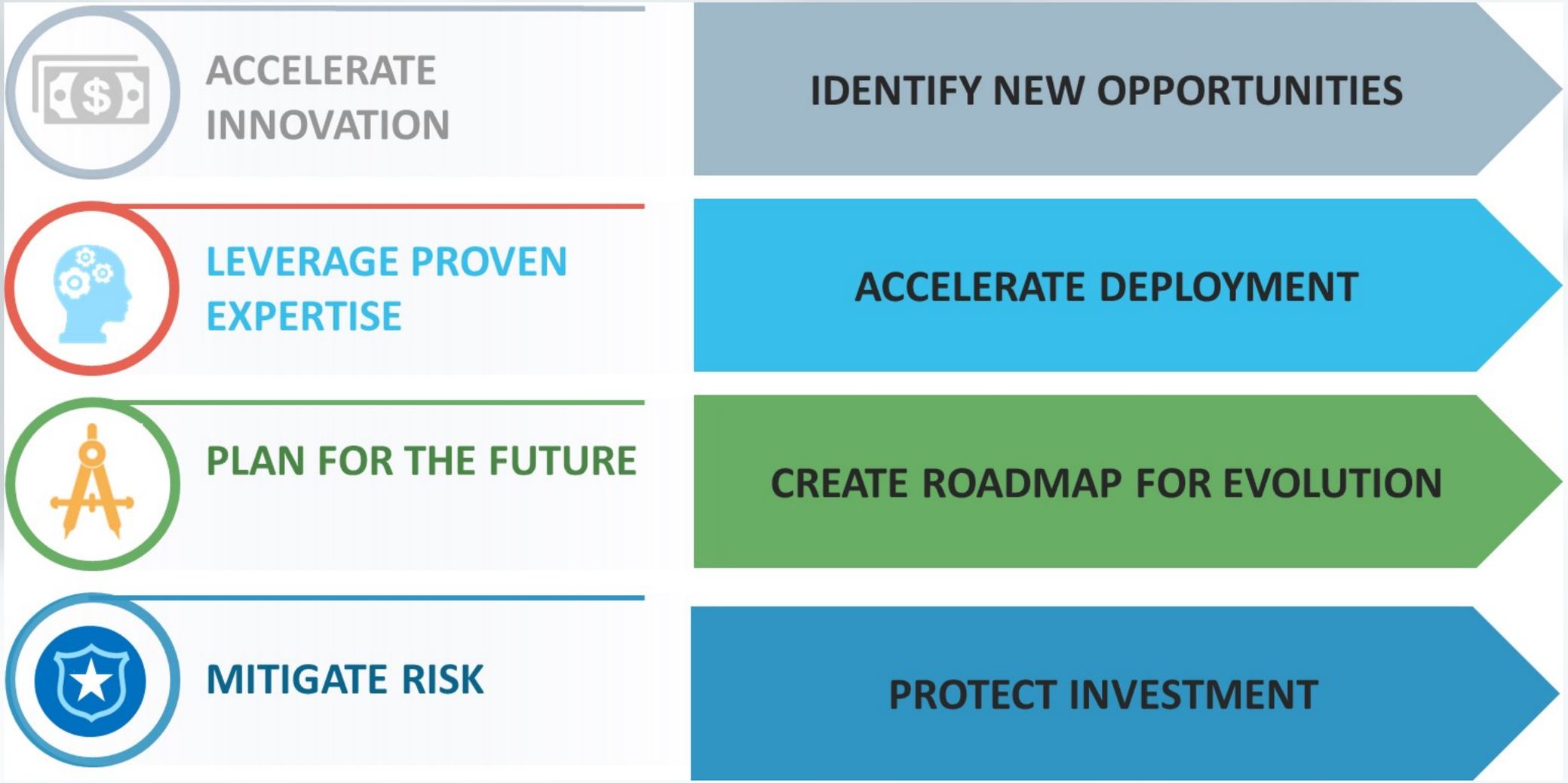
Automation

A Simple Idea, with Complex Concepts

Jim Price, Sr. Manager Professional Services
James Nickerson, Lead Developer Professional Services

Consider Today Within The Context Of…



Services Across Your Network Lifecycle

Enabling you to drive more value from the network

AGENDA

- Automation
  - What is it?
  - Why is it important?
- Solutions
  - Types of automation
  - Building blocks: Juniper & Open Source
- Examples
  - Zero Touch Provisioning (ZTP)

AGENDA (Continued)

- Tools
  - Puppet
  - Chef
  - Ansible
  - Netconf
  - SLAX & Juise
- Code Libraries
- Case Studies
- Resources
- Hands-on Exercises

Defining Automation

Webster defines Automation as:

The operation of an apparatus, process, or system by mechanical or electronic devices that take the place of human labor.

- **We define Automation as:**

The use of Standard's based tools (APIs, Scripting, Provisioning Tools, etc.) to perform tasks to meet customer requirements with little to no human interaction required.

Types of Automation:

What are you going to automate?

- Deployment Automation
- Migration Automation
- Operations Automation
- Configuration Generation Automation
- Network Compliance Auditing Automation
- Service Provisioning Automation
- Etc…

Why Automate

Automation: Why

Business value

Business Continuity

- Simplified operations
- Increased operations control
- Standardization
- Pro-active remediation
- Reduced remediation time
- Hardened certification

Reduce Cost

- OPEX savings:
  - Improved efficiency
  - Effective resource utilization
  - Reduced fees

Accelerate Business

- Network Build
- Certification time
- Service creation
- Service elasticity
- ROI

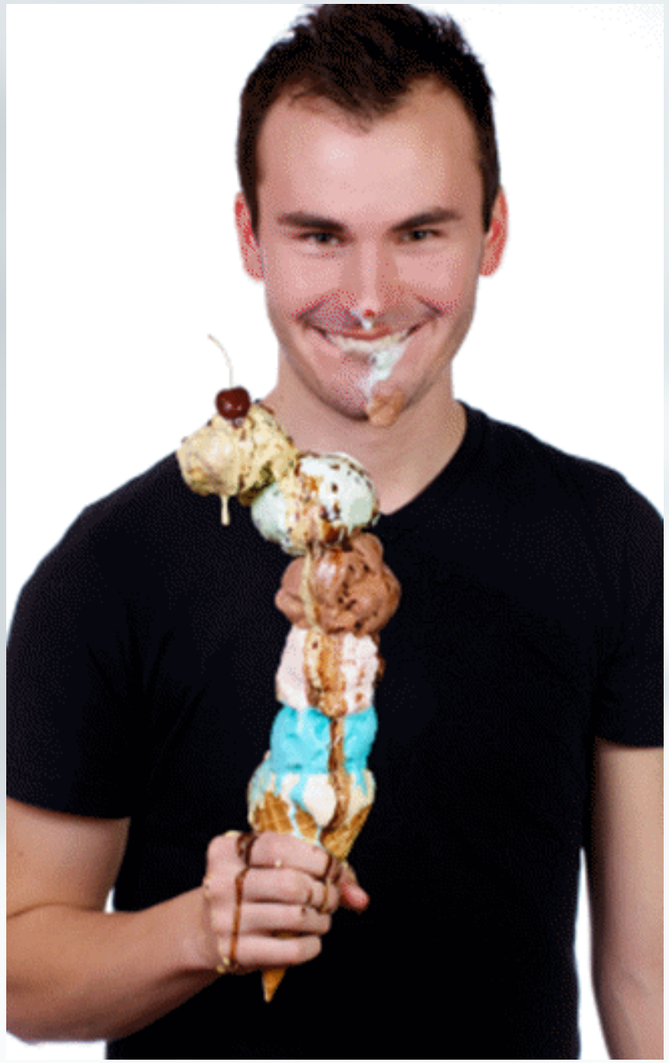
Automation Opportunities continue throughout the Network Lifecycle

PLAN: Design & Certify	BUILD: Implement & Migrate	OPERATE: Audit & Testing
• Design Validation • Product / code • Feature • Workflow	• Zero Touch Provisioning • Configuration Generation • Network Stand-up • Rapid Deployment Migration	• Validation • Configuration Compliance • Run State Compliance • Product Compliance

OPERATE: Product Lifecycle	OPERATE: Service Provisioning	OPERATE: Troubleshoot
• Software Upgrades • Updates Based on Events • Inventory • Alerts & Remediation	• Service Creation • Scale Up/Down • On/Off Box Coordination • Change Impact	• Fault Finding • Remediation • Escalation

How to Build Automation Solutions

Automation is Like Ice Cream



- Everyone want it
- Everyone wants something different
- No-one wants to make it
- No-one wants to clean up the mess

Automation: How do you want it to be?

The ice-cream analogy

Automation capabilities

BUILDING BLOCKS

The Grocery Store

SCRIPT IT

DIY with Kitchen-Aid

PRODUCT

Banana Split at Baskin Robins

Programing Expertise

Automation: How

Juniper building blocks

Network Director
Security Director
Service Now
Service Insight
Configlet

Contrail

On-box

Ruby-EZ
PY-EZ
Snapshot

ZTP
JEAP

Automation: How

Open source building blocks

Programmable Interfaces for Junos

- INNOVATION LEADERSHIP build on One Junos
  - Junos XML API – workflow automation
    - Junos Script
    - XSLT
    - SLAX
    - Netconf
    - Http
  - Junos SDK API – new features and functionality
    - Control Plane API
    - Data Plane API
    - Remote API
  - Junos Space API – intelligent and programmable NMS
  - Contrail API – SDN and NFV orchestration

Deployment Automation

Large Restaurant Chain

Customers Problem

- Traditional deployment of Juniper devices across 1000+ concept restaurants is time-consuming and error-prone

Solution

- Provide a one-touch provisioning solution – a custom web-based solution that is simple, reliable, flexible and scalable to rapidly configure and deploy Juniper devices

Project Challenges

Customers Challenges

- Aggressive schedule
- Integration into existing MS SQL database
- Incorrect data in the database and poorly built templates
- Device bootstrap process

Internal Challenges

- Learning curve – first of its kind
- Limited Initial information from customer
- Large number of moving parts

Solution at a Glance

- Simple
  - Custom Web-based Interface and management
  - Configurations based on minimal input about each site
  - Interface does not require high technical knowledge
- Reliable
  - Leverages pre-defined configuration templates
  - Standardized configuration applied based on selected criteria
  - Minimizes/eliminates configuration errors
- Flexible/rapid deployment
  - Multiple staging environments
  - Accelerated and hassle-free provisioning of multiple devices for multiple stores in parallel

Deploy New Equipment

Sounds simple enough?

Install Device/Role Specific Junos OS Software Image

Install Device/Role Specific Configuration File

But There's Always More Workflow…

Bootstrap
DHCP / TFTP

Install Device/Role Specific
Junos OS Software Image

Install Device/Role Specific
Configuration File

Register Device
with Inventory System

Make a Rapid Deployment Staging Center

So non-techie can stage gear on a bench

Now do it for a “Store”

Multiple devices, different roles

"Front-of-House"
Switch
EX2200

Bootstrap
DHCP / TFTP

Install Device/Role Specific
Junos OS Software Image

Install Device/Role Specific
Configuration File

Register Device
with Inventory System

"Back-of-House"
Switch
EX2200

Bootstrap
DHCP / TFTP

Install Device/Role Specific
Junos OS Software Image

Install Device/Role Specific
Configuration File

Register Device
with Inventory System

CorpVPN
Firewall Router
SRX240

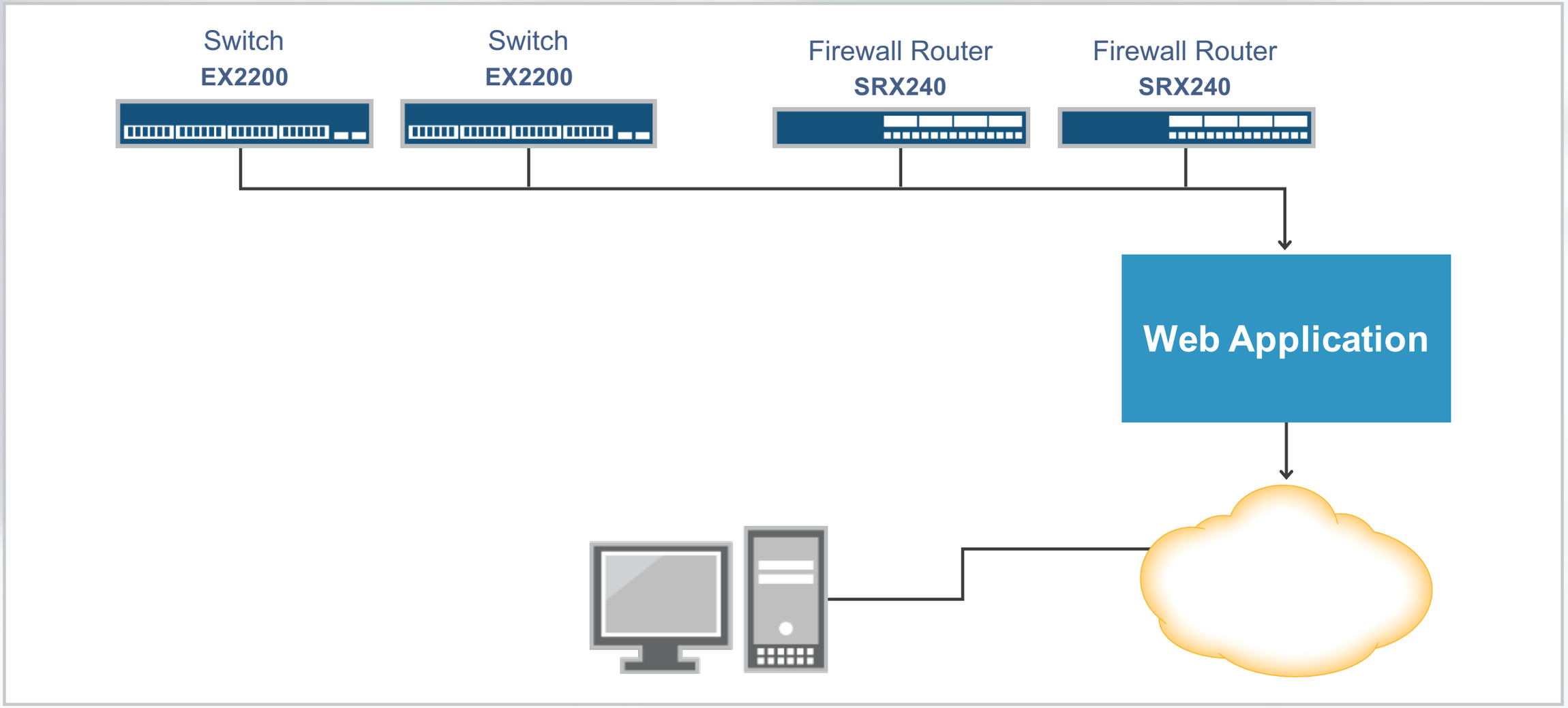
Bootstrap
DHCP / TFTP

Install Device/Role Specific
Junos OS Software Image

Install Device/Role Specific
Configuration File

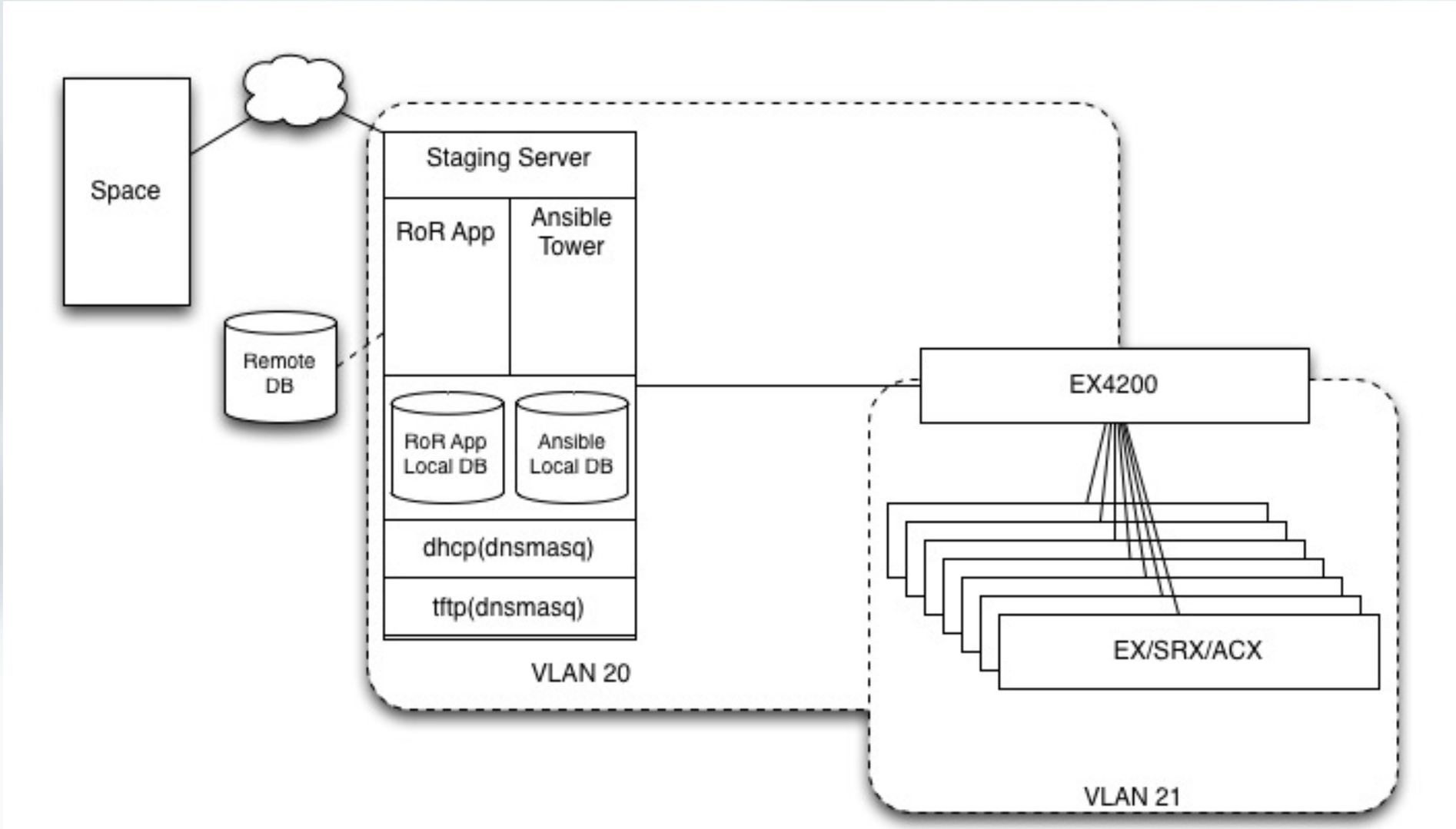
Register Device
with Inventory System

Component Overview



Staging Center Workflow

Overview


Space
Remote DB
Staging Server
RoR App
Ansible Tower
RoR App Local DB
Ansible Local DB
dhcp(dnsmasq)
tftp(dnsmasq)
VLAN 20
EX4200
EX/SRX/ACX
VLAN 21

Staging Flow

- Cable Device
- Bootstrap
- Provision
- Register
- Profit!

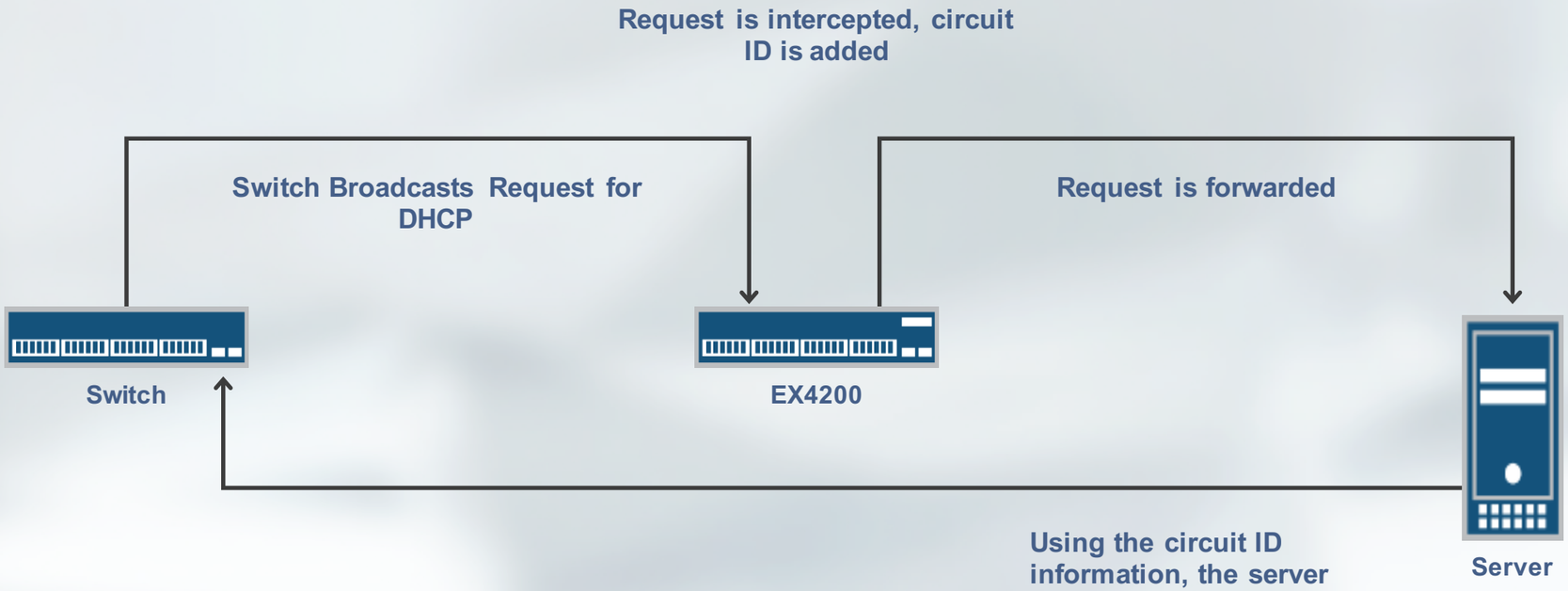
Cable Device

- Each port of the EX4200 corresponds to a port in a “bench” in the deployment dashboard.
- In this case it is Role specific…. Ge-0/0/0 is always an ex2200, ge-0/0/3 is always an srx100…. Etc
- Device is cabled and plugged in… when device comes up, starts autoconf/ztp/bootstrap automatically

Bootstrap

- Device Broadcasts for IP via DHCP….
- Broadcast contained to VLAN 21… EX4200 is setup as a DHCP relay. EX4200 applies circuit-id (“port number:vlan name) to DHCP request… and forwards to Staging Server
- Server determines what IP to issue based on circuit id….(same port always reachable on same ip)
- Server sends back DHCP reply with IP and Pointers to a TFTP server (self), and Conf filename.

