Netbox

Sorry, only the "I" in DDI

About me

- Johannes Luther / Network Consultant
- Company: IsarNet AG
- Focus: Cisco products
  - Wireless platforms
  - Datacenter (ACI)
  - Authentication stuff (Cisco ISE, 802.1X, NAC…)
  - Automation (Ansible, Custom Python)

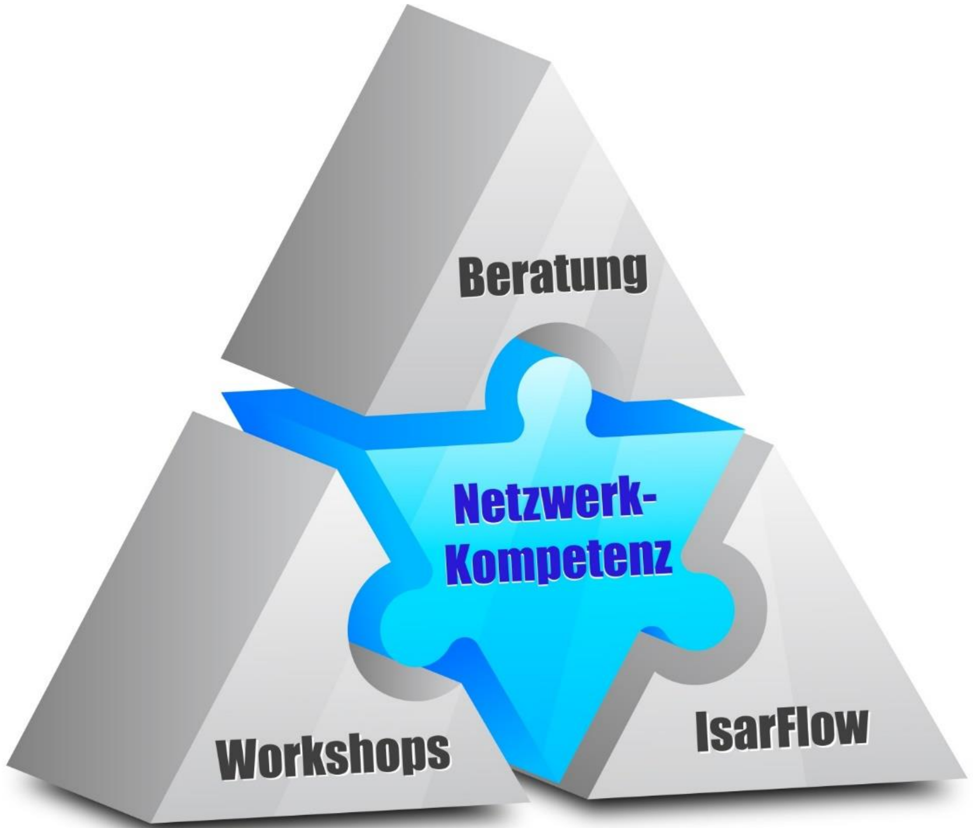
Contact

CISCO CERTIFIED CCIE Enterprise Infrastructure

CISCO CERTIFIED CCIE Enterprise Wireless

CISCO CERTIFIED CCDE DESIGN

IsarNet auf einen Blick



IsarNet auf einen Blick

- Gründung der IsarNet AG im Juli 1999 durch 5 erfahrene Netzwerkspezialisten
- Lokal präsent in München, Stuttgart und Dresden
- Breite Kundenbasis in allen Wirtschaftsbereichen
- Nov 2021: 29 Mitarbeiter
    - 14 Networking-Consultants
    - 12 Software-Entwickler / SE / TME / Support
    - 2x GF, 1x Buchhaltung

Unser Fokus

- Die IsarNet AG bietet hoch spezialisierte Beratung, Services, Tools und Workshops, in allen Bereichen des Networking
- Um diese Services bieten zu können, beschäftigen wir ausschließlich hoch motivierte und qualifizierte Mitarbeiter
- Konsequente Aus- und Weiterbildung gehören daher zu unseren grundlegendsten Prinzipien

- Die IsarNet Software Solutions GmbH ist als 100%ige Tochter der IsarNet AG für die Entwicklung und den Vertrieb netzwerknaher Softwarelösungen verantwortlich.

