Network automation in DC ČRA

Ansible, Git, CI/CD, ARISTA

CSNOG 2025 @ Zlín

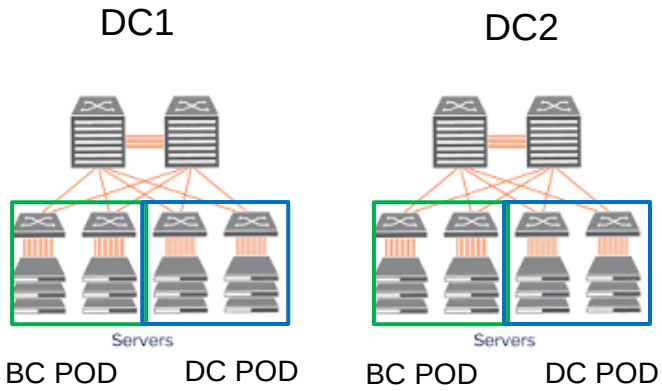
Radim Roška & Vojtěch Šetina

22.1.2025

ALTEPRO

CRA DC CORE project – **DC fabric**

- Leaf – spine components
  - Spines – 7260CX3-64-F (64x 100GE)
  - Leaves – 7050SX3-48YC8 (48x25GE + 8x100GE)
  - Border leaves with MACSEC- 7280SR3M-48YC8 (48x25GE + 8x100GE)
  - Management - ARISTA Cloud Vision Portal
- **BGP EVPN** as control plane and **VXLAN** as a dataplane
- 2 DCs acting as eventually single EVPN domain
- Basic design facts
  - Design based on ARISTA best practice – eBGP/eBGP
  - EVPN multihoming
  - Multitenant L2 / L3 unicast and multicast services



"A multi-DC, leaf-spine architecture with BGP EVPN and VXLAN provides scalability, flexibility, and best-practice Arista design."

CRA DC CORE project - **Automation Goals**

- Phase 1:
  - **ZTP** (Zero Touch Provisioning)
  - Introduce **SOT** (Source of Truth) & templates (AVD) based on new **LLD**
  - **Pre-validate LLD** in **containerlab**
  - Introduce **GIT**
  - Automate with **change control**
  - **Migration** from legacy to new SOT/network

CONTAINERlab

- Phase 2:
  - Improve GIT usage
  - **CI** pipeline for provisioning
  - Day 2 operations
    - Manual edits of **SOT** => trigger pipeline
    - CLI based **application** to edit **SOT** => trigger pipeline

Phase 1

1 rok

Phase 2

ANSIBLE

git

CloudVision

+

SOT editor

CI CD

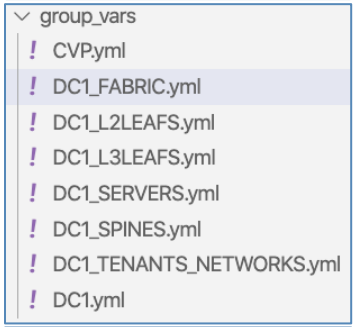
Automation - ARISTA AVD (data model)

Docs: https://avd.sh

Inventory

```
all:
  children:
    CVP: hosts: cv_server: ansible_host: 10.83.28.164
    DC1: children:
      DC1_FABRIC: children:
        DC1_SPINES: hosts:
          DC1-SPINE1: ansible_host: 10.255.0.11
          DC1-SPINE2: ansible_host: 10.255.0.12
# <etc.>
```

Variables




```
# Spine Switches
spine:
  platform: vEOS-LAB
  bgp_as: 65001
  # defines the range of acceptable remote ASNs from leaf switches
  leaf_as_range: 65101-65132
  nodes:
    DC1-SPINE1:
      id: 1
      mgmt_ip: 10.255.0.11/24
    DC1-SPINE2:
      id: 2
      mgmt_ip: 10.255.0.12/24
```

Playbooks

PlayBook

```
---
- name: Build Testing topology using Dev CVP servers
  hosts: cvp
  connection: local
  gather_facts: no
  tasks:
    - name: '#01 - Collect initial facts from
{{inventory_hostname}}'
      cv_facts:
      register: FACTS
    - name: '#02 - Configure Configlets on {{inventory_hostname}}'
      cv_configlet:
        cvp_facts: "{{FACTS.ansible_facts}}"
        configlets: "{{configlet_list}}"
        configlet_filter: ["ANSIBLE_TESTING"]
      register: CONFIGLETS
```

