NetDevOps: Network automation with Ansible

François Caen, Systems Engineer
fracaen@cisco.com

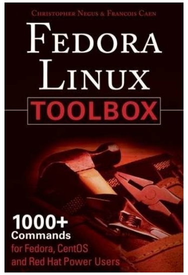
Agenda

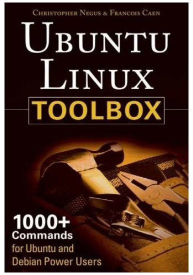
- Introduction
- Automation Motivation
- Tools: Ansible for CLI automation
- API’s: better machine communication
- Configuration Abstraction
- Conclusion

Who is this guy?

…and should I listen or look at my phone?

- 1999 – Linux sysadmin
- 2001 – FLOSS advocate / author
- 2003 – first LFNW!
- 2004 – Network Engineer
- 2015 – Cisco SE


Christopher Negus & Francois Caen
Fedora Linux Toolbox
1000+ Commands for Fedora, CentOS and Red Hat Power Users


Christopher Negus & Francois Caen
Ubuntu Linux Toolbox
1000+ Commands for Ubuntu and Debian Power Users

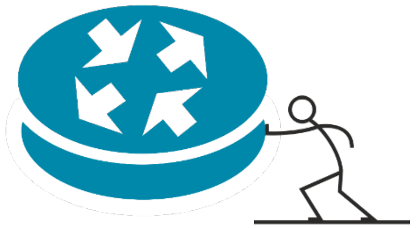
Motivators for Automation

Cloud-scale:

- Lots of Equipment:
  1000 Network Devices
- Multiple Operating Systems:
  - IOS, IOSXR, IOSXE, NXOS, ASA OS
  - Multivendor Security Appliances (WAF, DDoS, LB)
- Small team: 6 people
- Rapid Deployment
  - Several new Datacenters per year
  - Several Service Deployments requiring changes

Enterprise-scale:

- Daily repetitive tasks:
  - New device configuration
  - 3rd party NMS config
  - Change one config line on all your devices (NF collector,…)
- Monitoring:
  - Be alerted when a route goes away



What is Ansible

Ansible, an open source community project sponsored by Red Hat, is the simplest way to automate IT. Ansible is the only automation language that can be used across entire IT teams – from systems and network administrators to developers and managers.

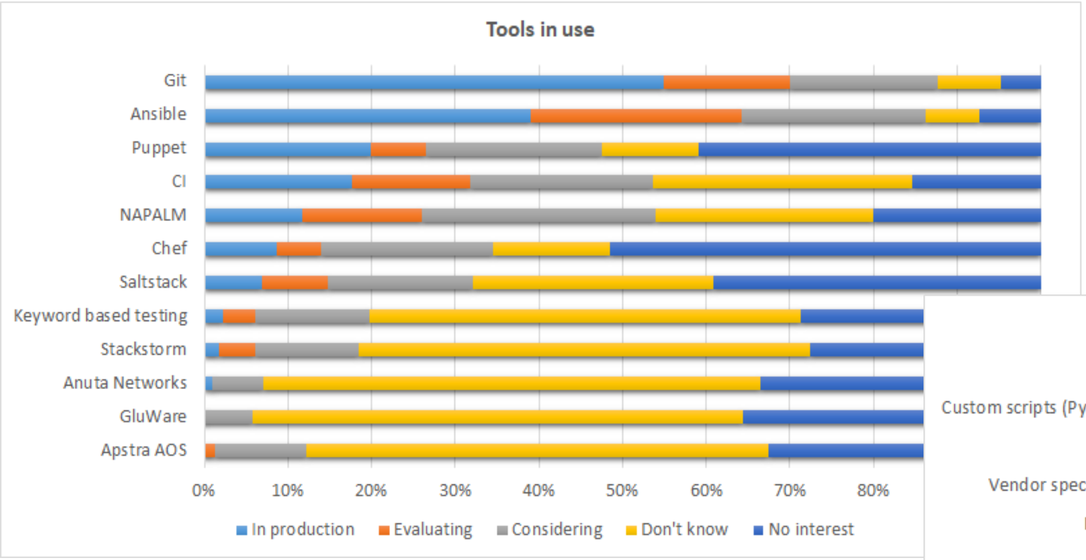
Ansible by Red Hat provides enterprise-ready solutions to automate your entire application lifecycle – from servers to clouds to containers and everything in between.