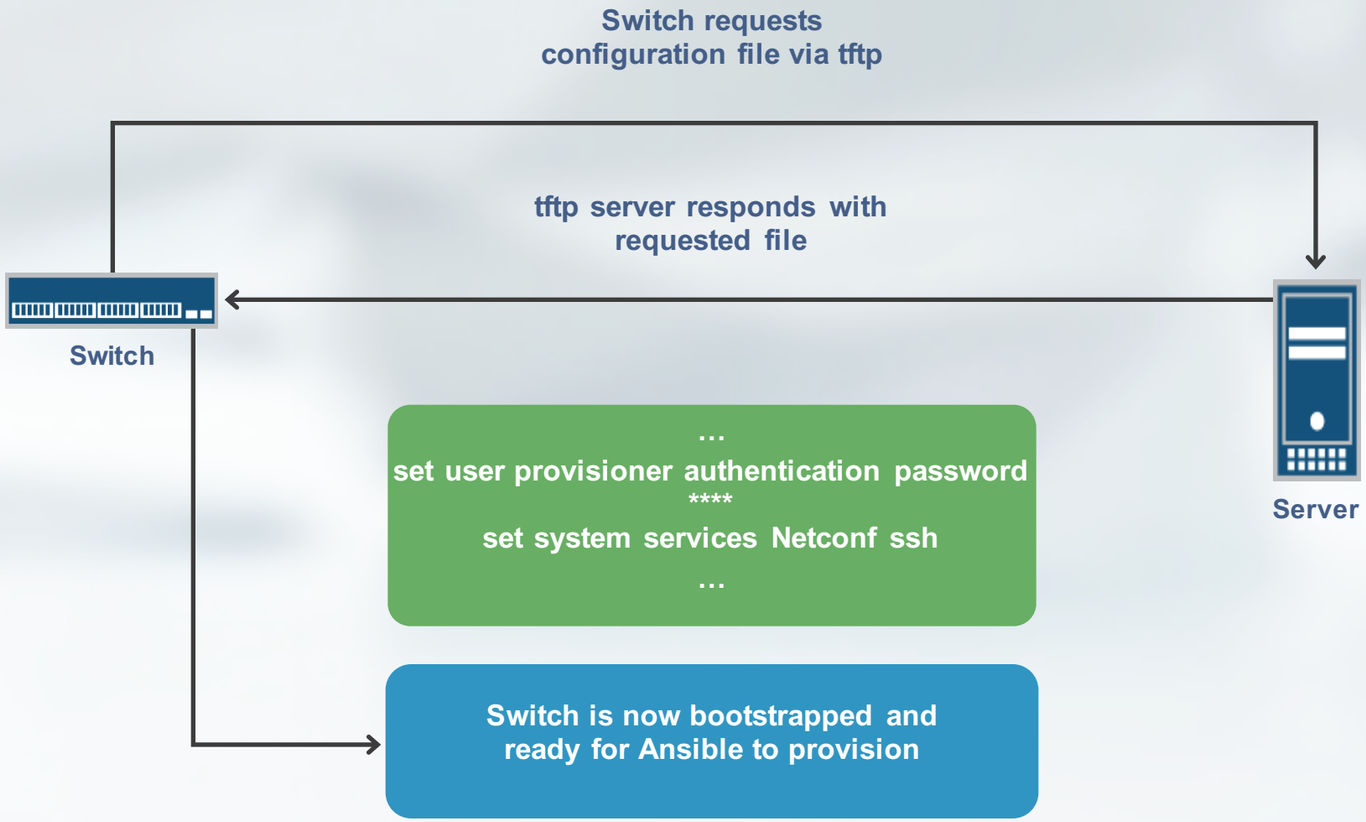
Bootstrap (Cont)



Bootstrap (Cont)

- Device retrieves configuration from TFTP server
- Configuration sets up user credentials, enables Netconf, and LLDP, and sets up IP as static (prevent interface flapping).
- Device is now ready to be provisioned.

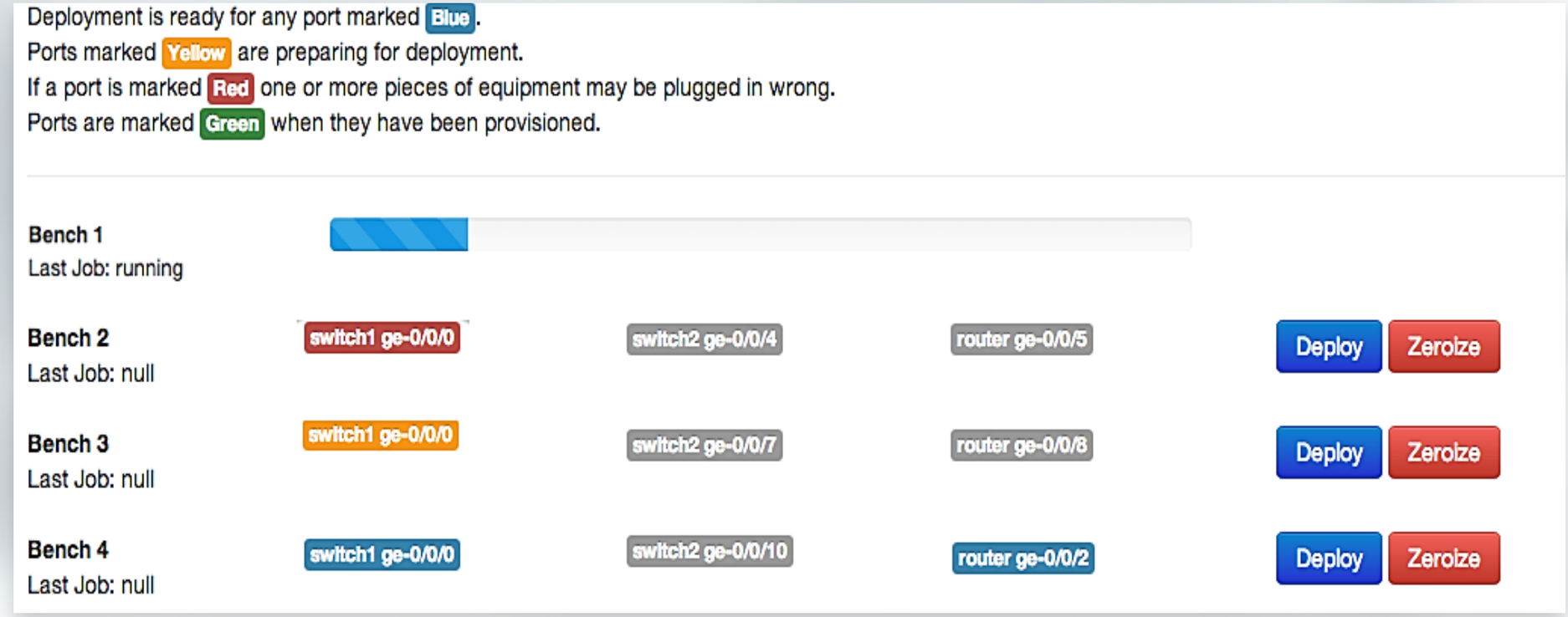
Bootstrap (Cont)



Provision

- Device is now sitting and waiting for provisioning.
- User Can view device status, job status, and issue command to start provisioning from RoR app dashboard.
- Dashboard is updated about device status via Netconf requests to the EX4200 about its LLDP neighbors.

Deployment Dashboard



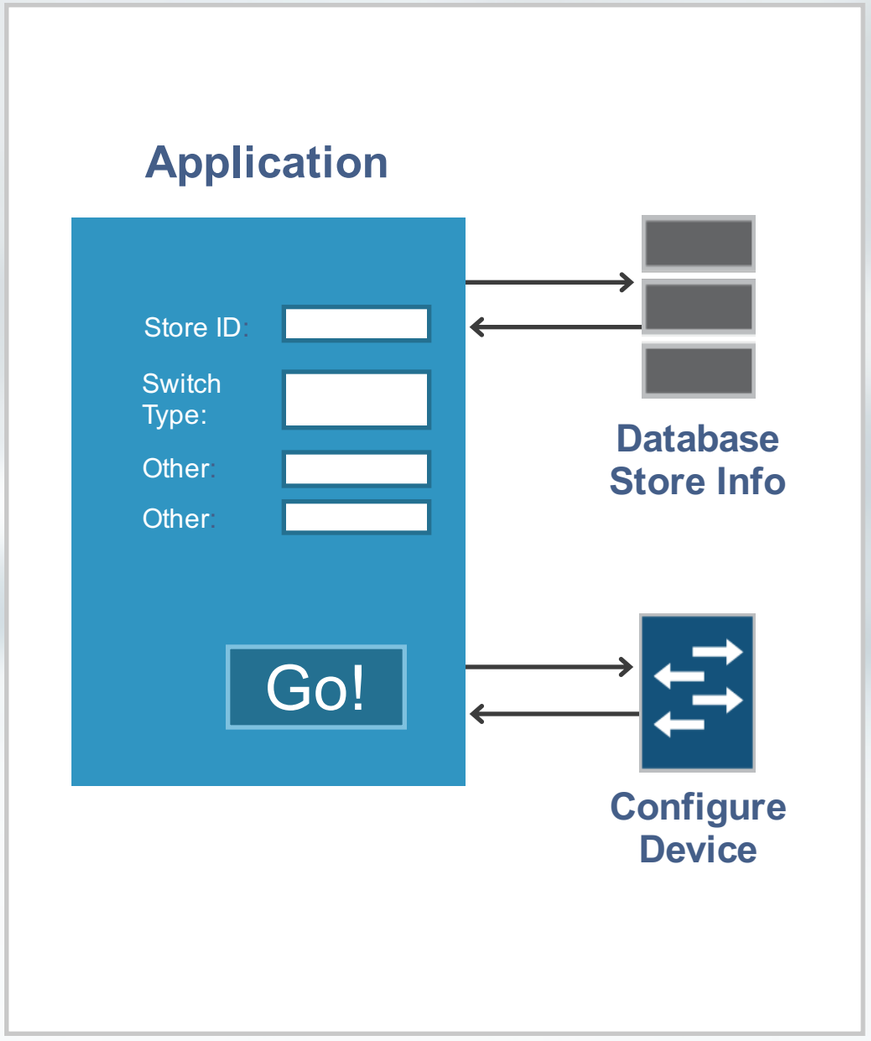
Provision

- User now selects to start provisioning against specified devices… user provides Restaurant Type and code and submits form.
- RoR app records the deployment in internal database, accesses remote database retreiving needed variables for configuration templates.
- RoR app then issues request to Ansible Tower to setup the list of devices to run against, and variables gathered previously.

Provision (cont)

- Ansible Tower Takes Over
- Ansible Tower queues job, and starts running defined “playbook”. Each Task in Playbook is ran in parallel.
- Playbook first checks for connection
- Then creates log directory, and generates configuration file based on variables and templates.
- Then Downgrades/Upgrades device if needed.
- Then applies generated configuration

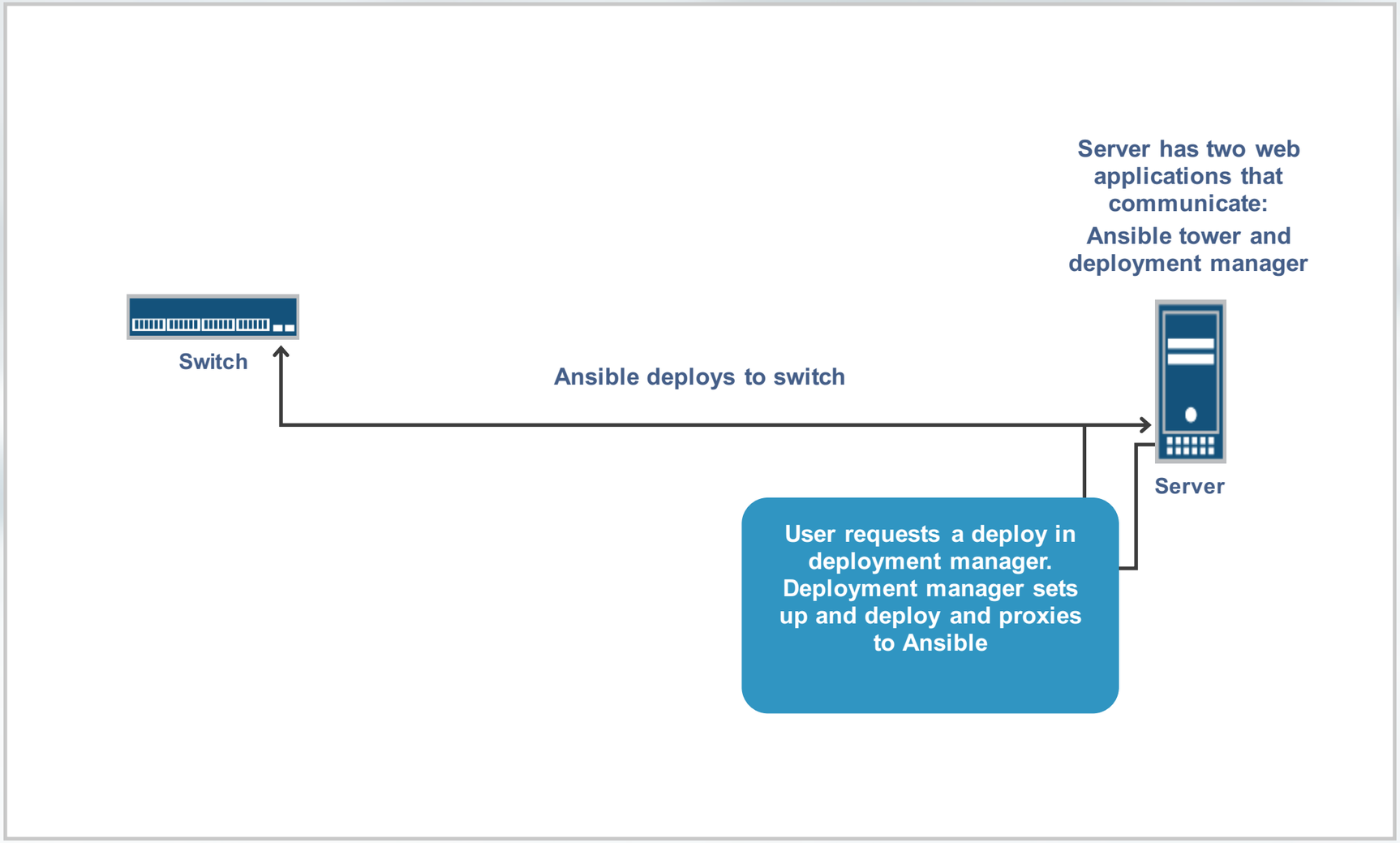
Deployment Workflow



1. Input Information into Application
2. Application Selects Configuration Template (World-Class / Type B vs. Type C)
3. Configuration Created with correct
   1. Model
   2. Host Name
   3. Mac Address
   4. IP Address
   5. Junos O/S Version

   Junos O/S Version updated (if needed)
4. Configuration Applied to Device
5. Device Rebooted to store Configuration
6. Device Ready for Shipping
7.

One-Touch Provisioning

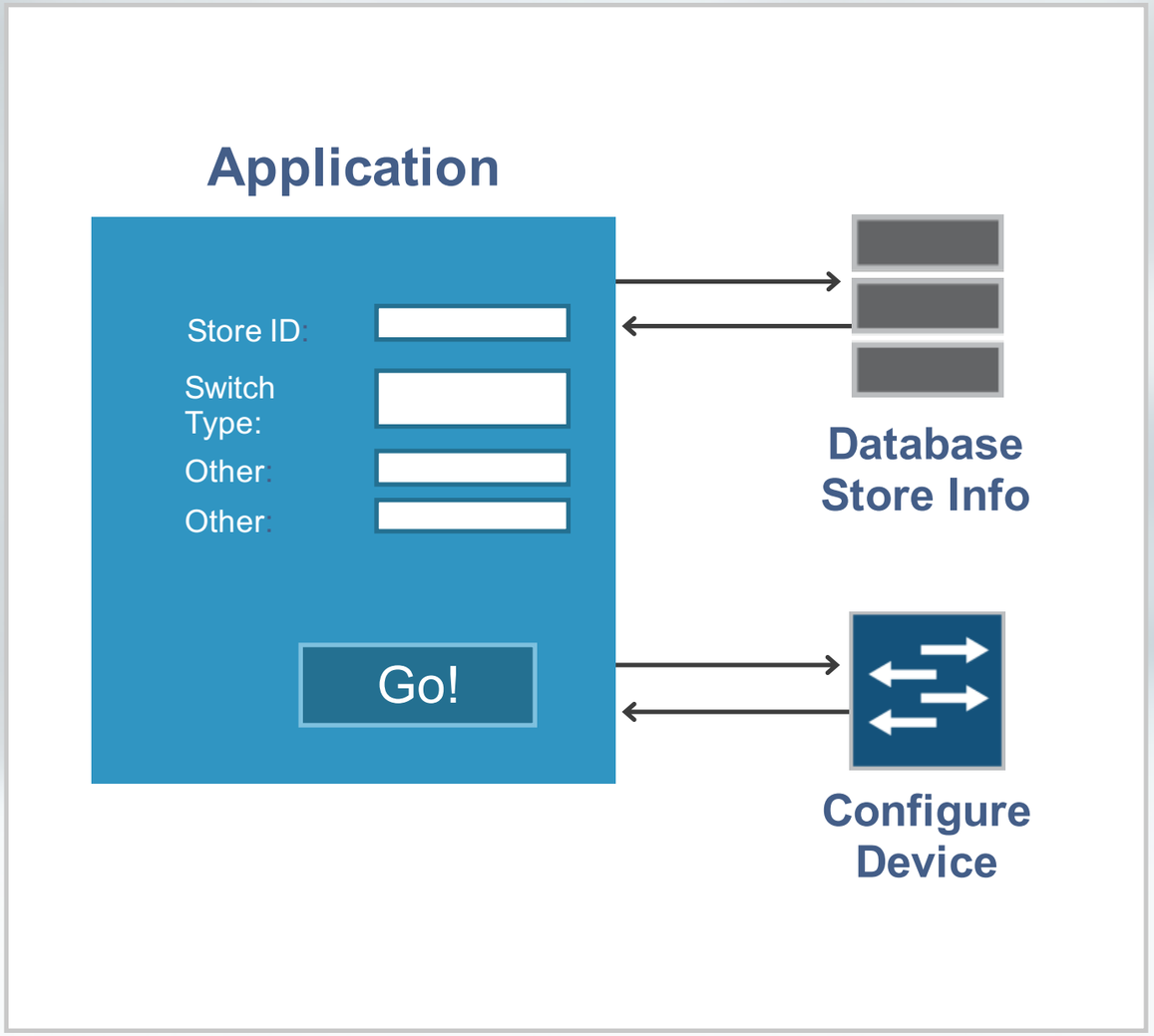


Provision/Register

- Finally, Playbook registers device in Junos Space

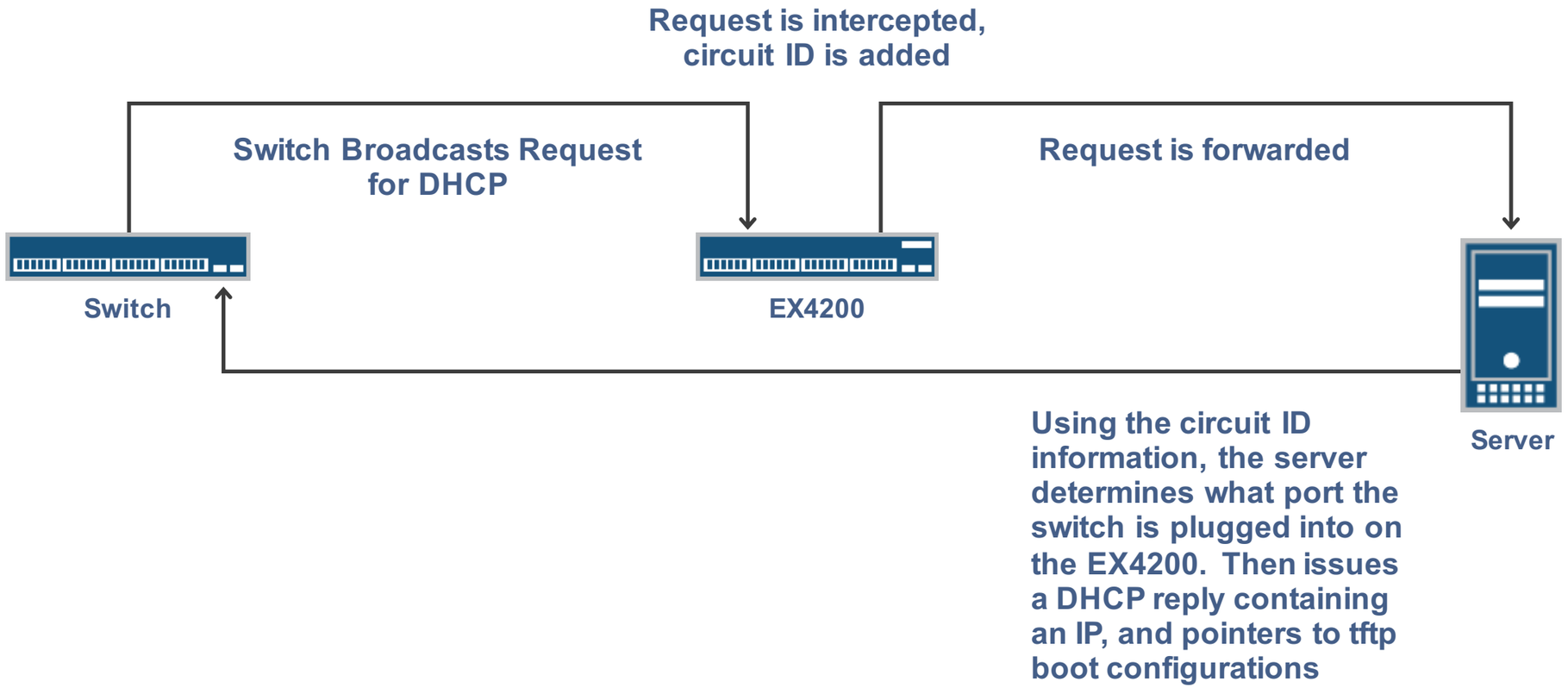
- Device is Ready to Be Shipped/Installed

