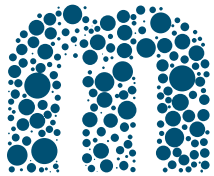
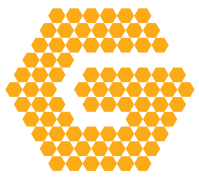
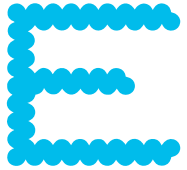






INTUITIVE

Cisco live!

December 4-7, 2018 · Cancun

#CiscoLiveLA

Unpacking the NetDevOps Toolbox

Hank Preston & Kevin Corbin
NetDevOps Guys
@hfpreston
@kecorbin

BRKDEV-1267

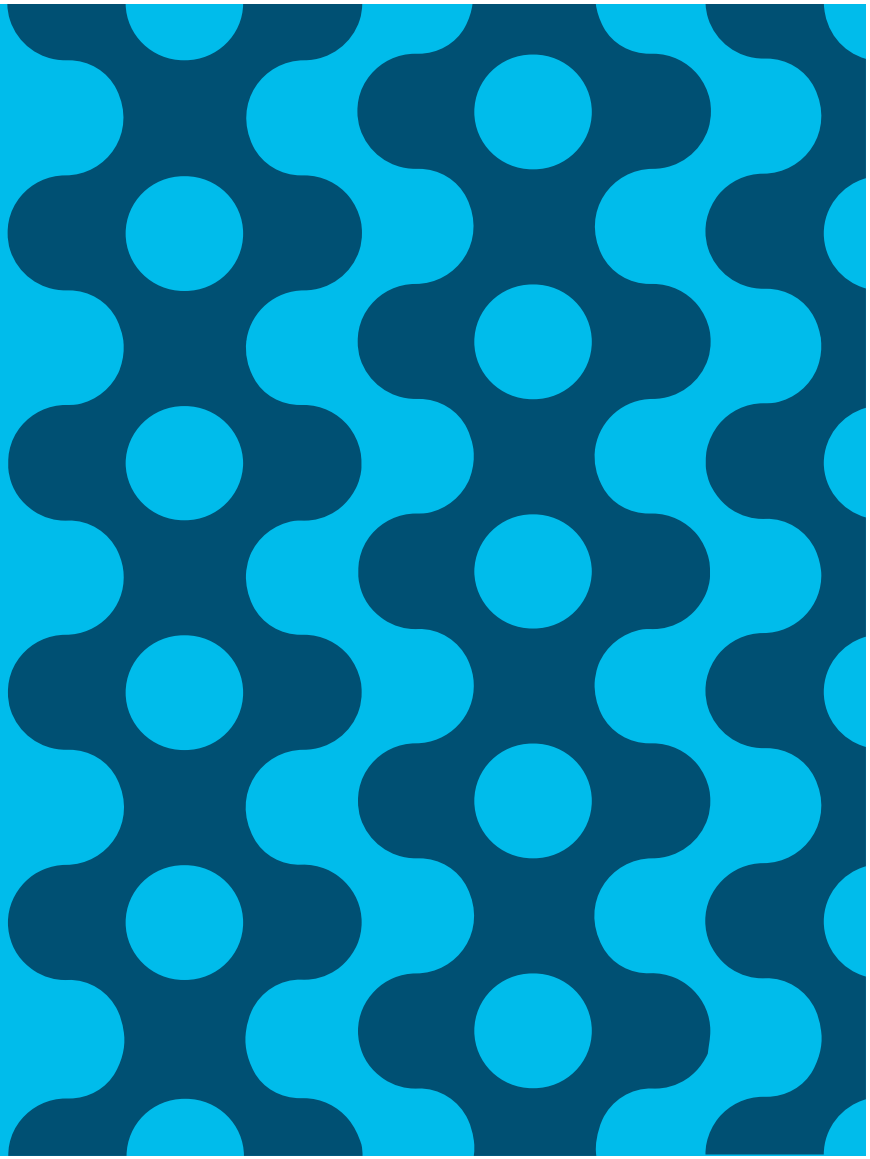
Cisco live!

#CiscoLiveLA

Agenda

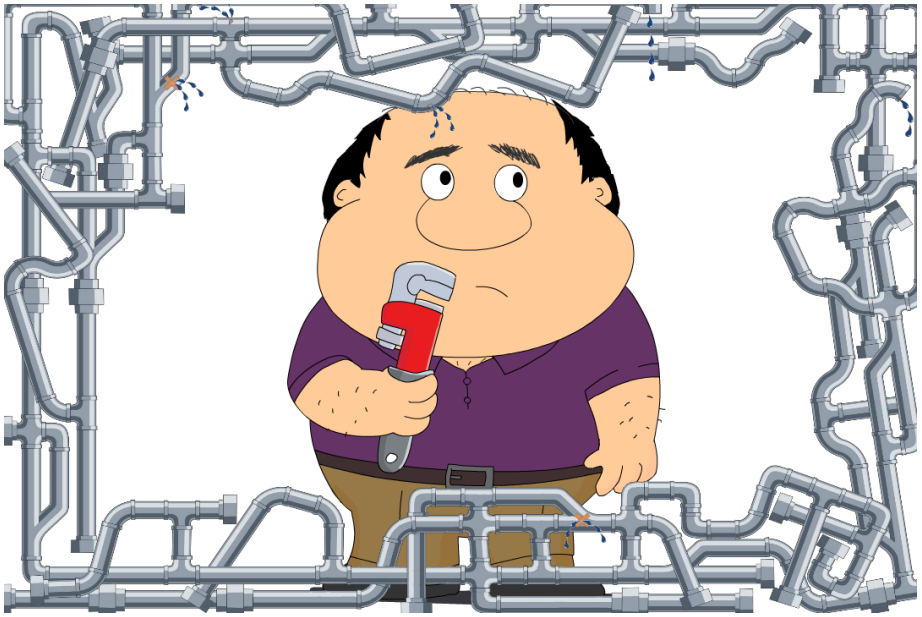
- What are we trying to achieve with NetDevOps?
- Picking our tools
  - Source Control
  - Environment Simulation
  - Configuration Management
  - Network Verification
- Review and Next Steps

What are we trying to achieve with NetDevOps?



Today's reality...

- Functional but considered fragile
- Network configuration more "art than science"
- Tribal knowledge of key engineers



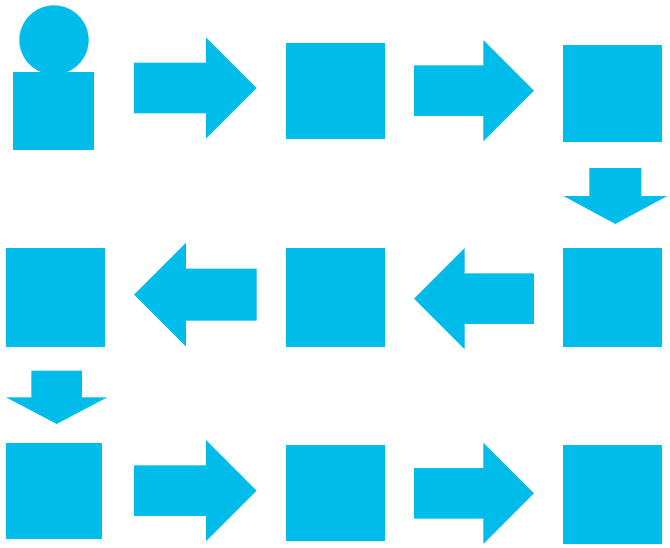
"Every time we implement a network change something goes wrong..."

"Isn't it great, our switch hasn't been rebooted in 6 years"

"We can't update/change the network, our business won't allow it"

* Paraphrased quotes from actual network operators

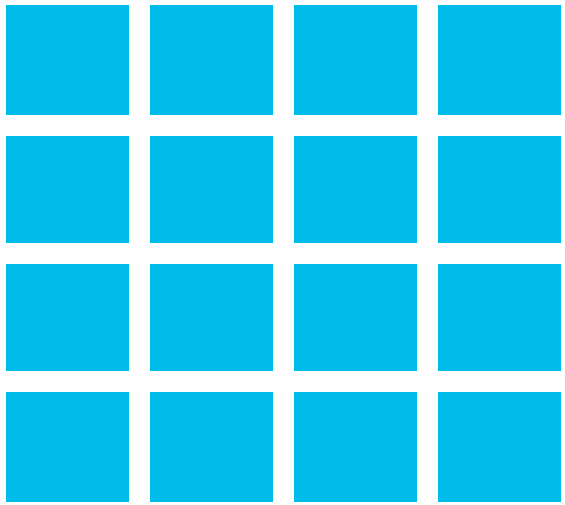
Today's Network Realities

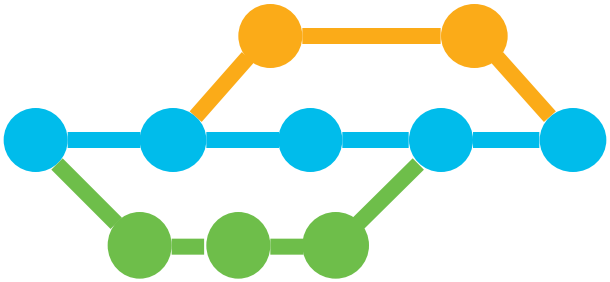


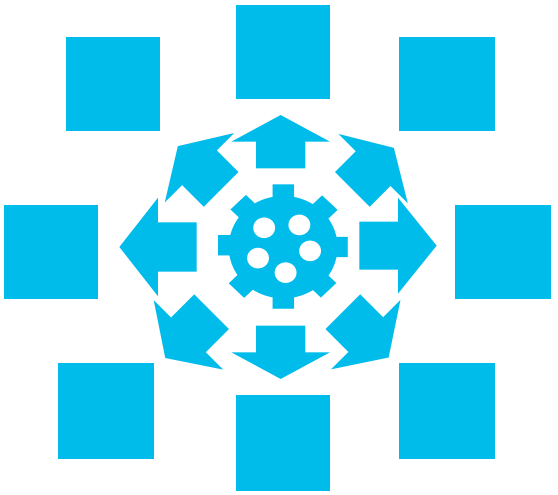
Sequential and Manual Infrastructure Provisioning

Snowflake Infrastructure with Time Capsules of Configurations

NetDevOps Will Deliver



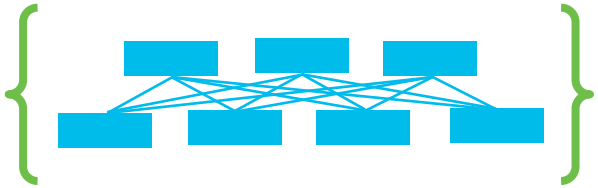




Consistent Version Controlled Infrastructure deployed with Parallel & Automated Provisioning

NetDevOps Operational Models

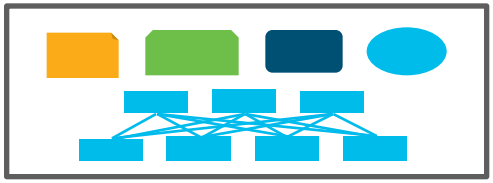
Network as Code



- Git based workflows
- Close alignment to software DevOps approaches
- Leverage abstractions, such as controllers, when possible

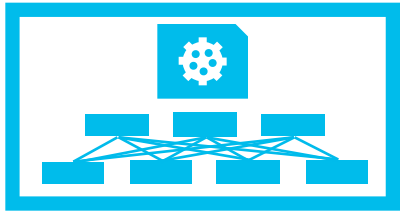
Our Focus in this Session

IT as a Service



- Service Catalog based workflows
- Deliver End User Self Service experience in “eStores”

Controller Driven



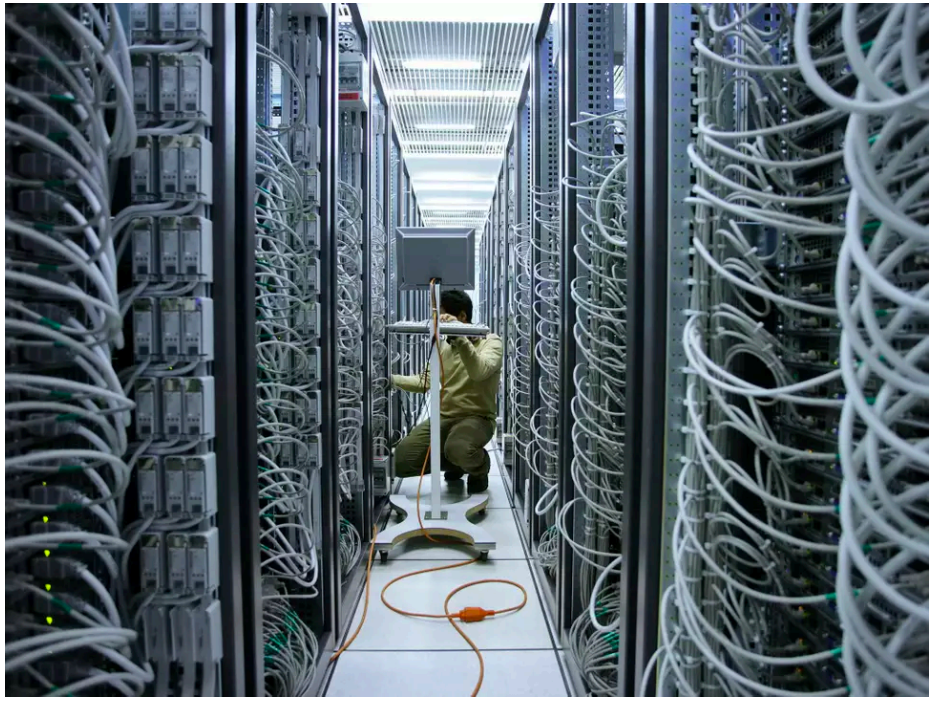
- Network Controller based workflows
- Evolving traditional network operation model

Principles of "Network as Code"

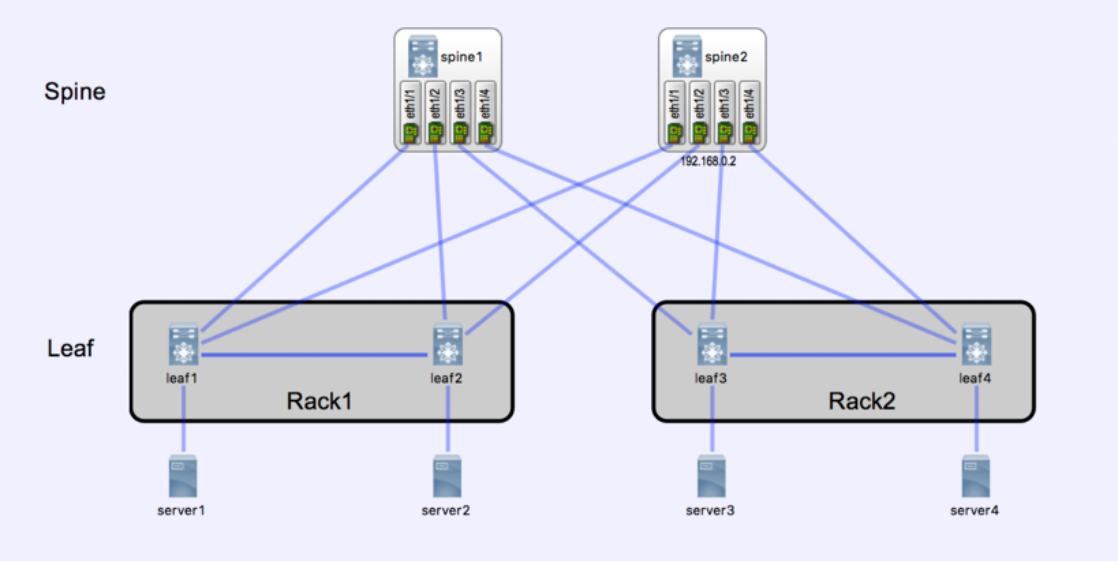
- Store Network Configurations in Source Control
- Source Control is *a* Source of Truth*
- Configuration Management Deployment through APIs

** The "Source of **Whole** Truth" for the network will likely be constructed from multiple systems holding specific details. For example, IP details would come from IP Address Management.*