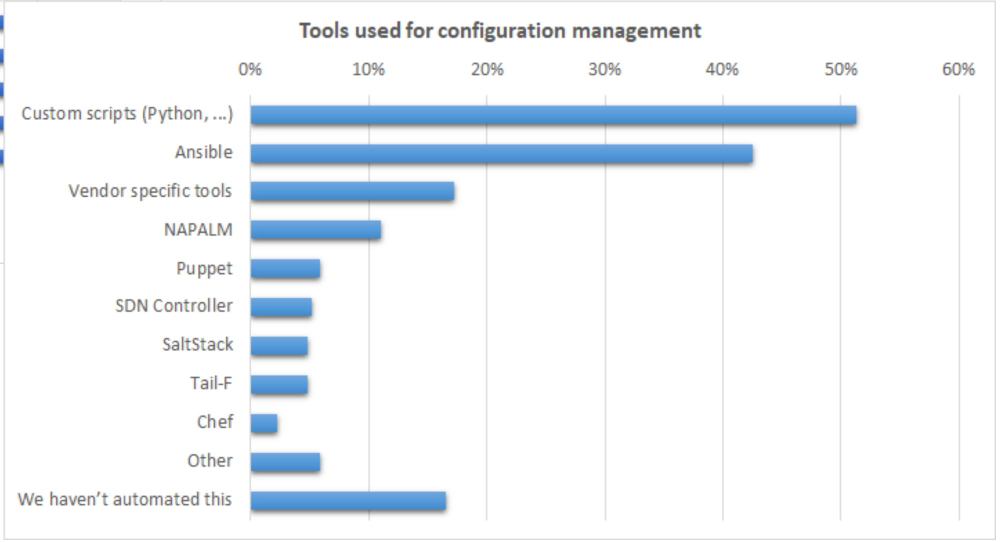
It uses no agents and no additional custom security infrastructure, so it's easy to deploy - and most importantly, it uses a very simple language (YAML, in the form of Ansible Playbooks) that allow you to describe your automation jobs in a way that approaches plain English.

Why choose Ansible?

- Agentless
- Server and support teams already using Ansible
- Infrastructure as code
- Simple to use and learn
- Community and vendor driven
- Modular framework, easily modified
- Leverage many common programming languages