Task

Play

```
- name: Build Switch configuration
  hosts: DC1_FABRIC
  connection: local
  gather_facts: no
  collections:
    - arista.avd
    - arista.cvp
  tasks:
    - name: generate intended variables
      import_role:
        name: eos_l3ls_evpn
    - name: generate device intended config and documentation
      import_role:
        name: eos_cli_config_gen
```

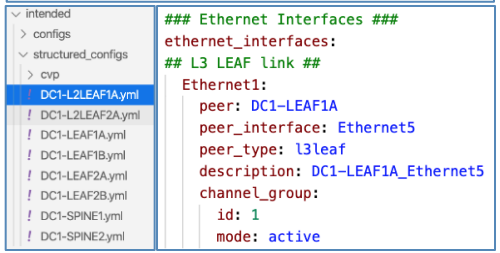
Modules, Roles, Filters

http://docs.ansible.com/ansible/latest/list_of_network_modules.html

Jinja2 Templates

Jinja

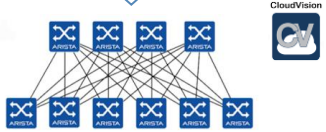
Structured Configs



Documentation

M↓

Intended Configs



Why AVD?

- Few lines in AVD => *“Lots of error-free config lines in multiple switches”*

```
23  # Define underlay and overlay routing protocol to be used
  - underlay_routing_protocol: ebgp
  - overlay_routing_protocol: ebgp
24+ underlay_routing_protocol: ospf
25+ overlay_routing_protocol: ibgp
```

```
$ git diff --numstat intended/configs/dc1-leaf1a.cfg
27      22      avd-examples/…/dc1-leaf1a.cfg
```


```
259 router bgp 65101
260    router-id 10.255.0.3
261    maximum-paths 4 ecmp 4
262    no bgp default ipv4-unicast
263    neighbor EVPN-OVERLAY-PEERS peer group
264+   neighbor EVPN-OVERLAY-PEERS remote-as 65101
265    neighbor EVPN-OVERLAY-PEERS update-source Loopback0
266    neighbor EVPN-OVERLAY-PEERS bfd
       neighbor EVPN-OVERLAY-PEERS ebgp-multihop 3
267    neighbor EVPN-OVERLAY-PEERS password 7 Q4fqtbqcZ7oQuKfuWtNGRQ==
268    neighbor EVPN-OVERLAY-PEERS send-community
269    neighbor EVPN-OVERLAY-PEERS maximum-routes 0
       neighbor IPv4-UNDERLAY-PEERS peer group
       neighbor IPv4-UNDERLAY-PEERS password 7 7x4B4rnJhZB438m9+BrBfQ==
       neighbor IPv4-UNDERLAY-PEERS send-community
       neighbor IPv4-UNDERLAY-PEERS maximum-routes 12000
```

Automation – **CRA SOT** Services - *overlay*

- Network services
  - VRFs, VLANs, routing, SVIs…
- Endpoints
  - “access ports”

