



# Powering Your Automation: A Single Source of Truth

TIM SCHREYACK



# Network Automation: The Dream

Click the easy button and your  
network is configured



## Network Automation: The Reality

- ▶ Varying vendor support
- ▶ Multiple, competing frameworks
- ▶ Nothing is "off the shelf"
- ▶ There is no easy button to create the easy button



# Automation vs Orchestration

Orchestration – codifying your processes

Automation – codifying your tasks

Orchestration is essentially stringing together multiple chunks of automation.

# Network Automation Overview

- ▶ **Bootstrap**
  - ▶ Deploy new equipment with minimal human involvement
  - ▶ Validate physical infrastructure
- ▶ **Operations**
  - ▶ Easy button for standard MACs
- ▶ **Device Configuration**
  - ▶ Template based versions of config in whole or in part
- ▶ **Version Control**
  - ▶ Keep templates in Github
  - ▶ Maintain branches or forks
  - ▶ Require peer review prior to merging to Master



# Automation Tools

## ▶ Automation Tools

- ▶ Ansible
- ▶ Puppet
- ▶ Salt
- ▶ NAPALM

## ▶ Orchestration tools

- ▶ Stackstorm
- ▶ RunDeck

# Common Methods of Data Input/Storage

## **The Really Really Bad:**

By hand  
Statically defined

## **The Still Pretty Bad:**

Spreadsheets  
Email

## **And Still Not So Great:**

Disconnected tools

IPAM

CMDB

DCIM



## A Better Way

# A Single Source of Truth Database

Store ALL of your static data in one database

Link formerly disconnected pieces of data together

Maintain only one copy of the data, instead of multiples



# What data to store?

- ▶ Physical and virtual Devices
- ▶ Links
- ▶ Data center info (sites, racks, pods, etc...)
- ▶ IP Addressing
- ▶ VLANs
- ▶ ASNs
- ▶ VRFs
- ▶ All your network and other infrastructure data



# Advantages

Now we can make connections between data that was formerly disconnected.

We can easily retrieve static values based on any number of criteria.

We can create resource pools of physical and logical items for use in our automation.



## A Brief Aside

### **Pets vs cattle**

But my resource is special – I need to name it and pet it and love it forever.

No, no, it's not. Treat all your resources as cattle.