The Promise of NetDevOps



Common Demo Topology Details

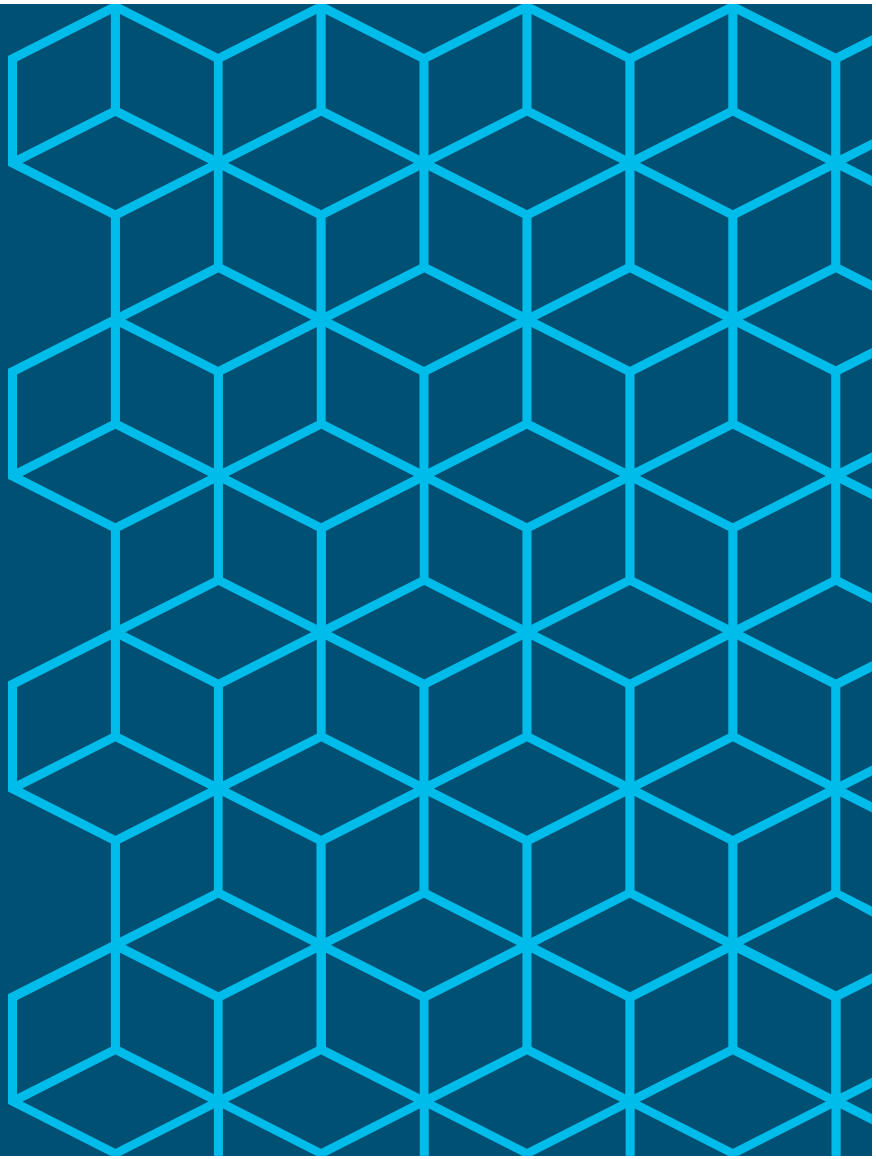


- Spine Leaf Architecture
- MP-BGP EVPN VXLAN
  - RFC 7432
- Fancy way of saying a next gen “VLAN”

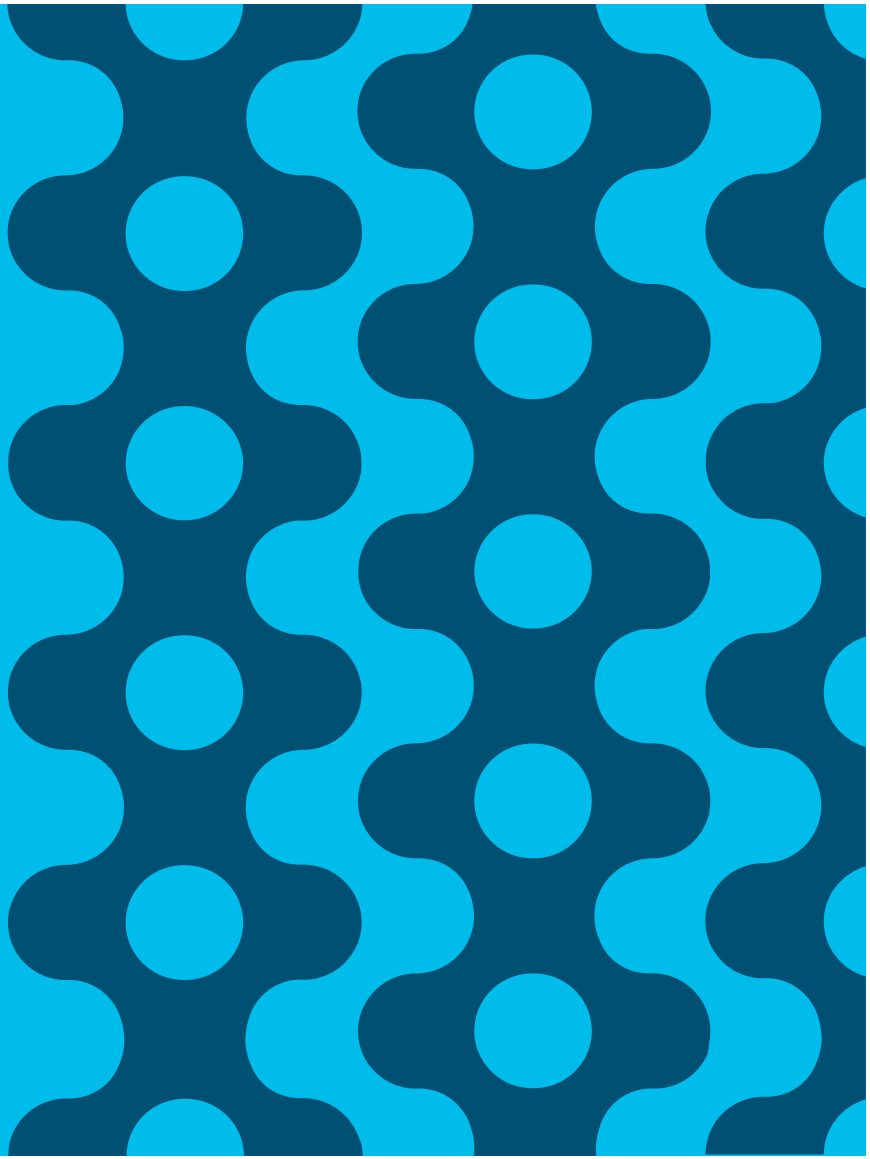
Topologies brought to you by:

VIRL

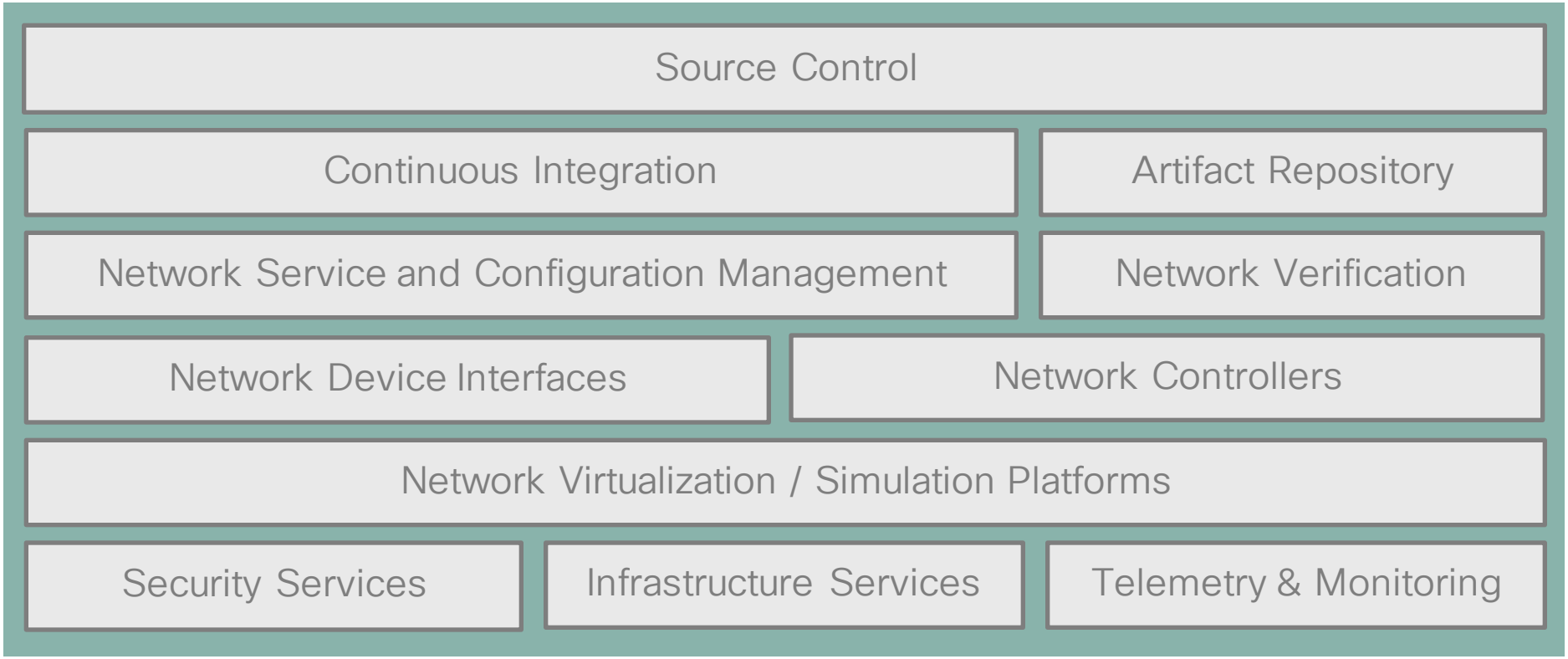
Demo



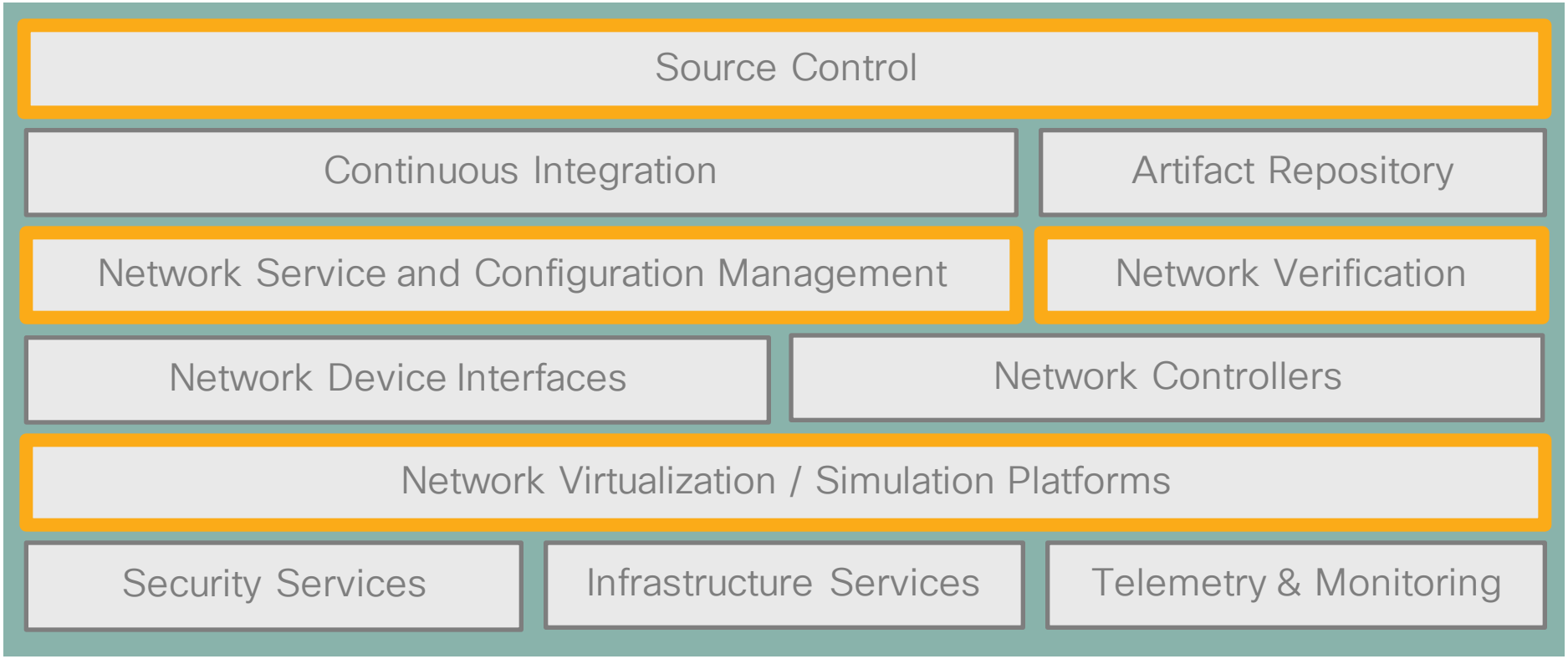
Picking our tools



The NetDevOps Engineers Tool Chest



Today's Session will Explore

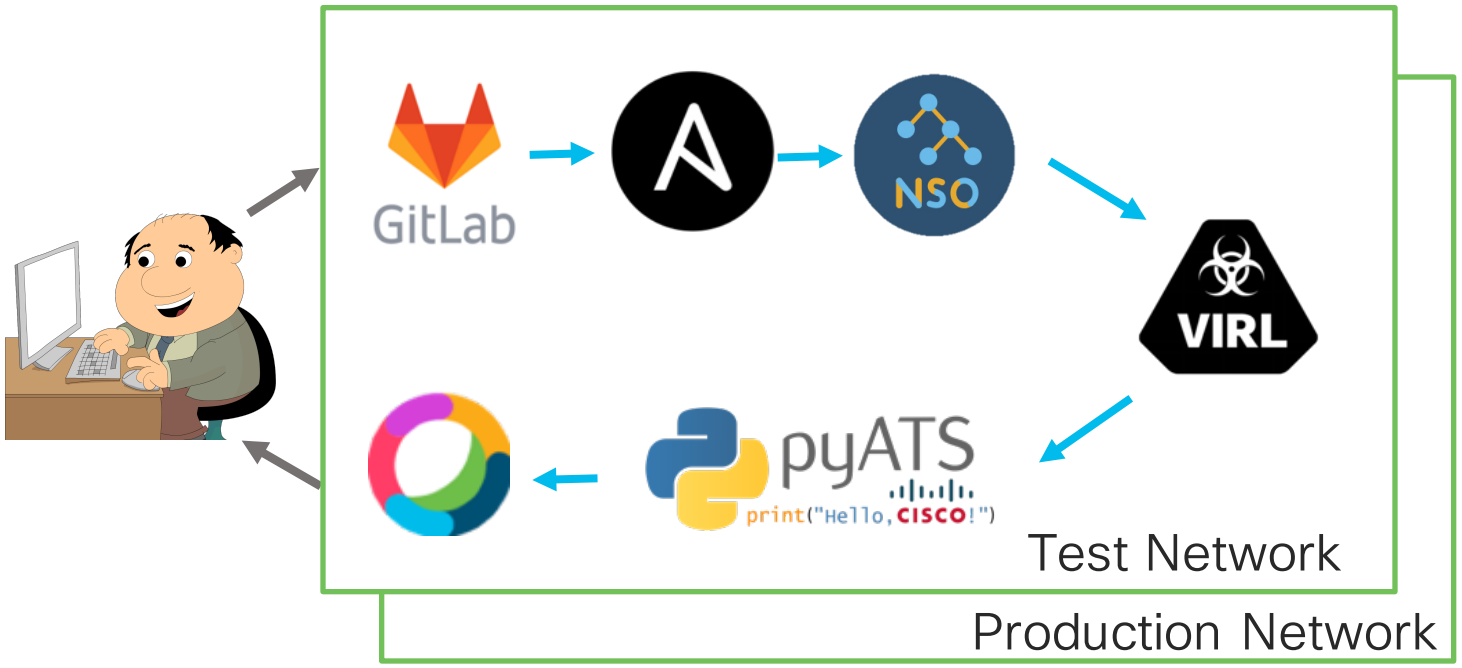


How to pick a tool? (Not in priority order)

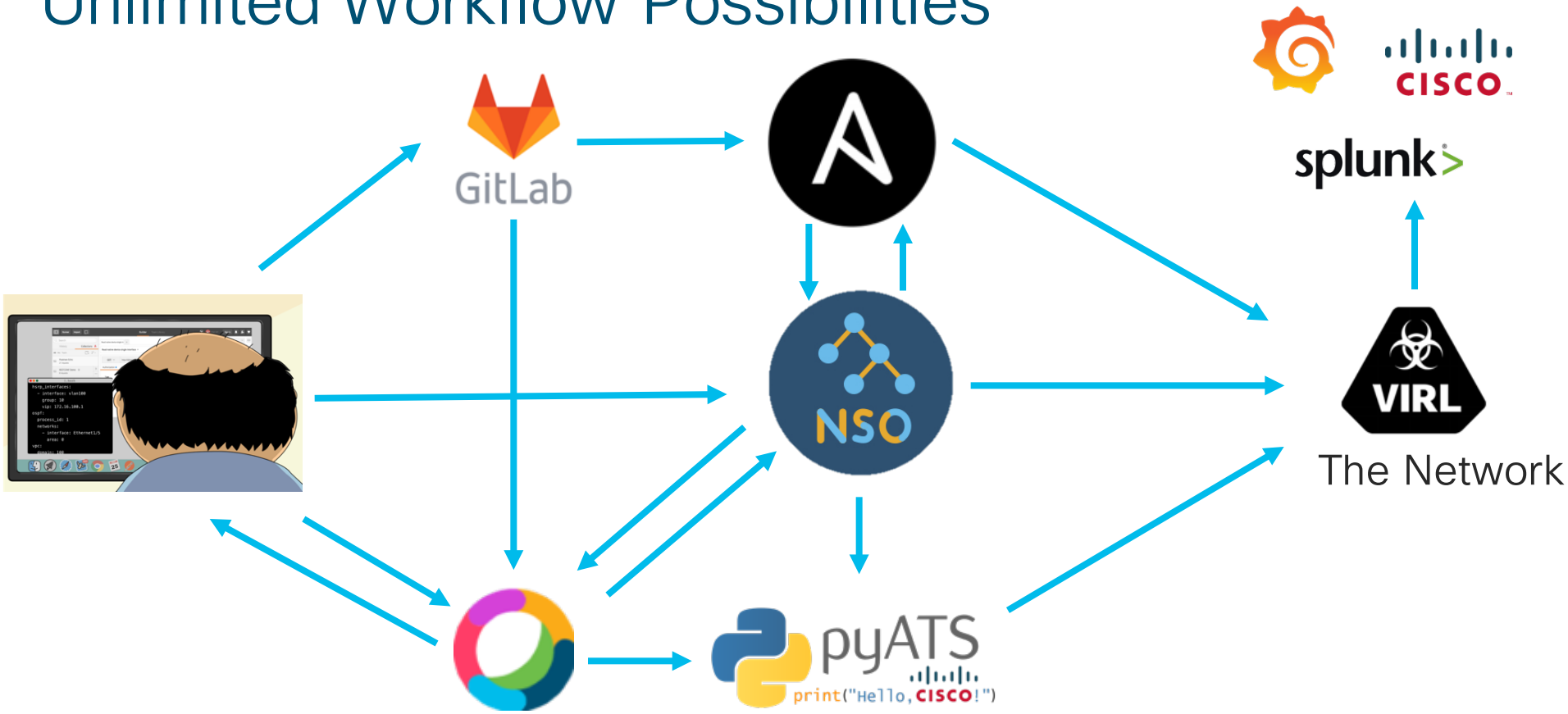
- Commercial vs Open Source
- Programming language
- Supported integrations
- Popularity in community
- Relevant examples
- Tool Capabilities
- Used elsewhere in organization

Often no one tool will fit, using multiple is okay too!

