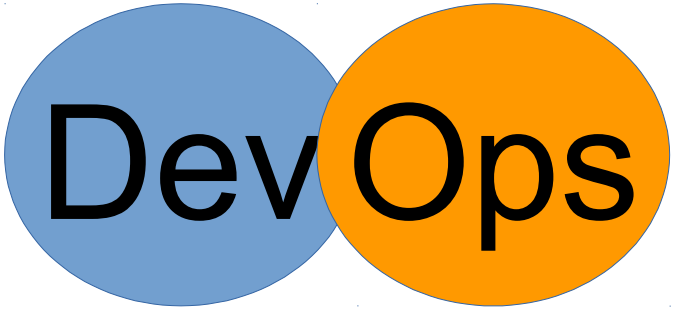
Central Orchestration of Network Infrastructure
NetOps meets DevOps

ken.wilson@opengear.com

DevOps – Quick overview

Remove the barriers between the development and operation teams



- Driven by goal of reducing application deployment overhead and increasing quality
- Focused on automated deployment and configuration
- Infrastructure should be treated as code and tested the same way

Context - Opengear

Build management appliances for data-centres and remote sites

- Used by Network administrators
- Primary uses
  - Serial connectivity to switches/routers/firewalls
  - Serial/USB connectivity to UPS/PDU
  - Provision of Out-of-band access to management networks
- Some server management via Ethernet/IPMI/Serial
- Often the primary access method for configuration of network equipment

NetOps - Current Situation at our customers

- Config snippets stored in a wiki
- Manually pasted into a SSH/Telnet/Serial console

NetOps – Current Situation (cont)

Pros

- Well understood
- Config CLIs will catch syntax and some logic errors
- Basic version control with snippets stored in a Wiki

Cons

- Everything else

NetOps – Current Situation (cont)

Hard to get a view of how your network is actually configured

- Architecture diagrams don't count

Tools like RANCID help bridge that gap

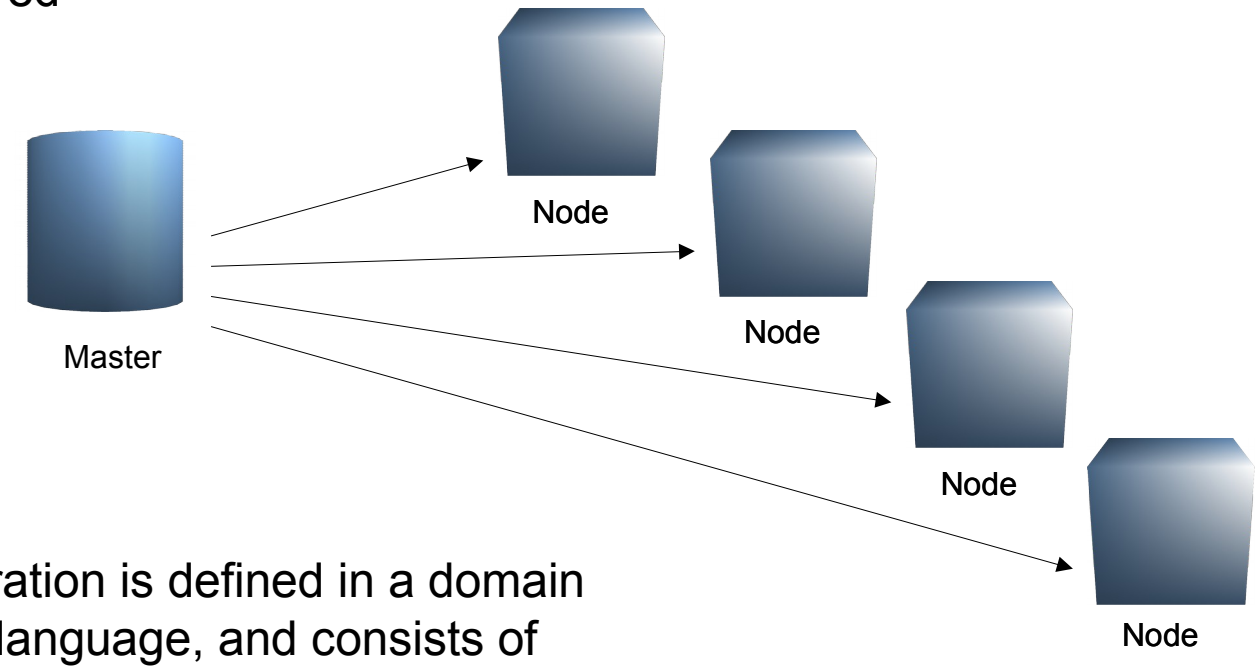
- Really Awesome New Cisco Config Differ
- Configs (and hardware information) are retrieved from devices, and stored in CVS
- Notifications on config or hardware changes

Helps solve the config backup and audit issues, but doesn't help deployment

Orchestration Tools

Orchestration Tools – Common Concepts

Configuration is deployed centrally from a master to the nodes being configured



Configuration is defined in a domain specific language, and consists of lists of actions.