# Source of Truth Examples

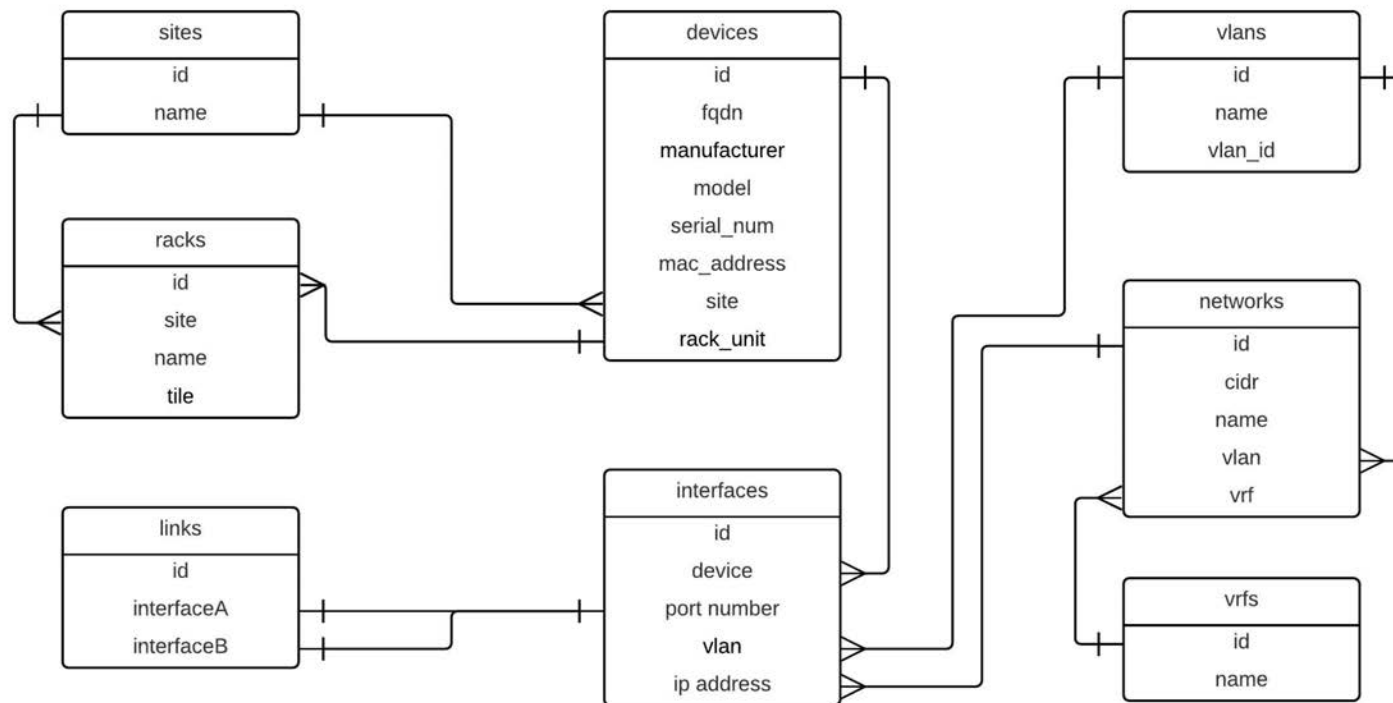
## DIY

- ▶ Mongo
- ▶ Postgres
- ▶ Puppet DB/ Hiera
- ▶ YAML/JSON Files (easy, but limited)

## Off the Shelf

- ▶ NetBox

# Source of Truth Example Structure



# A Practical Example

**Deploy a network update to configure a new physical server that will run web servers.**

- ▶ **Physical Requirements**
  - ▶ Switchports on two access switches
  - ▶ Cabling requirements between server and switches
- ▶ **Logical Requirements**
  - ▶ VLAN(s)
  - ▶ Public, Internal, and Management IP Addresses
  - ▶ FQDN(s)

## A Practical Example (cont.)

From resource pools in your single source of truth, you can now programmatically allocate almost everything:

### **Physical Resources**

Rack Unit location  
Switchports

### **Logical Resources**

VLANs  
IP Addresses

# A Practical Example (cont.)

**We can automatically generate:**

DC Ops request to rack and cable server  
Network configuration (switches/firewalls/etc...)  
DNS Entries



# A Practical Example (cont.)

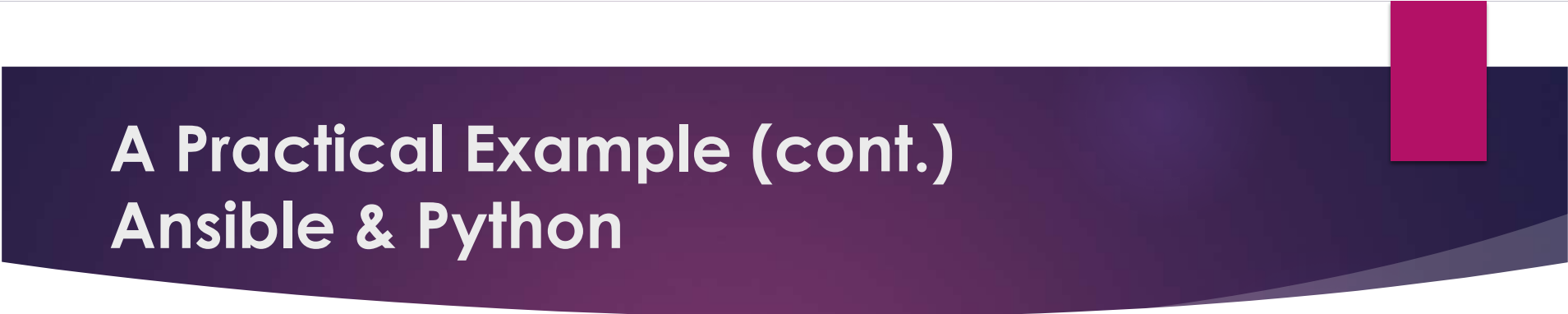
## Ansible & Python

### Ansible

- Short learning curve
- Agentless
- Open Source
- Broad vendor support

### Python

- Most common language (probably)
- Readable
- Supported in virtually all tools



## A Practical Example (cont.) Ansible & Python

Filter Plugins allow you to use Python to look up data from the single source of truth in your Ansible templates.

```
ansible/plays/filter_plugins/my_functions.py
```