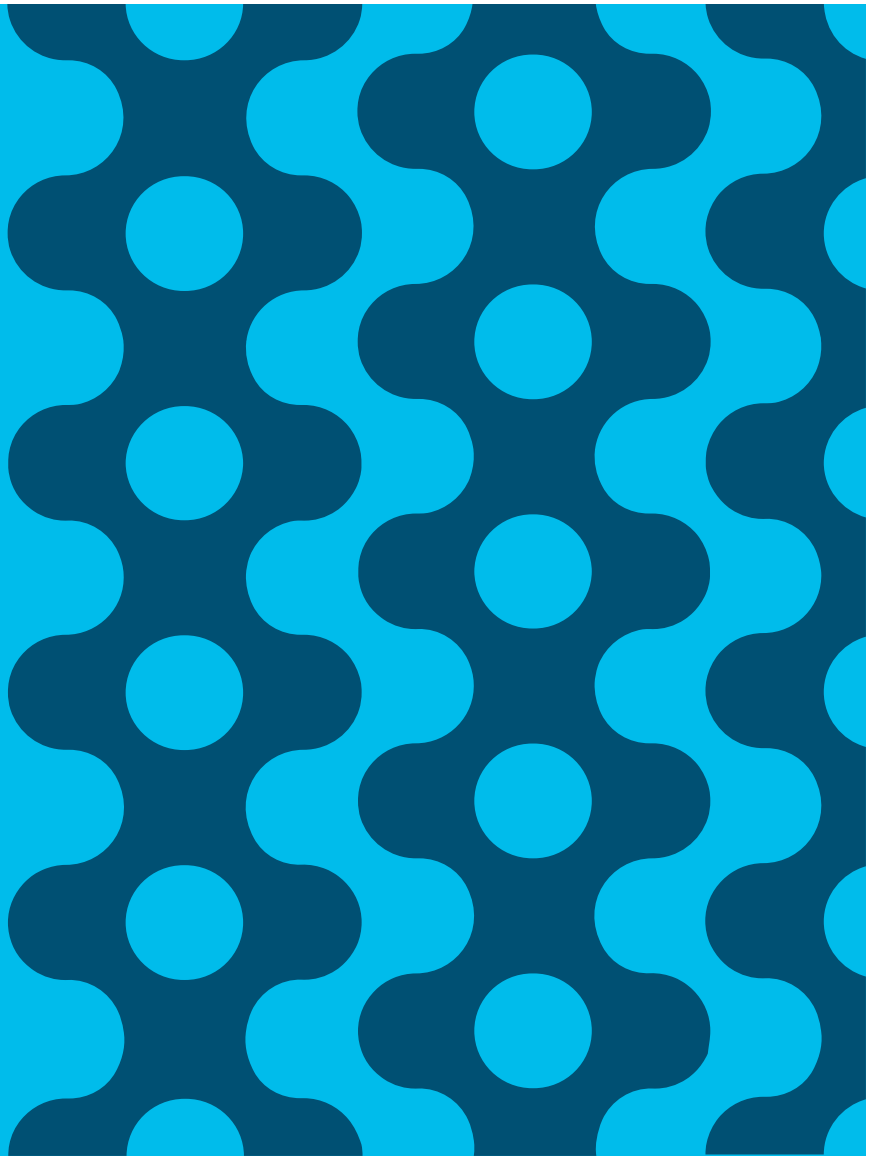
Our “opinionated” Stack + Workflow



Unlimited Workflow Possibilities

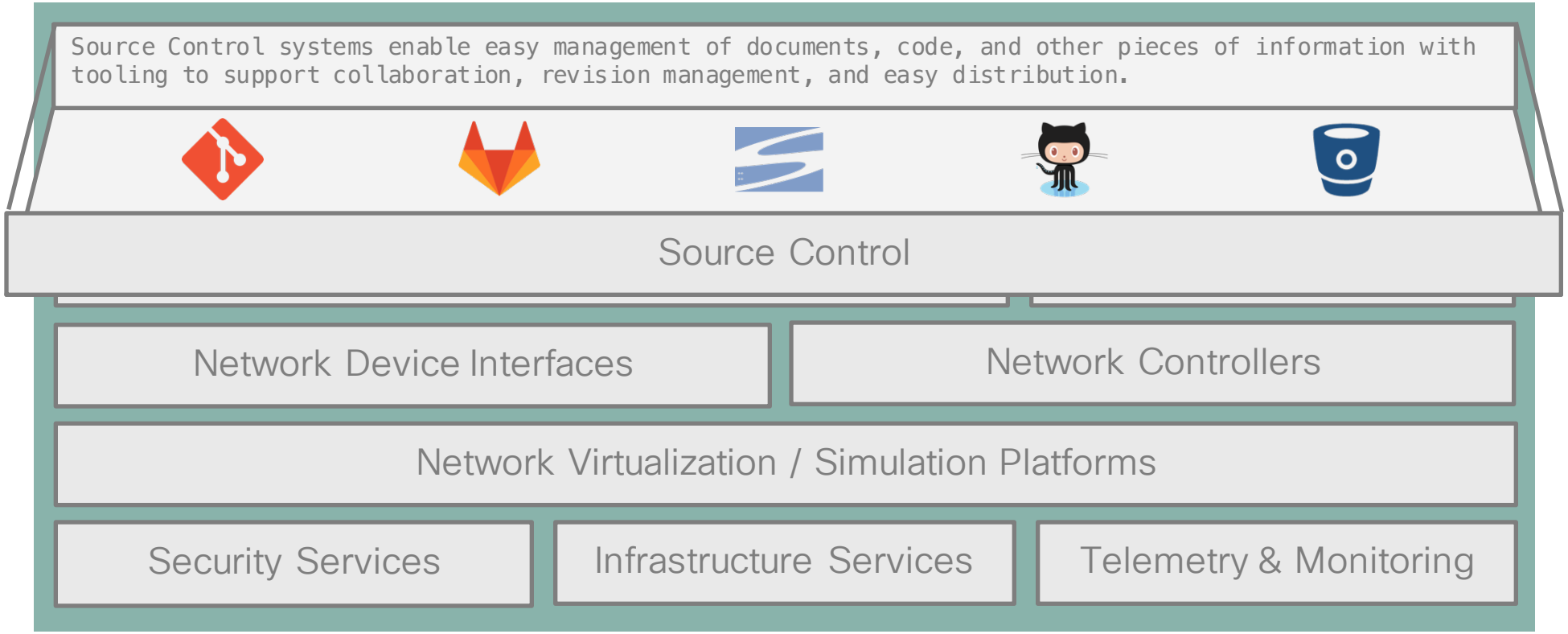


Source Control



Cisco live!

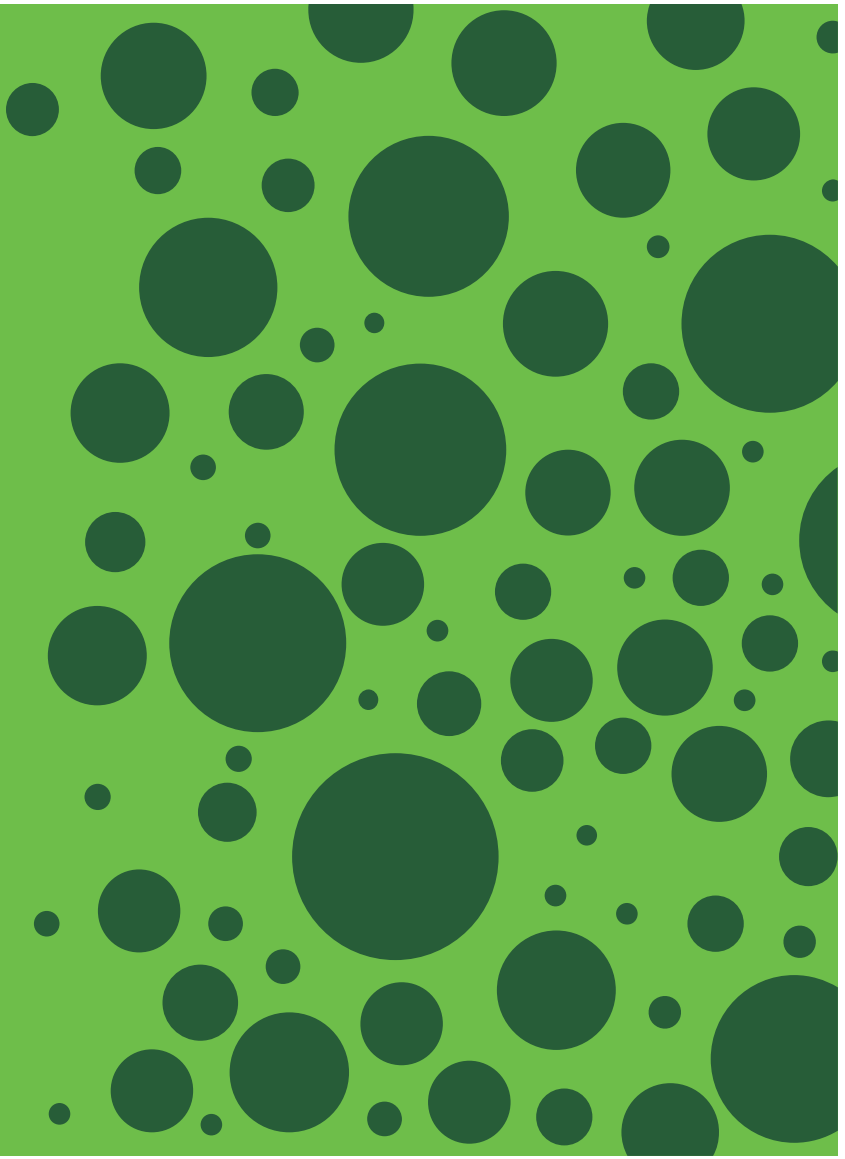
The NetDevOps Engineers Tool Chest



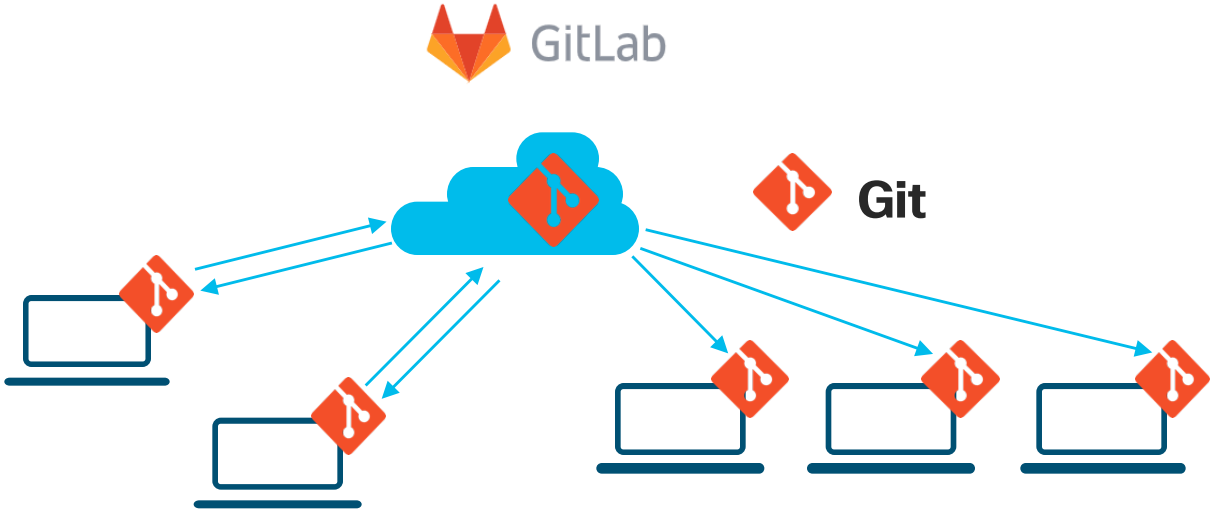
The Need for Version Control

- How do I make incremental changes and share my work with others?
- How do I go back to the version of this file from (yesterday, last week, last year, ...)?
- What changed between version X and version Y of a file?
- People have been making changes to the same file (or set of files)... How do I reconcile and merge all these changes?

Chosen tools:
git and GitLab



Git vs. GitLab



What do I “git” from Github/Gitlab/others?

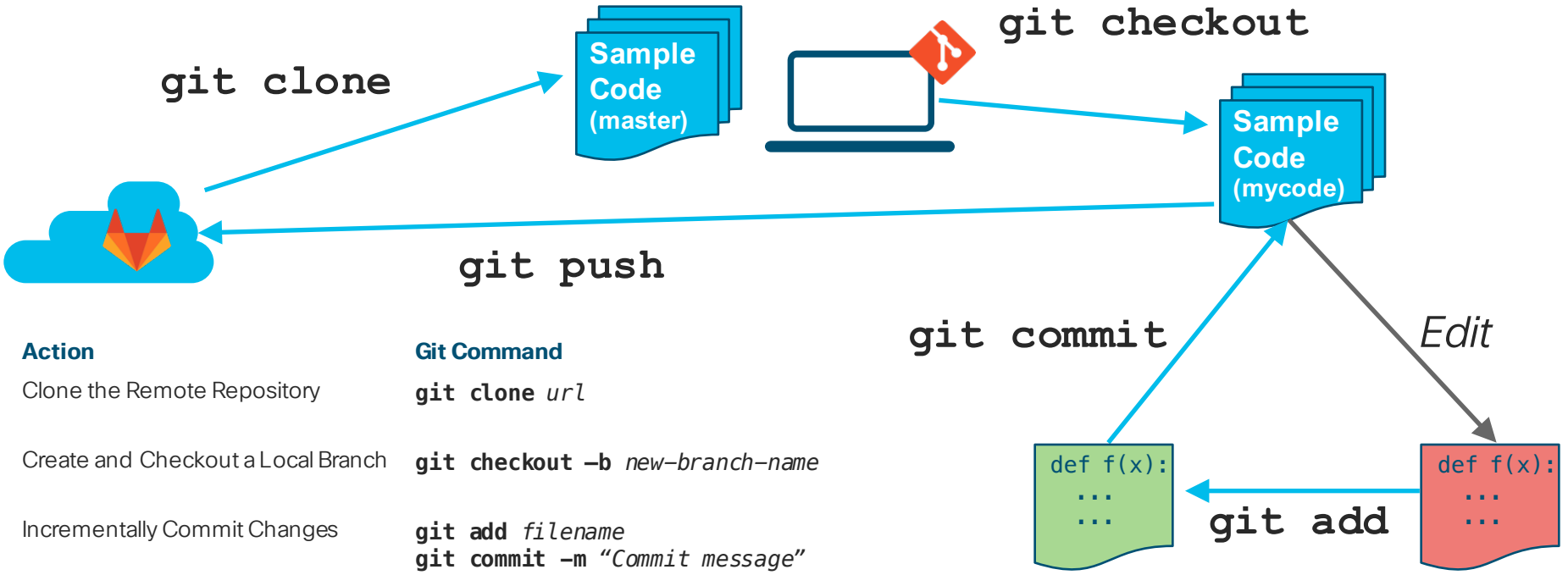
Project Management

- Issues
- Tasks
- Comments/Notes
- CI/CD integrations

Code Review

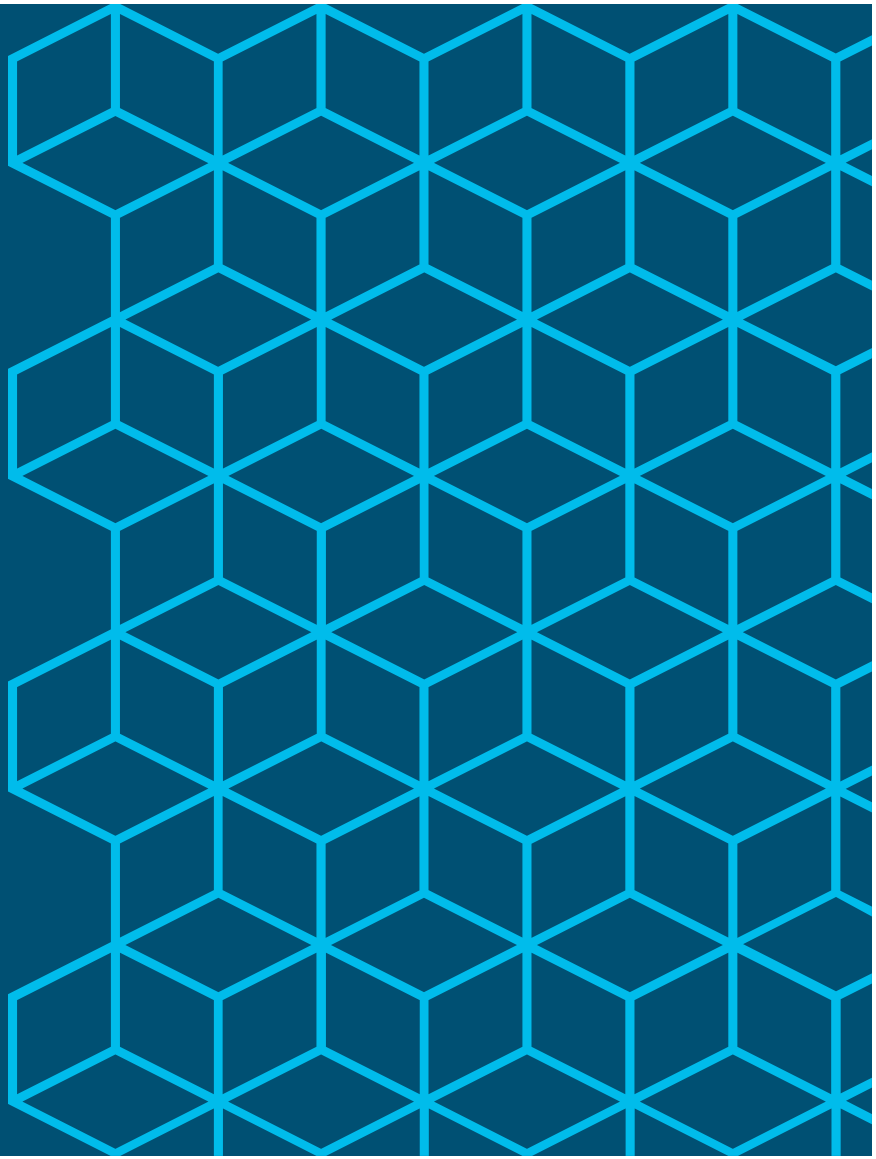
- Propose Changes – Pull/Merge request
- Request Review/Manage Feedback
- See Diff
- Branch management