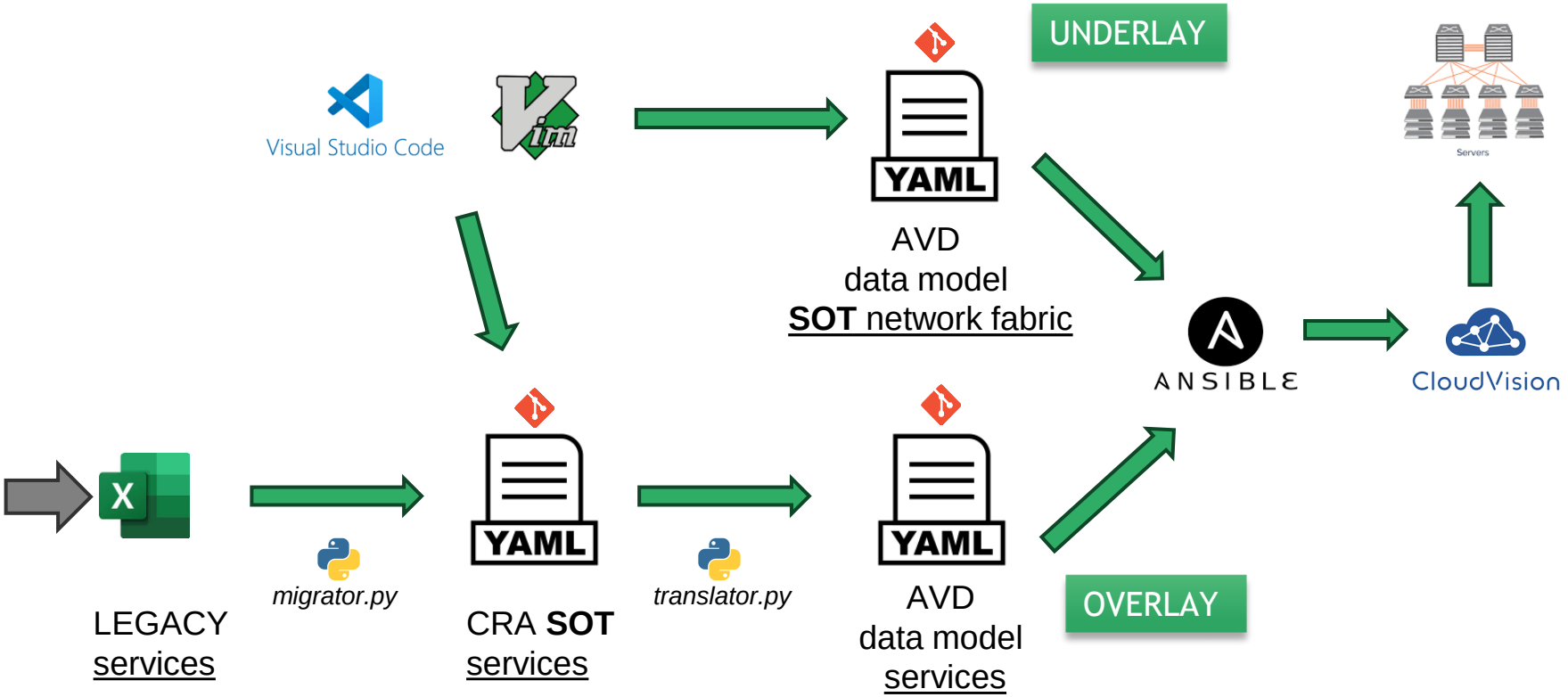
dccore-sot > ! cra_services.yaml

```yaml
tenants:
  - name: CRA
    vrfs:
      - name: cra-inet
        svis:
          - CRA_id: 123456
            name: EXAMPLE_service-1234
            CRA_service_type: service_l3_anycast
            vxlan: true
            enabled: true
            CRA_pods:
              TVPM_DC:
                CRA_vlan-id: 1234
                CRA_nodes: [dca-phdc-dc-l11, dca-p2dc-dc-l12]
            ip_address_virtual: xx.xx.xx.153/30
            ipv6_address_virtuals: ['xxxx:xxx:0:xxx::1/64']
```