Deployment Workflow

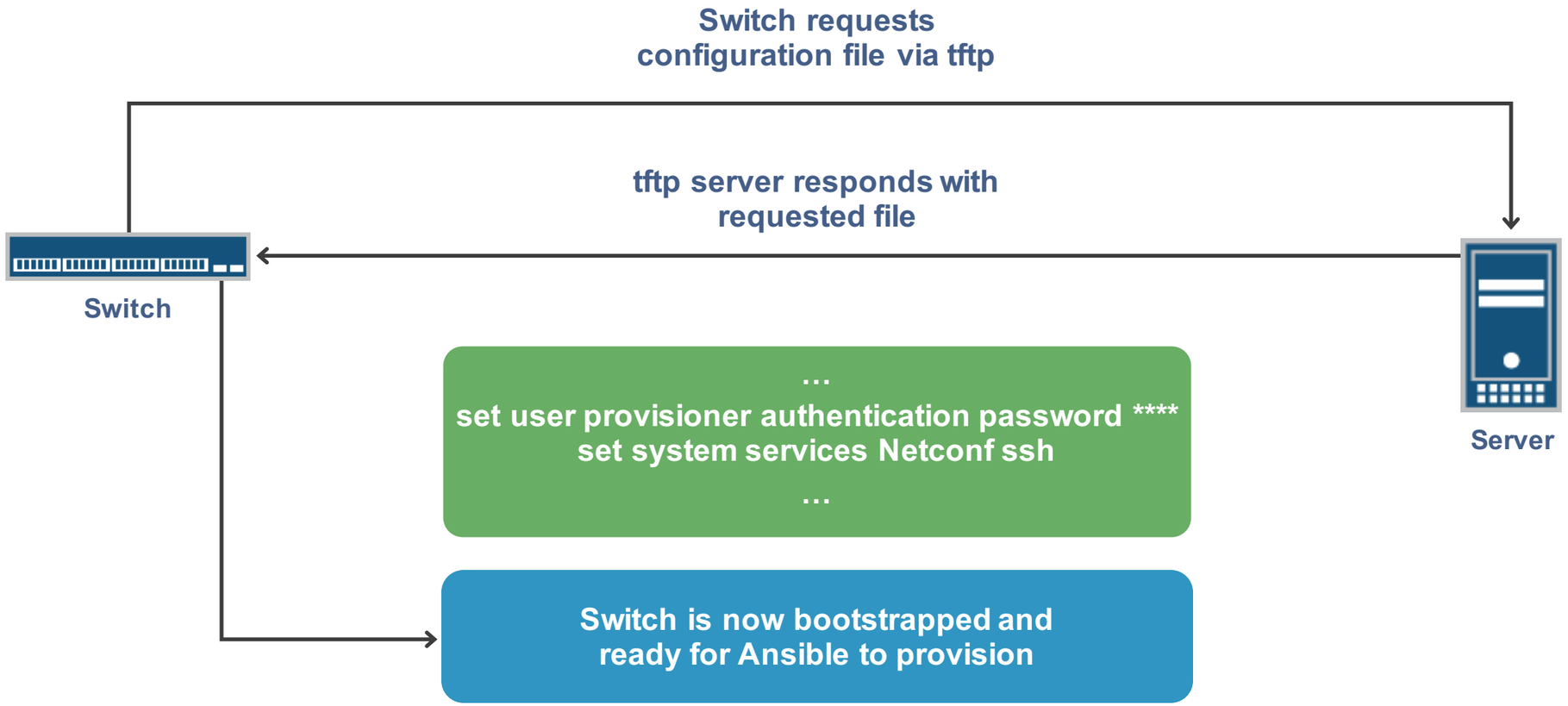


1. Input Information into Application
2. Application Selects Configuration Template (World-Class / Type B vs. Type C)
3. Configuration Created with correct
    1. Model
    2. Host Name
    3. Mac Address
    4. IP Address
    5. Junos O/S Version
4. Junos O/S Version updated (if needed)
5. Configuration Applied to Device
6. Device Rebooted to store Configuration
7. Device Ready for Shipping

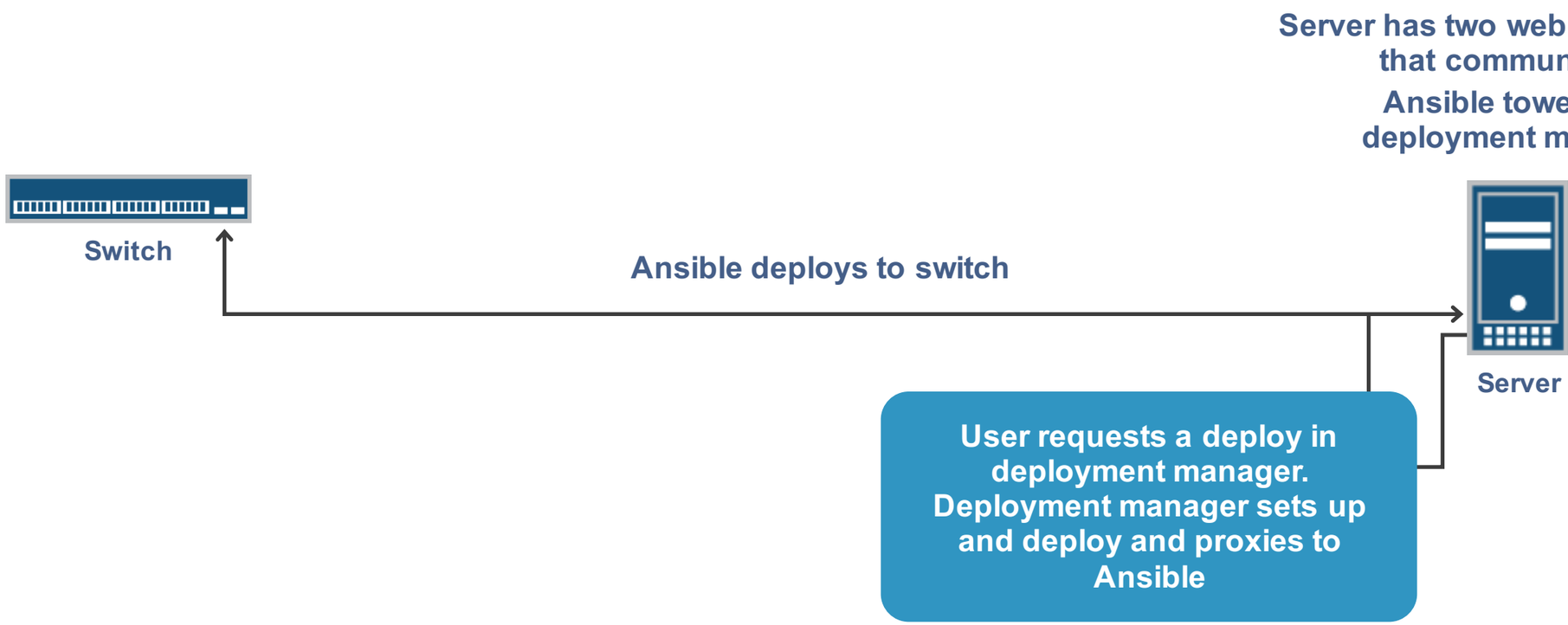
Bootstrap Step – 1



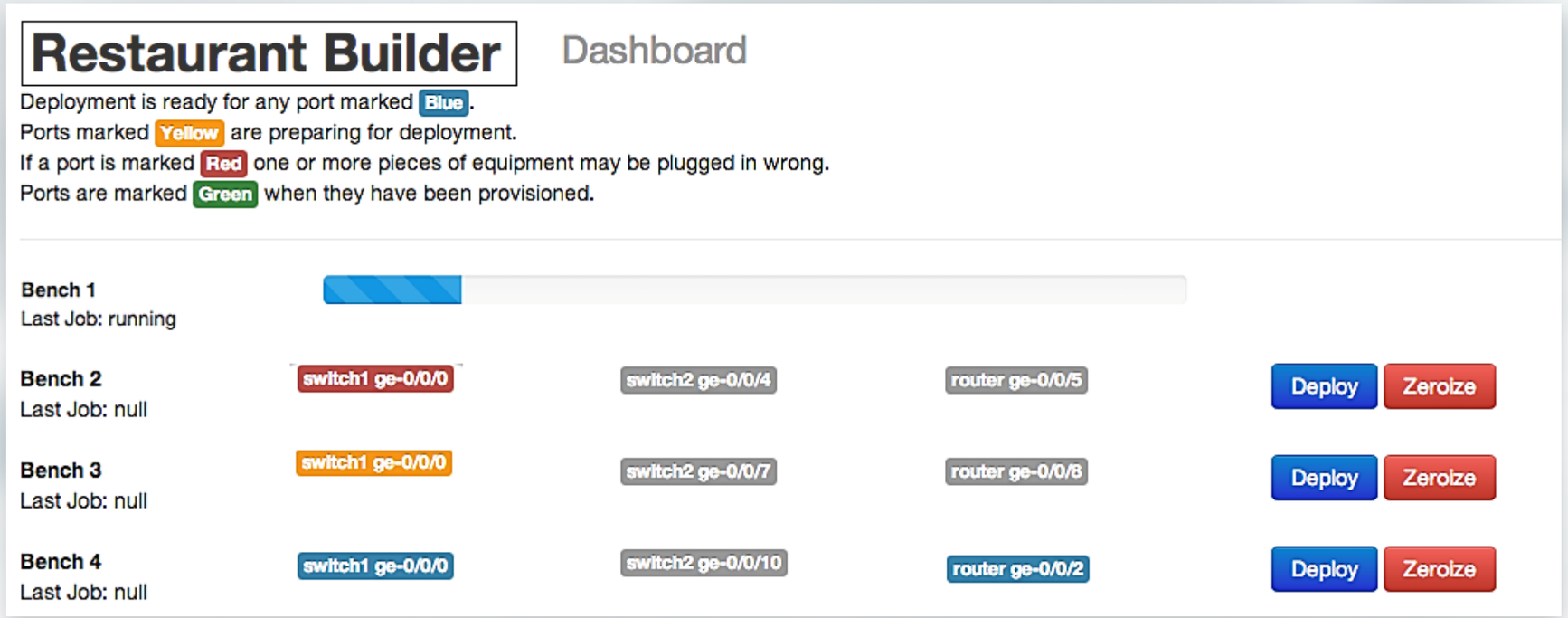
Bootstrap Step – 2



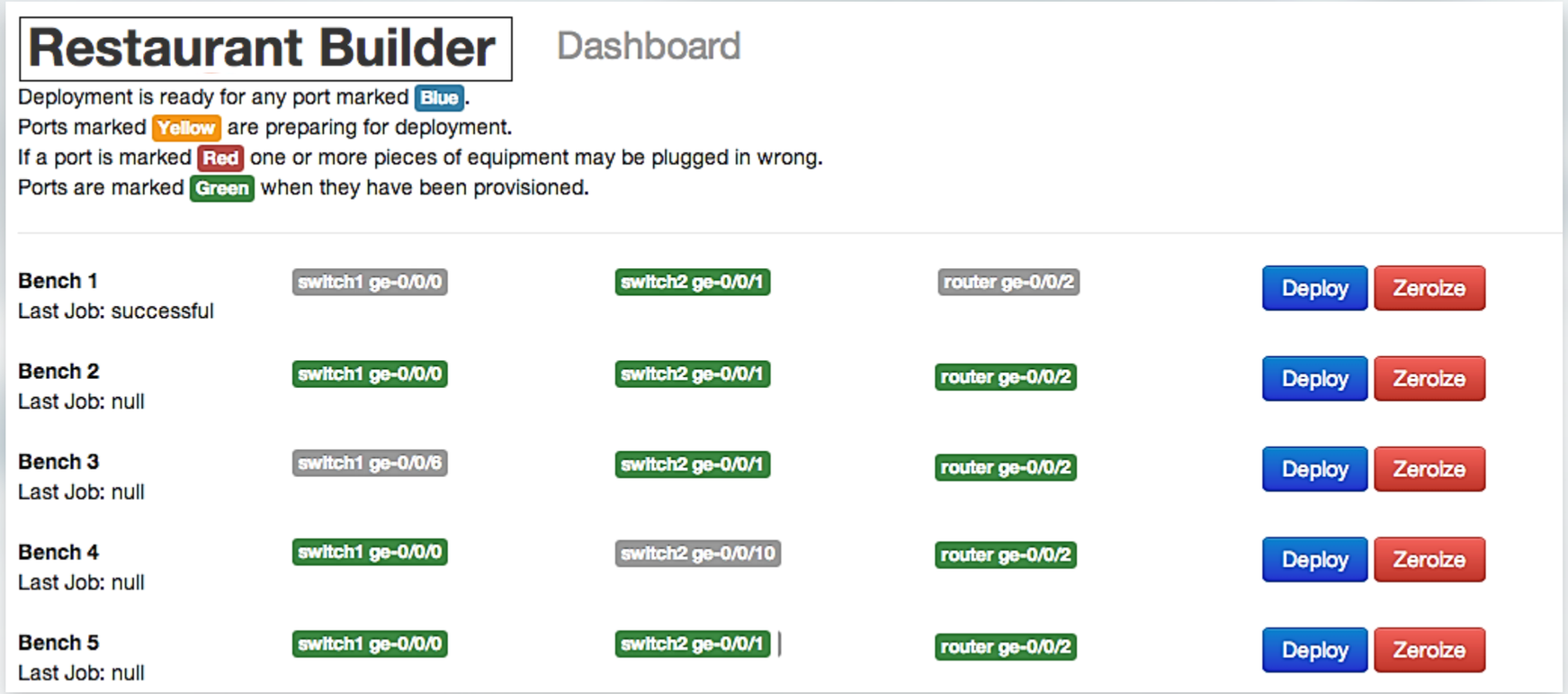
One-Touch Provisioning



Sample Dashboard – Deployment In Progress



Sample Dashboard – Deployment Successful



Project Conclusion

- Provided the Restaurant Builder Tool to automate and parallelize provisioning of Juniper devices
- Delivery
  - Tool was delivered to in July 2014
  - Provided on-site and remote deployment support
  - Provided training and post-delivery support
- Future
  - Tool can be easily generalized for other customers for rapid deployment of new devices
  - Tool can be optimized/cleaned-up further

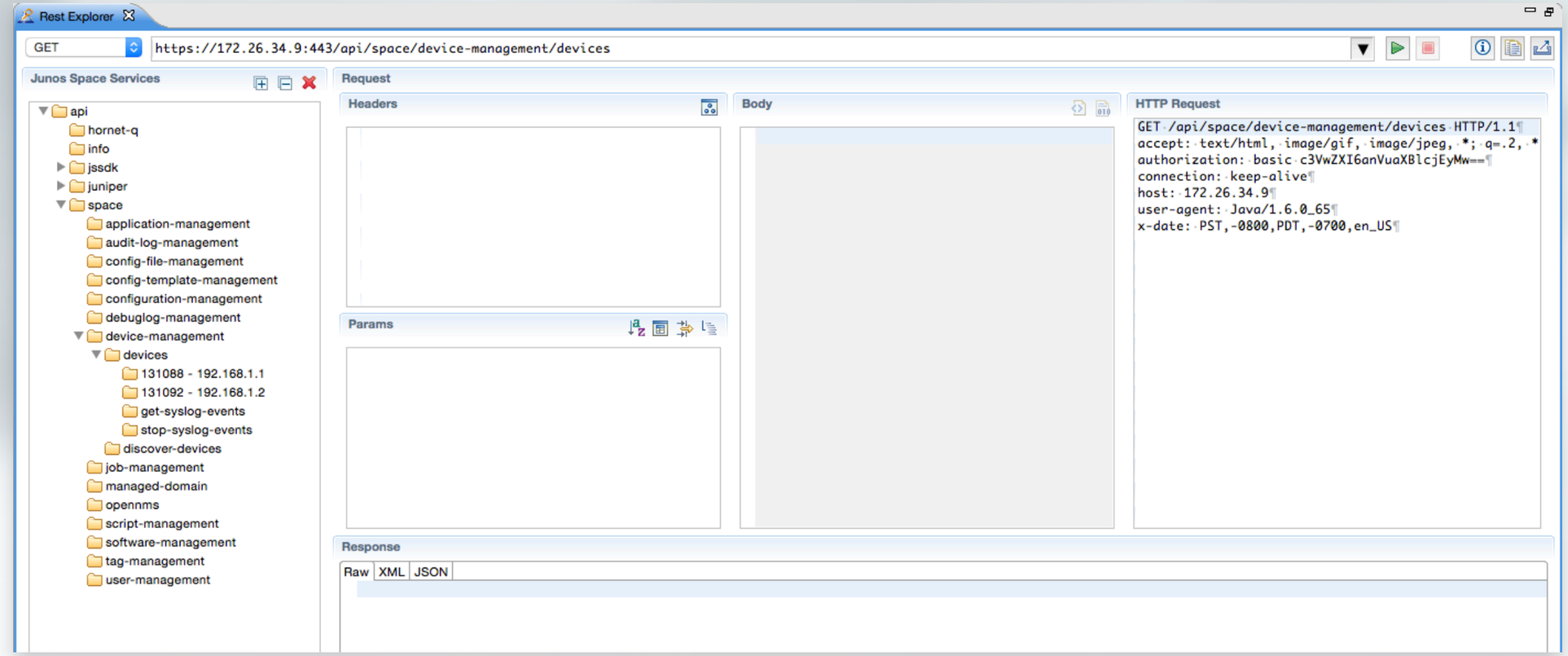
Junos Space

Leverage Junos Space Externally

- Junos Space and its various services provide a large amount of restful APIs
- The APIs can abstract connection and authentication details
- This enables the architect to leverage Junos Space in external scripts and programs
- Generally, if it can be done via Junos Space directly, it can be done via the APIs

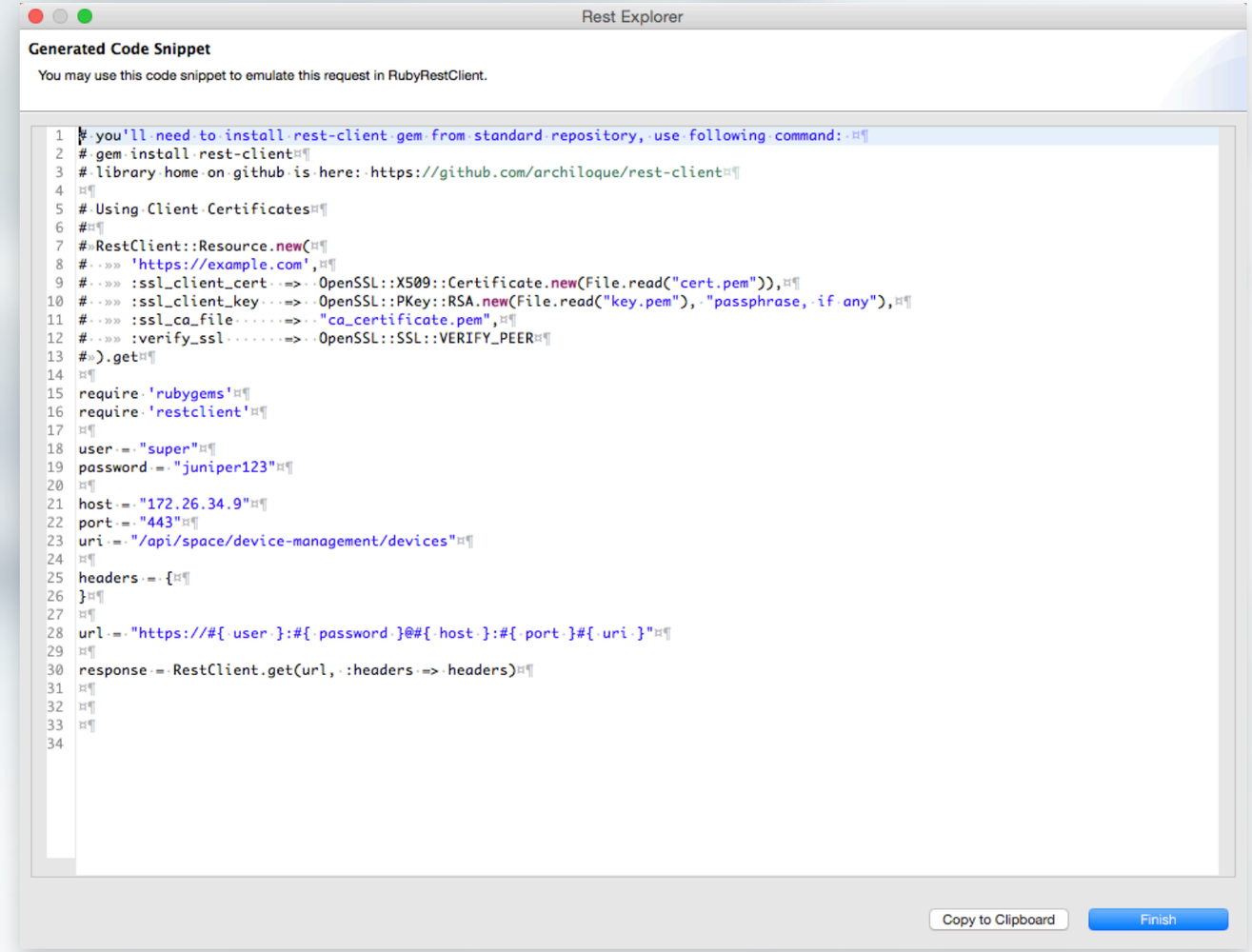
Junos Space API Explorer

- Junos Space SDK plugin for eclipse contains “Rest explorer”
- Can explore the APIs, figure out how to achieve the result, and what parameters are accepted



Junos Space API Code Generation

- Junos Space SDK plugin for eclipse has code generation for popular languages
- Implement and GO!
- Found in Rest Explorer



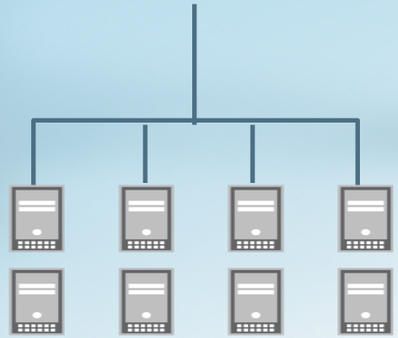
Zero Touch Provisioning

Network Provisioning

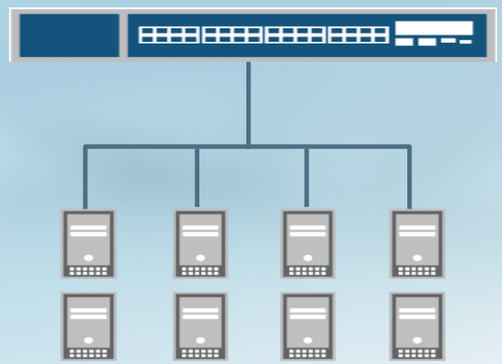
Load image and configure

EX & QFX Series Switches

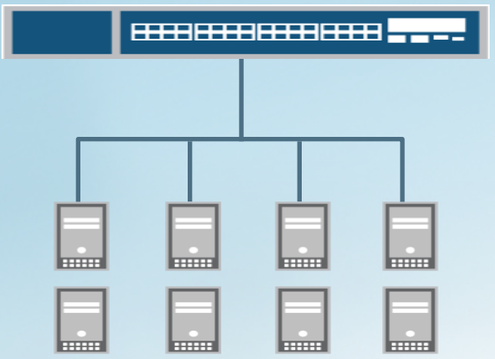
Time Consuming
Manual process requiring switches to be staged;
takes days



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.