NetDevOps Sample-Code Workflow



Step	Action	Git Command
1.	Clone the Remote Repository	**git clone** *url*
2.	Create and Checkout a Local Branch	**git checkout –b** *new-branch-name*
3.	Incrementally Commit Changes	**git add** *filename* **git commit –m** "*Commit message*"
4.	Push changes to Remote Repository when ready.	**git push**

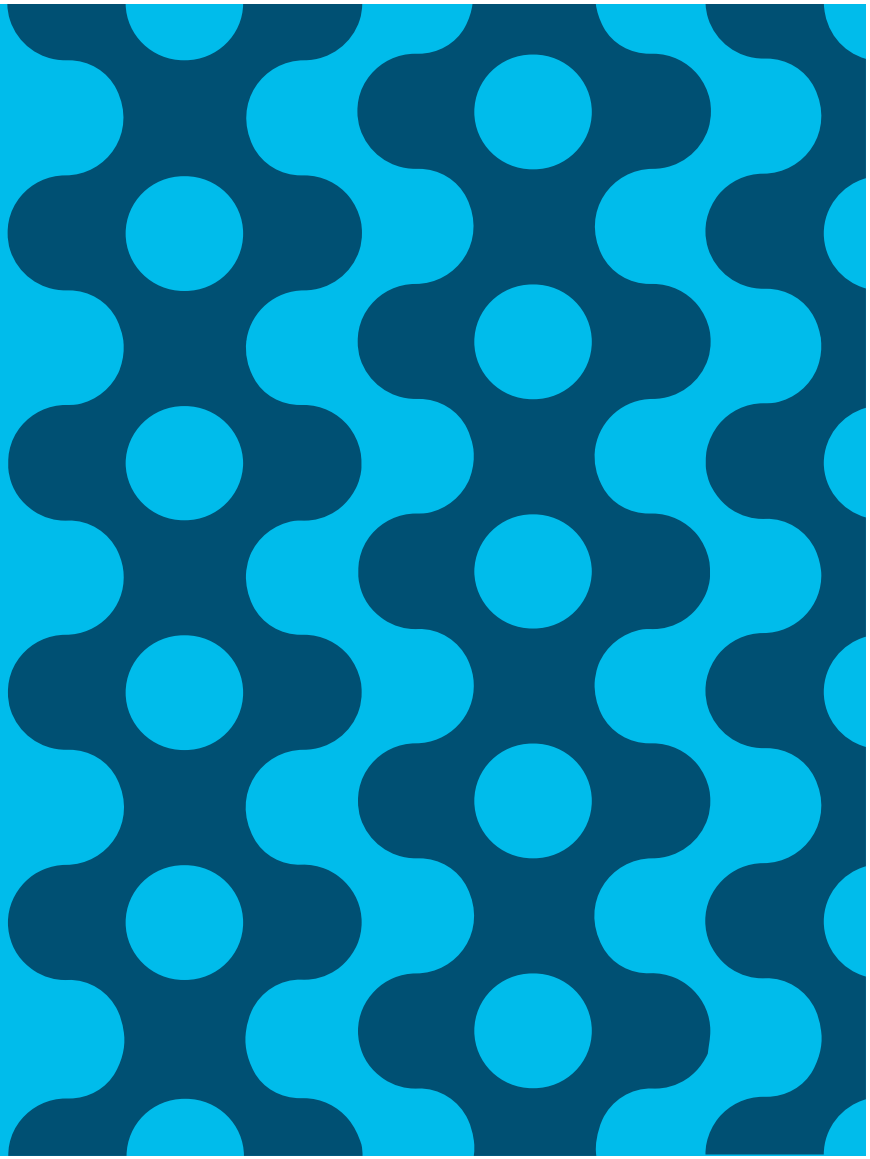
Demo



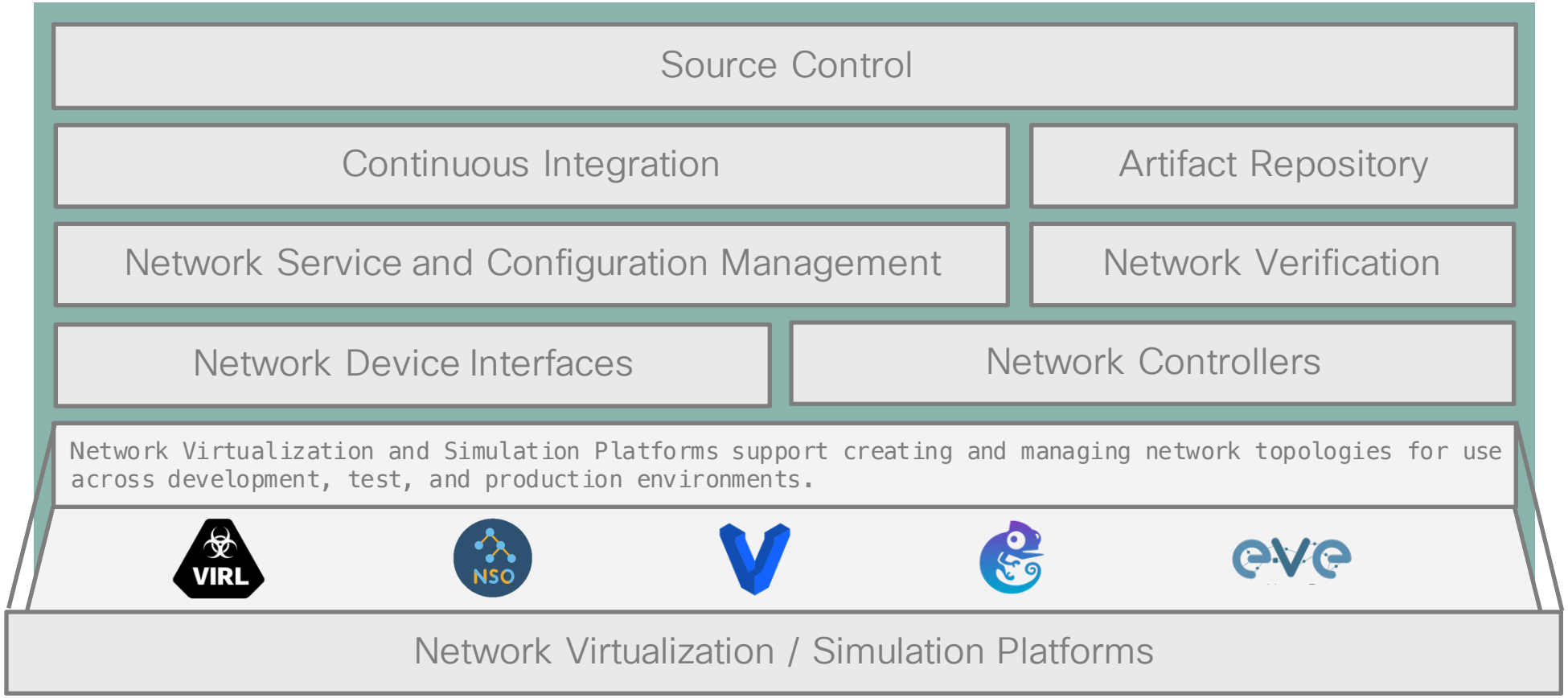
Useful Git Commands

Setup	Tell git who you are *one-time setup*	`git config --global user.name "your name"` `git config --global user.email your@email.com`
Clone	Clone ("download") a git repository	`git clone url`
Fetch	Fetch branch(es)/tag(s)	`git fetch repository`
Status	Check the Status of your local repository	`git status`
Checkout A Branch	Create and Checkout a local **Branch** Creates a "safe place" for your changes	`git checkout -b new-branch-name`
Add	Add a file to your next commit.	`git add filename`
Commit	Commit your changes.	`git commit -m "Your commit message."`
Push	Push changes to remote (central server)	`git push`
Merge	Combine changes from branch(es)	`git merge`

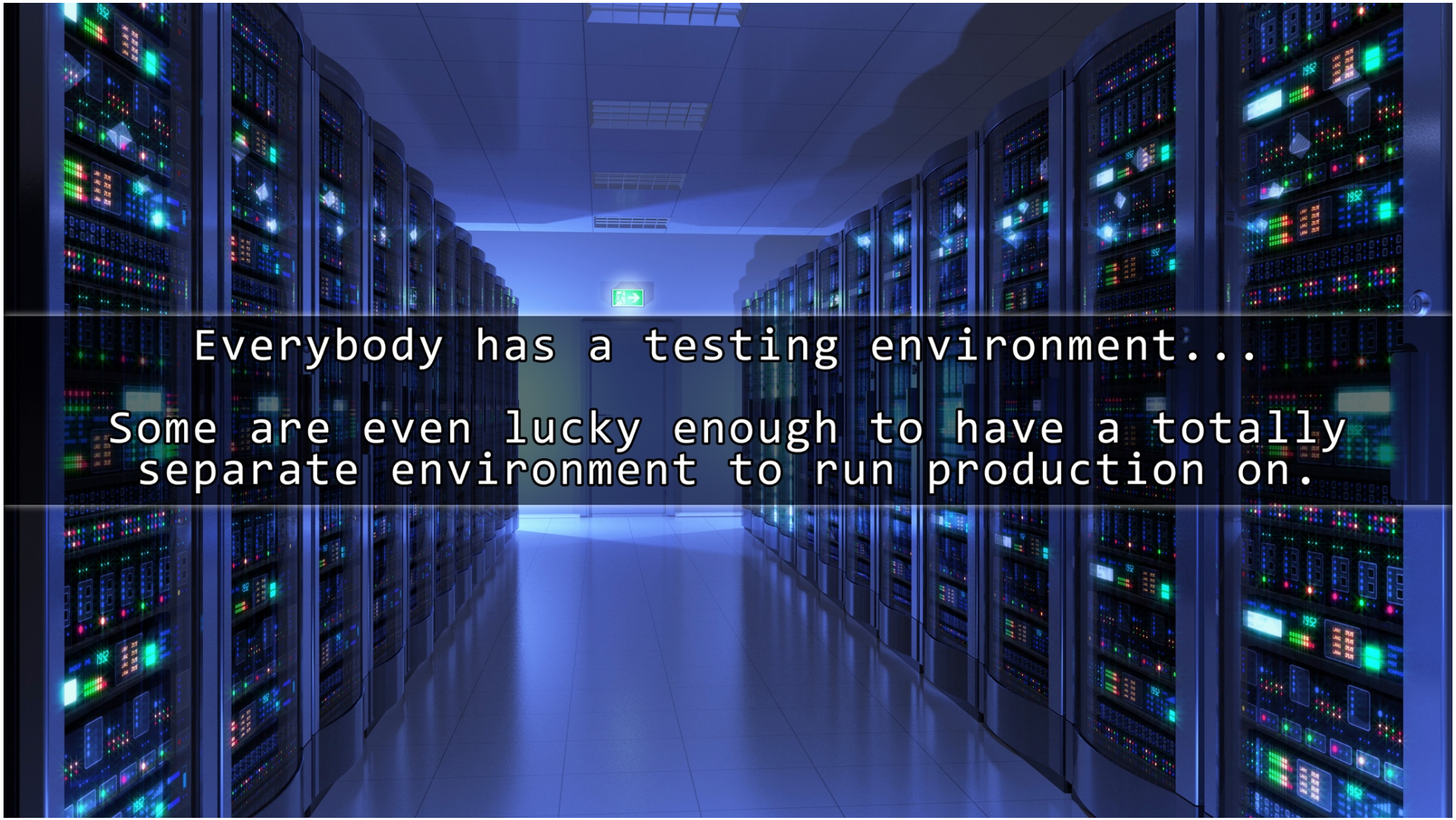
Environment Simulation

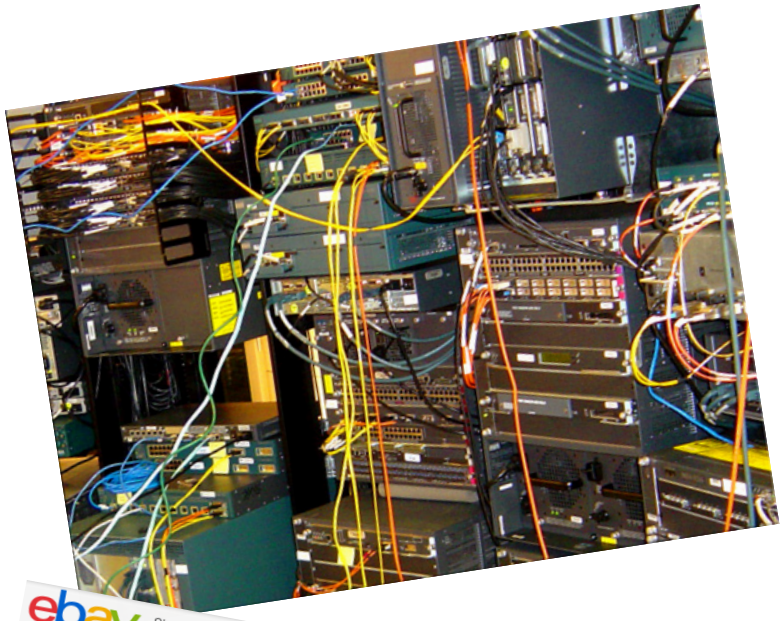


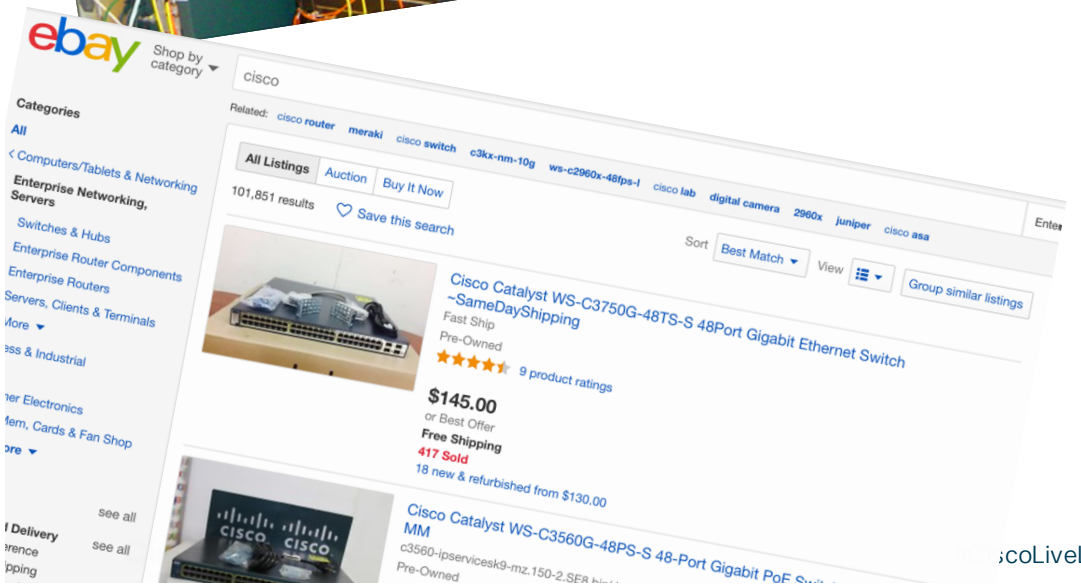
The NetDevOps Engineers Tool Chest



DevOps Environment Hierarchy


Everybody has a testing environment...
Some are even lucky enough to have a totally separate environment to run production on.