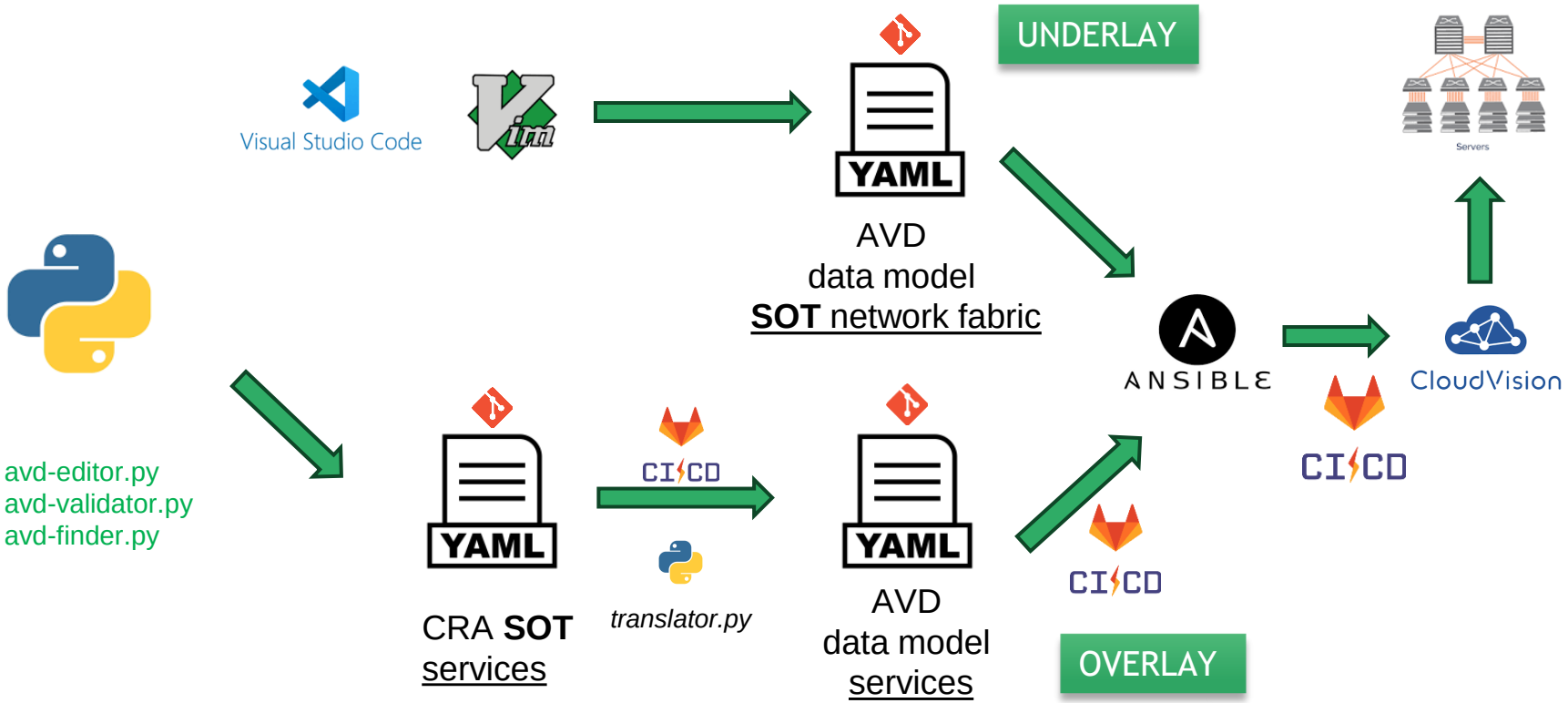
dccore-sot > ! cra_connected_endpoints.yaml

```yaml
# Servers
servers:
- name: SERVER_EXAMPLE_1
  adapters:
  - switch_ports: [Eth2, Eth2]
    switches: [cra-dc-l11, cra-dc-l12]
    endpoint_ports: [ens3f0, ens6f1]
    link_tracking: {enabled: true}
    port_channel:
      channel_id: 110
      mode: active
      description: SERVER_EXAMPLE_1
      lacp_timer: {mode: normal}
    ethernet_segment: {short_esi: '0dc1:0000:xxxx'}
    profile: PPROFILE-EXAMPLE
    vlans: 1234
    qos_profile: QOS_PROFILE_EXAMPLE
```

Automation – Phase 1 (SOT, AVD, CVP)



Automation – Phase 2 (python, GitLab CI)



Helper scripts – Finder

- avd_finder.py
  - Helps and simplifies searching across all yaml files (services, endpoints, port profiles)
  - Multiple filters and extended outputs

```
Finding all objects with CRA_vlan-id: 123 ...
```

SERVICE TABLES

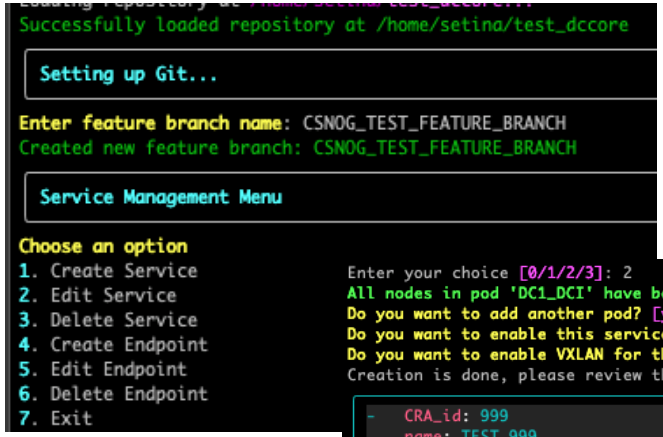
POD	VLAN	CRA_ID	SERVICE_NAME	TYPE	VXLAN	CRA_NODES
TVPM_DCI	123	10666123	test-rr	l2vlan	True	dca-phdc-bl01,dca-p6dc-bl02