⇨ IsarFlow – konsolidiertes Performance-Management für NetFlow, IPSLA, SNMP und cbQoS

Professionalität & Qualität

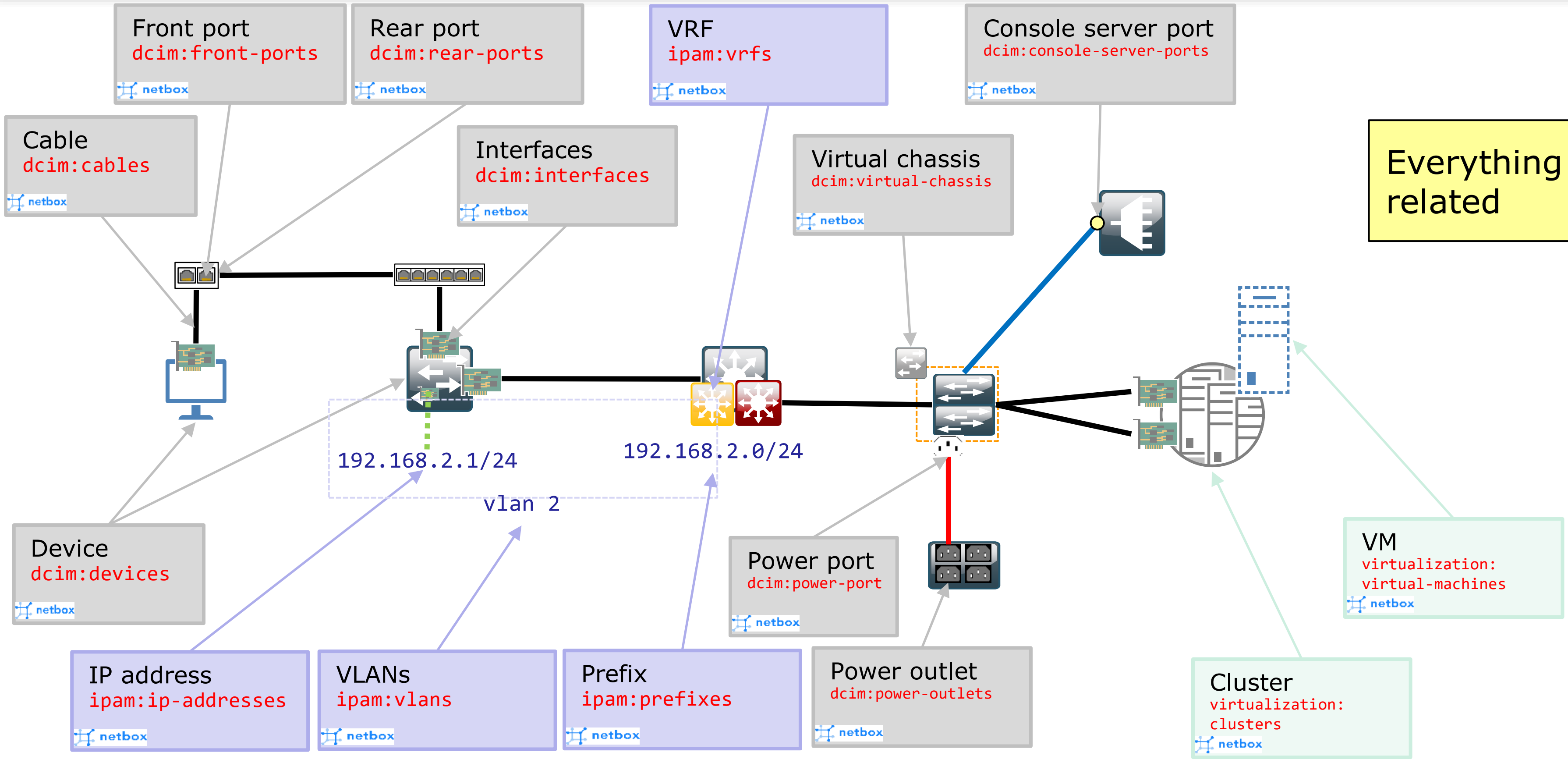
Die Zertifizierungen unseres Teams sprechen für sich:

- **34x Cisco Certified Internetworking Expert**
  - 21x CCIE Enterprise Infrastructure
  - 4x CCIE Service Provider
  - 7x CCIE Security
  - 2x CCIE Enterprise Wireless
- **14x Cisco Certified Design Expert**
- Verschiedene weitere Zertifizierungen, z.B.
  - AWS Certified Solutions Architect – Associate
  - Palo Alto Networks Certified Network Security Engineer
  - RIPE IPv6 Fundamentals – Analyst
  - VMware Certified Professional 6 – Network Virtualization

Netbox introduction

- Netbox (https://github.com/netbox-community/netbox) is not a DDI, it's an **IRM** (Infrastructure Resource Modeling) tool.
- It does **not provide network services** (by default)
- Runs on Linux and is based on the Django Python application framework.
- Open source
- Design goals:
    - Replicate the real world (documentation)
    - Single source of truth

Netbox network modeling

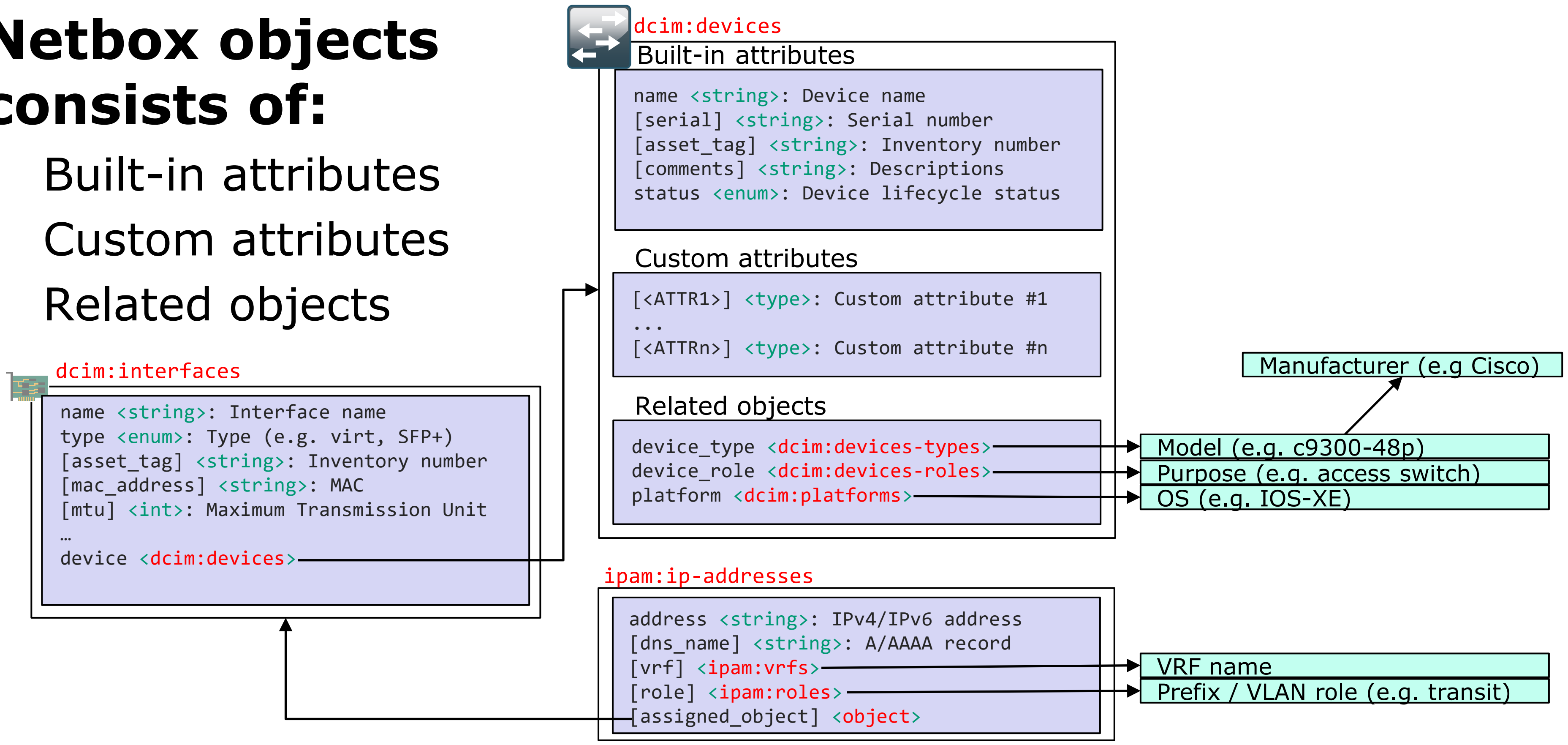


Everything is related

Netbox device details