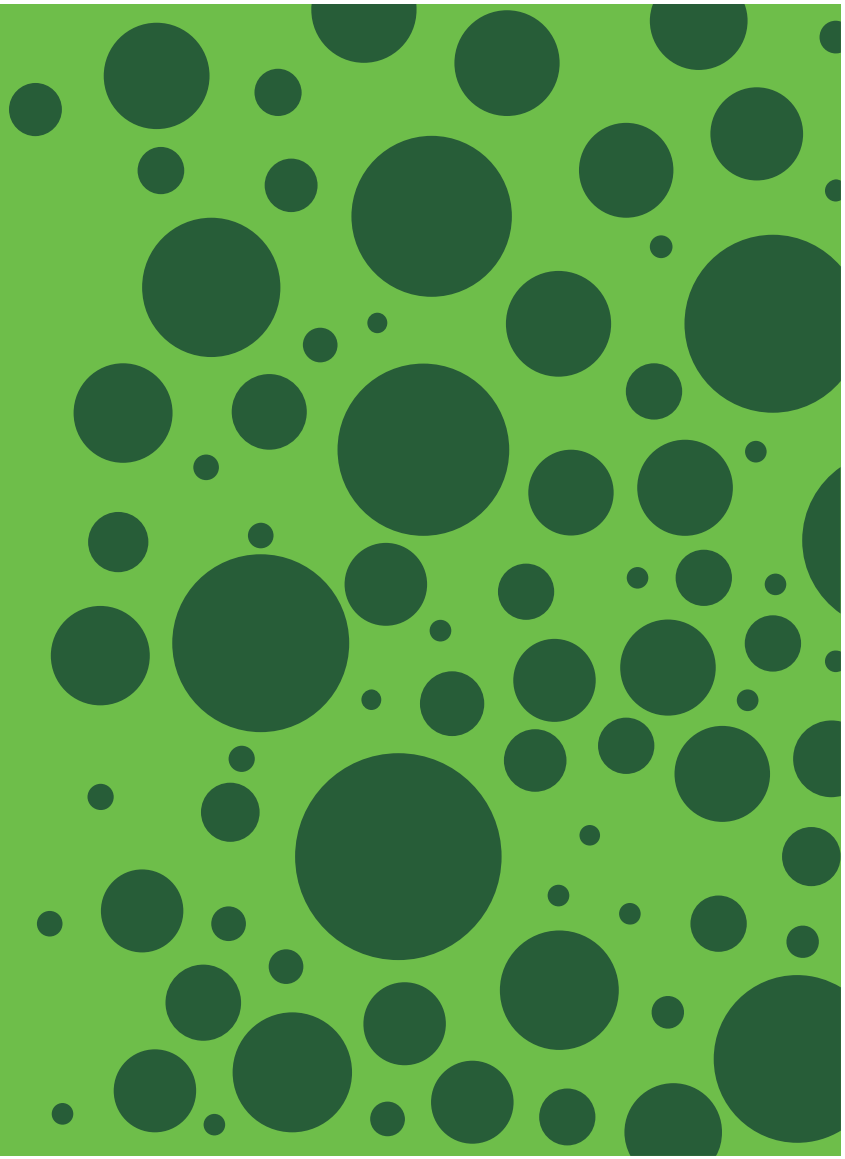


What do we do...

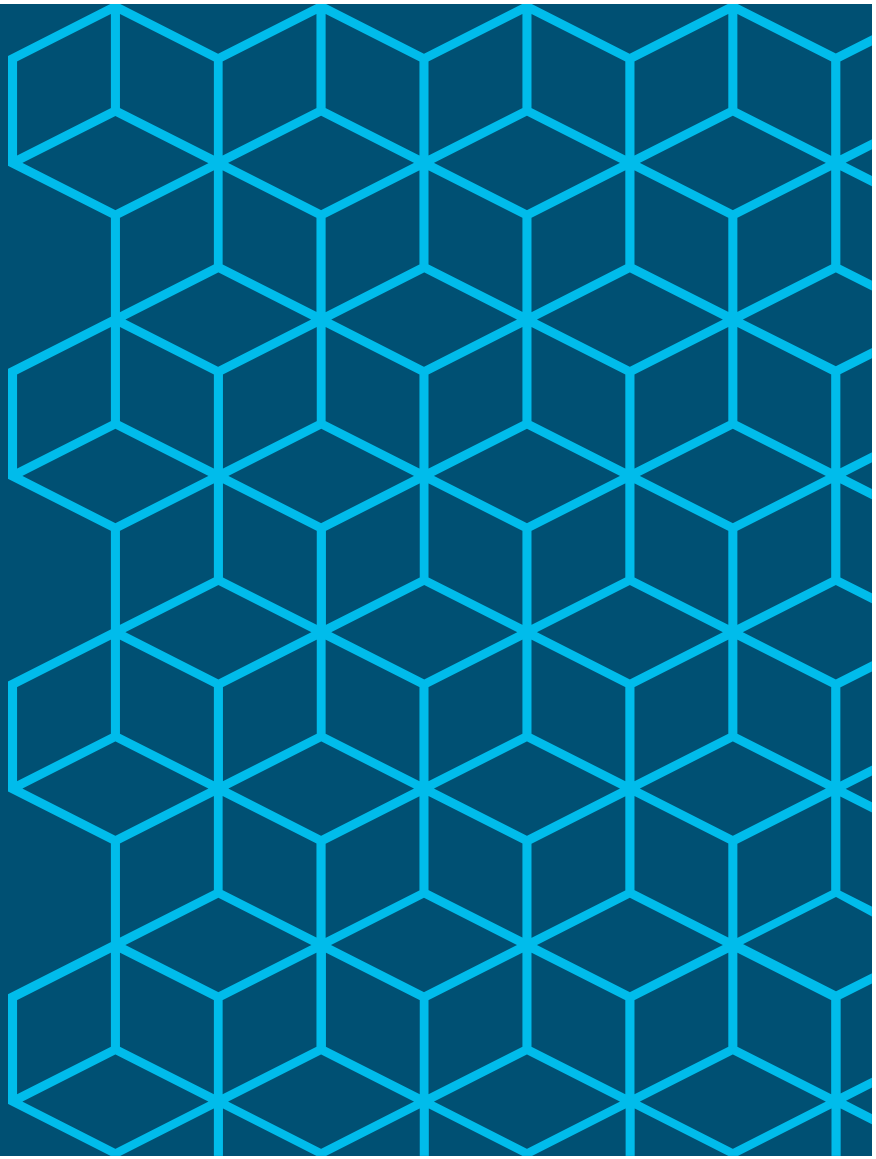
Tool selection criteria

- As close to production parity simulations
- Management, control, and data plane
- Include application workloads
- Vendor supported images
- Programmable and “Simulation as Code”

Chosen tool:
Cisco VIRL/CML



Demo



Cisco VIRL/CML

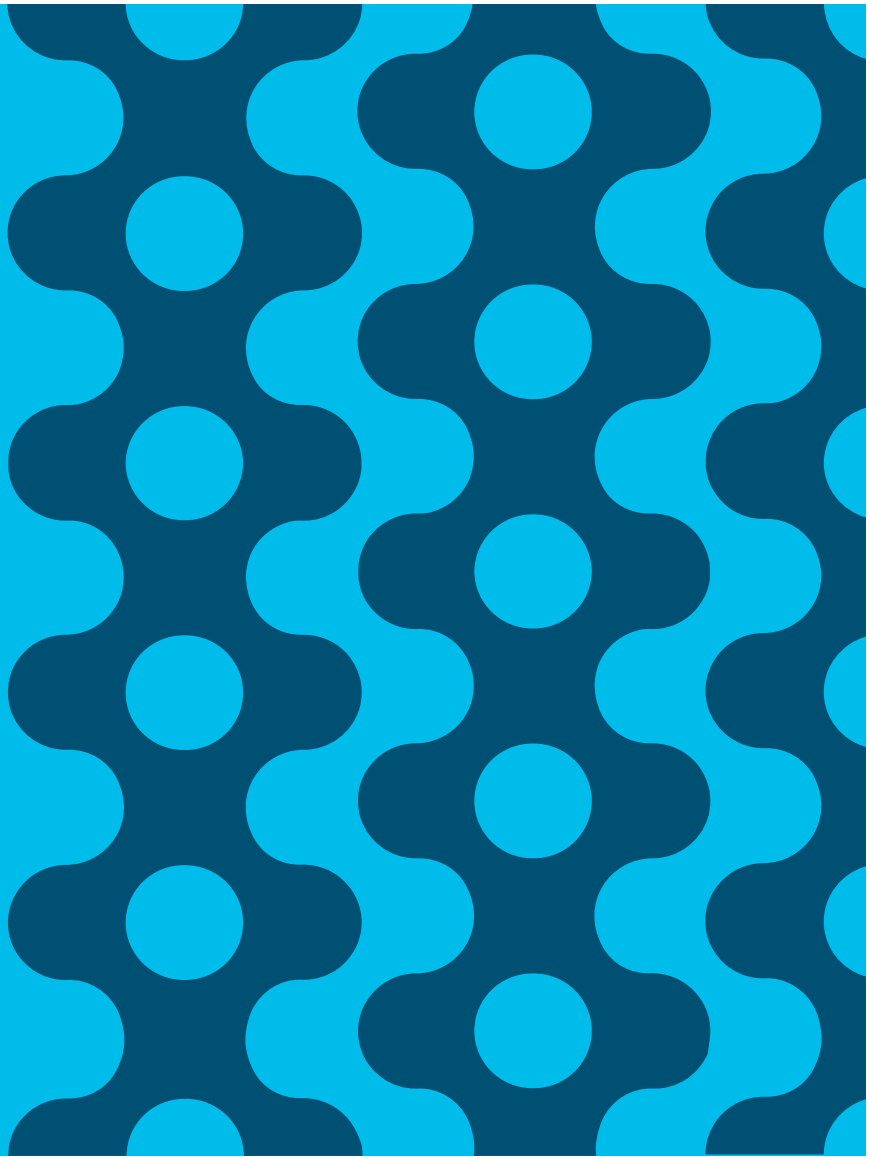
Dev and Test Environments NetDevOps Engineers Need TODAY!

- Enterprises are demanding increased reliability, this will mean Dev > Test > Prod
- The simulations in VIRL support multi-vendor, complex topologies that can mimic production, including apps
- Combine with “virlutils” and it’s the NetDevOps Engineers ideal toolset

```
(std) test\ $ virl nodes
Here is a list of all the running nodes
```

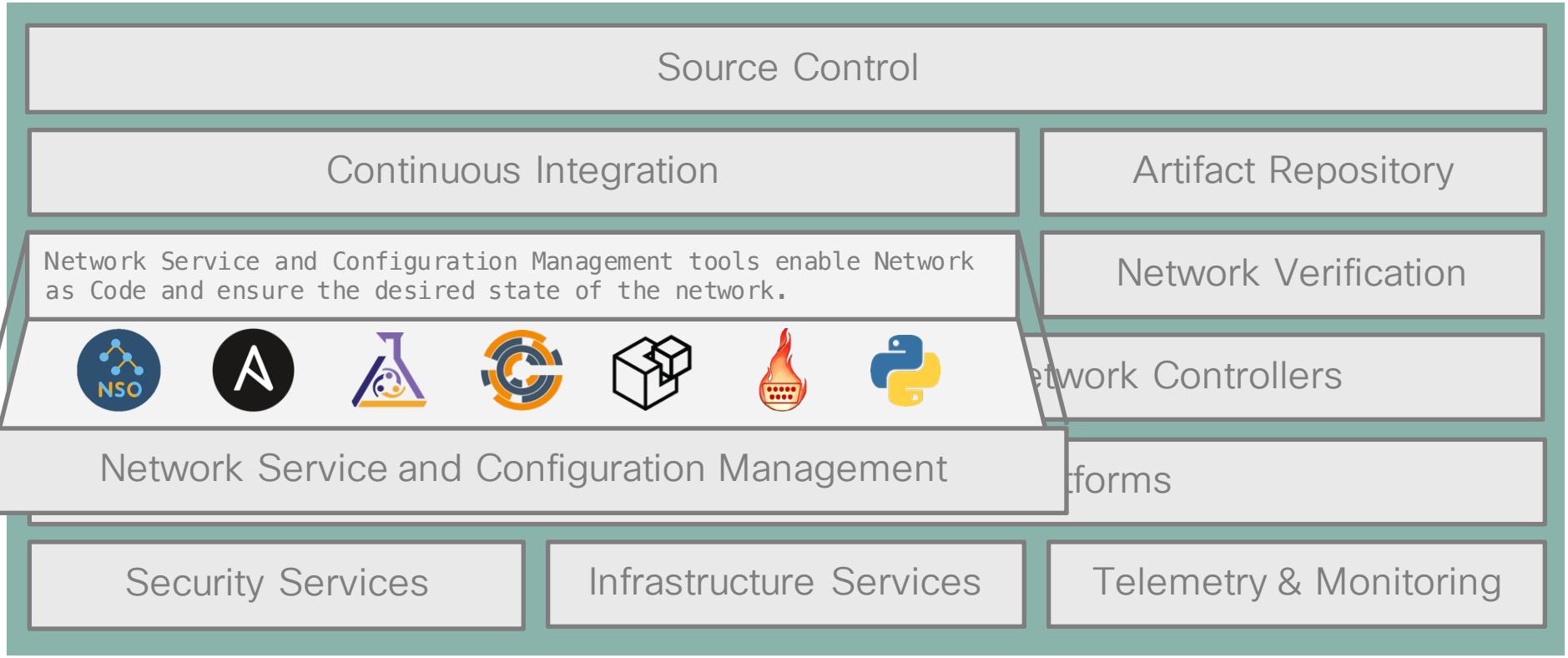
Node	Type	State	Reachable	Protocol	Manage
agg3	NX-OSv 9000	ACTIVE	REACHABLE	telnet	10.94.
agg4	NX-OSv 9000	ACTIVE	REACHABLE	telnet	10.94.
branch10-host11	server	ACTIVE	UNREACHABLE	ssh	10.94.
branch10-router10	CSR1000v	ACTIVE	REACHABLE	telnet	10.94.
core1	IOS XRv 9000 - extra mem	ACTIVE	REACHABLE	telnet	10.94.
core2	IOS XRv 9000 - extra mem	ACTIVE	REACHABLE	telnet	10.94.
leaf5	NX-OSv 9000	ACTIVE	REACHABLE	telnet	10.94.
leaf6	NX-OSv 9000	ACTIVE	REACHABLE	telnet	10.94.
route-generator	lxc-routem	ACTIVE	REACHABLE	ssh	10.94.
server7	server	ACTIVE	REACHABLE	ssh	10.94.
server8	server	ACTIVE	REACHABLE	ssh	10.94.
tgen10-4	lxc-ostinato-drone	ACTIVE	REACHABLE	ssh	10.94.
tgen5-1	lxc-ostinato-drone	ACTIVE	REACHABLE	ssh	10.94.

Configuration Management



Cisco live!

