbook to Load Junos Software

idempotent

install_os.yml

```
### ---------------------------------------------------------------------
### Install Junos OS image as specified by the host's
### junos_os_tag variable
### ---------------------------------------------------------------------

- hosts: all
  name: Junos OS image installation
  connection: local
  gather_facts: no
  vars_files:
    - /usr/local/junos/packages/catalog.yml

  tasks:
    - name: Junos OS install, please wait, this could take a bit ...
      junos_install_os: >
        host={{ inventory_hostname }}
        version={{ PACKAGE_TAGS[junos_os_tag].version }}
        package={{ PACKAGE_DIR }}/{{ PACKAGE_TAGS[junos_os_tag].package }}
        reboot=True
      notify:
        - Junos reboot
        - Junos wait for restart

  handlers:
    - name: Junos reboot
      pause: seconds={{ reboot_wait_time }}
    - name: Junos wait for restart
      wait_for: host={{ inventory_hostname }} port=830
```

`junos_install_os` performs installation of Junos OS only if the device does not have it already installed

`handlers` only called if OS is changed

Jinja2 Templates

What is Jinja2

- Jinja2 is a widely used templating engine for Python.
  - Design and implementation is intuitive
  - Template inheritance
    - Allows you to build a base “skeleton” template that contains all the common elements of your site and defines **blocks** that child templates can override.
  - Easy to debug
    - Line numbers of exceptions directly point to the correct line in the template.
  - Execution is very fast
  - Widely adopted

Template basics

Variables are delimited by {{ … }}

Expressions (conditionals, loops, macros, blocks) by {% … %}

```
system {
    host-name {{ host_name }};
    name-server {
        {% for dns_server in dns %}
          {{ dns_server }};
        {% endfor %}
    }
}
```

If statements

Template:

```
user {{ user.name }} {
    authentication {
     {% if user.passwd %}
          encrypted-password "{{ user.passwd }}";
     {% elif user.key %}
          ssh-rsa "{{ user.key }}";
     {% endif %}
    }
}
```



If statements – cont.

Rendered:

```
user bob {
    authentication {
        encrypted-password "$1$wDeov1";
    }
}
```

For loops

Template:

```
{% for interface_name in core_interfaces %}
        {{ interface_name }} {
            output-traffic-control-profile CORE-MAP;
    }
 {% endfor %}
```

For loops – cont.

Rendered:

```
xe-4/1/0 {
    output-traffic-control-profile CORE-MAP;
    }
xe-4/2/0 {
    output-traffic-control-profile CORE-MAP;
    }
xe-4/3/0 {
    output-traffic-control-profile CORE-MAP;
    }
```

Filters

- Filters use methods to manipulate data.

Template:

```
{% for interface_name in core_interfaces %}
        {{ interface_name }} {
            description “{{ desc|uppper }}”;
    }
{% endfor %}
```

Filters – cont.

Rendered:

```
xe-4/1/0 {
    description “UPSTREAM CON”;
    }
xe-4/2/0 {
    description “DOWNSTREAM CON”;
    }
xe-4/3/0 {
    description “MIRROR PORT”;
    }
```

Include

- Templates are able to directly include the content of files and other templates.
- This allows you to split a configuration into multiple pieces.

Template:

```
system {
    {% include “login.j2” %}
    {% include “services.j2” %}
}
```

Include – cont.

Rendered:

```
system {
    login {
        user rick {
      class super-user;
    }
    services {
        ssh;
    }
}
```