Netbox objects consists of:

- Built-in attributes
- Custom attributes
- Related objects



dcim:devices

Built-in attributes

```
name <string>: Device name
[serial] <string>: Serial number
[asset_tag] <string>: Inventory number
[comments] <string>: Descriptions
status <enum>: Device lifecycle status
```

Custom attributes

```
[<ATTR1>] <type>: Custom attribute #1
...
[<ATTRn>] <type>: Custom attribute #n
```

Related objects

```
device_type <dcim:devices-types>
device_role <dcim:devices-roles>
platform <dcim:platforms>
```

dcim:interfaces

```
name <string>: Interface name
type <enum>: Type (e.g. virt, SFP+)
[asset_tag] <string>: Inventory number
[mac_address] <string>: MAC
[mtu] <int>: Maximum Transmission Unit
…
device <dcim:devices>
```

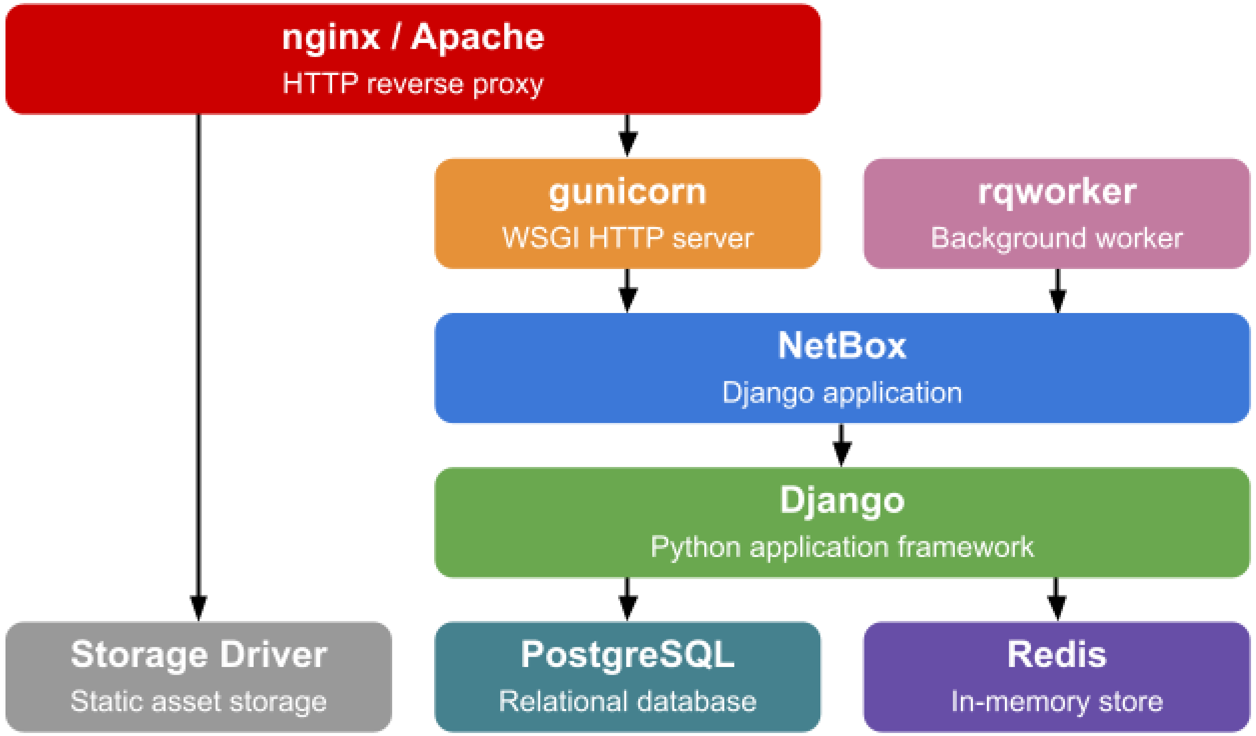
ipam:ip-addresses

```
address <string>: IPv4/IPv6 address
[dns_name] <string>: A/AAAA record
[vrf] <ipam:vrfs>
[role] <ipam:roles>
[assigned_object] <object>
```