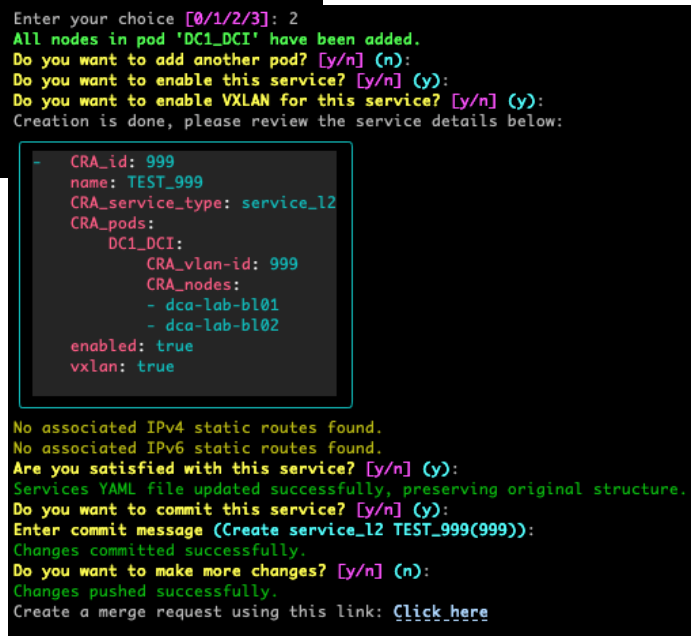
ENDPOINT TABLES

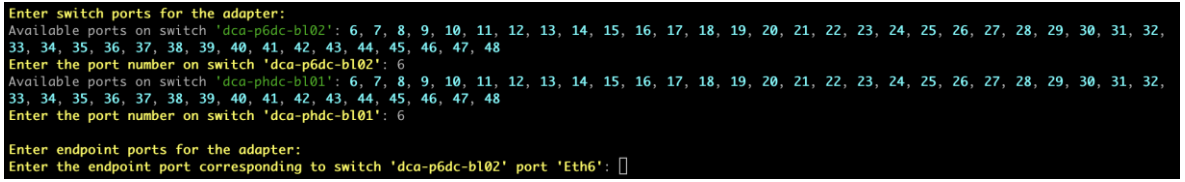
POD	ENDPOINT_NAME	TYPE	SWITCH	SWITCH_PORTS	ENDPOINT_PORTS	PORT_PROFILE	MODE
TVPM_DCI	test-rr-endpoint	servers	dca-phdc-bl01, dca-p6dc-bl02	Eth6, Eth6	Eth8, Eth9	PPROFILE-CUSTOMER-ACCESS	access

Helper scripts - Editor

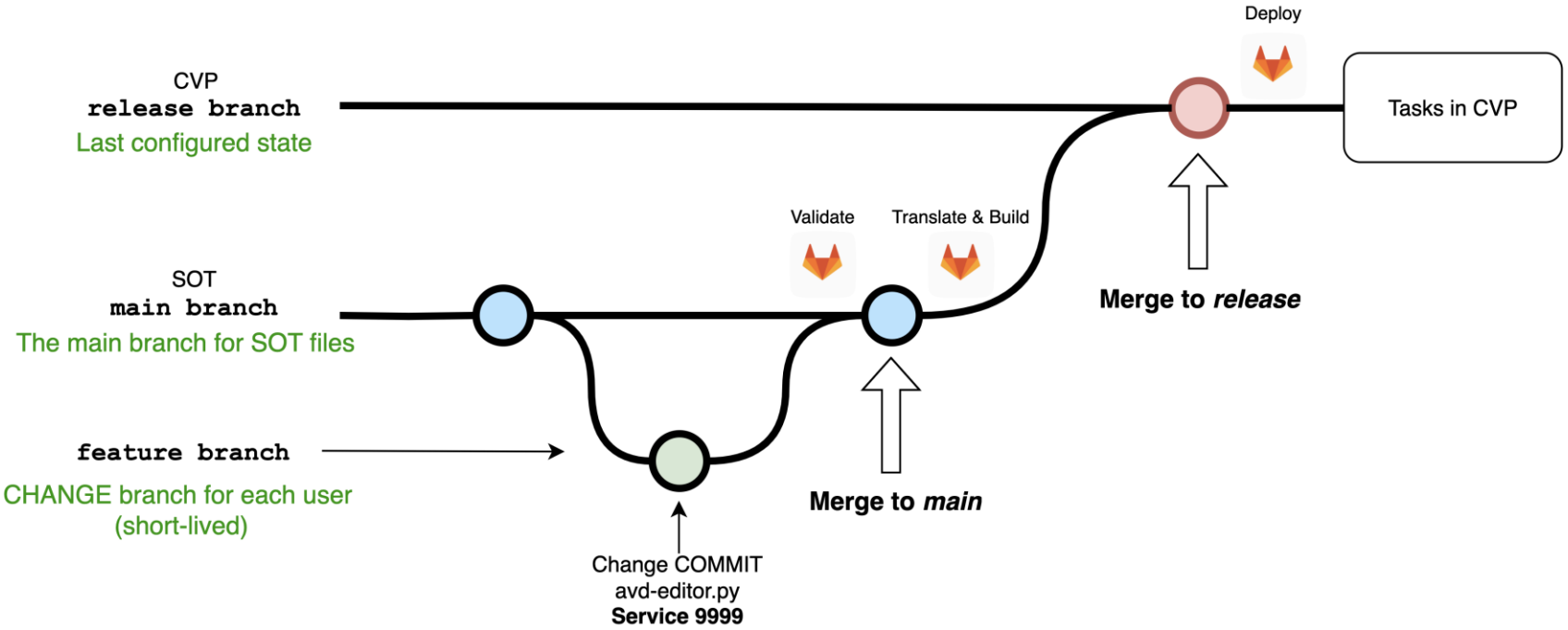
- avd-editor.py
  - Simplify changes on SOT yaml files
  - Real time validations
    - unique CRA_id
    - occupied switch ports
    - ...
  - Handles Git operations automatically
    - The user only answers simple questions