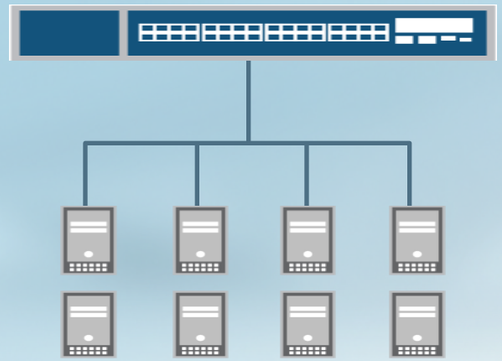
One of the first attempts in this regard was bootstrapping using TFTP standard RFC 906, published in 1984, which established the 1981 published Trivial File Transfer Protocol standard RFC 783 to be used as the standard file transfer protocol for bootstrap loading. It was followed shortly after by the 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

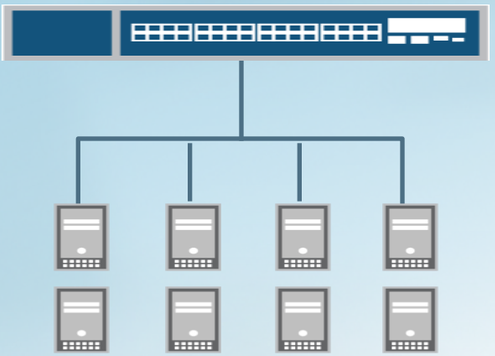
EX & QFX Series Switches

Auto Image and Configure

Auto Image and Configure



Auto Image and Configure

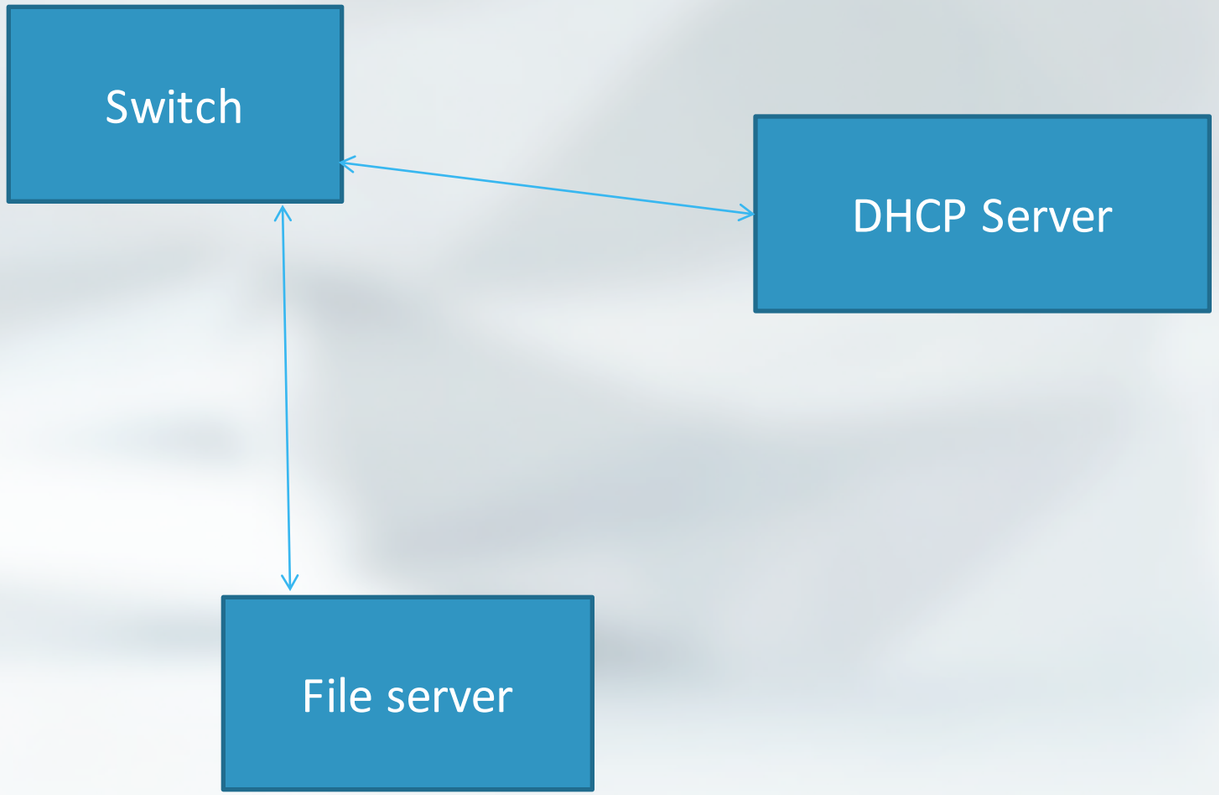


Increase Speed of Deployment & Provisioning

Reduce Configuration Errors

Reduce OPEX

Topology



Junos ZTP Topology – Components

Device:

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

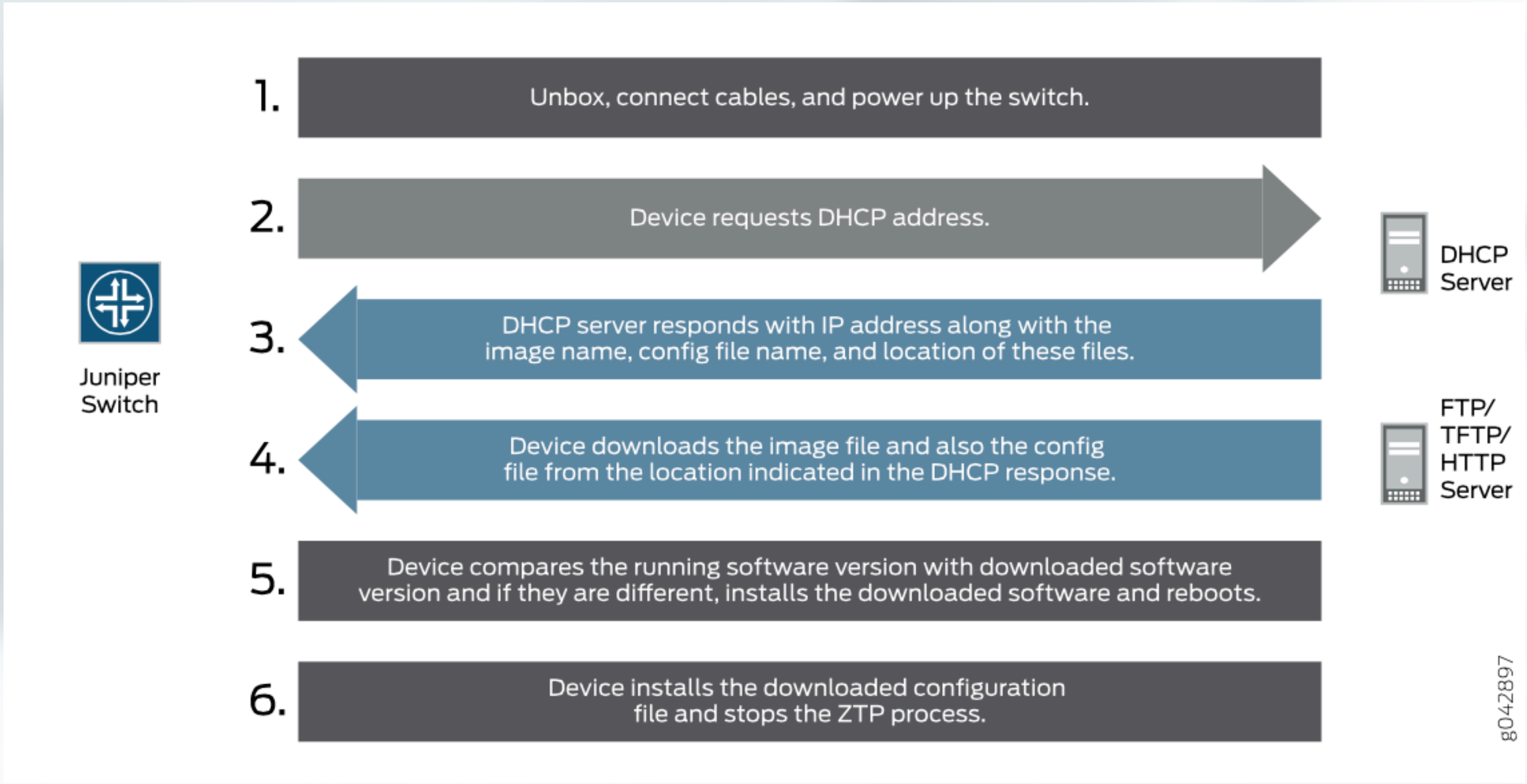
DHCP Server:

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

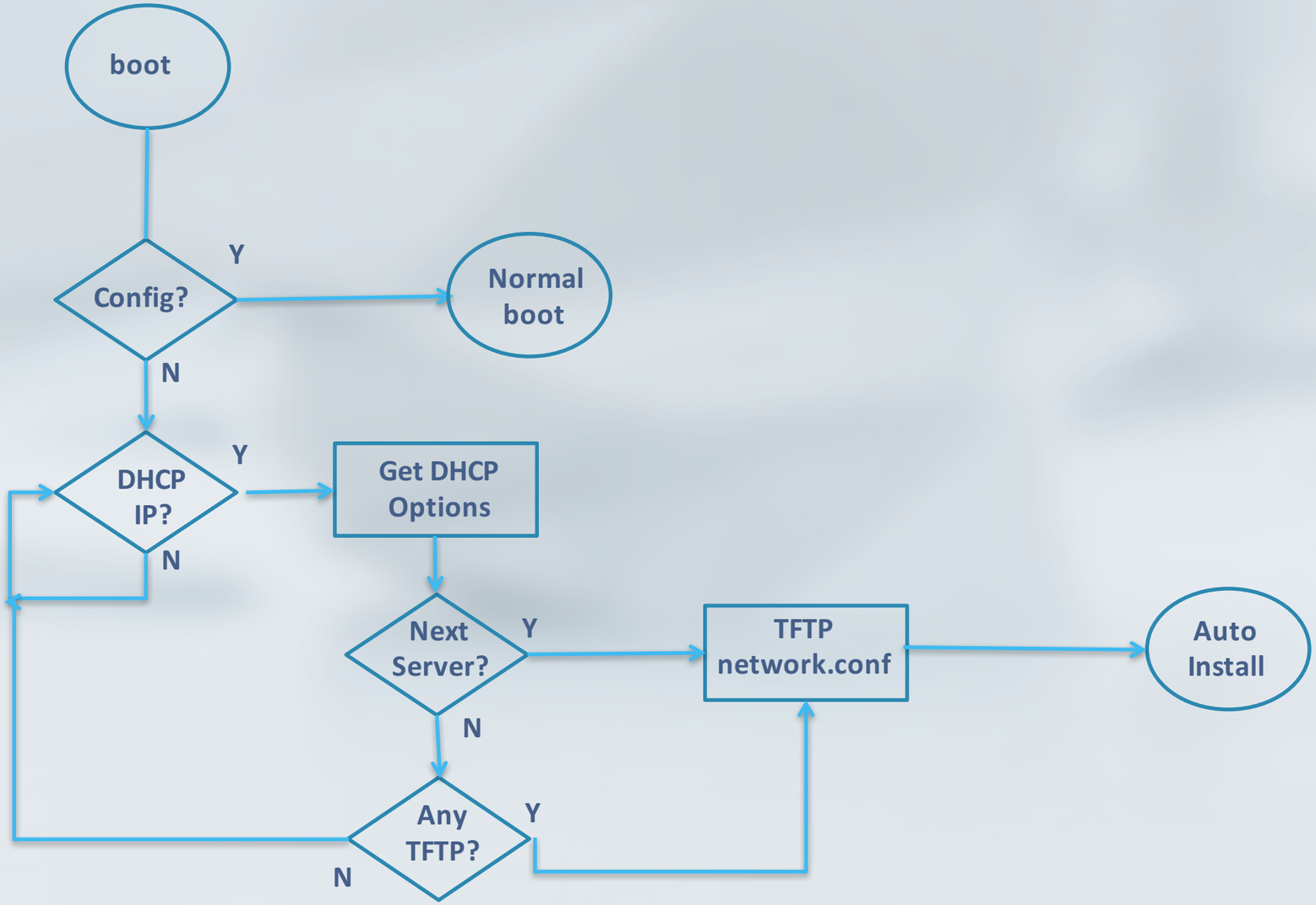
File Server:

- 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

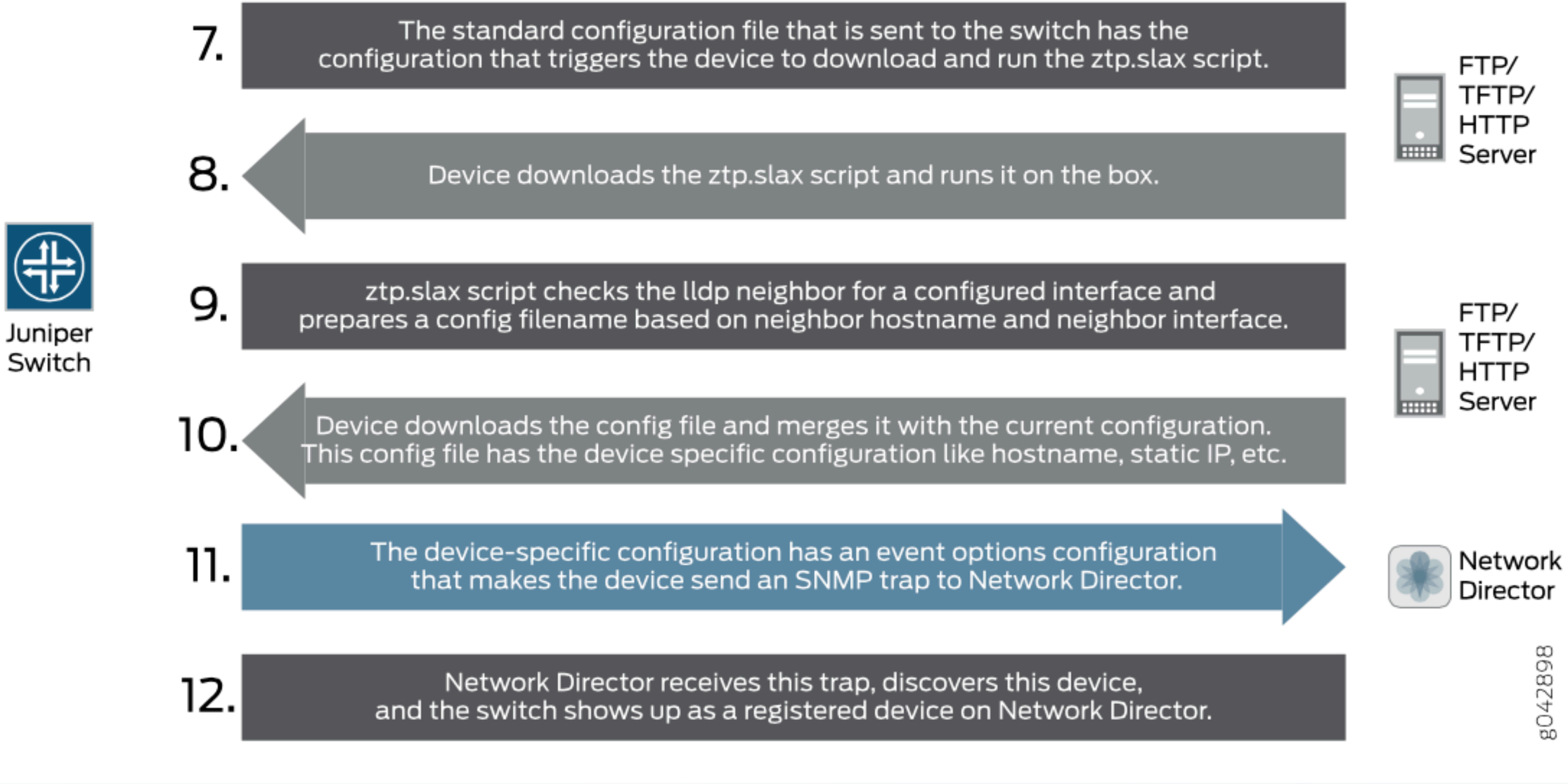
DNSMASQ DHCP/TFTP Server Configuration

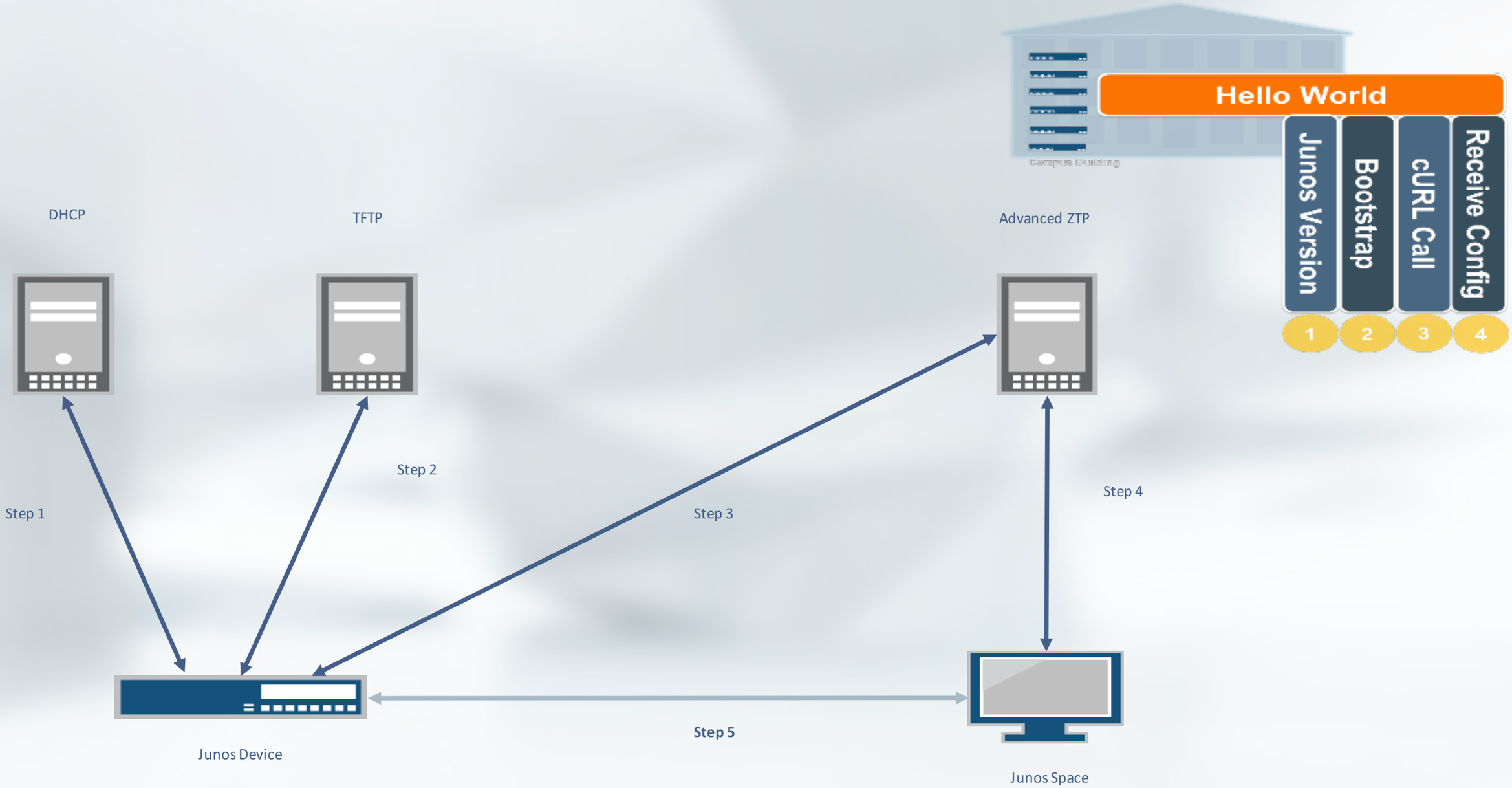
```
dhcp-leasefile=/var/lib/misc/dnsmasq.leases
log-facility=/var/lib/misc/dnsmasq.log
log-dhcp
port=0
enable-tftp
tftp-root=/tftpboot
dhcp-option=150,10.250.30.10
dhcp-vendorclass=set:ex2200-clients,Juniper-ex2200-24t # Set tags based on Vendor Class
dhcp-host=set:hostname,ma:ca:dd:re:ss # set tags 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 <>

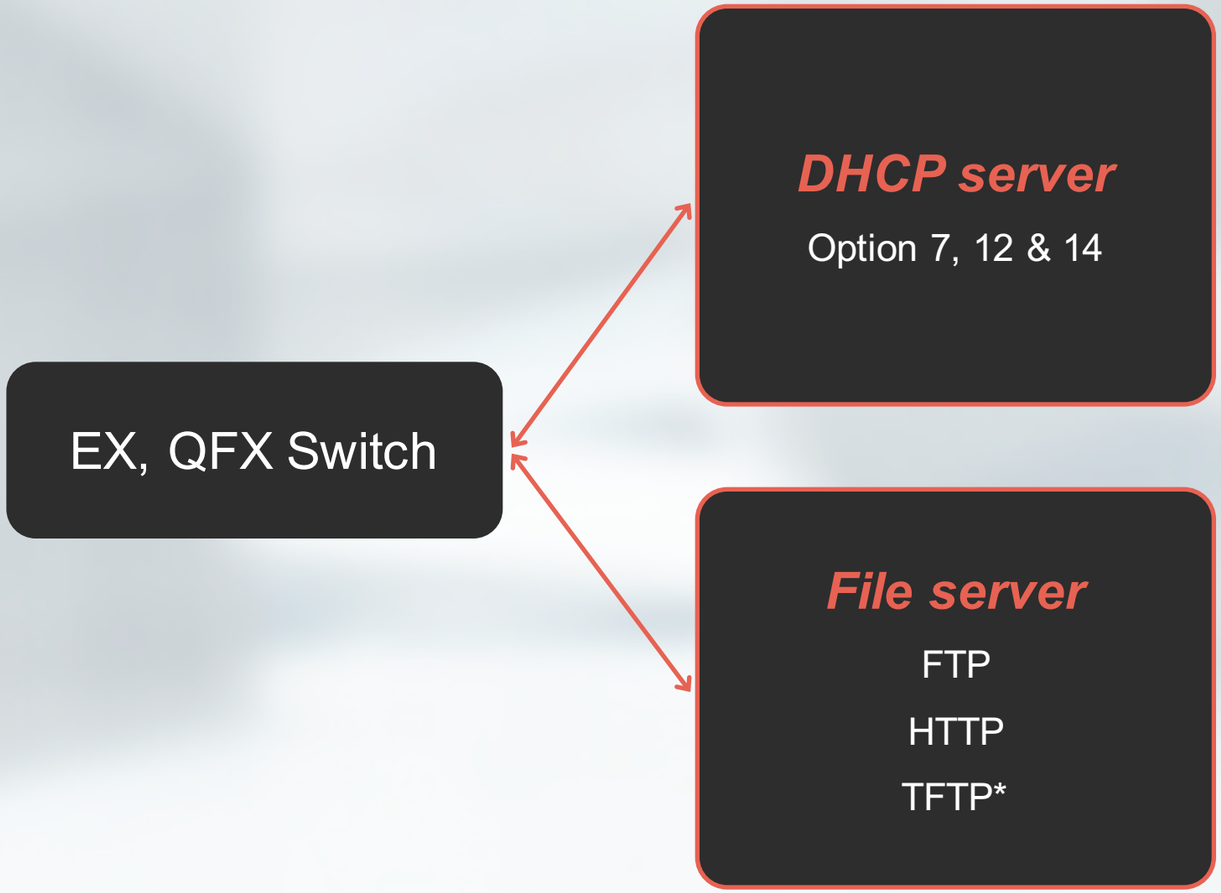
Further Troubleshooting

- 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

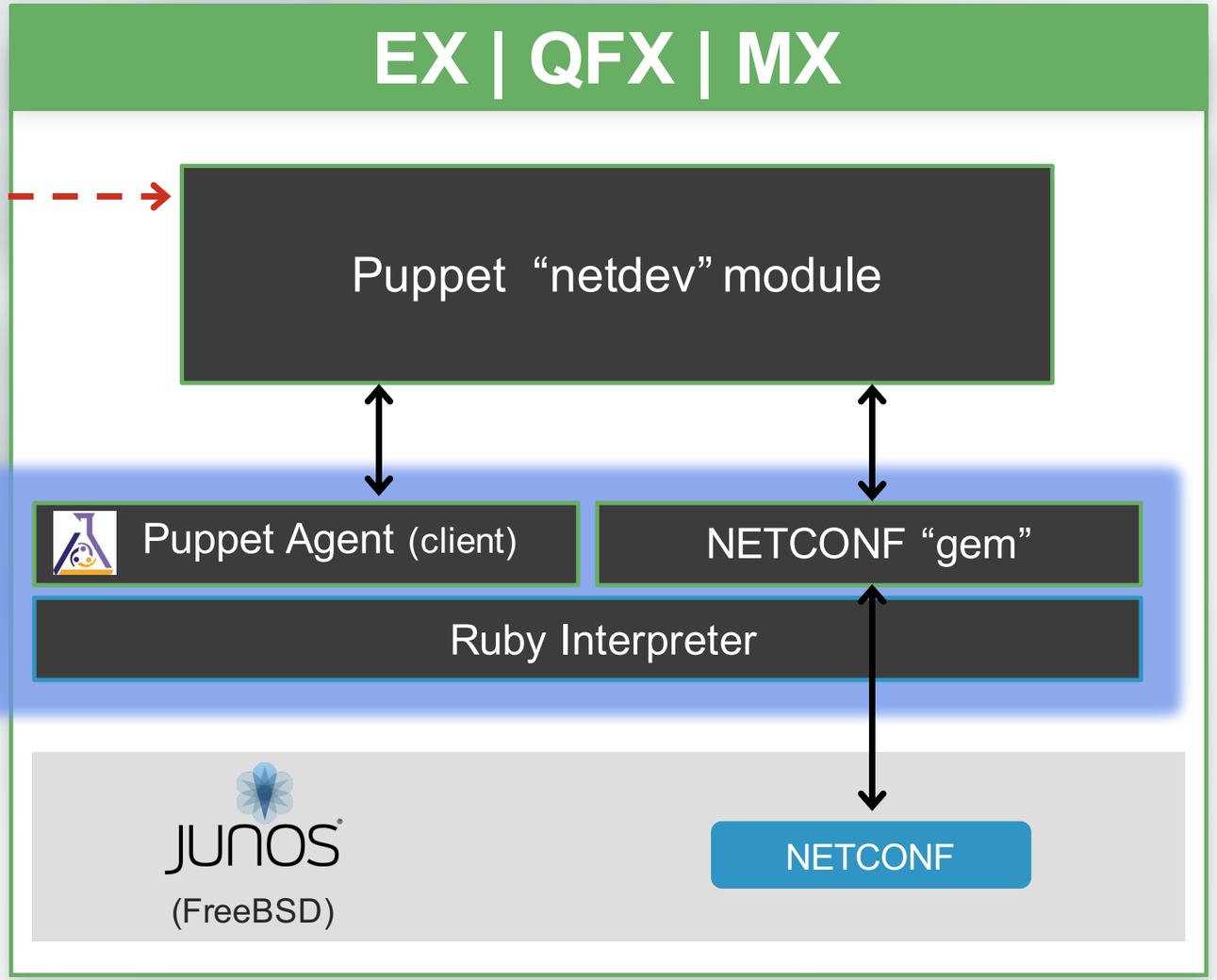
- How Puppet works
  - Puppet language is declarative (based on ruby), rather than procedural, meaning you tell Puppet what results you want, rather than how to get there
  - Its language is clear, simple and concise, easy for pretty much anyone to read and understand
  - This clarity facilitates easier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

Configure: Puppet

Puppet Master
(server)

"netdev"

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



JUNOS SDK

jpuppet package

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

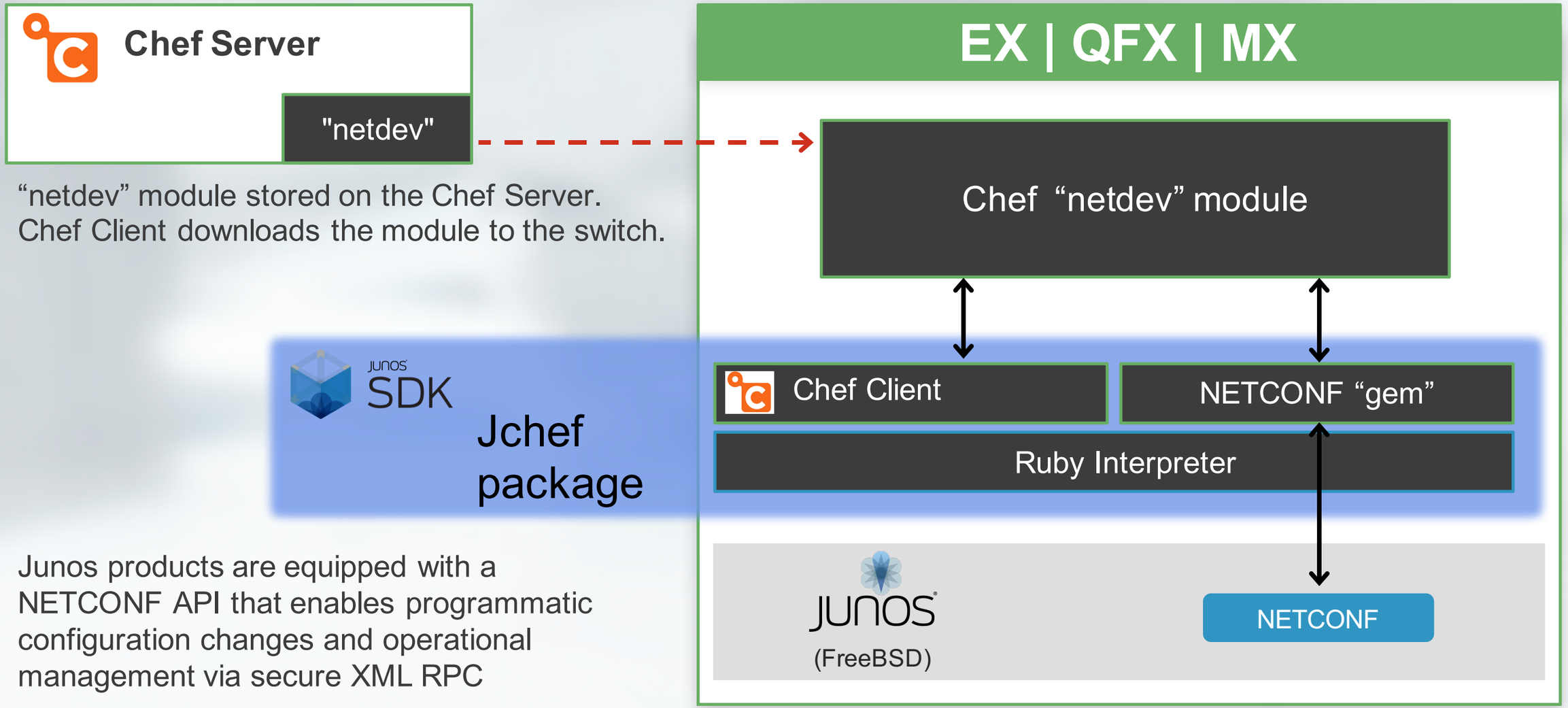
What is Chef?

- 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/

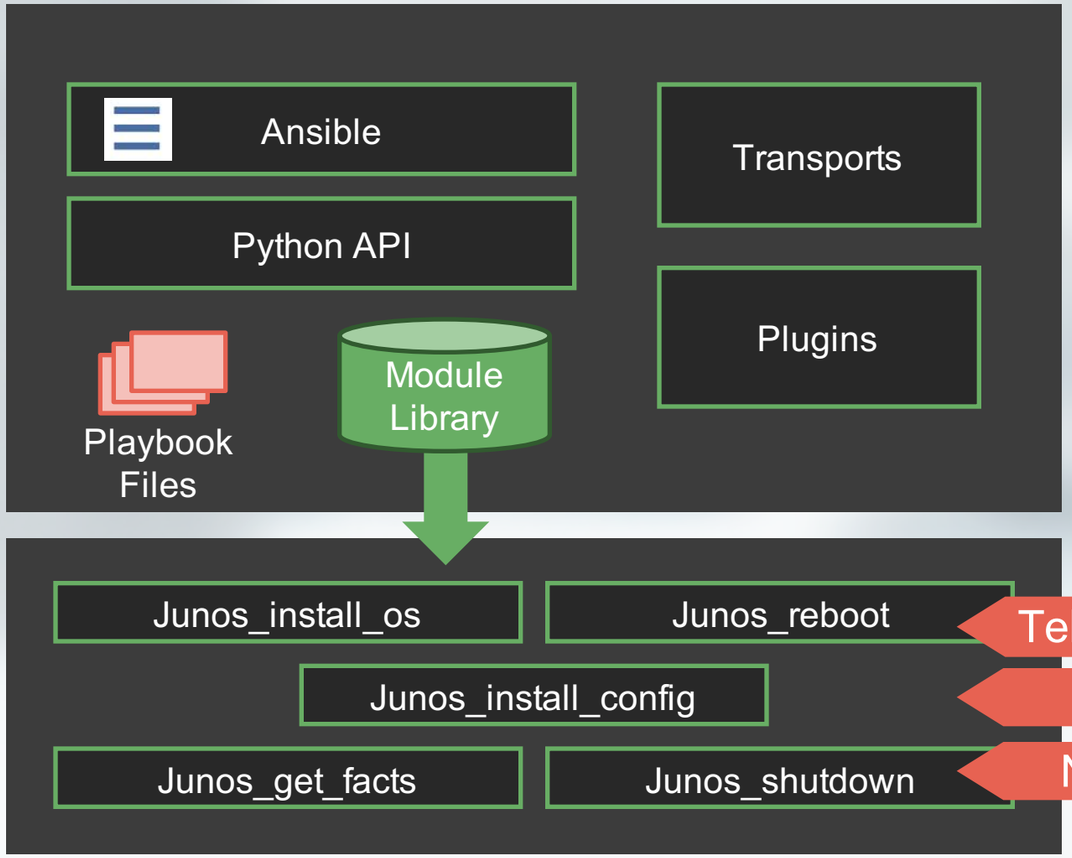
Compatible with:
All Junos Platforms

Ansible – Simple IT Orchestration

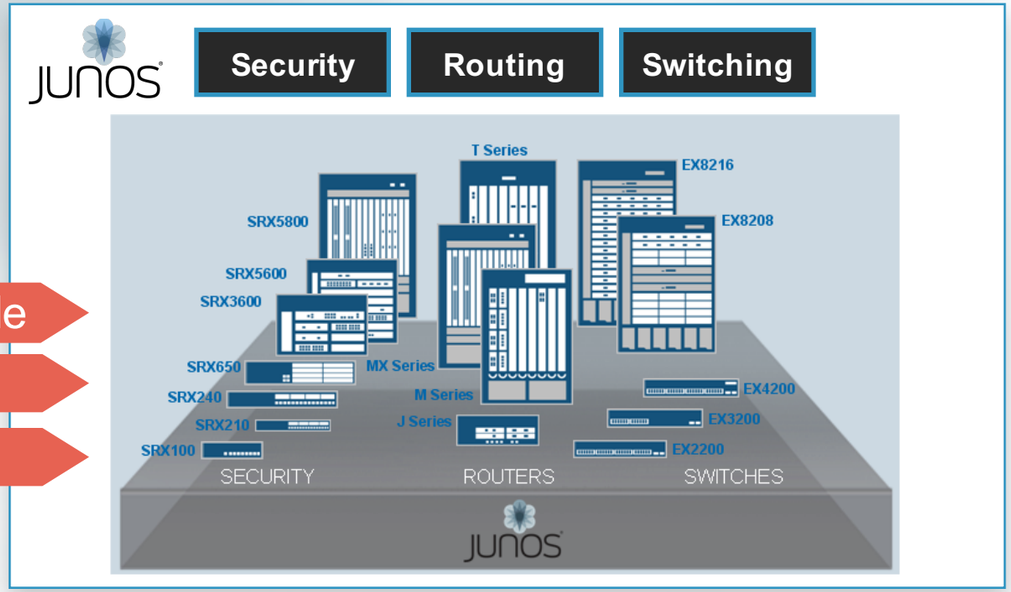
- Ansible is a simple IT automation engine that automates provisioning, configuration management, application deployment, and intra-service orchestration
- Designed for multi-tier deployments, Ansible models your IT infrastructure by describing how all of your systems inter-relate, rather than just managing 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;
    }
}
```

Yaml

What is yaml

YAML is a human readable mechanism for storing data that is easily mapped to common data types.

It supports hierarchical structures

Whitespace delimited (like Python) so files have consistent look.

Program language agnostic – supported by most.

Yaml - Example

```
---
ipsec_monitor_interval: 5
ipsec_monitor_threshold: 5
zones:
  untrust:
    name: untrust
    interfaces:
      - lo0.0
      - ge-0/0/1.0
      - ge-0/0/4.0
      - ge-0/0/5.0
```

YAML files always start with 3 hyphens

Yaml – data types

- YAML automatically detects data types. You can do programmy stuff with them, but we won't be covering that.

```
---
boolean: true
float: 105.7
integer: 33
string: This is a string
string2: "This is also a string"
```

Yaml - lists

- Lists can be in blocks using a hyphen+space to begin new items

```
---
- MX
- SRX
- QFX
```

- Or inline delimited by comma+space and enclosed in square brackets

```
---
[MX, SRX, QFX]
```

Yaml – Dictionary/Associate Array

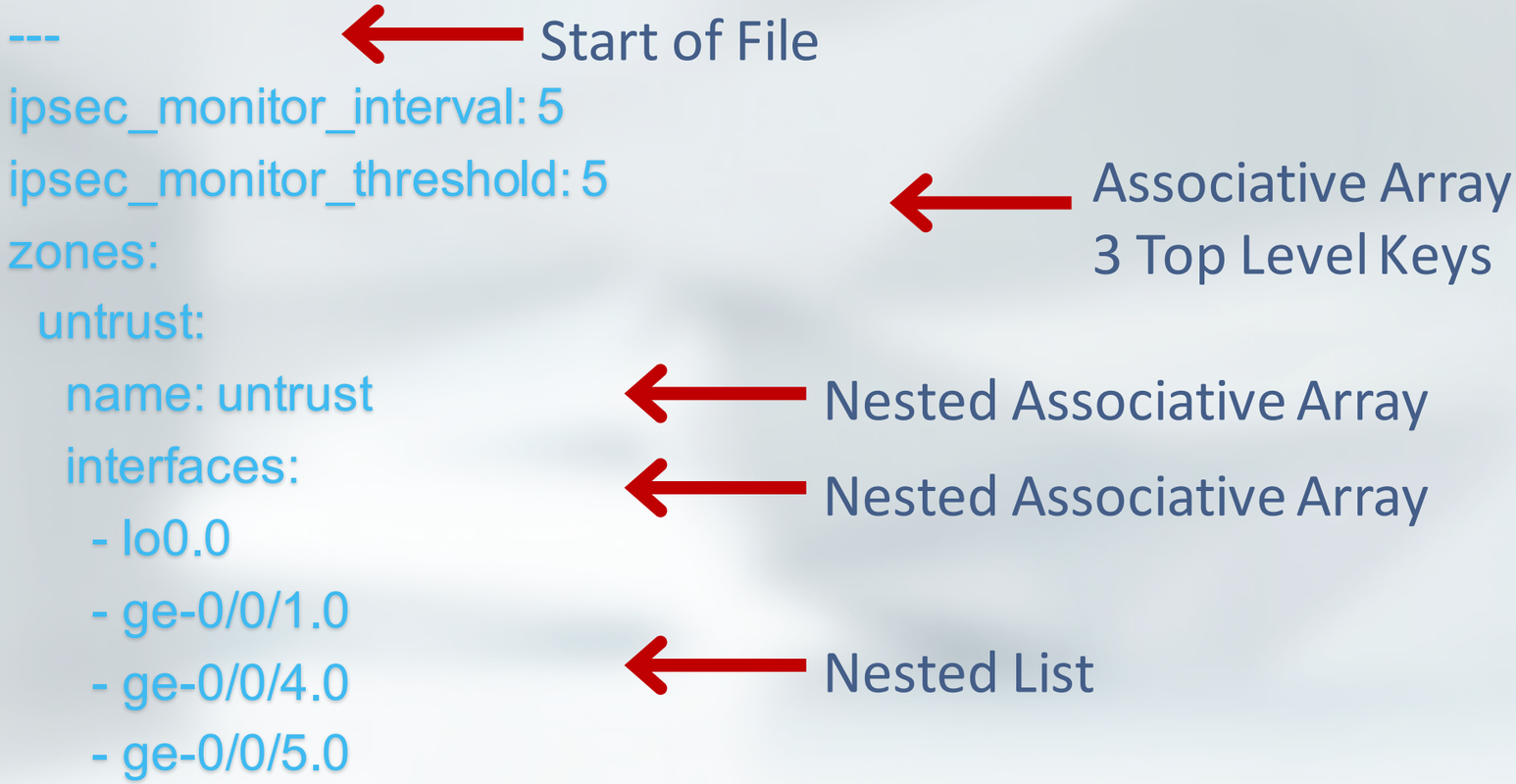
Associate Arrays can be in blocks with new lines and indention with keys and values separated by colon+space

```
---
    name: lo0
      logical_unit: 185
      inet_address: 10.1.1.185/32
```