Netbox additional objects

- Organizational models
    - Sites and site groups (e.g. buildings)
    - Regions: Group sites and site groups
    - Locations: Rooms, storage places, **rack rows**
- Rack layouts and documentation
- Tenants: Organization grouping
- Tags: Predefined tags for multiple reasons

Netbox architecture



Netbox interfaces

- Because Netbox shall act as the single **source of truth** it needs interfaces that other tools can talk to it:
  - **REST-API**: OpenAPI (documented using Swagger)
    ⇨ Pull/Query information from Netbox

    ⇨ Configuration interface
  - **GraphiQL-API**: Read only API with a custom data model

    ⇨ Pull / Query information from Netbox
  - **Webhooks**: Trigger based notifications
    ⇨ Push information from Netbox

Netbox interfaces

- User facing interfaces:
  - Web UI
  - CLI (`nbshell`)

- Other available interfaces and integrations:
  - Pynetbox (Python API client library)
  - Ansible modules and plugins (netbox.netbox). Examples:
    - netbox.netbox.nb_inventory (Ansible dynamic inventory source)
    - netbox.netbox.netbox_device (Create, update or delete devices)
    - netbox.netbox.nb_lookup (Queries and returns elements)
  - And some more

```
# On Netbox server: Run shell
netbox/manage.py nbshell

# Query a device
deviceObj = Device.objects.get(name='myDevice')
# Print the device name and type
print(f'{deviceObj.name} ({deviceObj.device_type})')
myDevice (C9200-24P)

# Query devices in a specific rack
rack = 'myRack01'
deviceList = Device.objects.filter(rack__name=rack)
for deviceObj in deviceList:
  # Print the device name and type
  print(f'{deviceObj.name} ({deviceObj.device_type})')

# Bulk create loopback0 interfaces on multiple devices (by list)
mgmtInterfaceId = 'Loopback0'
devices = [ 'myDevice01', 'myDevice02' ]
for device in devices:
  Interface(name=mgmtInterfaceId, type="virtual", device=Device.objects.get(name=device)).save()

# Bulk create loopback0 interfaces on multiple devices (by role)
mgmtInterfaceId = 'Loopback0'
deviceRole = 'Distribution switch'
devices = Device.objects.filter(device_role_id=DeviceRole.objects.get(name=deviceRole).id)
for device in devices:
  # Skip if device is in virtual chassis and it is not the master device
  if (device.virtual_chassis_id) and (device.name != device.virtual_chassis.master.name):
    print(f'{device}: is virtual chassis member and not master ... skipping device')
  else:
    if not device.interfaces.filter(name=mgmtInterfaceId).exists():
      print(f'{device}: interface {mgmtInterfaceId} does not exist ... creating interface')
      Interface(name=mgmtInterfaceId, type="virtual", device=Device.objects.get(name=device)).save()
    else:
      print(f'{device}: interface {mgmtInterfaceId} does exist')
```