Orchestration Tools – Common Concepts

Action

- Install a package
- Start a service
- Copy a file
- Configure a network port
- Customisable with variables locally defined or discovered from the node

Actions are run in a defined order
Actions are idempotent
Action definitions can also be used for audit

Orchestration Tools – Common Concepts

Variables

Can be defined in multiple places

- Locally in the action definition
- Locally in the node definitions
- Dynamically by querying the remote node

Used in the action definition, or for populating templated configuration files

```
{master:0}[edit]
puppet@choc-qfx5100-c# show protocols bgp
group external {
    type external;
    local-address 20.20.20.20;
    peer-as 200;
    neighbor 30.30.10.10;
    neighbor 30.30.10.11;
}
```



```
[root@chef-lnx04 templates]# cat bgp.text.erb
<% @bgp.each do | name, hash | %>
protocols {
    bgp {
        group <%=name%> {
            type  <%= hash['type']%>;
            local-address <%= hash['local-address']%>;
            peer-as <%= hash['peer-as']%>;
            <% hash['neighbor'].each do | neighbor | %>
                neighbor <%=neighbor%>;
            <% end %>
        }
    }
}
<% end %>
```

Orchestration Tools – Common Concepts

Version Control

Actions and Node configuration should be version controlled

- Usually left to the implementer (apart from Chef)
- Allows revision tracking of infrastructure configuration
- Makes it easier to integrate a review process into network config operations
- Easier roll-back to known good configurations if required

Puppet

- Maintained by Puppet Labs – initial release was in 2005
- Java master process
- Ruby agent that runs on the node
- Communications over certificate secured SSL TCP connection
  - Agent generates the certs, and requires the master to authorise

Nomenclature

- Action = Resource
- List of Actions (and logic for selecting nodes) = Manifest
- Variables = Facts

Puppet – Network Device Support

- Cisco NXOS/IOS-XR
- Arista EOS
- Huawei CloudEngine
- Cumulus Linux
- Juniper JunOS
- Mellanox

Chef

- Maintained by Chef (formerly OpsCode) – initial release was around 2008
- Erlang/Ruby master process
- Ruby agent that runs on the node
- Communicates via TCP connections to a variety of services
- Authentication via certificates

Nomenclature

- Action = Recipe
- List of Actions = Cookbook
- Variables = Attributes
- Mapping of Cookbooks to Nodes = Run List

Chef – Network Device Support

- Cisco NXOS/IOS-XR
- Arista EOS
- Cumulus Linux
- Juniper JunOS

Ansible

- Maintained by RedHat – initial release was in 2012
- Python master process, “Agent-less”
- Master process pushes agent code to the node during operations
  - Requires Python to run
- SSH is used as connection mechanism
- Authentication is via SSH shared keys

Nomenclature

- Action = Play
- List of Actions = Playbook
- Variables = Facts
- Mapping of Plays to Nodes = Defined in the Playbook

Ansible – Network Device Support

- Cisco NXOS/IOS-XR
- Arista EOS
- Cumulus Linux
- Juniper JunOS

Commonalities in Bindings

- Multi-vendor is a nice idea, but quite restricted
- Puppet and Chef have netdev – focused on L1/L2 Switch configuration
  - Primarily pushed by Juniper, adding support for others (Cisco, Arista, Mellanox)
- Building blocks are
  - netdev_interface – physical interface abstraction
  - netdev_l2_interface – used for creating/deleting layer 2 interfaces
  - netdev_lag – used for creating/deleting link aggregation groups
  - netdev_vlan – used for creating/deleting VLANs
- Any more complexity means vendor specific bindings

Example - Puppet

```
node "jd.mycorp.com" {
  netdev_device { $hostname: }

  netdev_vlan { "Pink":
    vlan_id => 105,
    description => "This is a pink vlan",
  }

  netdev_vlan { "Green":
    vlan_id => 101,
  }

  netdev_vlan { "Red":
    vlan_id => 103,
    description => "This is the native vlan",
  }

  netdev_l2_interface { 'ge-0/0/19':
    untagged_vlan => Red,
  }

  netdev_l2_interface { 'ge-0/0/20':
    description => "connected to R1-central",
    untagged_vlan => Red,
    tagged_vlans => [ Green, Pink ],
  }
```

Example - Chef

```
# Filename “netdev_access_switch/vlan_create.rb”
netdev_vlan “Pink” do
        vlan_id 105
        description  “This is a pink vlan”
        action :create
end
netdev_vlan “Green” do
        vlan_id 101
        action :create
end
netdev_vlan “Red” do
        vlan_id 103
        description  “This is the native vlan”
        action :create
end
netdev_l2_interface “ge-0/0/19” do
        untagged_vlan “Red”
        vlan_tagging false
        action :create
end
netdev_l2_interface “ge-0/0/20” do
        description “connected to R1-central”
        untagged_vlan “Red”
        tagged_vlans [“Green”, “Pink”]
        vlan_tagging true
        action :create
end
```

```
# Run List
{
  "name": "access_switch_jd_mycorp_com",
  "chef_environment": "_default",
  "normal": {
  },
   "run_list": [
                "recipe[netdev_access_switch::vlan_create]"
  ]
}
```

Barriers to entry

NetOps

- Ain’t broke, why fix
- Vendor support
  - Closed ecosystem is better for them
- In-house expertise
  - $$CONSULTANTS$$

Vendors

- Hard to pick the winning horse
- Can be a challenge to embed the agents
- Resource constraints
- Lock in

Futures

- Systems/Vendors will provide more consistent interfaces
  - Netdev is a start
- DevOps will become the norm
- Time to skill up :)

Questions?