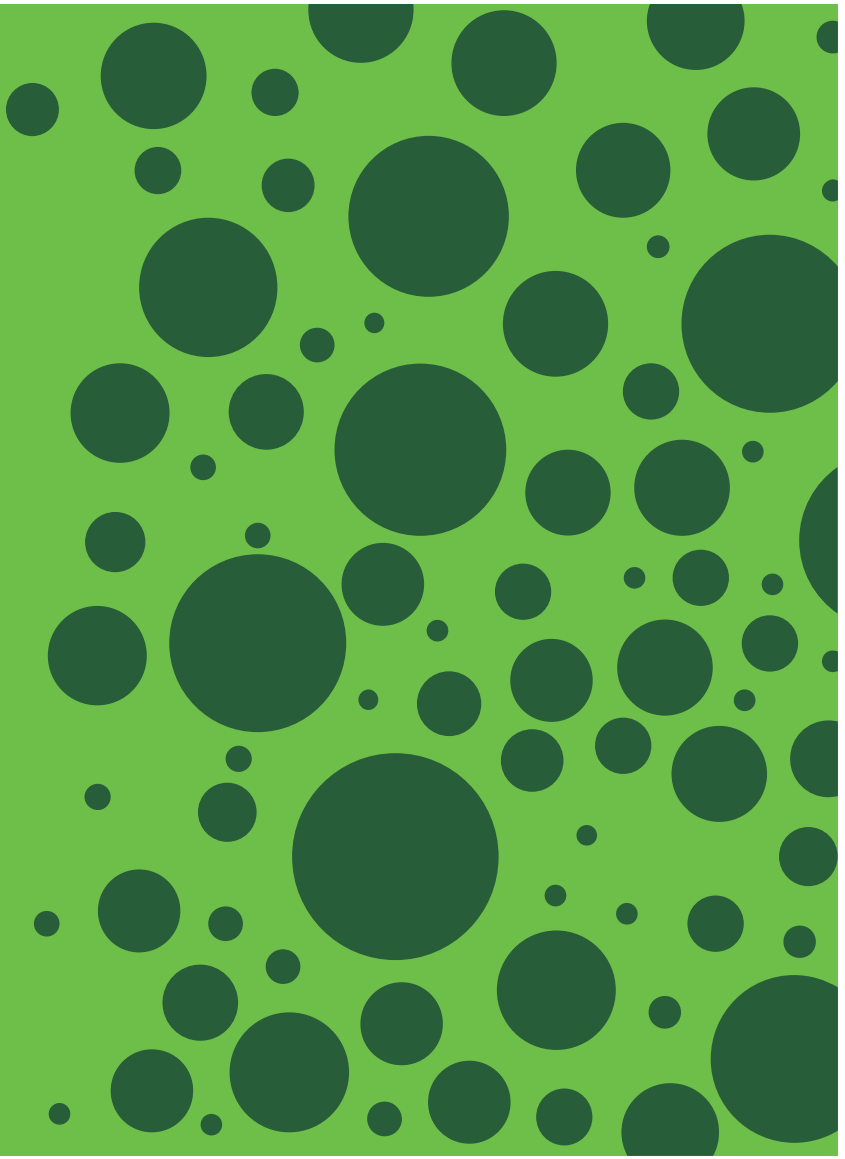
The NetDevOps Engineers Tool Chest



Tool selection criteria

- Robust support for network configuration management uses cases
- Device AND service management
- Configuration validation
- Large heterogeneous network topologies
- High performance

Chosen tools:
Cisco NSO + Ansible



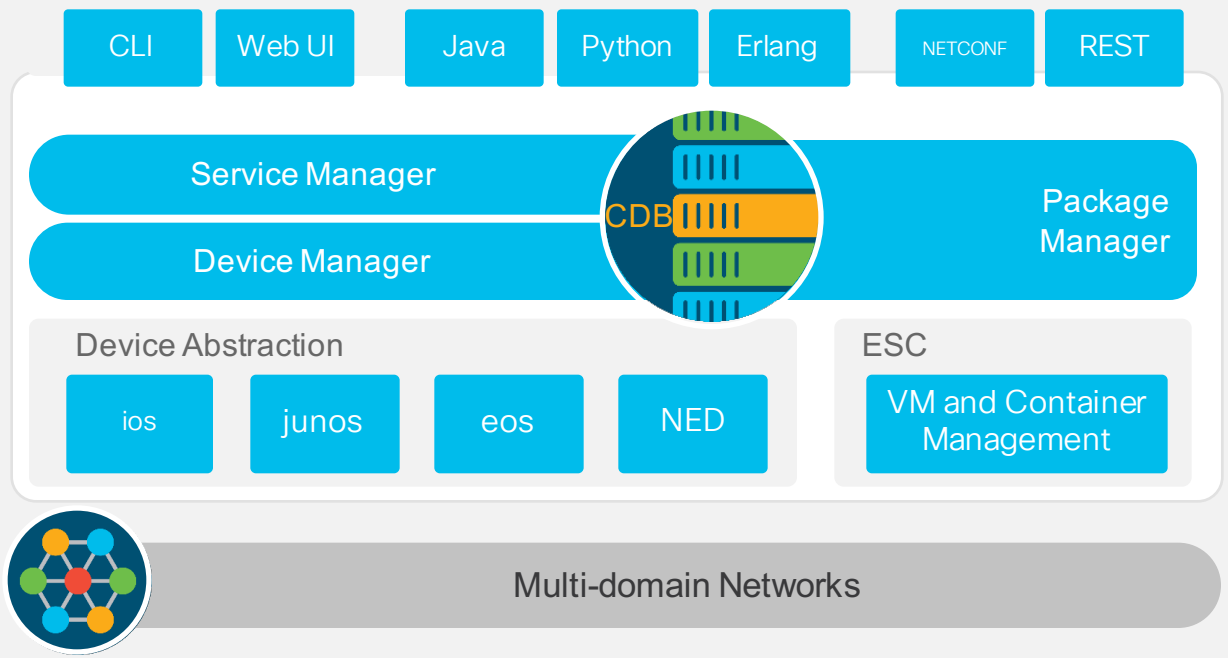
Key aspects of this tool

- **Integration** with other systems
  - North, South, East, West
- No device changes required – **brownfield**
- Flexibility to map into **many workflows**
  - Portals, ITSM, BSS, OSS
  - Other configuration management tools – **Ansible!**
- **Developer Experience!**

https://developer.cisco.com/site/nso/

Get started with Cisco NSO on your laptop or in your lab environment today

Cisco NSO - Architecture



- **Model-driven**, end-to-end service lifecycle and **customer experience focused orchestration**
- **Application Server** which runs applications in the form of **packages**
- **Seamless** integration with order managers and ITSM systems
- Loosely-coupled and **modular architecture leveraging open APIs** and **standard protocols**
- Orchestration across multi-domain and multi-layer for **network-wide**, centralized **policy** and **services**

Broadest Multivendor Support

Over 100 Supported NEDs

Packages

The fundamental building block of NSO

- Three primary package types
  - Network Element Driver (NED) – model of network device + way to manage it
  - Services – defines something that you do or provide with a network
  - Tools – other packages that don't fall into the above categories (e.g. device discovery)
- Two consumption models
  - Customer Facing Service (CFS) – stuff business people care about
  - Resource Facing Service (RFS) – stuff infrastructure people care about
- Core Function Packs
  - A curated set of packages for a given use case.
    - SD-WAN
    - NFVO
    - Secure Agile Exchange

The Network API and CLI

Fixes these chronic issues:

- Lack of automation, Managing device configuration
- Quality issues in delivery
- Inflexibility to change existing configuration (create and delete only)
- CLI Scripting–inflexible and high fallout

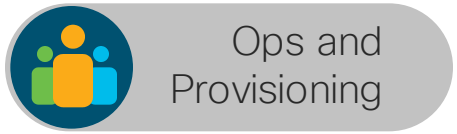
- A YANG-Based Configuration Database
- device manager do all the tedious stuff
  - sync-from/sync-to/compare Devices
- Rich set of Northbound APIs rendered from the database / devices
- Consistent and Network-wide CLI, UI, REST
- Start with CLI but gradually introduce others e.g. Python, REST, for scripting trivial tasks
- Transaction-safe operations and rollback!
- **Device Configuration Management AND Accurate network configuration state**
- Choice of technology up to the **consumer**!!

Choose Your Own Adventure

Automation

Day-to-day management of rapidly growing, complex networks

Solutions
- Compare-Config
- Config Templates
- Config Policies
- Compliance Reports



Customer Experience

Provisions services and manages service quality in networks

Solutions
- Custom actions
- CDB Subscriptions
- Verifications
- Triggers

Time-to-Market

Develops new network services on demand

Solutions
- ITSM / BSS Integration
- Service Models
- Orchestrated Assurance

Network API
Utilize a single interface to all network devices

Operations Abstraction
Leverage one central API for all services

Transformation
Develop your network services catalog

What is a service?

Customer Facing Service

"Top Down"
a.k.a The Overlay

- MPLS L3 VPN
- Data Center VLAN
- Branch Deployment
- VXLAN Tenant

Resource Facing Service

"Bottom Up"
a.k.a The Underlay

- PE Router
- Distribution Switch
- Data Center VXLAN Fabric
- DMZ/PCI/STIG Device
- SNMP

Service Package Components

Separation of Concerns

- YANG – defines the services data model, API, CLI, and constraints in high level terms.
- Logic – (**optional**) python/java code which collects additional information (variables), performs verification on instances
- Templates – how the service is rendered on one or more device/types
- package-meta-data.xml – minimum required NSO version, package dependencies, components provided

Config, **Deploy**, **Validate w/ organizational, standards, process, best Practices**

Our VXLAN Fabric Service

```
vxlan-fabric fabric1
 pim-pool      100.2.1.0/24
 loopback-pool 10.2.1.0/24
 interior-pool 172.16.0.0/16
 spines spine1
  connections leaf1
   interface 1/1
  !
  connections leaf2
   interface 1/2
  !
  connections leaf3
   interface 1/3
  !
  connections leaf4
   interface 1/4
  !
 !
```

- Resource Allocation
- Service Birth Certificate
- Verfication / Sync API's
- Subscriptions

```
admin@ncs# vxlan-fabric fabric1 verify
result Spine spine1, OSPF status: 4/4, BGP STATUS: 4/4 - READY
Spine spine2, OSPF status: 4/4, BGP STATUS: 4/4 - READY
Leaf leaf1, OSPF status: 2/2, BGP STATUS: 2/2 - READY
Leaf leaf2, OSPF status: 2/2, BGP STATUS: 2/2 - READY
Leaf leaf3, OSPF status: 2/2, BGP STATUS: 2/2 - READY
Leaf leaf4, OSPF status: 2/2, BGP STATUS: 2/2 - READY
SUMMMARY: BGP AND OSPF OK

ready true
admin@ncs#
```

Our VXLAN Tenant Service

Who uses it and for what?

YANG Model

```
import vxlan-fabric {
  prefix vxlan-fabric;
}

list vxlan-tenant {
  key name;
  leaf name {
    tailf:info "Unique service id";
    tailf:cli-allow-range;
    type string;
  }

  leaf fabric {
    type leafref {
      path "/vxlan-fabric:vxlan-fabric/vxlan-fabric:name";
    }
    mandatory true;
  }
```

Appearance
what is its abstraction, what is its API, how can it be used?

Logic

```
@Service.create
def cb_create(self, tctx, root, service, proplist):
    self.log.info('Service create(service=', service._path, ')')
    # Allocate identifiers
    vniis = self.allocate_vniid(root, service, tctx.username)
    self.log.info('vniids: {}'.format(vniis))
    vlans = self.allocate_vlanid(root, service, tctx.username)
    self.log.info('vlans: {}'.format(vlans))
    # Ready to push config
    vars = ncs.template.Variables()
    # Tenant specific settings
    vars.add('VRF', service.name)
    # Hardcoded values
    vars.add('BGP_AS', '65001')
    vars.add('NVE_ID', '1')
    template = ncs.template.Template(service)
```

Meaning + Semantics
what does it provide, what does it consume?

Templates

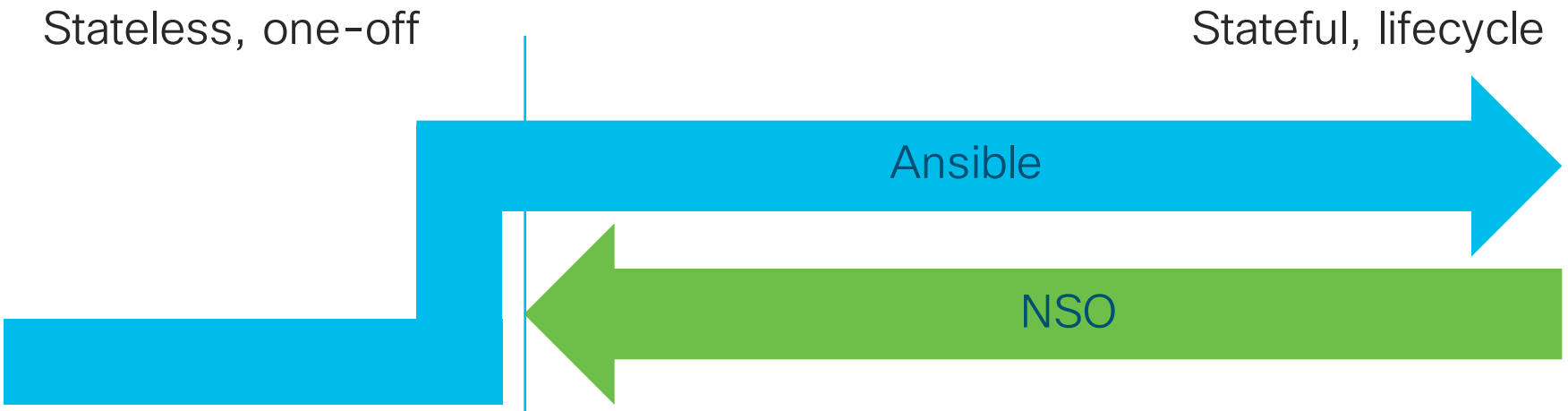
```
<vlan xmlns="http://tail-f.com/ned/cisco-nx">
    <vlan-list>
        <id>{$VLAN_ID}</id>
        <vn-segment>{$VNI_ID}</vn-segment>
    </vlan-list>
</vlan>
<evpn xmlns="http://tail-f.com/ned/cisco-nx">
    <vni>
        <id>{$VNI_ID}</id>
        <l2/>
        <rd>auto</rd>
        <route-target>
            <method>import</method>
            <rt>auto</rt>
        </route-target>
        <route-target>
            <method>export</method>
            <rt>auto</rt>
        </route-target>
    </vni>
</evpn>
```

Manifestation
How is this implemented?

Model granularity

- Granularity is chosen to:
  - reduce the complexity of each model
  - allow different teams to own different models
  - allow independent life-cycles
- One top model describes the customer facing service
- Lower models describes intermediate resource facing services
- Auxiliary models might be used as needed

Hammers and Screwdrivers



- Update SSH keys
- NTP/AAA/DNS
- Static day 1 config

- API-driven (CRUD) interactions
- Network services (l3vpn, service chains)
- Full lifecycle

Strengths

Ansible

Key rollover

SNMP Community

Install Application

Single-shot config

Patch applications

NSO

Audit configuration

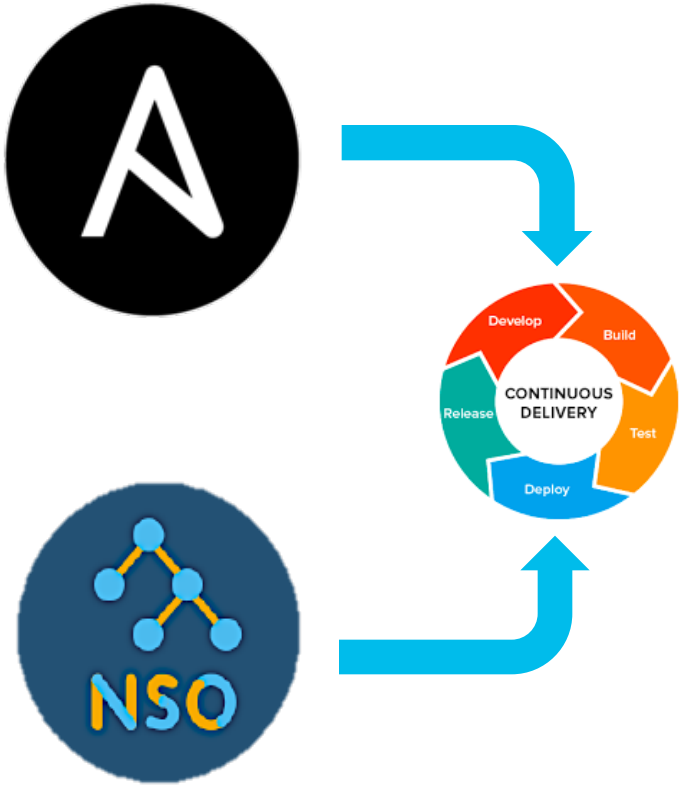
Transactions / Rollback

Multi-device services

Minimal diff production

Service reconciliation

Roles and Responsibilities



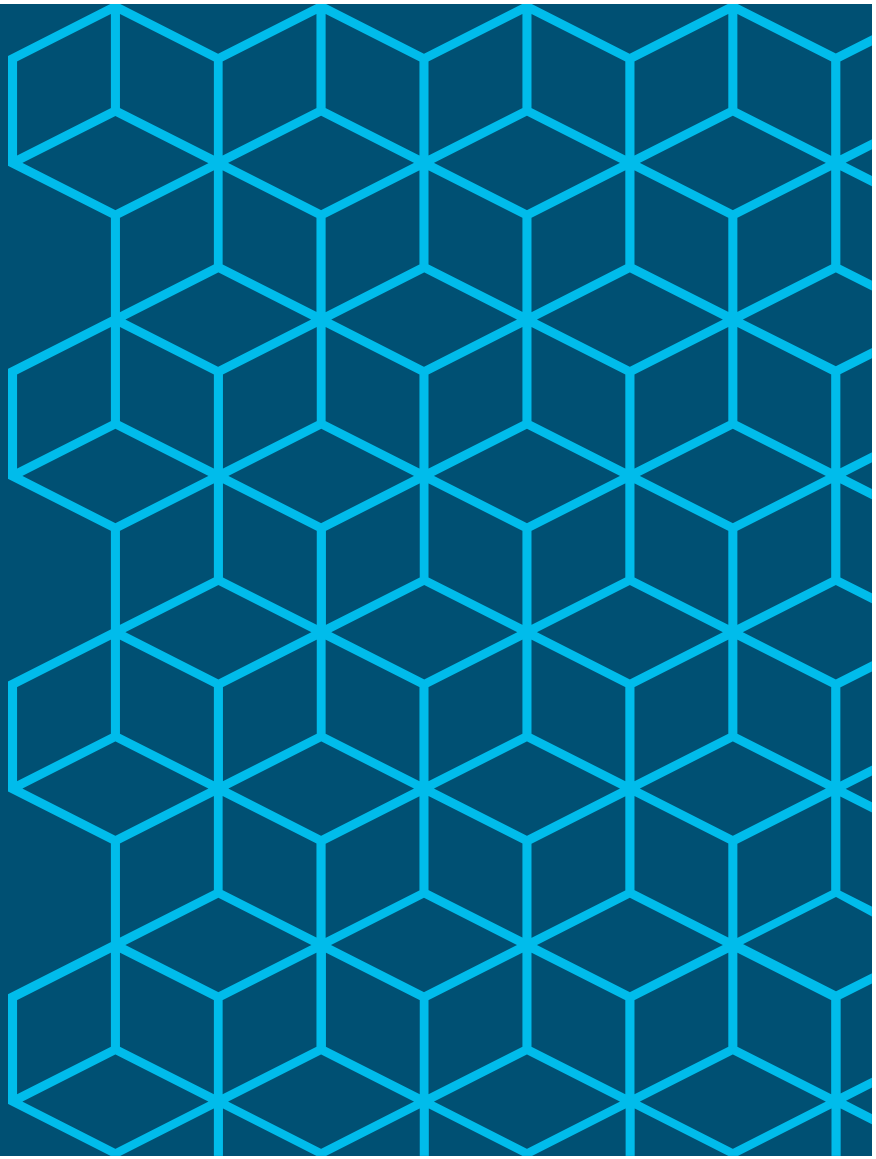
DevOps teams
- Owns lifecycle of playbooks