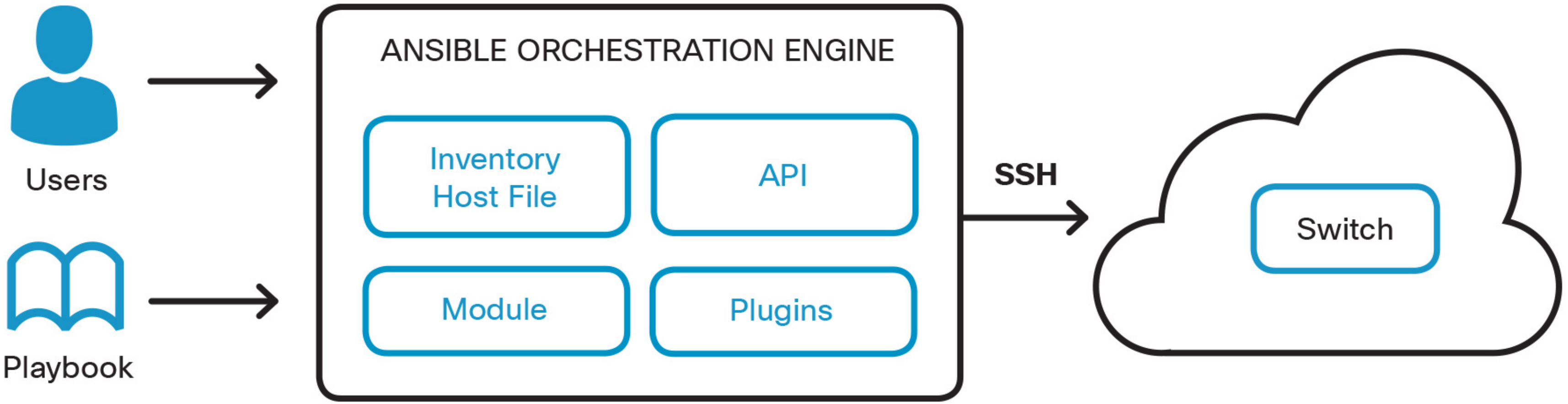
NetDevOps Fall 2016 Survey



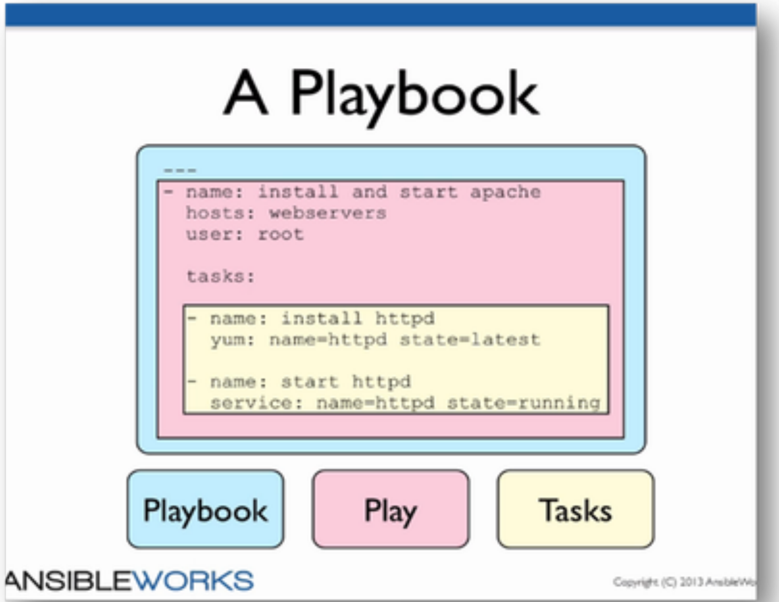


Source:
https://interestingtraffic.nl/2017/03/27/insights-from-the-netdevops-fall-2016-survey/

Ansible Configuration Management Workflow



Ansible Terms

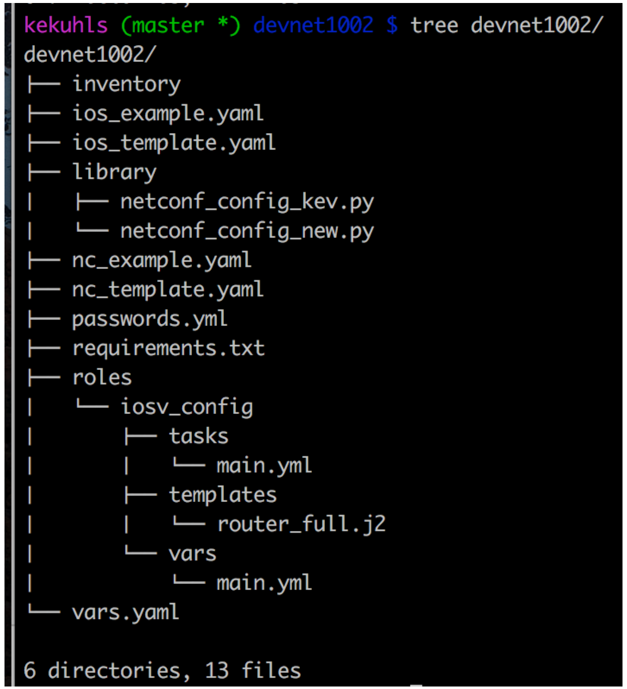


A Playbook

```
---
- name: install and start apache
  hosts: webservers
  user: root

  tasks:
  - name: install httpd
    yum: name=httpd state=latest

  - name: start httpd
    service: name=httpd state=running
```



```
kekuhls (master *) devnet1002 $ tree devnet1002/
devnet1002/
├── inventory
├── ios_example.yaml
├── ios_template.yaml
├── library
|   ├── netconf_config_kev.py
|   └── netconf_config_new.py
├── nc_example.yaml
├── nc_template.yaml
├── passwords.yml
├── requirements.txt
├── roles
|   └── iosv_config
|       ├── tasks
|       |   └── main.yml
|       ├── templates
|       |   └── router_full.j2
|       └── vars
|           └── main.yml
└── vars.yaml

6 directories, 13 files
```