Or inline delimited by comma+space with keys and values separated by colon+space enclosed in curly brackets

```
---
    {name: lo0, logical_unit: 185, inet_address: 10.1.1.185/32}
```

Yaml – Example Deconstructed



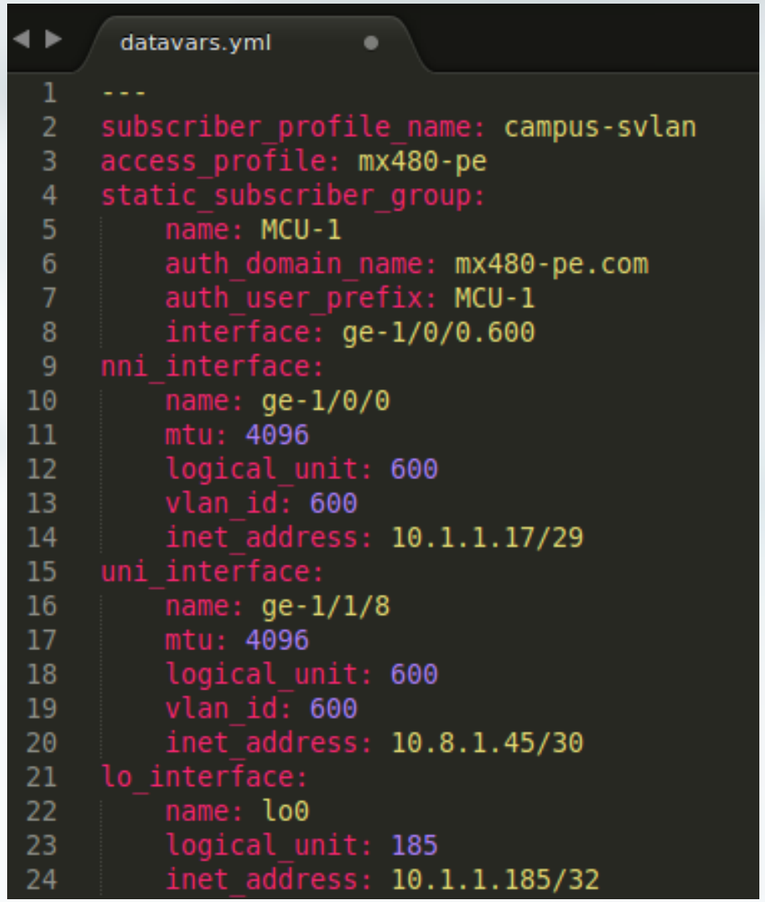
Yaml - gotchas

- Whitespace indented, but DOES NOT MIXED WHITESPACE!
  - Suggested format is to use (2) spaces for indents
    - Not hard requirement, but must be the same throughout a file

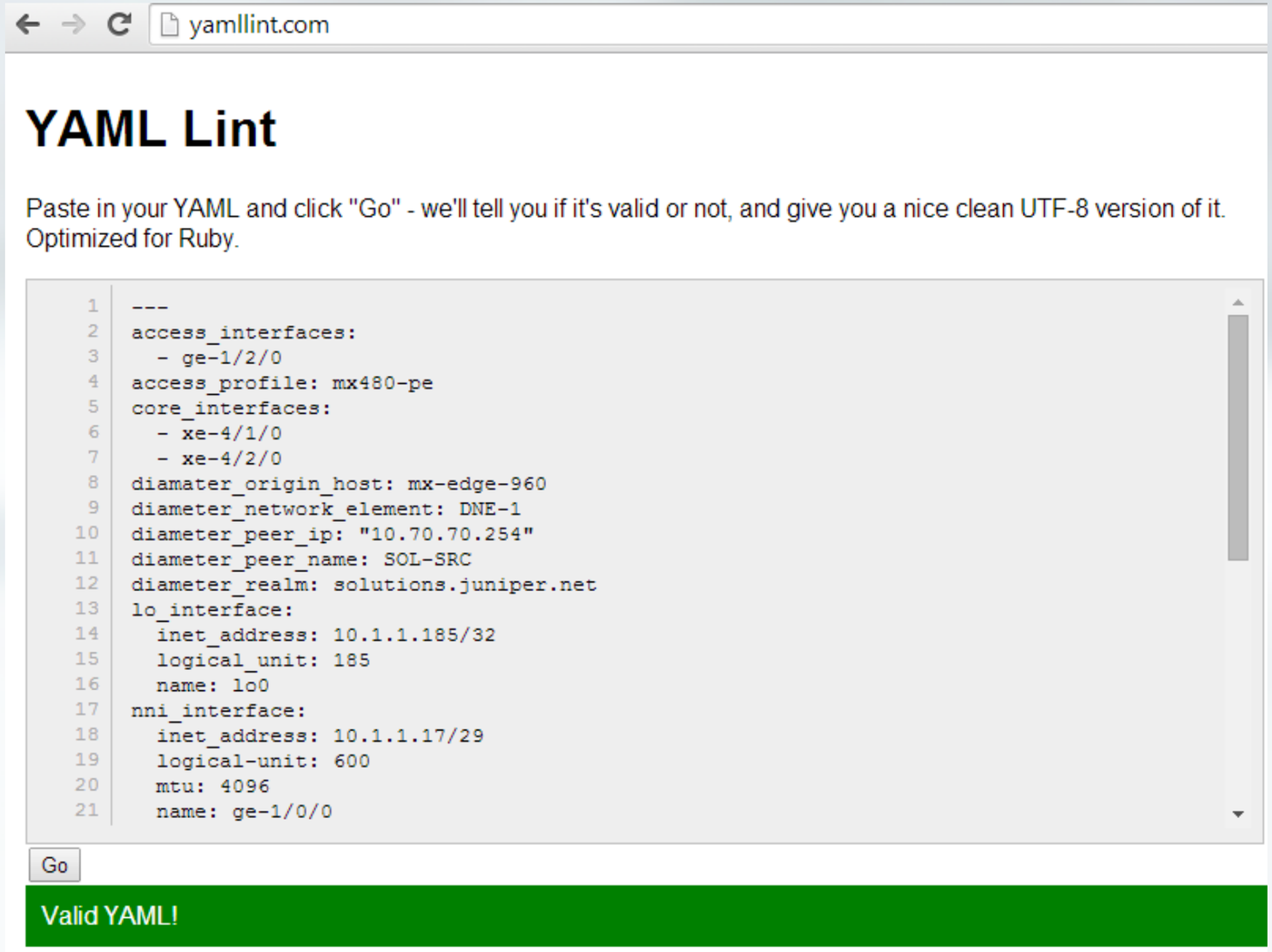
- Comments begin with a number sign # and separated by whitespace

Yaml - editor

- Using an editor with YAML support can make your life easier. Here is an example of Sublime 3



Yaml – Check validity (Linting)



NETCONF

Compatible with:
All Junos Platforms

NETCONF Explained

What is NETCONF?

- XML Device API implemented over SSH subchannel
- SSH –p 830 user@device –s netconf
- Communicate using XML RPC
- Configuration Commands & Operational Commands

Off-Box Workflow Automation

NETCONF – XML over SSH

- Secure and connection oriented … SSHv2 as transport
- Structured and transaction based … XML as RPC request / response
- User-class privilege aware … Native to Junos

Management System

NETCONF XML PROTOCOL (RFC4741)

Automate config changes, remote invocation of operational commands, collection of logs

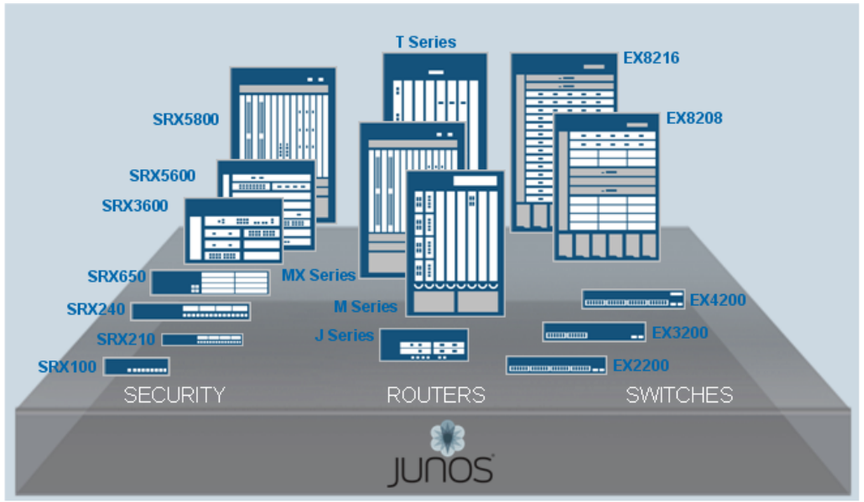
Secure TCP/IP connections via SSHv2 (RFC4742)

XML

NETCONF client libraries exist for a number of programming languages such as Java, Perl, Ruby, Python, and even SLAX !

JUNOS

Security | **Routing** | **Switching**



Junos Configuration

- The following illustrates the Junos configuration as an XML document for a VLAN named “Accounting”

```
[edit]
admin@EX# show vlans Accounting | display xml
```

CLI

```
[edit]
admin@EX-dc1# show vlans Accounting
description "Accounting Department";
vlan-id 500;
interface {
    ge-0/0/12.0;
    ge-0/0/13.0;
    ae0.0;
}
```

XML

```
<configuration>
   <vlans>
      <vlan>
         <name>Accounting</name>
         <description>Accounting Department</description>
         <vlan-id>500</vlan-id>
         <interface>
            <name>ge-0/0/12.0</name>
         </interface>
         <interface>
            <name>ge-0/0/13.0</name>
         </interface>
         <interface>
            <name>ae0.0</name>
         </interface>
      </vlan>
   </vlans>
</configuration>
```

Junos Operational Commands

- The following illustrates the "`show vlans`" operational command in XML:

```
admin@EX> show vlans | display xml rpc
```

```
admin@EX> show vlans | display xml
```

XML RPC COMMAND

```
<rpc>
  <get-vlan-information>
</rpc>
```

XML RESPONSE

```
<rpc-reply>
  <vlan-information …>
    <vlan-terse/>
    <vlan>
        <vlan-instance>0</vlan-instance>
        <vlan-name>VLAN-blue</vlan-name>
        <vlan-create-time>Wed Nov 16 12:42:03 2011
        </vlan-create-time>
        <vlan-status>Enabled</vlan-status>
        <vlan-owner>static</vlan-owner>
        <vlan-tag>500</vlan-tag>
        <vlan-tag-string>500</vlan-tag-string>
        <vlan-index>2</vlan-index>
        <vlan-protocol-port>Port Mode</vlan-protocol-port>
        <vlan-members-count>1</vlan-members-count>
        <vlan-members-upcount>0</vlan-members-upcount>
        <vlan-detail>
          <vlan-member-list>
            <vlan-member>
              <vlan-member-interface>ge-0/0/19.0</vlan-member-interface>
            </vlan-member>
          </vlan-member-list>
        </vlan-detail>
    </vlan>
    …
</rpc-reply>
```

Demo

Documentation:
http://www.juniper.net/documentation/en_US/junos13.2/information-products/pathway-pages/netconf-guide/netconf.html

SLAX & JUISE

LIBSLAX

- Junos OnBox scripting language
- Based on XSLT
- Exported SLAX from Junos
  - Standalone language
  - Open-Source (githib.com/juniper/libslax)
- Imported back into JUNOS-12.2, 12.3
  - Re-imported on each release

SLAX Scripts

Operation Script

- Create Custom Commands
- Diagnose Network Problems
- Controlled Configuration Change

Event Script

- Automate Event Responses
- Correlate Events

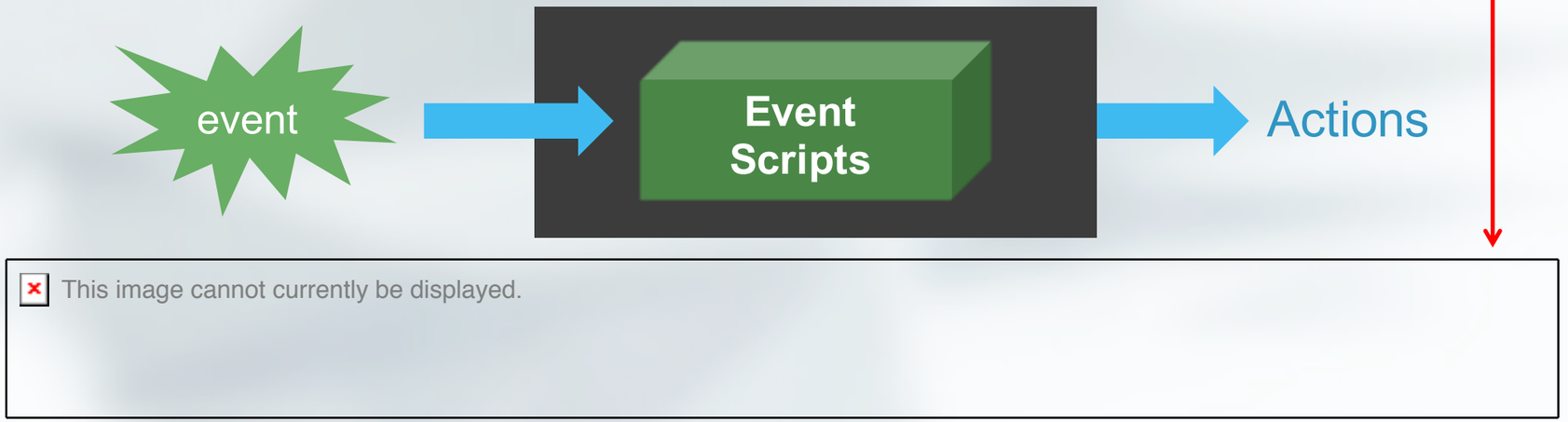
Commit Configuration Script

- Assure compliance to business rules network/security policies

Event Automation

Event scripts are triggered by events on the device

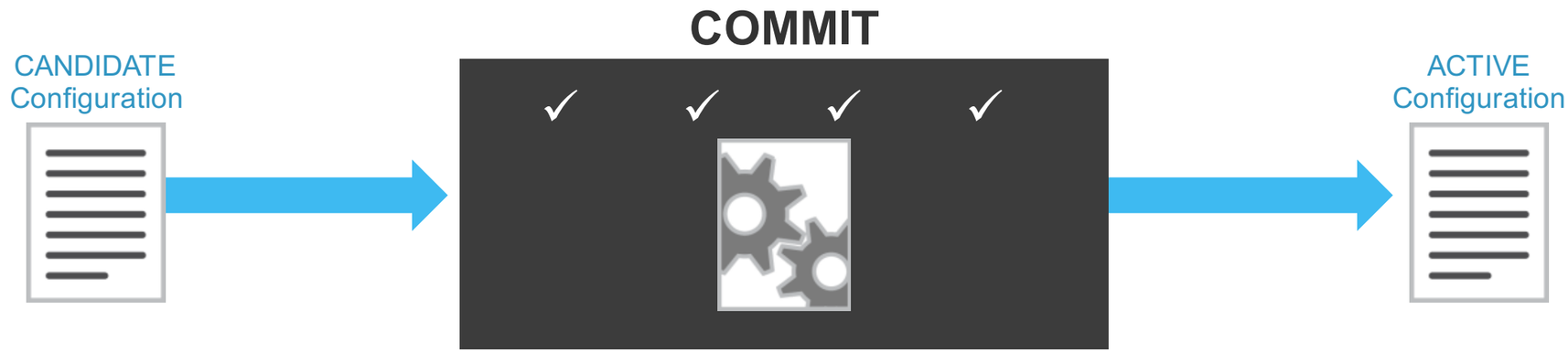
- Gather relevant troubleshooting info and correlate events from leading indicators
- Automate event responses with a set of actions

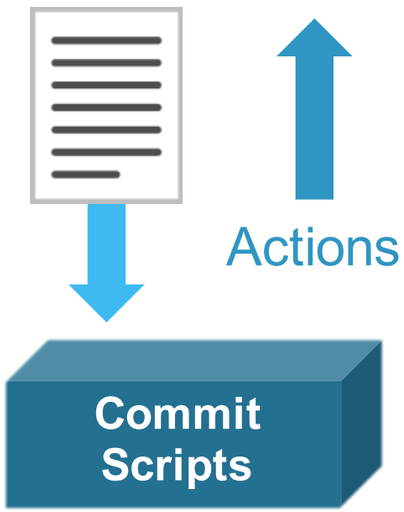


Benefits

- ✓ Automate time-of-day configuration changes
- ✓ Speed time-to-resolve to reduce the downtime
- ✓ Automate response to leading indicators to minimize impact

Configuration Automation



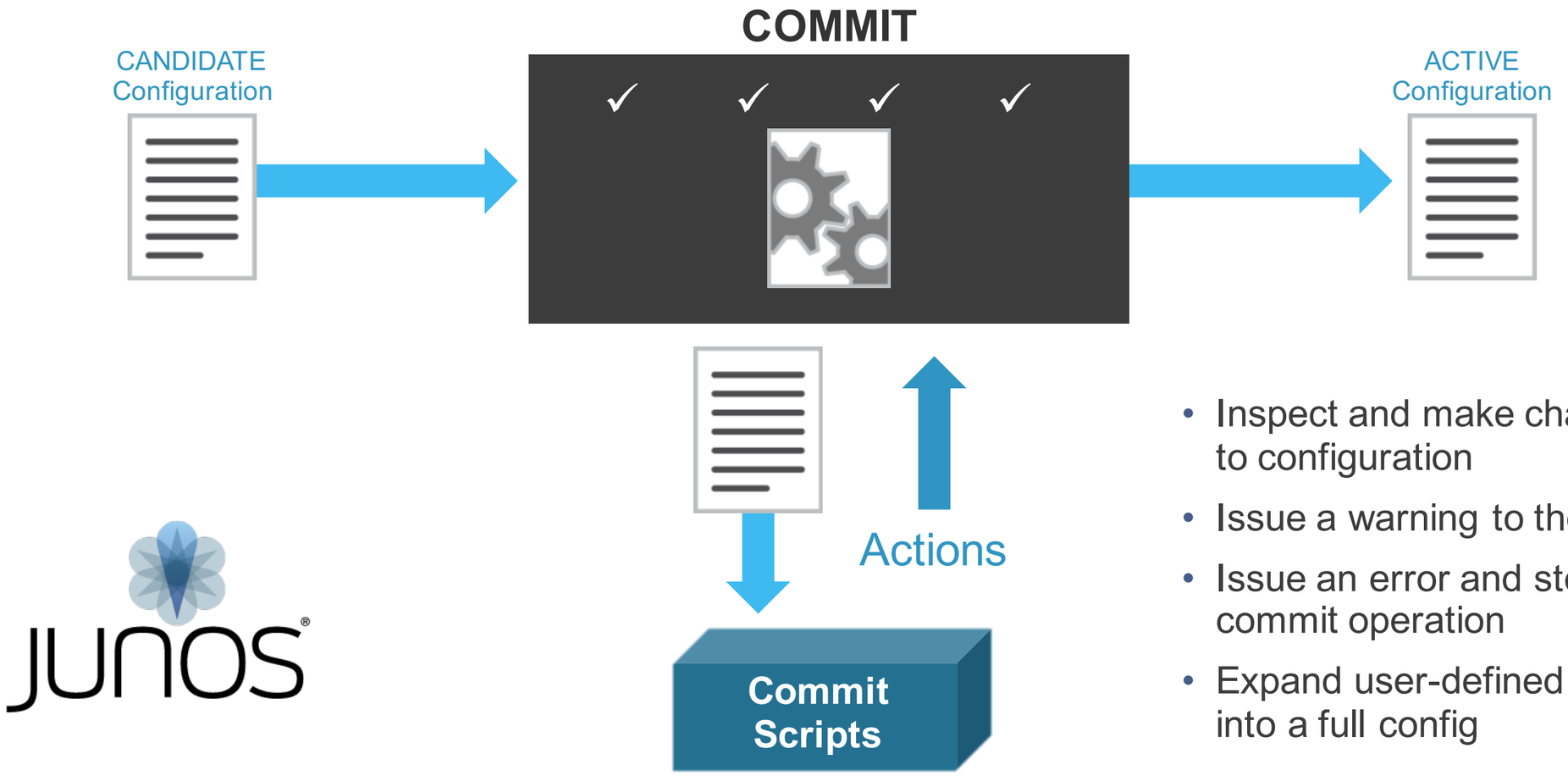


- Abstract complex configuration into a simple set of base commands
- Enforce best practices and business rules

Benefits

- ✓ Assure compliance to business rules and policies
- ✓ Provide change management to avert and correct errors
- ✓ Simplify and speed setup of complex configuration

Configuration Automation



- Inspect and make changes to configuration
- Issue a warning to the user
- Issue an error and stop the commit operation
- Expand user-defined macros into a full config