YANG becomes contract language between teams across infrastructure cycles:
- Network Requirements from apps provided in YAML-format
- New services published as REST-interface update (versioned)

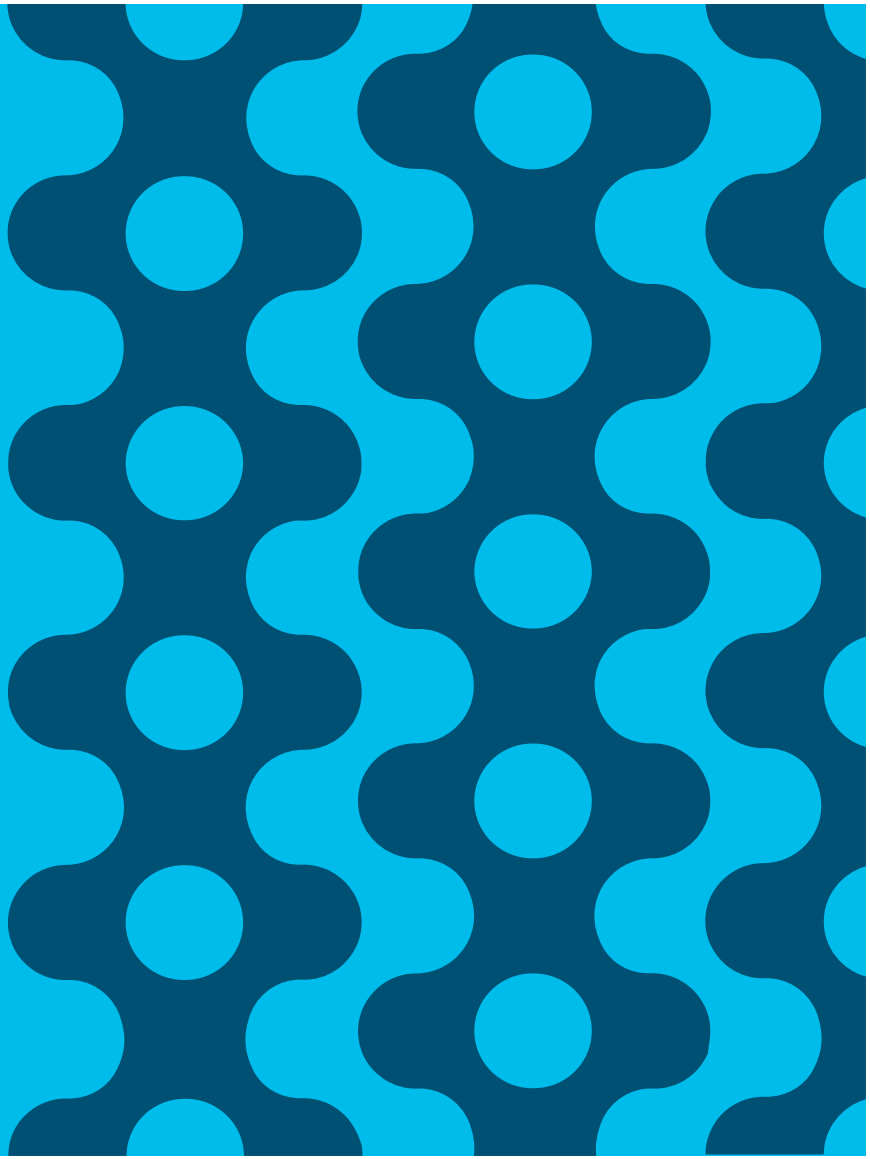
NetDevOps teams:
- Owns lifecycle of network services

Demos

- Rollback Tenant Deploy
- Generate “playbooks”
- Network Hygiene

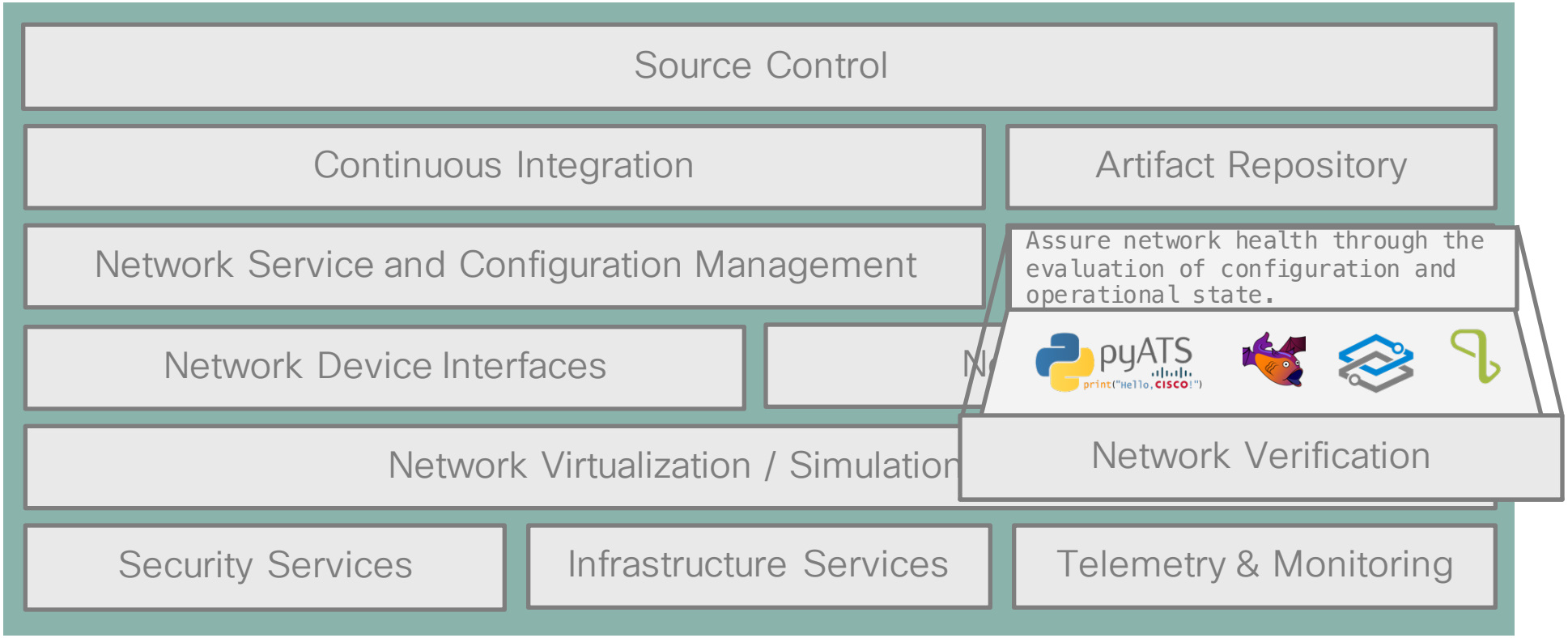


Network Verification



Cisco live!

The NetDevOps Engineers Tool Chest



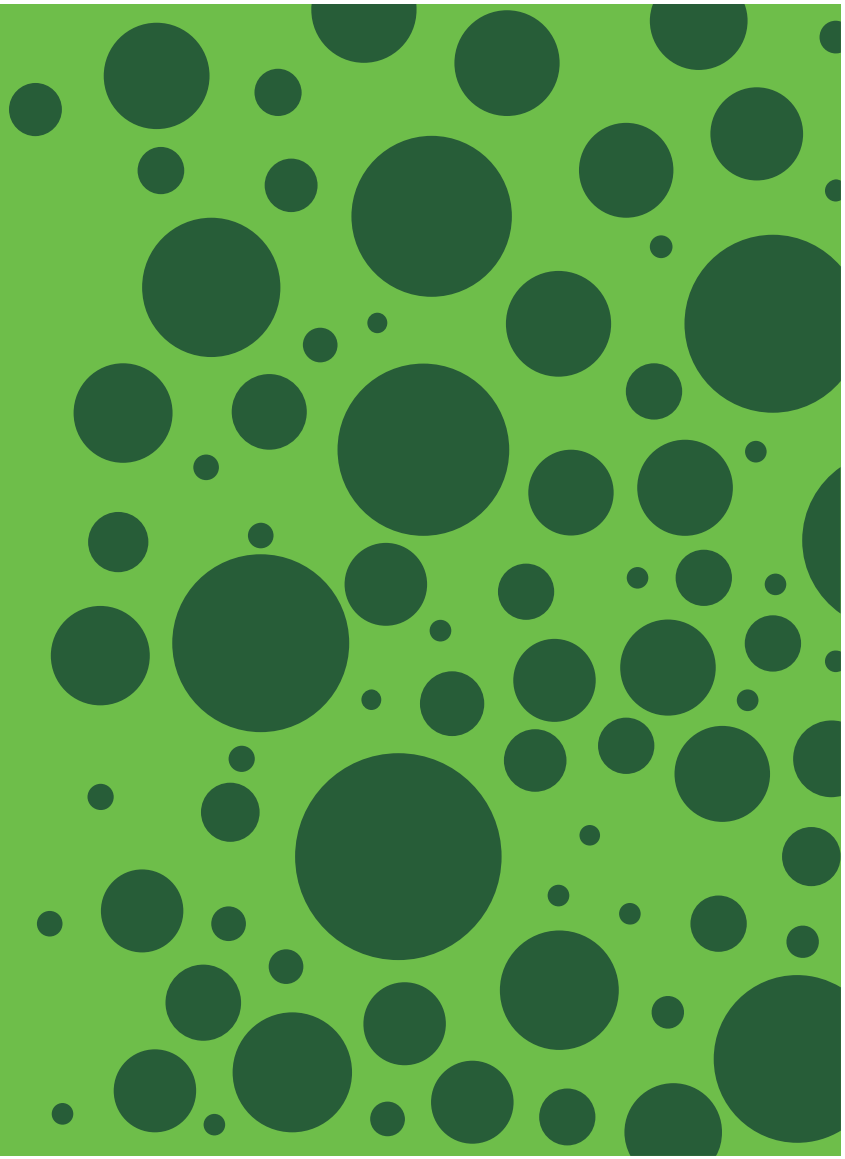
Tool selection criteria

- **configurational** and **operational**
  - **offline** or **online** analysis
- **what-if?**
  - a packet walked into a network…
  - "that link" failed?
- **simple** and **extensible**
  - are all links up?
  - was the change successful?
  - will the change be successful?
- **trigger failures** and **recover from them**

Desired State Actual State Great Success!

User friendly **abstractions** for ChatOps and Natural Language Processing

Chosen tool:
pyATS and Genie



Key aspects of this tool

- Powerful **network testing / verification** ecosystem
- Extensive library of **pre-built CLI parsers**
  - generate new parsers where needed
- Wide range of **reporting** options
  - CLI logs -> Pipeline Friendly -> Web Dashboard
- Integrate with **3rd party** python libraries
  - traffic generators/packet captures/web page clicks

`genie.ops`
`show`

`genie.sdk`
`clear ip bgp`
`show ip bgp`

`genie.conf`
`conf t`

pyATS, 2014 - present

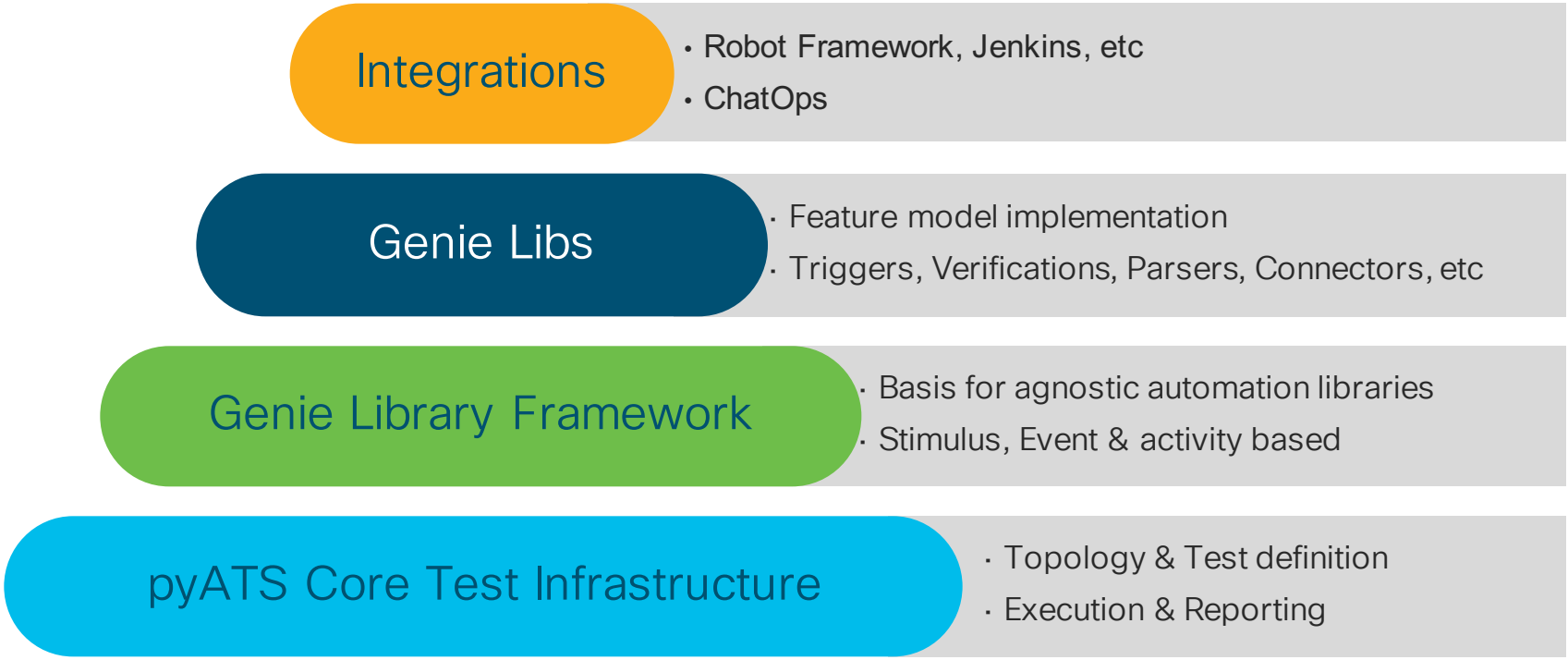
- Launched internally in Cisco engineering late 2014
- Quickly became the most adopted test framework within Cisco
- Running in sanity, regression, solution labs, etc.

3000+ Internal Engineers/consumers

5,000,000+ LoC Testcases/Scripts

1,000,000+ runs / month

pyATS Framework



pyATS - The toolbox

- pythonic infrastructure & object-oriented programming paradigm
- providing the most fundamental, common toolset needed by everyone
- Agile software development methodology
- Capable of leveraging existing tools and libraries

https://developer.cisco.com/docs/pyats/

"There's a difference between knowing the path, and walking the path." - Morpheus

Genie High Level Features

`genie.conf`

```
conf t
```

`genie.ops`

```
show
```

`genie.sdk`

```
clear ip bgp
show ip bgp
```

Provides *feature-centric* object models

- Focuses development effort on writing test cases & suites
- Shields the end scripter from explicit CLI/YANG-RPCs

Objects are *agnostic*

- Works across management interfaces: CLI, YANG, XML, etc.
- Handles feature differences between images, releases, platforms, etc.