Ansible for Networking

```
- name: load new acl into device
  ios_config:
    lines:
      - 10 permit ip host 1.1.1.1 any log
      - 20 permit ip host 2.2.2.2 any log
      - 30 permit ip host 3.3.3.3 any log
      - 40 permit ip host 4.4.4.4 any log
      - 50 permit ip host 5.5.5.5 any log
    parents: ip access-list extended test
    before: no ip access-list extended test
    match: exact
    provider: "{{ cli }}"
```

Jinja Template

Contains **variables** and/or **expressions** which get replaced with values when *rendered*

# **Simple Variable Replacment**
hostname {{sitecode}}-fw

# **Variable Replacement based on Dictionary**
route outside 0.0.0.0 0.0.0.0 {{config['vlan101']['ip'][1]}}

# **Variable Replacement by Filter**
route outside 0.0.0.0 0.0.0.0 {{ external_net_cidr | ipaddr('1') | ipaddr('address') }}"

# **Loop Through set of data to create multiple lines**
{%for route in config['routes'] %}
route oob-vpn {{config['routes'][route]['network']}} {{config['routes'][route]['mask']}} {{config['vlan90']['ip'][1]}}
{% endfor %}

# **Conditional Statements**
{% if config['vlan41'] is defined %}
route dmzext {{config['vlan41']['ip'][0]}} {{config['vlan41']['ip'].netmask}} {{config['vlan102']['ip'][1]}}
{endif %}

Yaml

- Structure to define:
  - dictionary (unordered set of key value pairs, lists)
  - list of items
  - key value pair

```
# A sample employee record
name: Francois Caen
job: Systems Engineer
employed: True
languages:
    French: Native
    English: Fluent
    German: Novice
    python: Novice
education: Maitrise
favorite drinks:
    - Beer
    - Gin
```

Ansible 2.x Exercise

Configuration Management Today: CLI

Human Friendly

Task Oriented

Easy To Replay

No Special Tools

Syntax format changes

No Structured output

No Error Reporting

No Transaction manageme nt

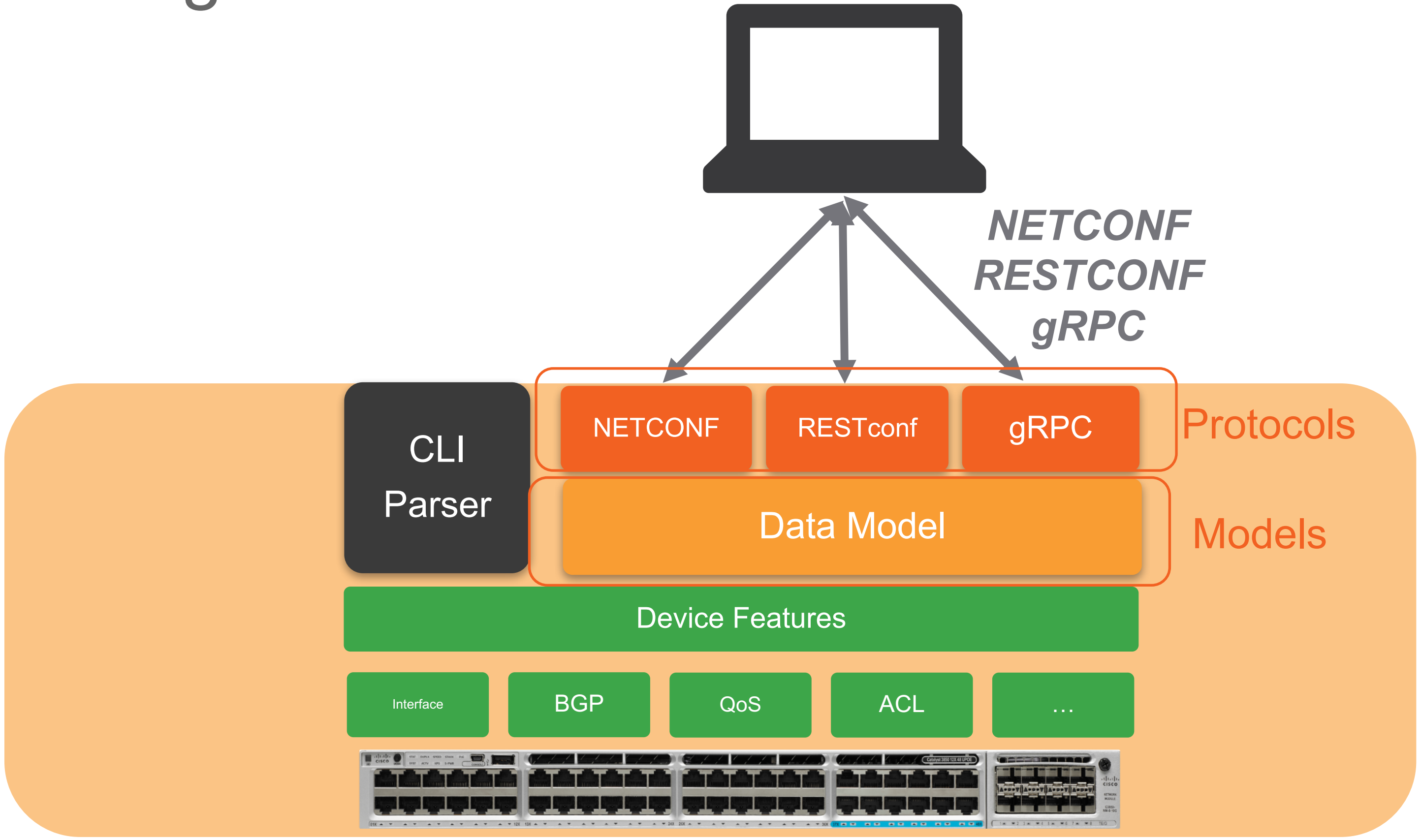
Alternatives to CLI automation?