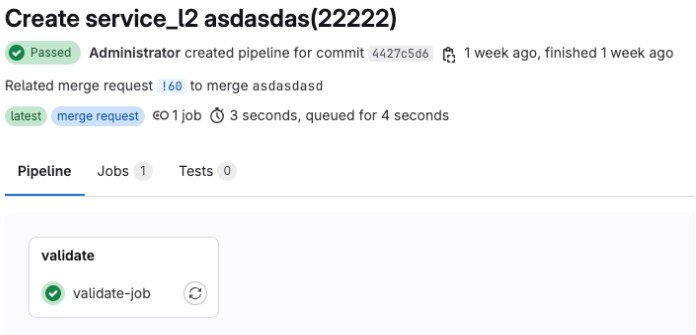
Workflow – Create service 9999

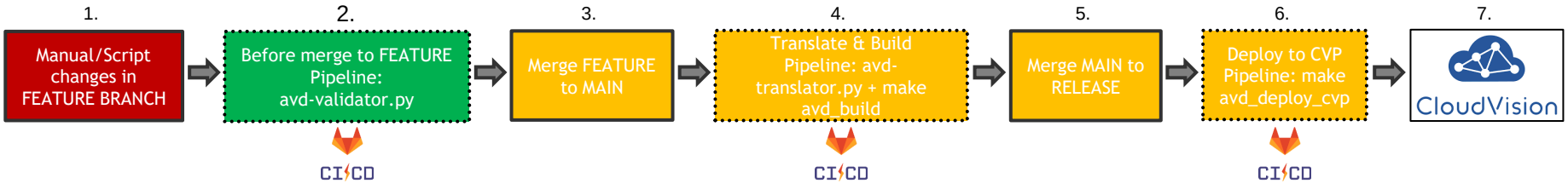


Validation Pipeline

- Validations before the merge to MAIN
  - Lower error rate in production repository
  - Predefined set of validations (Warning & Errors)
  - Pipeline fails when error occurs

validator.py



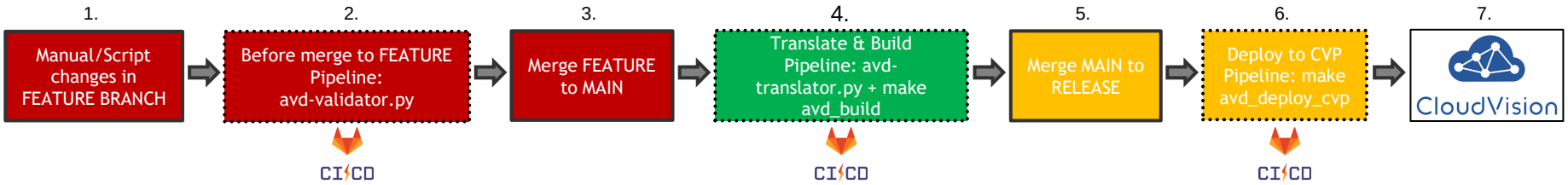


Translate & Build Pipeline

- Both stages creates separate commits
- 2 stages:
  - **Translate**
    - Translate CRA files to AVD using avd-translator.py script
  - **Build**
    - Build intended configs using AVD playbook
    - "make avd_build" in avd repository