Create Python functions and classes to access your database.

# A Practical Example (cont.)

## Ansible & Python

```
switch_template.j2
```

```
interface Ethernet{{ fqdn | allocate_port(pool='webservers') }}
    switchport access vlan {{ site | get_vlan(pool='webservers') }}
    no shut
```

# A Practical Example (cont.)

## Ansible & Python

```
my_functions.py
```

```
def allocate_port(fqdn, pool):
```

```
    <code>
```

```
    return port_number
```

```
def get_vlan(site, pool):
```

```
    <code>
```

```
    return vlan_number
```

# A Practical Example (cont.)

## Ansible & Python

```
my_play.yml
```

```
---
```

```
- hosts: "web_switches"
  connection: local
  vars:
    fqdn: "my_new_server.mydomain"
    site: "my_site"

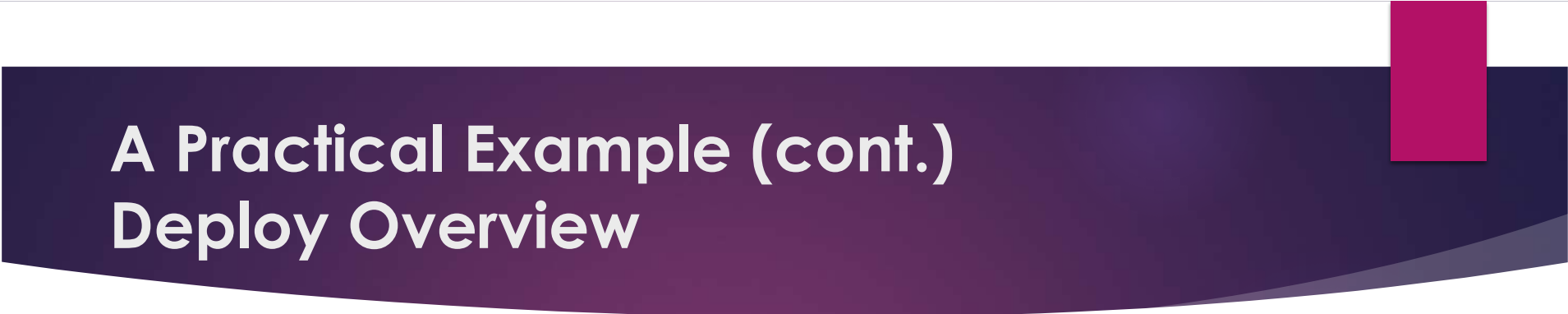
  tasks:
    - name: Generate Switch Configuration
      template:
        src: "my_template.j2"
        dest: "my_switch_config.cfg"
```

# A Practical Example (cont.)

## Ansible & Python

```
my_switch_config.cfg
```

```
interface Ethernet12
  switchport access vlan 101
  no shut
```



# A Practical Example (cont.)

## Deploy Overview

- ▶ Ansible has broad network support
- ▶ Fairly easy to configure using plays to deploy configuration
  - ▶ Take snapshot (depending on Network OS support)
  - ▶ Deploy config (SSH, API)
  - ▶ Save changes
  - ▶ Rollback on failure
- ▶ Can use orchestration tools to handle deploying to multiple devices based on events



# Conclusion

- ▶ Network automation is becoming mainstream
- ▶ A paradigm shift is required to take full advantage of the possibilities
- ▶ Start by identifying key processes that can be automated
- ▶ Don't be afraid to rethink how those processes work
- ▶ Start writing code!



# Questions