APIs – Application Programming Interfaces

*“A **set of Function Calls** that allow talking to a system”*

- Programming Building block
- APIs can have various **Properties**
  - **Transport** (SSH, HTTP)
  - **Encoding** (XML, JSON, ProtoBuffer)
  - **Data structure** (Data Models)
- Some **Examples** of APIs
  - The Twitter API
  - The Java API

Open Programmable Interface



Encoding Formats

“XML, JSON, YAML are ***Text-file formats*** *used to* ***store structured data*** *for embedded and Web applications”*

XML vs JSON vs YAML

```
<interfaces xmlns:="“[…]yang:ietf-interfaces”>
    <interface>

        <name>eth0</name>
        <type>ethernetCsmacd</type>
        <location>0</location>
        <enabled>true</enabled>
        <if-index>2</if-index>

    </interface>
</interfaces>
```

```
{
    "ietf-interfaces:interfaces": {
        "interface": [
            {
                "name": "eth0”,
                "type": "ethernetCsmacd”,
                "location": "0”,
                "enabled": true,
                "if-index": 2
            }
        ]
    }
}
```

```
---
ietf-interfaces:interfaces:
    interface:
        name: eth0
        type: ethernetCsmacd
        location: 0
        enabled: true
        if-index: 2
```

NETCONF definition

*"NETCONF is **a protocol defined by the IETF** to **install**, **manipulate**, and delete the **configuration of network devices**"*

V 1.0

- RFC 4741 1.0 Base NETCONF Protocol
- RFC 4742 NETCONF over SSH

V 1.1

- RFC 6241 – 1.1 Base NETCONF Protocol
- RFC 6242 – NETCONF over SSH

Extensions

- RFC 5277 Notifications
- RFC 5717 Partial Locking
- RFC 6243 With defaults
- RFC 6020 YANG

2006

2011

NETCONF Operations

Main Operations		Description
<get>	(close to ‘show ?’)	Retrieve running configuration and device state information
<get-config>	(close to ‘show run’)	Retrieve all or part of specified configuration datastore
<edit-config>	(close to ‘conf t’)	Loads all or part of a configuration to the specified configuration datastore

Other Operations	Description
<copy-config>	Replace an entire configuration datastore with another
<delete-config>	Delete a configuration datastore
<commit>	Copy candidate datastore to running datastore (ex: XR)
<lock> / <unlock>	Lock or unlock the entire configuration datastore system
<close-session>	Graceful termination of NETCONF session
<kill-session>	Forced termination of NETCONF session

<get-config>

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config>
    <source>
      <running/>
    </source>
    <filter>
      <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
        <interface>
          <name>GigabitEthernet0</name>
        </interface>
      </interfaces>
    </filter>
  </get-config>
</rpc>
```

<get-config> Response