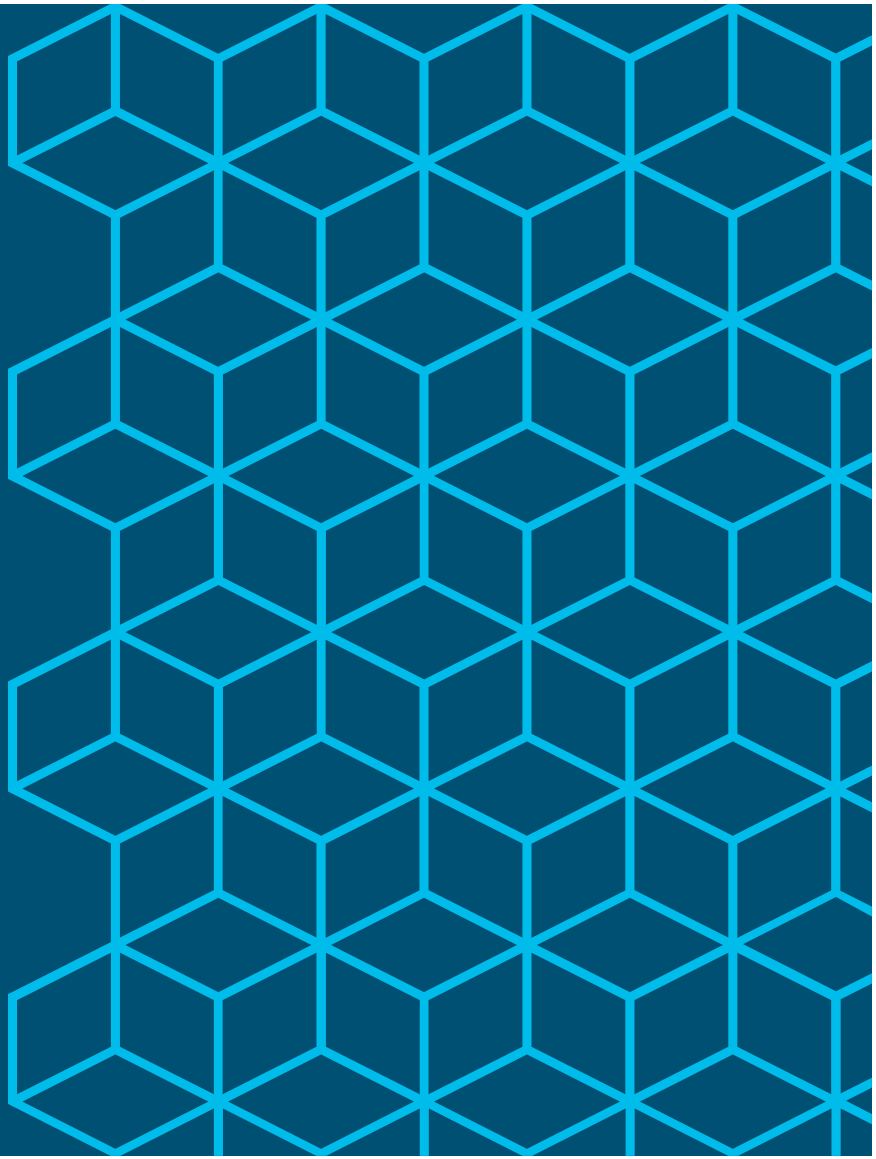
Genie is *plug & play*

- Use only the classes you need
- SDK's triggers and verifications **plug directly into pyATS as test cases and sections**

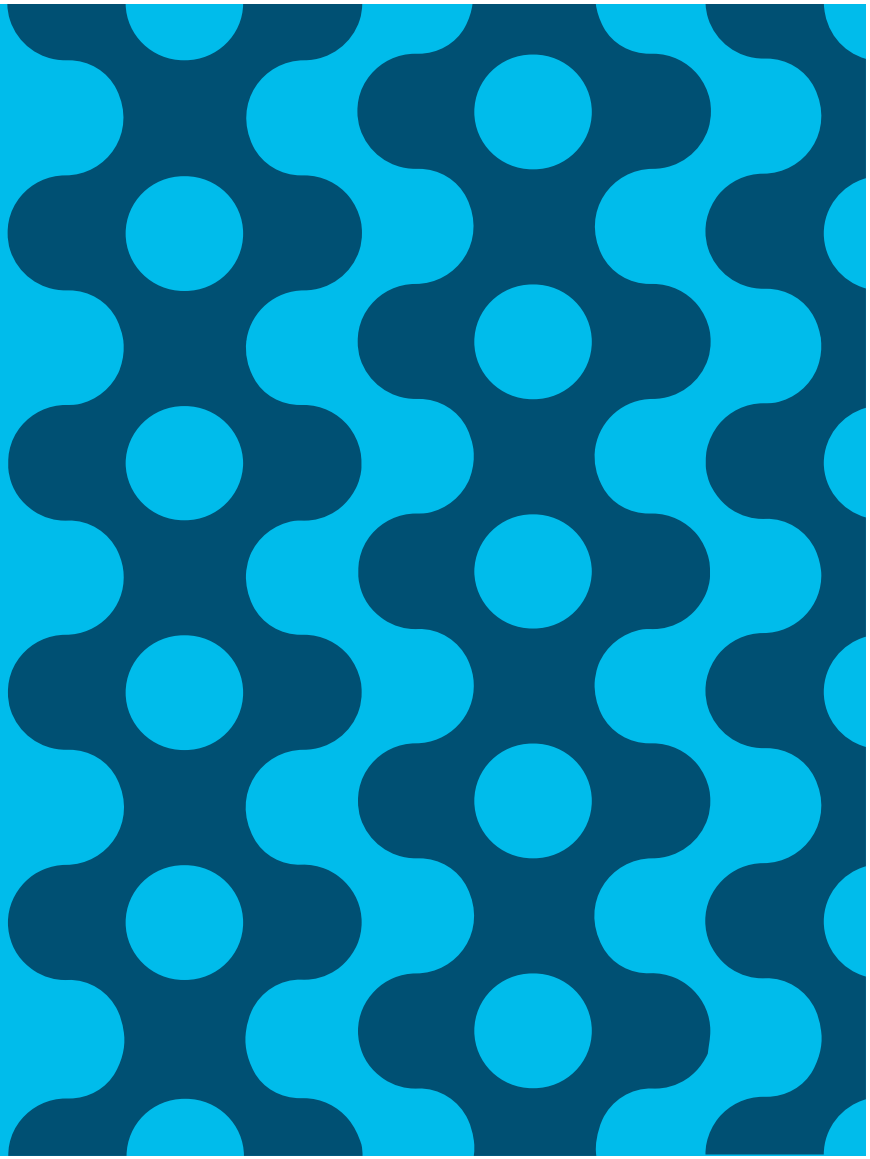
Genie is *extensible*

- Inherent & extend whenever needed
- Modify only what's required & accommodate for deltas between release/image/etc.

Demo

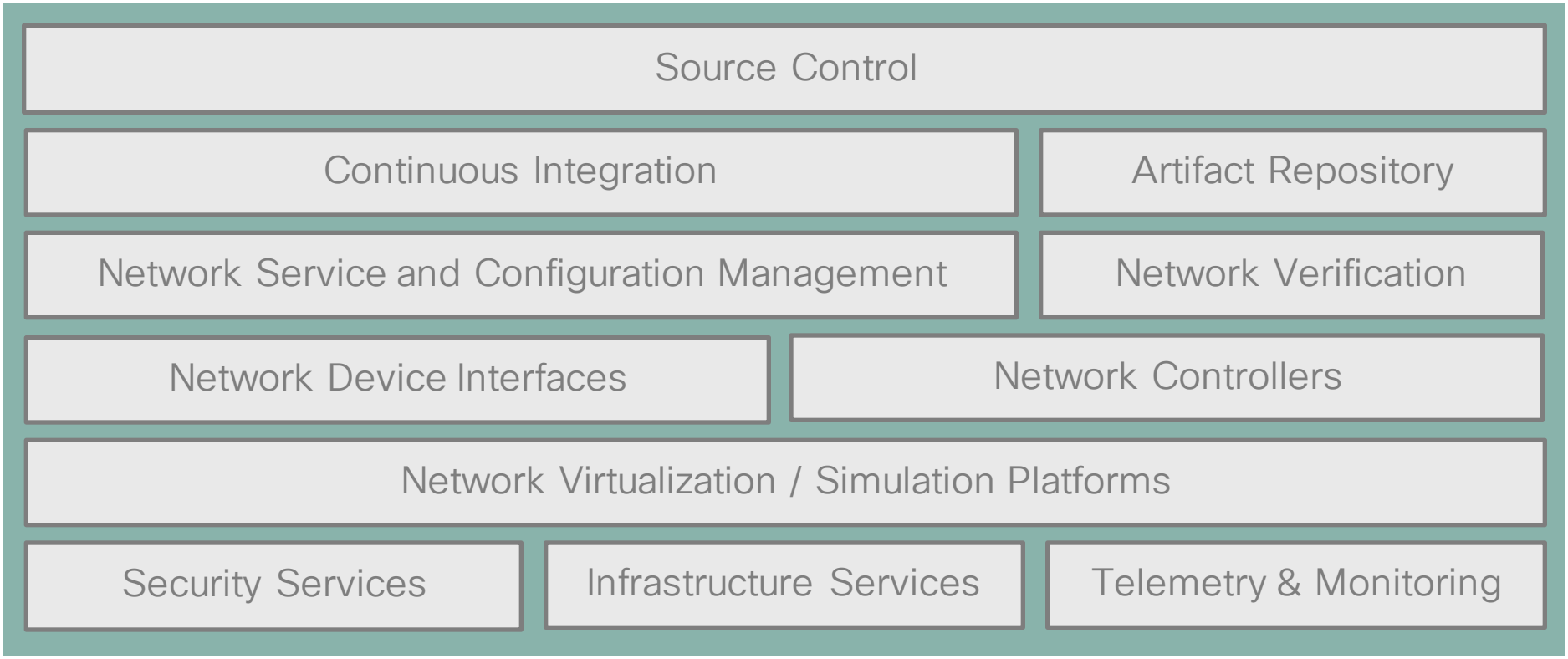


Review and Next Steps

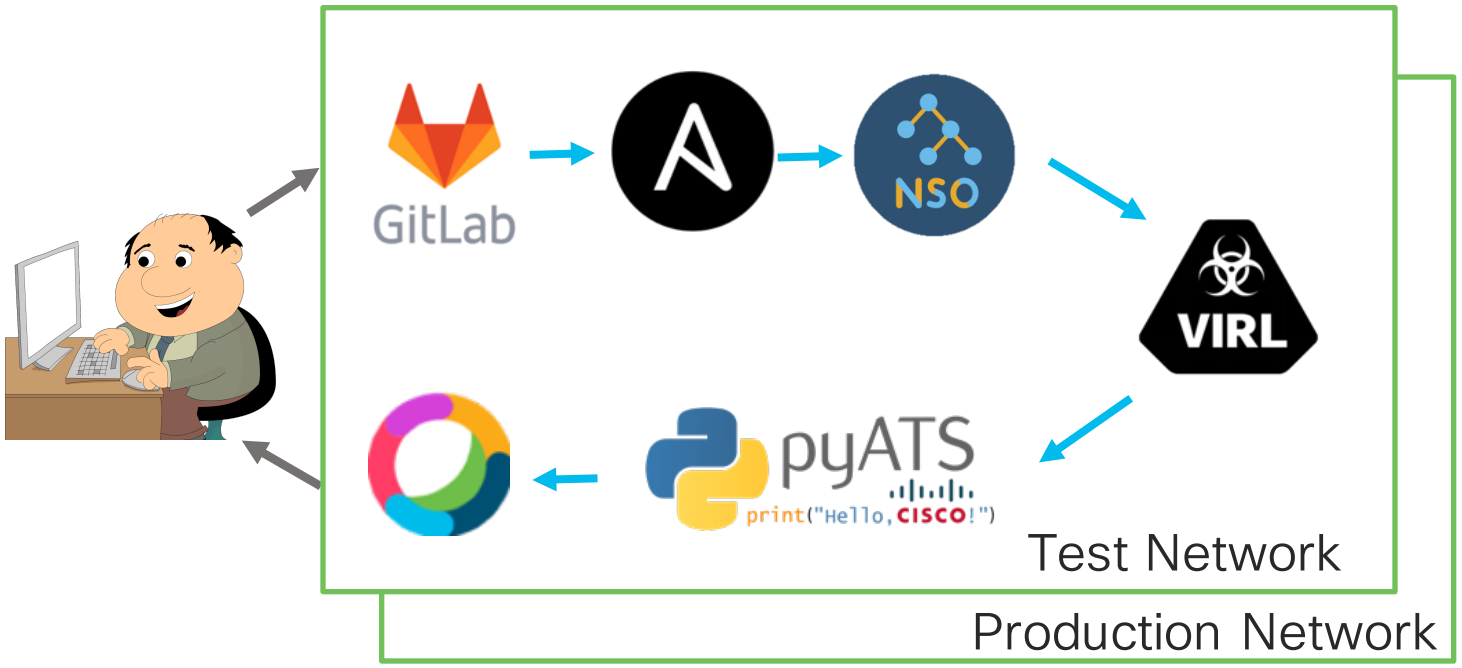


Cisco live!

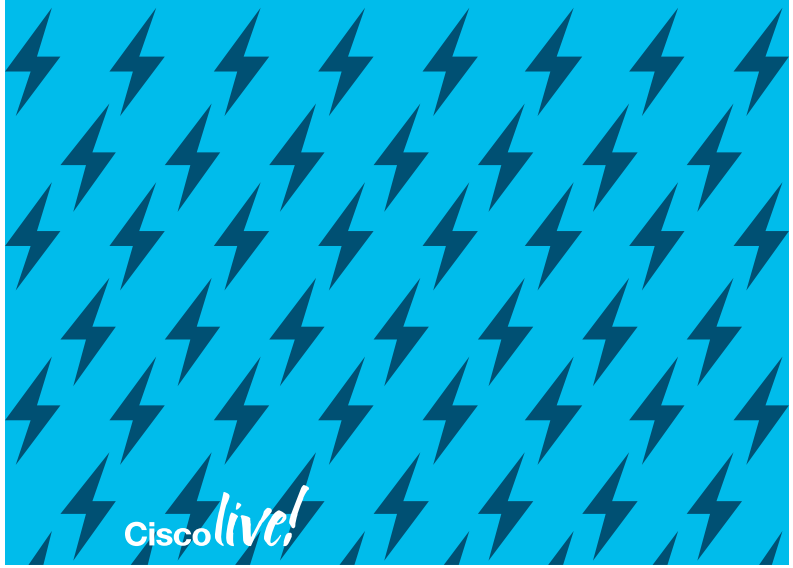
The NetDevOps Engineers Tool Chest



Our “opinionated” Stack + Workflow



Agenda



- What are we trying to achieve with NetDevOps?
- Picking our tools
  - Source Control
  - Configuration Management
  - Environment Simulation
  - Network Verification
- Review and Next Steps

More Resources

- git
  - What's the big deal about Source Control? Why your network configurations should be in "git"
- Cisco NSO
  - Introduction to Network Services Orchestrator – the single API and CLI for your netwo`rk
- VIRL & virlutils
  - NetDevOps development environments with Vagrant, VIRL and Cisco NSO
- pyATS
  - Profile, test and verify your network is running smoothly with pyATS
- NetDevOps CICD
  - It's Not a Dream, NetDevOps CICD Pipelines Can Develop, Test, and Deploy Network Configurations Today
  - DEVNET-1296 - NetDevOps CICD Pipelines Can Develop, Test, and Deploy Network Configurations Today

Cisco Webex Teams

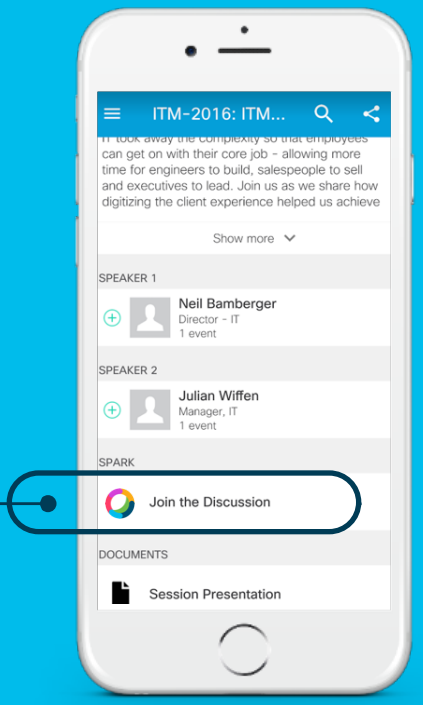
Questions?

Use Cisco Webex Teams (formerly Cisco Spark) to chat with the speaker after the session

How

1. Find this session in the Cisco Live Mobile App
2. Click "Join the Discussion"
3. Install Webex Teams or go directly to the team space
4. Enter messages/questions in the team space

Webex Teams will be moderated by the speaker until December 10, 2018.



cs.co/ciscolivebot#BRKXXX-xxxx

Thank you

Service Definition

- What are the inputs required to instantiate a service?
- Can reference / inherit other model definitions

```
list vxlan-leaf {
  description "RFS model of VXLAN leaf";

    leaf as-number {
      type uint16;
      mandatory true;
    }

    leaf pim-ip {
      mandatory true;
      type inet:ipv4-address;
    }

    leaf loopback0 {
        mandatory true;
        type inet:ipv4-address;
    }
```

Logic

- Code which **enriches** or **augments** inputs to the service
- **Network configuration database** is available for **querying**, **changing**.
- Or **query external systems** – IPAM, etc

```
class LeafService(Service):
    @Service.create
    def cb_create(self, tctx, root, service, proplist):

        vars = ncs.template.Variables()
        vars.add('HOSTNAME', service.name)
        vars.add('GROUP_RANGE', service.group_range)
        vars.add('SSM_LIST', service.ssm_list)
        vars.add('AS_NUMBER', service.as_number)
        vars.add('PIM_IP', service.pim_ip)

        vars.add('LOOPBACK0_IP', service.loopback0)
        vars.add('LOOPBACK1_IP', service.loopback1)

        self.log.info('Vars', vars)
        template = ncs.template.Template(service)
        template.apply('vxlan-evpn-base', vars)
        template.apply('vxlan-evpn-leaf', vars)
```

Templates

- Maps **service input parameters** to **variables** derived from, or computed in package model / logic into XML **device(s) configuration**
  - multiple device/types supported
- Additional logic available
  - Loops/Conditionals

```
<bgp>
  <id>{$AS_NUMBER}</id>
  <router-id>{$LOOPBACK0_IP}</router-id>
   <template>
     <peer>…
   </template>

 <?foreach {/links/router-id}?>
  <neighbor>
    <id>{../router-id}</id>
    <inherit>
      <peer>VTEP-PEERS</peer>
    </inherit>
  </neighbor>
<?end?>
</bgp>
```

INTUITIVE