Configuration source

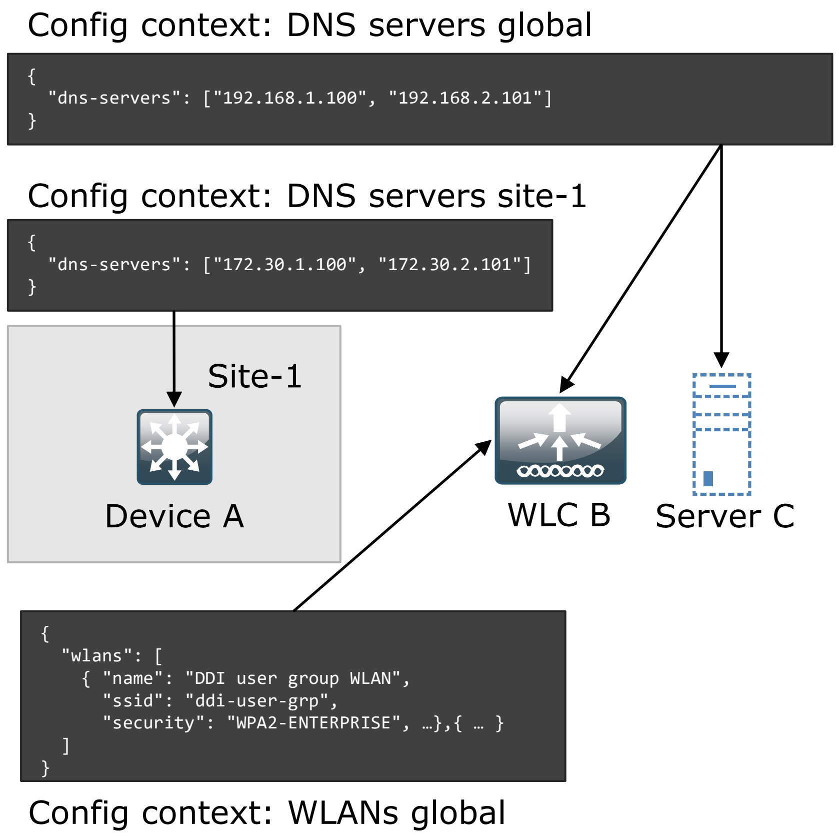
- All object instances can be queried using the REST-API to provide an input for configuration such as:
  - Device type and role
  - Interfaces
  - IP addresses, VLANs, VRFs
  - …
- *What about additional data, which could be relevant for configuration purposes?*

Configuration source

- **Custom fields**
  - Name
  - Type (Text, Int, Bool, List, Dropdown)
  - Assigned object
  - Validation limits / RegEx match
  - Assinged per object (no hierarchy)
  - Might not scale for lots of configuration logic

Configuration source

- **Config context**
  - Associate additional data to a group of devices by region, site, device type, role …
  - Hierarchical rendering
  - JSON input


Config context: DNS servers global
Config context: DNS servers site-1
Site-1
Device A
WLC B
Server C
Config context: WLANs global

```
{
  "dns-servers": ["192.168.1.100", "192.168.2.101"]
}
```

```
{
  "dns-servers": ["172.30.1.100", "172.30.2.101"]
}
```

```
{
  "wlans": [
    { "name": "DDI user group WLAN",
      "ssid": "ddi-user-grp",
      "security": "WPA2-ENTERPRISE", …},{ … }
  ]
}
```