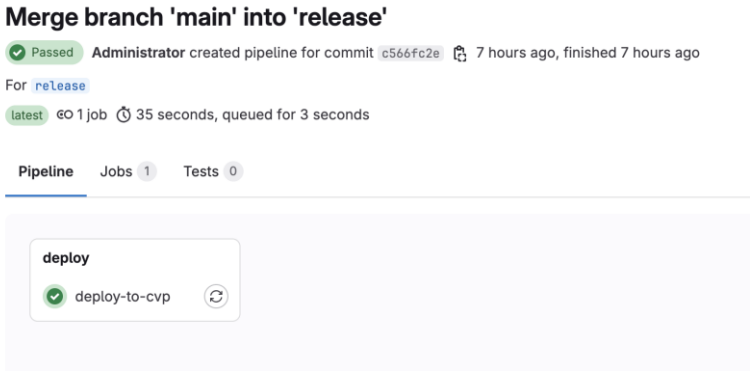
translator.py

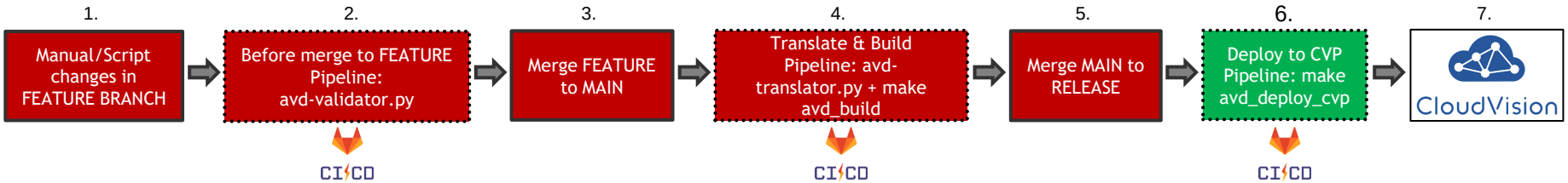
ANSIBLE



Deploy to CVP Pipeline

- Final stage
  - "make avd_deploy_cvp"
  - Deploy intended config files to CVP
    - Ends with updated tasks in CloudVision
    - Can execute automatic documentation (using avd playbook)





DEMO - User scenario – create a service

- Change request
  - VLAN - 999
  - VRF – cra-inet
  - POD – TVPM_DCI (dca-phdc-bl01, dca-p6dc-bl02)
  - L3 interface IP - 100.64.20.2/29
  - + Static route

- Switch config:

```
vlan 999
  name CSNOG
interface Vlan999
  description CSNOG
  vrf cra-inet
   ip address virtual 100.64.20.2/29

interface Vxlan1
  vxlan vlan 999 vni 9999

router bgp 65101
  vlan 999
    rd 999
    route-target both 0:999
    redistribute learned

ip route vrf cra-inet 192.168.1.0/24
   100.64.20.2 name 999_CSNOG
```

CONTAINERlab

How CRA Benefits from Automation

- Improved **efficiency**
- **Reduced** risk of **errors**
- Config compliance & **validations**
- Day 2 **configurations** can be done by **L1/L2 engineers**
- Full Configuration **visibility & control**
- Audit-Ready & **complete** change history
- Enables **rollbacks**
- Opens path for further automations & testing
- Testing in a **virtual environment**

Conclusion

- Complex networking justifies automation
  - SOT → template → **standardized** configs
  - Its not just about efficiency, but also about validations & testing
- Easier than ever to begin
  - Start small & virtual (https://arista-netdevops-community.github.io/avd-cEOS-Lab/)
  - Step by step approach
- Start is **EASY**, production is **HARDER**
  - Phase-by-phase deployment in production
  - Create GUI/TUI to edit SOT = auto-validate SOT

Thank you

www.altepro.cz

...

ALTEPRO solutions a.s.
Na Maninách 1092/20, Praha 7

+420 220 611 111
info@altepro.cz