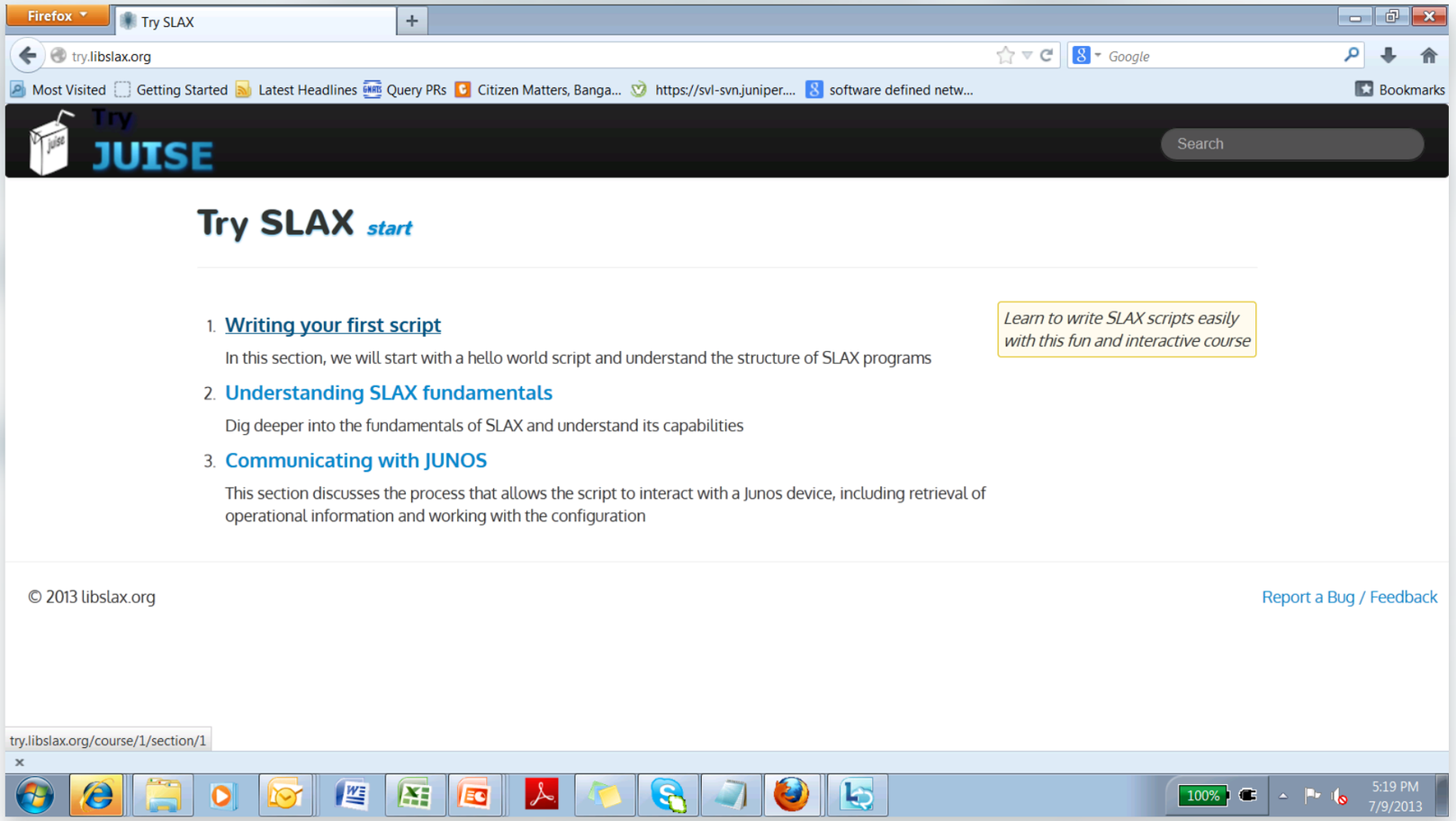
JUISE

- JUNOS UI Scripting Environment
- Open-Source (githib.com/juniper/juise)
- The JUISE project allows scripts to be written, debugged, and executed outside the JUNOS environment
- Tools for developers are available, including a debugger, a profiler, a call-flow tracing mechanism, and trace files

```
% juise @my-router my-script name1 val1 name2 val2
```

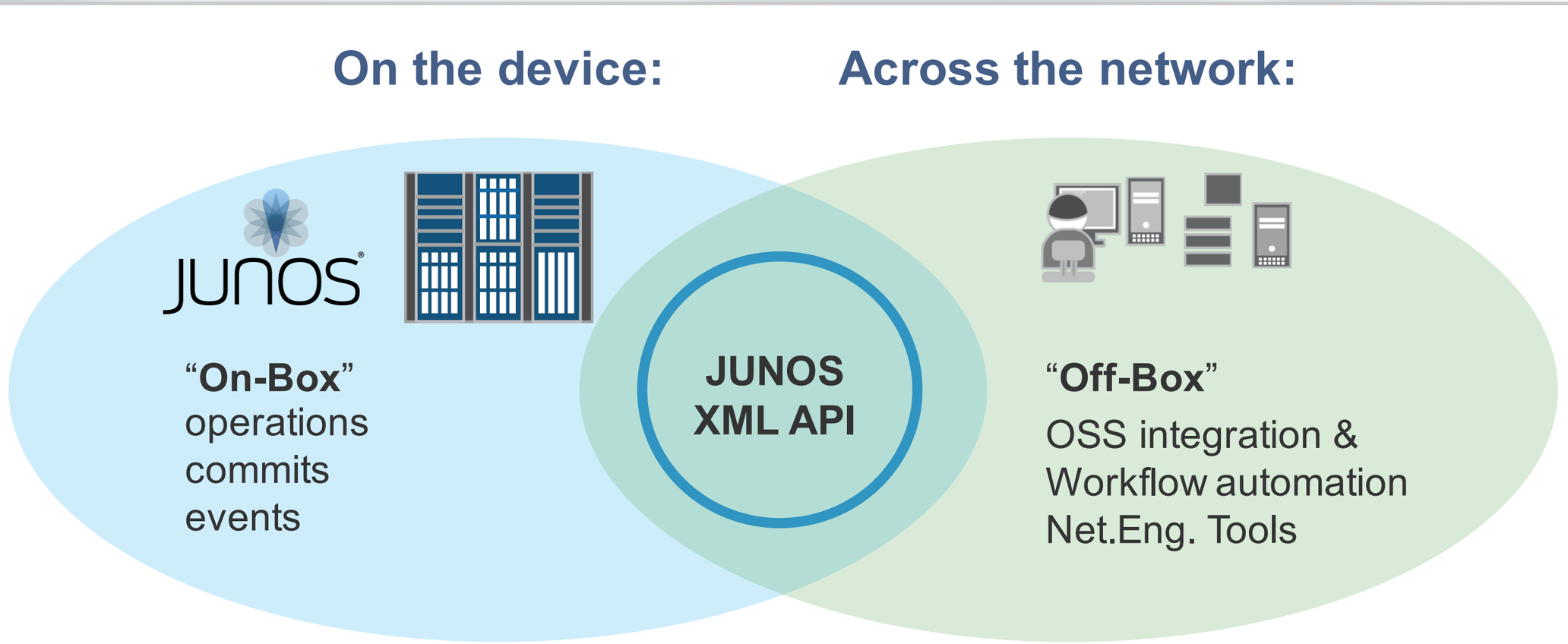
Try SLAX

- http://try.libslax.org
- Interactively learn and try SLAX programming language from web browser



Junos Workflow Automation

Bootstrap your automations with On-Box scripting, complete them with Off-Box



Code Libraries

Programming Language Support

- NETCONF
  - PERL
  - Java
  - Ruby
  - Python
- SLAX
  - JUISE (JUNOS UI Scripting Environment)
- EZ API
  - Python
  - Ruby

Programming Language Support

- Perl
  - Available from Juniper Support Software download:
  - http://www.juniper.net/support/downloads/?p=netconf#sw
- Ruby
  - Available from RubyGems.org, maintained by Juniper Networks
  - gem install netconf
- Python
  - Available from open source project, not affiliated with Juniper Networks
  - https://github.com/vbajpai/ncclient
- Java
  - Available from Juniper Support Software download:
  - http://www.juniper.net/support/downloads/?p=netconf#sw
- SLAX
  - Available from Google code, maintained by Juniper Networks
  - http://github.com/Juniper/libslax/
  - http://github.com/Juniper/juise/

Also all found on http://github.com/Juniper

Introducing "JUNOS PyEZ"

Open and Extensible "micro-framework"

- Remote device management and "fact" gathering
- Troubleshooting, Audit and Reporting
  - Operational data
  - Configuration data
- Configuration Management
  - Unstructured config snippets and templates
  - Structured abstractions
- Generalized utilities for file-system, software-upgrade, secure file copy (scp), etc.
- RPC Meta-Programming

Pyez – a layered approach

Python Shell

interactive

Python script

simple → complex

IT Frameworks

Custom Applications

- Native Python data types (hash/list)
- Junos specific not required
- XML not required

junos-pyez

open-source, Juniper

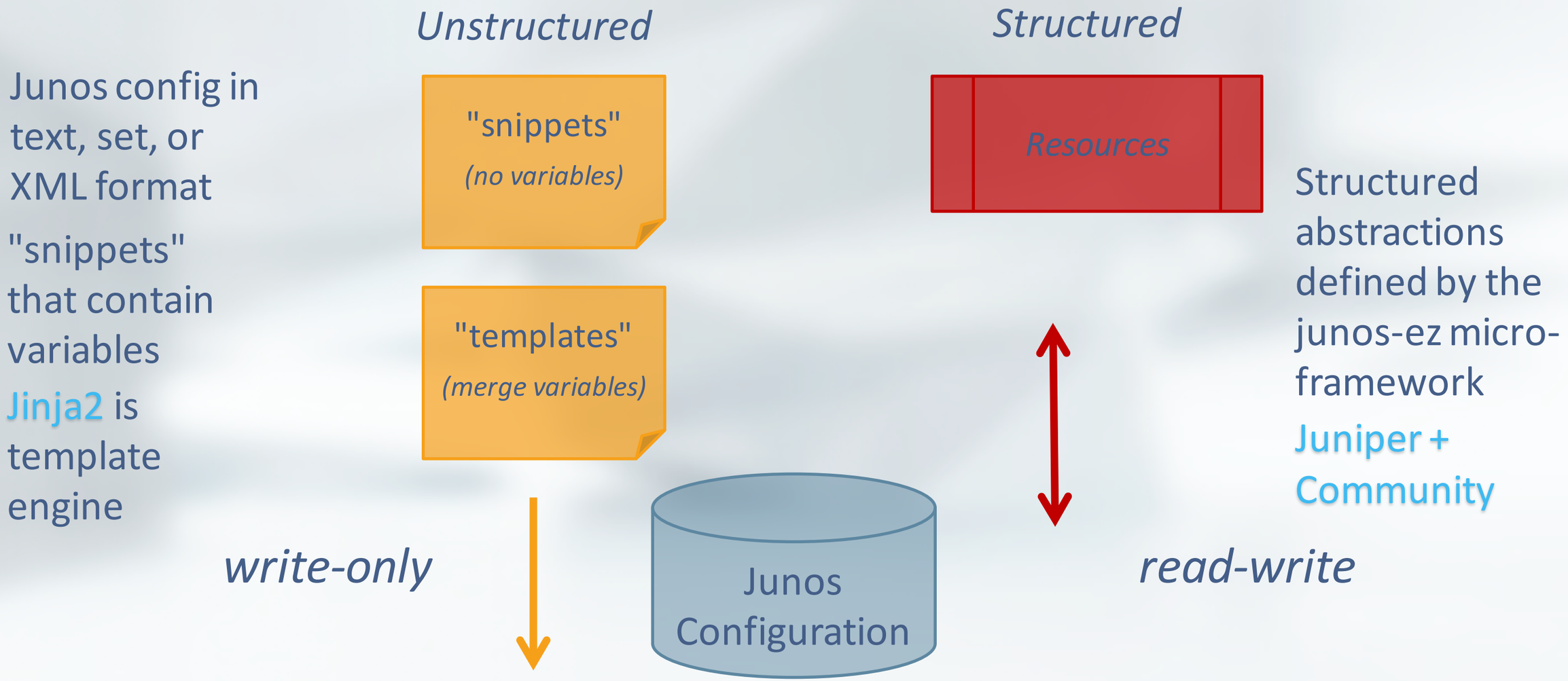
- Junos specific
- Abstraction Layer
- micro-framework

ncclient

open-source, Community

- NETCONF transport only
- Vendor Agnostic
- No abstractions

Pyez – configuration changes



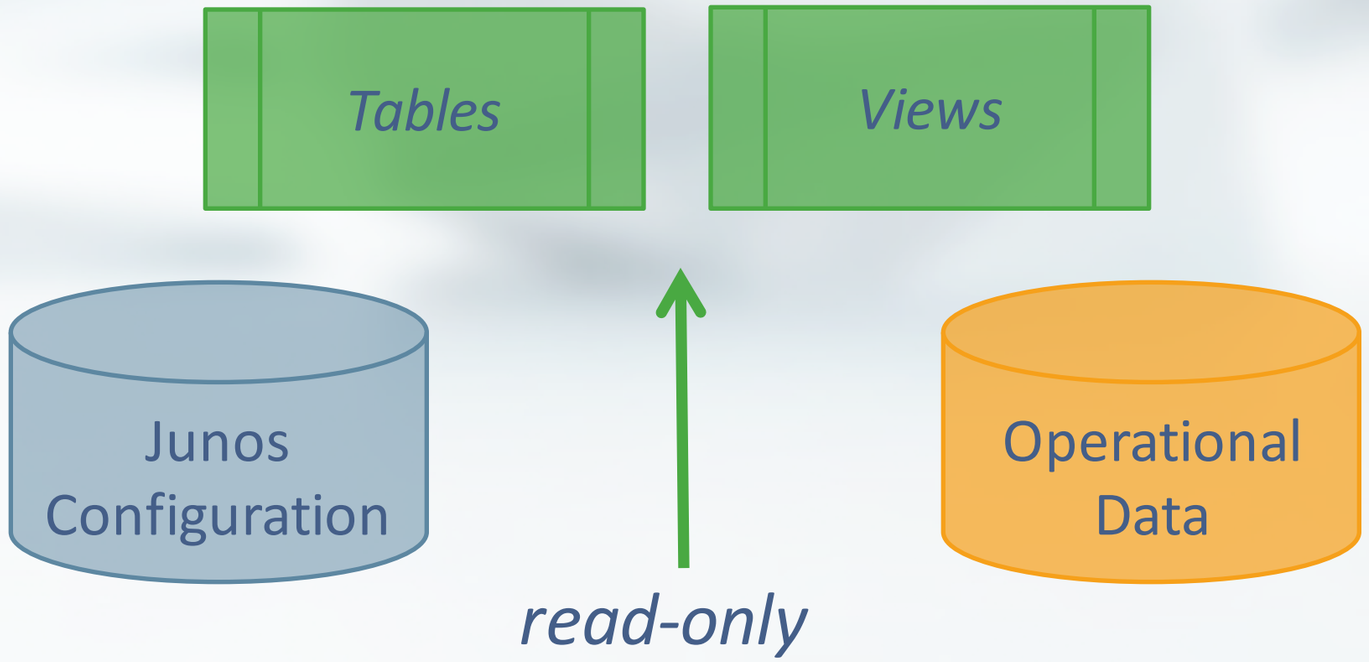
Pyez – troubleshooting, audit, reporting

Easily retrieve data and extract as native Python

Conceptually like database tables and views that define the fields of data you want

Structured abstractions defined by the Junos PyEZ micro-framework

No coding required to create abstractions



Pyez – “Hello world”

```
from jnpr.junos import Device
dev = Device(host='srx', user='eddie', passwd='password123')
dev.open()
Device(srx)
dev.facts
{'domain': None, 'hostname': 'firefly', 'ifd_style': 'CLASSIC', 'version_info':
junos.version_info(major=(12, 1), type=X, minor=(46, 'D', 15), build=3), '2RE': False,
'serialnumber': 'aaf5fe5f9b88', 'fqdn': 'firefly', 'virtual': True, 'switch_style': 'NONE', 'version':
'12.1X46-D15.3', 'HOME': '/cf/var/home/rick', 'srx_cluster': False, 'model': 'FIREFLY-PERIMETER',
'RE0': {'status': 'Testing', 'last_reboot_reason': 'Router rebooted after a normal shutdown.', 'model':
'FIREFLY-PERIMETER RE', 'up_time': '8 minutes, 51 seconds'}, 'personality': 'SRX_BRANCH'}
dev.close()
```

Pyez – Connecting To devices

- PyEZ supports connecting to Junos devices using standard user/pass, SSH keys, Agents and Forwarding.

"I am Vinz, Vinz Clortho, Keymaster of Gozer...Volguus Zildrohoar, Lord of the Seboullia. Are you the Gatekeeper?"

Pyez – Connect examples

Username and Password

```
dev = Device(host='srx', user='eddie', passwd='password123')
```

Host, user, and passwd

Connecting to a device via the SSH Agent or Actively loaded environment key

```
dev = Device('srx')
```

No user, key, or password
host keyword can also be omitted

Connecting to a device via the default SSH key + Key password

```
dev = Device(host='srx', passwd='juniper')
```

Host and passwd only

Pyez – Connect examples cont.

Keyfile + password

```
dev = Device(host='srx', ssh_private_key_file='/keys/pkey',
 passwd='juniper')
```

Host, key file and password

Keyfile w/out password

```
dev = Device(host='srx',
    ssh_private_key_file='/keys/nopass‘)
```

Host and key file only

Pyez – disable facts

- The automatic gathering of facts can be disabled. This is useful in situations where bugs or missing device types prevent facts from gathering.

```
dev = Device(host='srx', gather_facts=False)
dev.open()
Device(srx)
dev.facts
{}
```

Pyez – check connection (Probe)

- To check if NETCONF can be reached the device can be probed. This is particularly useful in re-connecting after a reboot.

Default timeout of 5 seconds

- dev.probe()
- False
- dev.probe(timeout=30)
- False

Specify timeout

- dev = Device(host='srx', auto_probe=5)
- dev.open()
- jnpr.junos.exception.ProbeError: ProbeError(srx)

Probe timeout before opening

Pyez – Config utilities

- Included in the framework are various utilities. We will specifically cover some of the more common Configuration functions.
  - Load
  - Pdiff
  - Commit_check
  - Commit
  - Rollback

Pyez – load

- Loads changes into the candidate configuration.
- Changes can be in the form of strings (text, set, xml), XML objects, and files.
- Files can be either static snippets of configuration or Jinja2 templates.

load() does **NOT** commit the change, it only loads it.
To commit changes use **commit()**

Set is not supported in JUNOS < 11.4

Pyez – config set

```
from jnpr.junos import Device
from jnpr.junos.utils.config import Config
dev = Device('srx')
dev.open()
set_commands = """
set system host-name myRouter
set system domain-name shermdog.com
"""
cu = Config(dev)
cu.load(set_commands, format='set')
<Element load-configuration-results at 0xb5f2048c>
```

Import Config util

Set commands in multi-line string

Create Config object

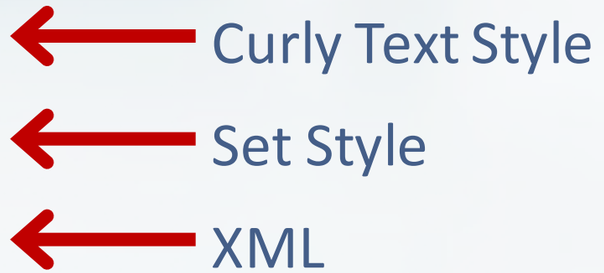
Load configuration onto device

Pyez – config from file

- Configuration data can be loaded from a file on the local system.
- The file extension can be used to determine the format, or it can be specified.

```
cu.load(path='config-example.conf')
cu.load(path='config-example.set')
cu.load(path='config-example.xml')
```


```
cu.load(path='config-example', format='text')
cu.load(path='config-example', format='set')
cu.load(path='config-example', format='xml')
```



Pyez – config from template

- The framework can render templates and load them.
- Variables can be stored in Python dictionary or loaded from a file.

```
tvars = dict(host_name='rick',
    domain_name='shermdog.com')
```

Data for variables

```
cu.load(
    template_path="config-example-template.conf",
    template_vars=tvars)
```

Path to template and vars

Pyez – pdiff / diff

- After a configuration has been loaded the diff (patch-format) between the current candidate and the provided rollback can be retrieved.

- pdiff automatically calls *print* for debugging

cu.pdiff()

By default compare rollback 0

```
[edit system]
-  host-name firefly;
+  host-name rick;
+  domain-name shermdog.com;
```

Pyez – pdiff / diff cont.

- cu.pdiff(3) ← A rollback id can be used to compare different rollbacks

```
[edit system]
-  host-name firefly;
+   host-name rick;
+   domain-name shermdog.com;
+   time-zone America/Chicago;
```

- The diff can also be stored into an object using diff()

```
diff = cu.diff()
```

Pyez – rollback

- The candidate config can be rolled back to either the last active config or a specific rollback number.

```
cu.rollback()
  True
cu.pdiff()
  None
cu.rollback(1)
  True
cu.pdiff()
[edit system]
-  time-zone America/Chicago;
```

By default, roll back to last active - 0

This shows the config *is* rolled back

A rollback id can also be used

Pyez – commit_check & commit

- Configuration changes can be checked prior to commit

```
cu.commit_check()
  True
```

If the commit-check results in warnings, they are not reported (at this time). A **jnpr.junos.exception.CommitError** is thrown

```
cu.commit ()
  True
cu.commit(comment="Doc",
    confirm=2)
True
```

← Comments and confirm timeout can be set

Demo

Documentation:
http://www.juniper.net/documentation/en_US/junos13.2/information-products/pathway-pages/netconf-guide/netconf.html

http://github.com/Juniper

Migration Automation

Tier 1 ISP

Customers Problem

- Manual migration 26000+ services from existing M320 routers to new MX480/960 routers tedious, cumbersome and error-prone

Interface with Space

- Customer wants to leverage existing Space (Network Management) Platform to perform migrations/configuration changes. A Space based application (Port Migration Tool) is developed to mitigate the problem

Customer Problem

What part of the configuration is needed to only migrate ge-0/0/2?