```
<rpc-reply message-id="urn:uuid:bdb1189e-4480-11e6-8507-fa163e2846a4"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
      <interface>
        <name>GigabitEthernet0</name>
        <type xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">ianaift:ethernetCsmacd</type>
        <enabled>true</enabled>
        <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
          <address>
            <ip>172.26.170.253</ip>
            <netmask>255.255.254.0</netmask>
          </address>
        </ipv4>
        <ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip"/>
      </interface>
    </interfaces>
  </data>
</rpc-reply>
```

<edit-config>

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
        <interface>
          <name>GigabitEthernet0/0/0</name>
          <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
            <address>
              <ip>1.1.1.1</ip>
              <netmask>255.255.255.255</netmask>
            </address>
           </ipv4>
        </interface>
      </interfaces>
    </config>
  </edit-config>
</rpc>
```

<edit-config> Response

```
<rpc-reply message-id="urn:uuid:d5fa0c22-4484-11e6-9132-fa163e2846a4"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

<edit-config> Rollback

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
        <interface>
          <name>GigabitEthernet0/0/0</name>
          <description>DID-ROLLBACK-WORK</description>
          <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
            <address>
              <ip>blah</ip>
              <netmask>255.255.255.255</netmask>
            </address>
           </ipv4>
        </interface>
      </interfaces>
    </config>
  </edit-config>
</rpc>
```

<edit-config> Rollback Response

```
<error-info xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <bad-element>ip</bad-element>
</error-info>
```

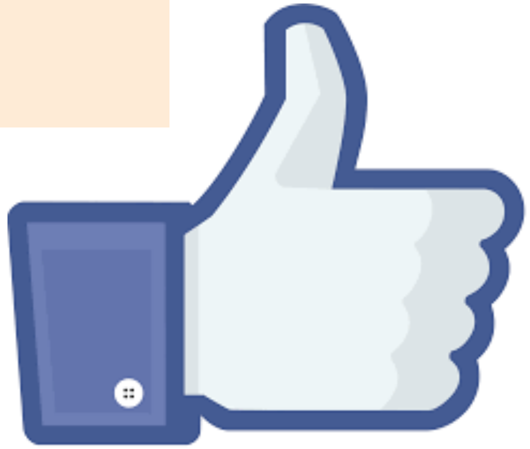
Ansible for Networking

```
- name: configure new ntp server
  netconf_config:
  xml: |
  <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
   <system xmlns="urn:ietf:params:xml:ns:yang:ietf-system">
     <ntp>
       <enabled>true</enabled>
         <server>
           <name>ntp1</name>
           <udp><address>127.0.0.1</address></udp>
         </server>
       </ntp>
     </system>
   </config>
```

NETCONF Exercise

Three Things to Like about NETCONF

1. Capability discovery, model download
2. Transactions
3. Notifications



Platform Support

Up-to-date version lives at: https://developer.cisco.com/site/standard-network-devices/

	Shipping	
IOS-XR	**IOS-XR 5.3.0 +** • ASR 9000 • NCS 1000 • NCS 5000	-
IOS-XE Nova	**IOS-XE 3.9.1 +** • Catalyst 4500	-
IOS-XE 16.x.1	**IOS-XE 16.3.1 +** • Catalyst 3650 • Catalyst 3850 • ASR 1000 • ISR 4400 • CSR 1000v • ISRv	
NX-OS	**“E” Release +** • Nexus 3000 • Nexus 9000	

Configuration Abstraction

Infrastructure as Code Example

Variable structure to represent Campus Fabric

```
fabric:
  - tenant_name: DEVELOPMENT
    tenant_id: 103
    ints:
      - vlan_id: 3240
        name: "10_103_240_0-DATA"
        subnet: " 10.103.240.0/24"
  - tenant_name: EMPLOYEE
    tenant_id: 101
    ints:
      - vlan_id: 1240
        name: "10_101_240_0-DATA"
        subnet: " 10.103.240.0/24”
      - vlan_id: 1241
        name: "10_101_241_0-VOICE"
        subnet: " 10.101.241.0/24”
```

Infrastructure as code Exercise

Next steps

- DevNet Express class
  Seattle June 6-7, other locations worldwide
  http://devnetevents.cisco.com

