

# **Фулстек-сетевик на минималках: Поднимаем WEB сервис управления VLAN`ами**

**GetNet**

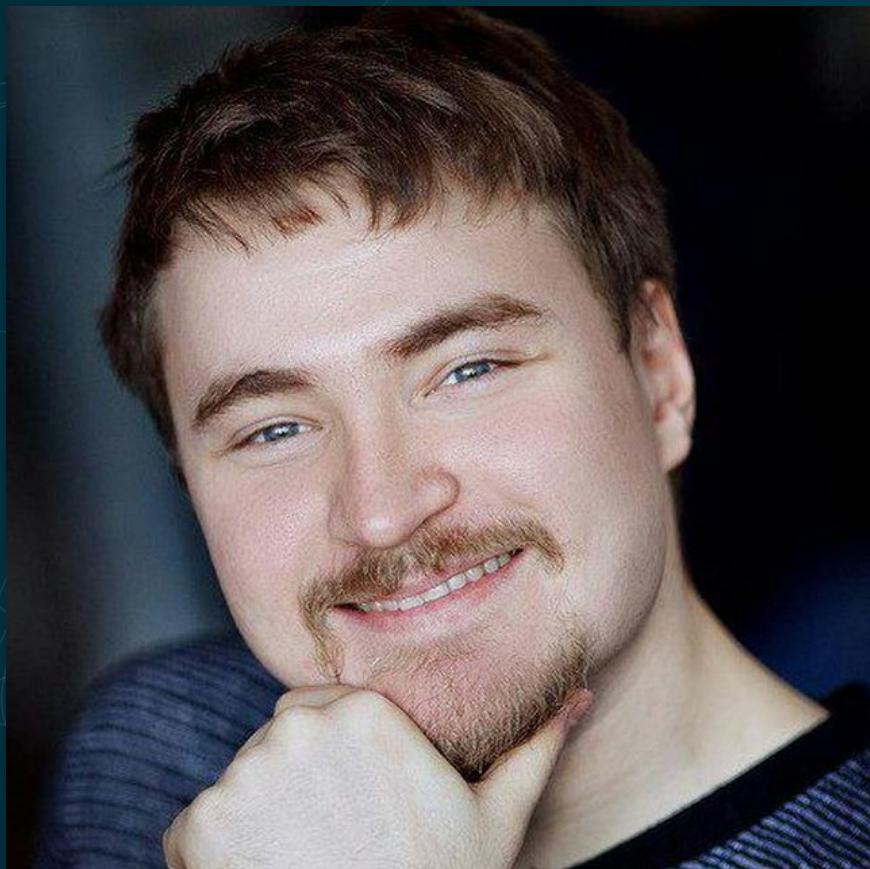
Конференция для  
IT-специалистов

# Обо мне:

---

Роман Карауланов

- Начинал карьеру в операторе связи;
- Прошел путь в Яндексе от дежурного сетевого инженера до инженера эксплуатации сетей dataцентров;
- На данный момент - сетевой инженер в MWS Cloud Platform;
- Занимаюсь автоматизацией и администрированием фабрики ЦОДов;



# Что будем делать?

---

**Access VLAN Deployer** – WEB сервис для переключения VLAN на access портах коммутатора Cisco, который позволит:

- Выводить таблицу с портами, их description и текущими VLAN
- Иметь возможность выбрать VLAN который существует на свиче
- Переключать несколько портов за раз

QR код на страницу проекта:



# С какими инструментами познакомимся:

---

**Flask** — легковесный фреймворк для создания приложений

**Gunicorn** — WSGI сервер для запуска приложения на Flask

**Jinja2** — шаблонизатор для создания динамических текстовых файлов на основе шаблонов и данных

**Re** — модуль для работы с регулярными выражениями

**Netmiko/Scrapli** — модули для подключения к сетевому оборудованию

**Docker** — платформа для создания и запуска контейнеров

**Git** — популярная система контроля версий

# Flask

- Служит для создания web-сервисов и API;
- Простой и гибкий: минимум кода для старта;
- Имеет встроенный web-сервер (только для целей разработки и тестов);
- Не сложный для обучения;
- Совместим с WSGI серверами (gunicorn, uWSGI, и т.д.);

```
from flask import Flask # Импортируем Flask

app = Flask(__name__) # Создаём приложение

@app.route('/') # Главная страница
def hello():
    return "Hello, Network Engineer!" # Ответ сервера

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000) # Запуск сервера
```

# Flask. Пример

```
from flask import Flask
import subprocess

app = Flask(__name__)

@app.route('/ping/<ip>') # URL вида /ping/8.8.8.8
def ping(ip):
    result = subprocess.run(['ping', '-c', '4', ip], capture_output=True, text=True)
    return f"Ping to {ip}:<br><pre>{result.stdout}</pre>"

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

← → ⌂ ⓘ 127.0.0.1:5000/ping/8.8.8.8

Ping to 8.8.8.8:

```
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: icmp_seq=0 ttl=93 time=29.065 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=93 time=24.015 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=93 time=24.133 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=93 time=23.948 ms