Try it out

- Public Demo: https://demo.netbox.dev/
  ⇨Wiped daily
- Do-it-yourself-demo: Docker

```
# Prerequisites: Docker, docker-compose and Internet
# Linux example

# Prepare
cd /srv/docker
mkdir netbox-demo
cd netbox-demo

# Pull netbox-docker repository
git clone -b release https://github.com/netbox-community/netbox-docker.git .

# Minimal netbox configuration for DEMO (!!!) installation
tee docker-compose.override.yml <<EOF
version: '3.4'
services:
  netbox:
    ports:
      - 8001:8080
EOF

# Pull required docker images
docker-compose pull

# Start containers (add "-d" to run in background / daemonized)
docker-compose up

# Netbox ready on http://<DOCKER-HOST-IP>:8001
```

! Do not use this in production, because:
- No SSL/TLS
- Default credentials (admin/admin)
- Default backend credentials (DB, Redis)
- Default API token: 0123456789abcdef0123456789abcdef01234567

Try it out: Add demo data

Do-it-yourself-demo: Add demo data

```
# Prerequisites: Running Netbox docker

# Prepare
cd /srv/docker/netbox-demo

## Get demo data
wget https://github.com/netbox-community/netbox-demo-data/raw/master/netbox-demo-v3.0.json


## Load data
docker cp netbox-demo-v3.0.json "$(docker-compose ps -q netbox)":/opt/netbox/netbox/netbox-demo.json
docker-compose exec netbox bash -c "source /opt/netbox/venv/bin/activate && ./manage.py loaddata netbox-demo.json"
```

Try it out: Add demo data

Do-it-yourself-demo: Enrich demo data

The demo data does not contain IP addresses, so we add some IPs

```
# Prerequisites: Running Netbox docker

# Run nbshell
docker-compose run --rm netbox /opt/netbox/venv/bin/python /opt/netbox/netbox/manage.py nbshell

# Netbox shell
prefixRole = 'Access - Data'
dataPrefixes = Prefix.objects.filter(role=Role.objects.get(name=prefixRole).id)
deviceSeq = 1

for dataPrefix in dataPrefixes:
  for x in range(0,9):
    nextFreeIp = dataPrefix.get_first_available_ip()
    ip = IPAddress(address=nextFreeIp)
    ip.dns_name = f'client{deviceSeq}.example.test'
    ip.save()
    deviceSeq+=1
```

Example: Ansible inventory

Using Netbox as an Ansible inventory source

```
# Prerequisites: Running Netbox docker, Netbox API token, Python and Ansible

cd ~
mkdir ansible-netbox-demo
cd ansible-netbox-demo

## Optional: Install Ansible in a virtual environment
python3 -m venv venv
source venv/bin/activate
pip install ansible

## Create ansible inventory
tee inventory-netbox.yml <<EOF
plugin: netbox.netbox.nb_inventory
api_endpoint: http://netbox.isarnet.lab:8001
token: 0123456789abcdef0123456789abcdef01234567

validate_certs: False
config_context: True
flatten_config_context: True
group_names_raw: True
interfaces: True
query_filters:
  - role: 'access-switch'
group_by:
  - sites
EOF

## Optional: Activate virtual environment (if no already loaded)
source venv/bin/activate

## Show inventory summary
ansible-inventory -i inventory-netbox.yml --graph
```

```
## Show inventory details
ansible-inventory -i inventory-netbox.yml --list

{
    "_meta": {
        "hostvars": {
            "dmi01-akron-sw01": {
                "custom_fields": {},
                "device_roles": [
                    "access-switch"
                ],
                "device_types": [
                    "c9200-48p"
                ],
                "is_virtual": false,
                "local_context_data": [
                    null
                ],
                "locations": [],
                "manufacturers": [
                    "cisco"
                ],
                "racks": [
                    "Comms closet"
                ],
                "regions": [
                    "us-oh",
                    "us",
                    "north-america"
…
```

isarNet

Professional Networking