M Series Switch

```
ge-0/0/1
interfaces {
    ge-0/0/1 {
        unit 10 {
            bandwidth 500m;
            vlan-id 10;
            family inet {
                address 119.225.62.148/30;
            }
        }
    }
}
routing-options {
    static {
        route 119.225.143.32/29 {
        next-hop 119.225.62.149;
        community no-export;
    }
}
```

```
ge-0/0/2
interfaces {
    ge-0/0/2 {
        unit 30 {
            accounting-profile INET-IFU;
            family inet {
            filter {
                input INET-IN-Block-Martians-and-No-Monitor;
                output INET-OUT-No-Monitor;
            }
            policer {
                output INET-Policing-BW-8m;
            }
            address 119.225.142.69/30;
            }
        }
    }
}
accounting-options {
    file USAGE-ACCOUNTING {
        ...
    }
    interface-profile INET-IFU {
        file USAGE-ACCOUNTING;
    }
}
firewall {
    family inet {
        filter INET-IN-Block-Martians-and-No-Monitor {
            ...
        }
    policer INET-Policing-BW-8m {
        ...
    }
}
```

Project Challenges

Design

- Limited Core Routing expertise
- Restricted access internal service provisioning toolset
- Limited understanding of design approach
- Lengthy design phase (almost 6-8 weeks)

Development

- Weak Space SDK and API Documentation
- Some Libraries (XML-CurlyBrace format conversion) not available in 13.1 SDK
- Attempted to Port 13.2 Libraries (XML-CurlyBrace format conversion) to 13.1
- Finally Juniper PS helped develop an custom external format conversion tool to solve the problem

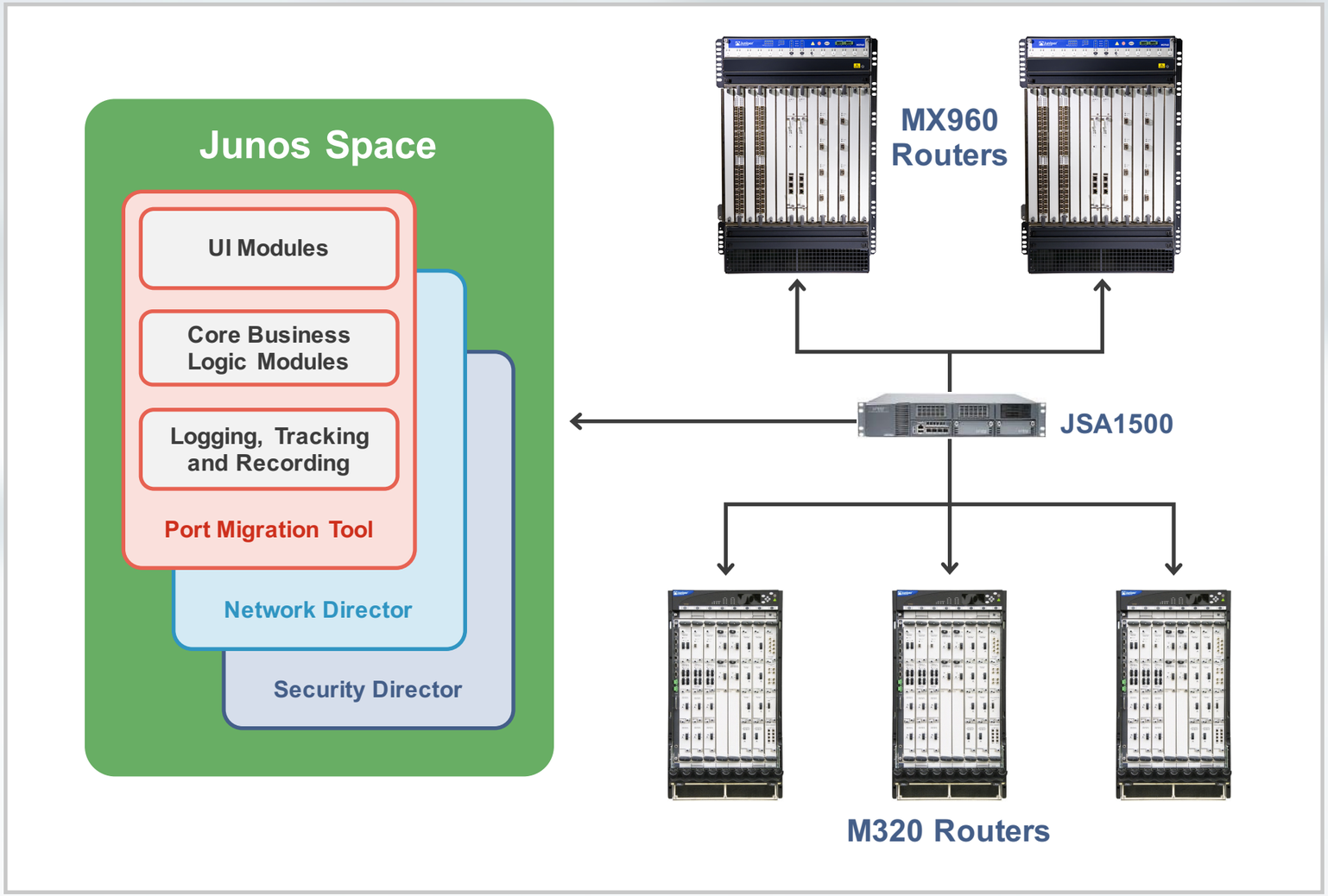
Testing

- Limited access to physical M320 and MX routers
- Restricted access to internal service provisioning toolset
- Lengthy testing phase due to the time zone difference and restricted production grade device access

High Level Port Migration Tool Design

- Requirements:
  - Graphical User Interface
  - Deployed as a plug-in/app inside Junos Space
  - GUI to prompt for selection of single-port or multi-port migration
  - GUI to display two columns: one column for selection of source router/ports; target router/port(s)
  - GUI to have 1 click button to display the transformed configuration and SQL statements
  - GUI to have 1 click button to push the transformed configuration to target router
  - GUI to provide the ability to store and download the transformed config and SQL statements
  - Logs all transactions with date stamp
  - Generate report on each port migration with status of each service

Port Migration Tool Deployment Architecture



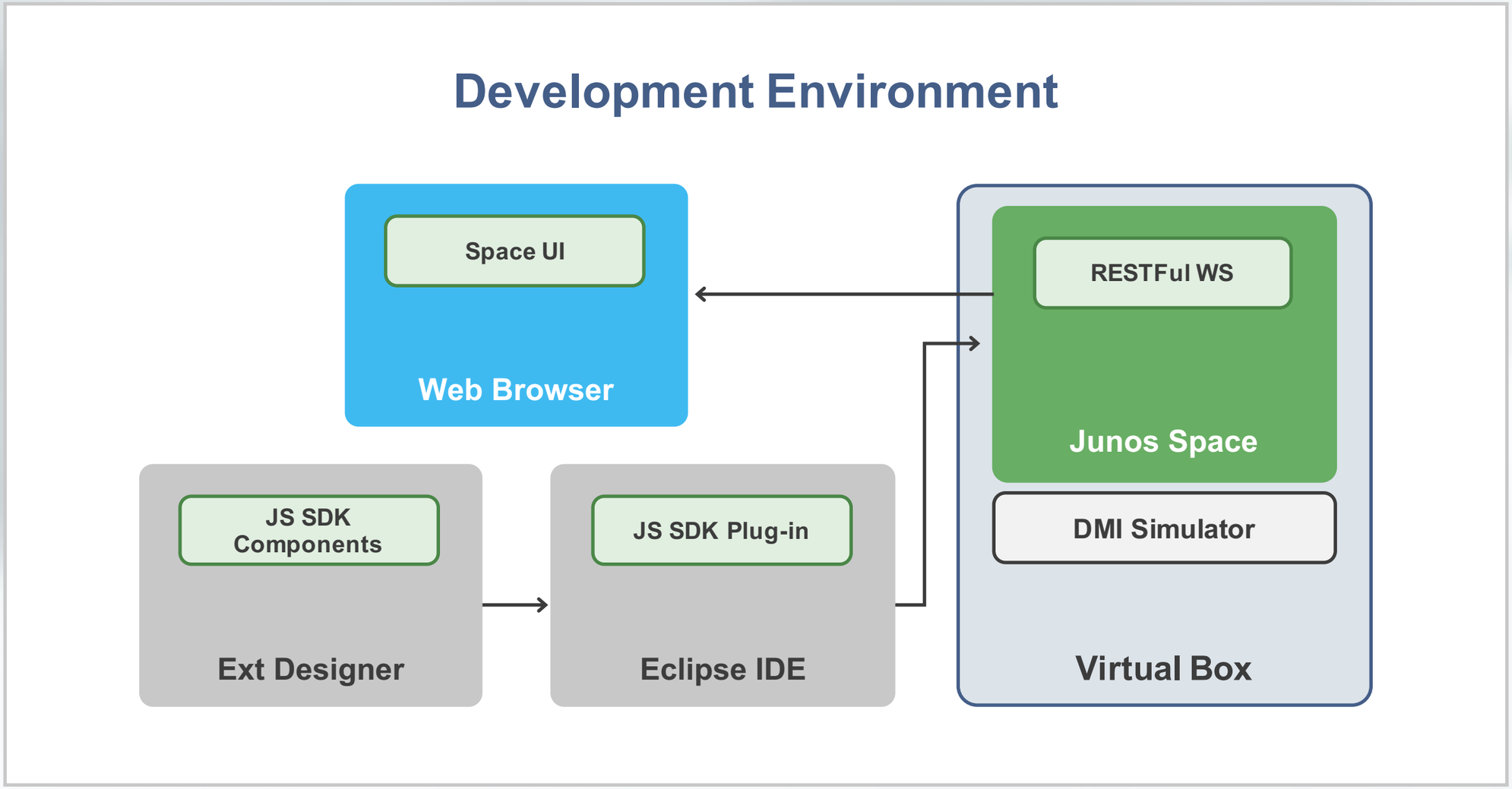
Port Migration Tool Components

- Deployed under existing Junos Space platform running on JSA1500 appliance
- Can be launched from logging into the Junos Space platform
- Requires the source M-Router and target M-Router to be managed by Junos Space platform
- Components:
  - UI Modules: Enables/Guides the user to navigate, provide inputs to the tool and initiate actions
  - Core Business Logic Modules: Business logic to convert/transform/display M-Config to MX-Config. Based on service types
  - Non-Core Business Logic Modules: Logging, Tracking, Reporting functions

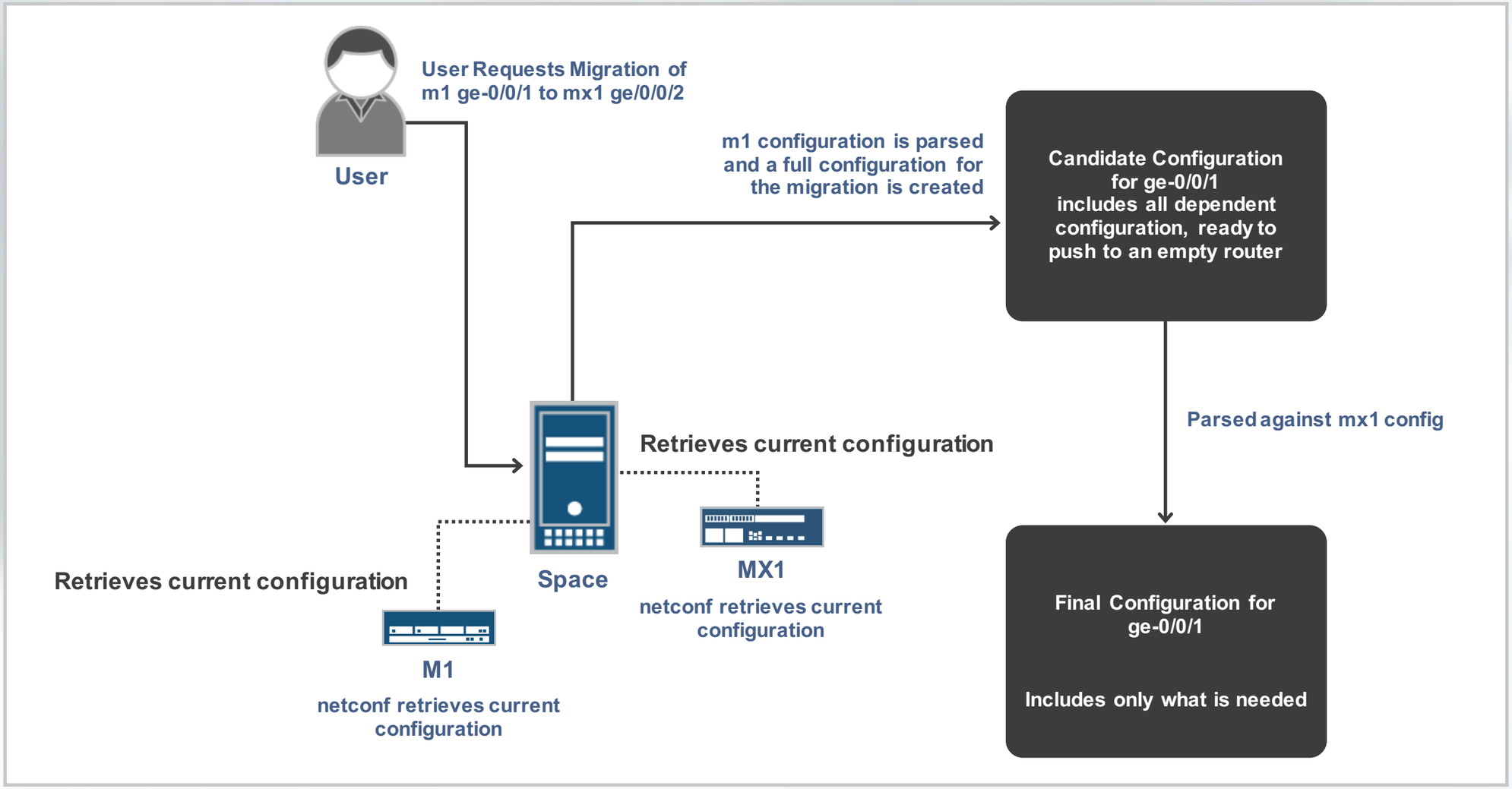
Port Migration Tool Details

- The Migration Tool is developed utilizing:
  - Junos Space 13.1
  - Junos Space SDK
  - Eclipse with Junos Space Plug-in
  - VirtualBox
  - DMI Simulator with virtual MX and M (for initial development)
  - MX and M Routers with Junos 12.3 (for final and DVT)
  - Production M-router configurations used for validation testing

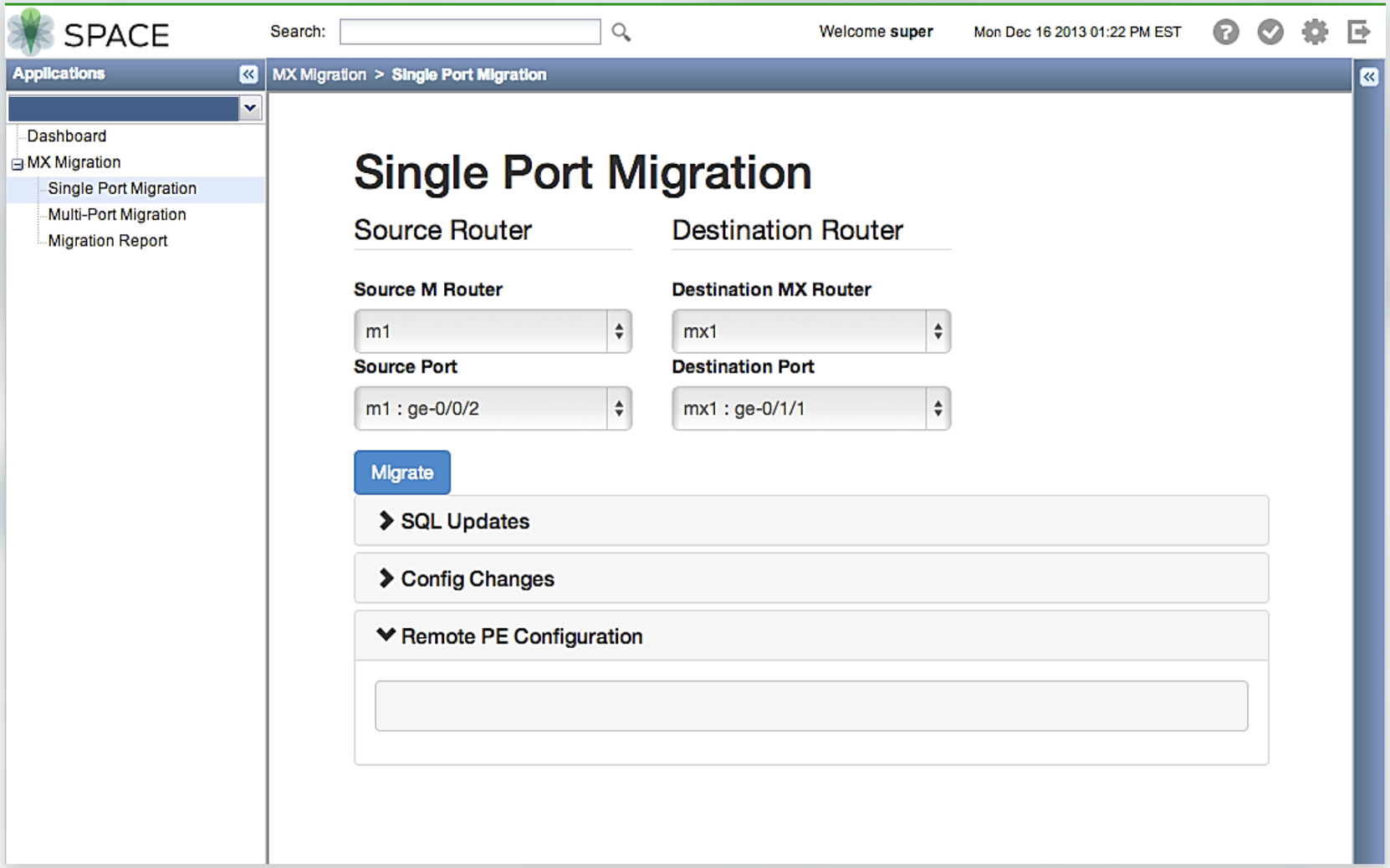
Development Environment



Project Functional Flow



Port Migration Tool in Action



Project Conclusion

- Provided Port Migration Tool to automate the migration of ports from M320 to MX routers
- Delivery
  - Juniper is training and assisting with initial 3 migrations
  - Tool is in production and was successfully put into use to perform multiple production migrations
- Future
  - New version of tool with Auto-push/rollback of configurations to support remote PEs
  - A project is underway to generalize the tool to fit other customers

Resources

Additional Resources

http://github.com/Juniper

http://developer.juniper.net

http://forums.juniper.net

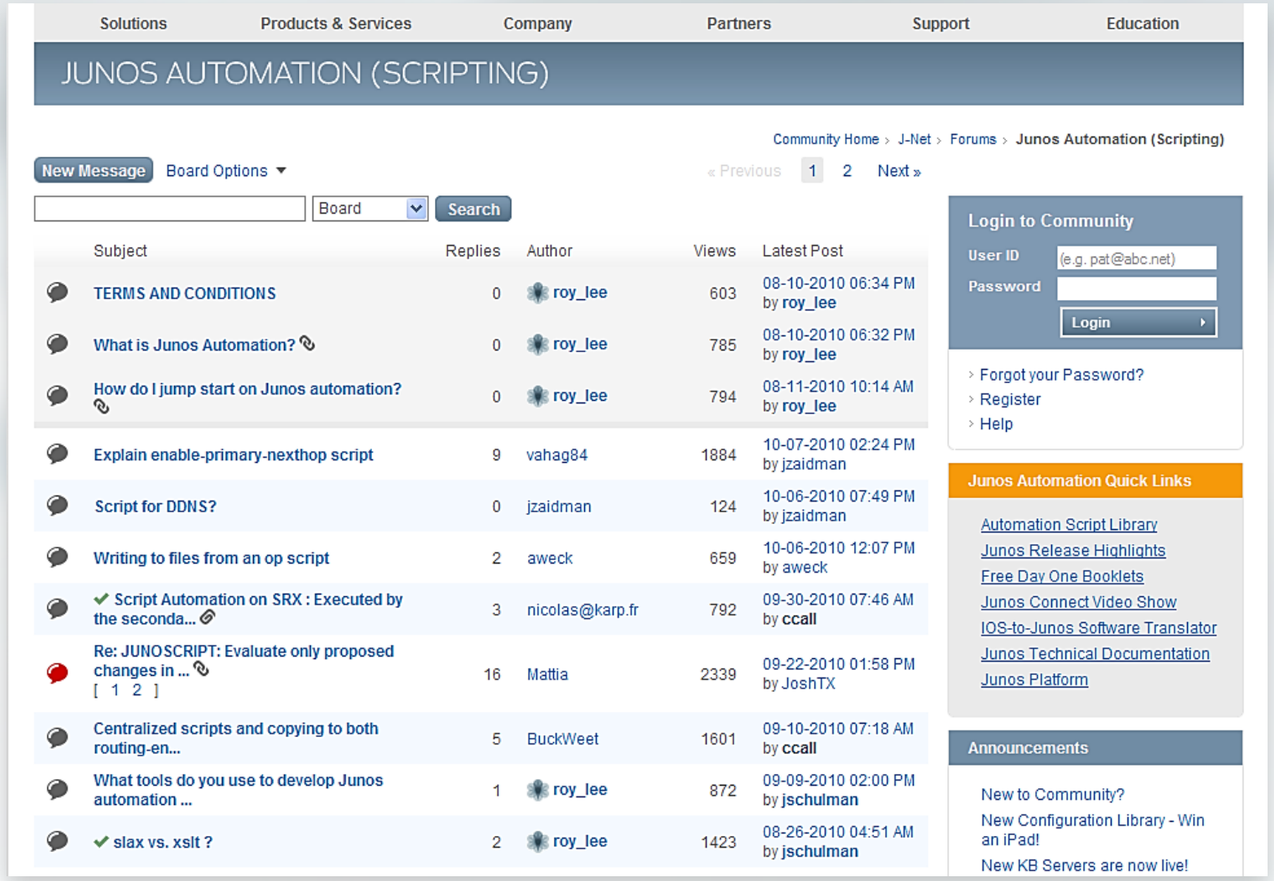
http://pathfinder.juniper.net/feature-explorer/

https://learningportal.juniper.net

On-line Support Community

http://forums.juniper.net Select “Junos Automation (Scripting)”

The Juniper “J-Net” Forum is a message board resource monitored by Junos Automation experts.



Video Training

https://learningportal.juniper.net

Automation Lab Setup