- Cisco Live:
  US: Las Vegas June 25-29
  Cancun, Barcelona, Melbourne

References

Cisco DevNet – http://developer.cisco.com
Ansible – http://www.Ansible.com
Jinja - https://kontrolissues.net/2016/01/14/intro-to-jinja2/
YAML - http://www.yaml.org/start.html
VIRL – Virtual Internet Routing Lab (how the demos were deployed) http://virl.cisco.com/
dCloud – The Cisco Demo Cloud (where the demos were deployed) https://dcloud.cisco.com/
(ask your Cisco account team or partner for access)
Source code in Github:

- Clone exercises from session: git clone git@github.com:kuhlskev/devnet1002.git
- Ansible Networking - https://github.com/ansible/ansible-modules-core/tree/stable-2.1/network
- Napalm - https://github.com/napalm-automation/napalm
- Cisco Config Repo

Blogs:

- https://pynet.twb-tech.com/
- http://jedelman.com
- https://networklore.com/

Q & A

Cisco live!

Thank You

Building the Environment

This is a rough guideline how to bring up / prepare the entire environment.

- Git client
- VirtualBox 5.0.28
- Vagrant 1.8.7 (be aware of this issue)
- Docker 1.13.1
- cdrtools (in particular mkisofs)
- a build environment (e.g. compiler, make, ...), suggest to use MacPorts or Brew if running on a Mac
- Clone the iso-xrv-x64-vbox repository from GitHub
- IOS XE image from Cisco.com (e.g. here, then go to *IOS XE Software* and download the Denali-16.3.2 .iso file in the *Latest* tree branch, ~350MB in size)

Building the Environment (cont)

Building the Vagrant Box

- Go to the directory where you cloned the iso-xrv-x64-vbox repository. Start the Vagrant box image build by running the following command
- iosxe_iso2vbox.py -v ~/Downloads/csr1000v-universalk9.16.03.02.iso
- This will take a while. When done, you need to install the resulting box into Vagrant:
- vagrant box add --name iosxe1002 csr1000v-universalk9.16.03.02.box
- (See the output at the end of the script. It has the exact location of the generated box file and also the command to add / replace the Vagrant box file).

Configure and Start Routers

The next steps are required to prepare configuration disks for the routers

- Clone this repo from GitHub into a new directory: https://github.com/kuhlskev/devnet1002
- Make sure that the Vagrant box name matches the one configured in the Vagrant file
- Ensure you have the required tools installed
- run make to create the ISO files with the router configurations
- Bring up the routers using vagrant up (brings up both) or vagrant up rtr1 to only start rtr1

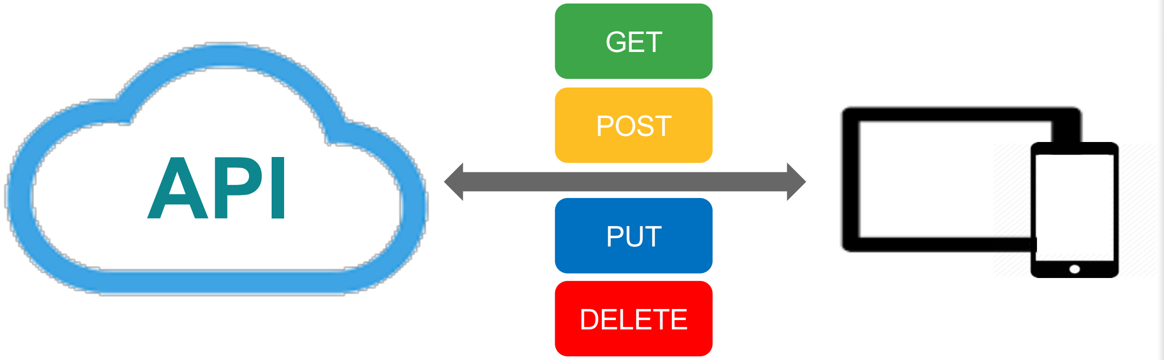
Protocols

	NETCONF	RESTconf	gRPC
Standard	IETF RFC 6241	IETF draft	Open source
Optimized for Networking	✓	✓	✗ (designed for generic client/server communications)
Session Oriented	✓	✓	✓
Connection Oriented	✓	✗	✓
Connection Security	SSH or SOAP	HTTPS	HTTPS

REST (REpresentational State Transfer)

RESTful
GET |PUT | POST | DELETE

*"A **framework** for **client-server** communications"*



- **not a standard**
- Architectural constraints
  - Client–server
  - Stateless
  - Cacheable
  - Layered system
  - Uniform interface