--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 23.948/25.290/29.065/2.180 ms
```

# Gunicorn

Легковесный WSGI сервер для работы с приложениями, например написанных при помощи Flask или Django:

- Простой production-ready сервер.  
Просто начать использовать;
- Умеет запускать копии приложения (workers) для увеличения производительности и управлять ими;
- Умеет работать с воркерами разных типов (sync/async);
- Умеет автоматически перезапускать упавший воркер;

```
gunicorn --workers 4 --bind 0.0.0.0:8000 app:app
# где --workers 4 – количество процессов
# --bind 0.0.0.0:8000 – адрес и порт
# app:app – путь к приложению
# Еще пару интересных параметров:
# --reload – перезагрузка приложения при изменении кода
# --timeout 300 – таймаут запроса
# --access-logfile – лог доступа
# --error logfile – лог ошибок
# --log-level – уровень логирования
# --capture-output – перехват вывода
```

# Jinja2

Шаблонизатор для создания динамических текстов на основании шаблонов и набора данных:

- Python-like синтаксис, легок в освоении;
- Имеет два основных вида синтаксиса: для логики - `{% ... %}`, и для переменных - `{{ ... }}` ;
- Поддерживает иерархию шаблонов через `{% include %}` и `{% macros %}`
- Поддерживает условные операторы и циклы: `{% if .. %}`, `{% for .. %}`
- Фильтры вида: `{{ var | lower }}`

```
# {# Циклы #}
# {% for item in items %}
# ...
# {% endfor %}
# {# Условные операторы #}
# {% if "hello" in world %}
# ...
# {% endif %}
# {# Переменные #}
# {% set var = "World!" %}
# Hello {{ var }}
# {# Импорт #}
# {% import "header.html" %}
```

# Jinja2. Берем данные

---

```
interfaces = [
    {
        "name": "FastEthernet1/1",
        "description": "Development",
        "ip_address": "192.168.1.1",
        "subnet_mask": "255.255.255.0"
    },
    {
        "name": "FastEthernet1/2",
        "description": "Sales",
        "ip_address": "192.168.2.1",
        "subnet_mask": "255.255.255.0"
    },
]
```

# Jinja2. Берем шаблон

---

```
template = """
{% for interface in interfaces %}
interface {{ interface.name }}
description {{ interface.description }}
ip address {{ interface.ip_address }} {{ interface.subnet_mask }}
no shutdown
{% endfor %}
"""
```

# Jinja2. Рендерим

---

```
j2 = Template(template)
result = j2.render(interfaces=interfaces)
print(result)
```



# Jinja2. Done

---

```
▶ Apple-MacBook-Pro:~/access-vlan-deployer ➜ master ?1 python3 examples/jinja_example.py
```

```
interface FastEthernet1/1
description Development
ip address 192.168.1.1 255.255.255.0
no shutdown

interface FastEthernet1/2
description Sales
ip address 192.168.2.1 255.255.255.0
no shutdown
```

# Re

Модуль в Python для работы с регулярными выражениями. Проверяет текст на вхождение шаблона, описанного при помощи специальных метасимволов – regexp:

- `re.search(regexp, text)`  
Ищет первое совпадение
- `re.match(regexp, text)`  
Проверяет в начале строки
- `re.split(regexp, text)`  
Делит строку по определенному шаблону
- `re.sub(regexp, target, text)`  
Ищет и меняет текст по шаблону на target
- `re.findall(regexp, text)`  
Возвращает все совпадения списком
- `reg = re.compile(regexp)`  
`reg.search(text)`  
Заранее компилирует регулярку для более быстрой работы при многократном вызове

## # Символы

`\d` – любая цифра  
`\w` – любая буква, цифра, подчёркивание  
`\s` – пробел

`\D` – всё кроме цифр  
`\W` – всё кроме букв  
`\S` – всё кроме пробелов  
. – любой символ

^ – начало строки  
\$ – конец строки

## # Квантификаторы

\* – 0 или более символов  
+ – 1 или более символов  
? – 0 или 1 символ

## # Группы

{n} – n символов  
{n,m} – от n до m символов  
{n,} – n или более символов  
{n,m} – от n до m символов включительно

## # Группировка

() – группировка  
[] – один из символов в скобках  
[a-z] – один символ от a до z  
[0-9] – одна цифра от 0 до 9  
(cisco|juniper) – строка cisco или juniper

# Re. Берем вывод команды

```
data = """
    Mac Address Table
-----
(*) - Security Entry      (M) - MLAG Entry
(MO) - MLAG Output Entry  (MI) - MLAG Input Entry
(E) - EVPN Entry          (EO) - EVPN Output Entry
(EI) - EVPN Input Entry

Vlan      Mac Address        Type      Ports
----      -----              -----     -----
100       e49d.73cf.f270    dynamic   eth-0-46
100       882f.6419.5f00    dynamic   eth-0-28
100       882f.641b.0000    dynamic   eth-0-30
100       5c17.8334.0ecc    dynamic   eth-0-38
100       5c17.8334.01e8    dynamic   eth-0-37
"""

```

# Re. Берем вывод команды

```
for line in data.split("\n"):
    if mac_address := re.search(r"(\w{4}\.){3}", line):
        # Например: e49d.73cf.f270
        # \w{4} – 4 символа
        # \. – 0 или 1 точка
        # (\w{4}\.){3} – группировка для блока из 4 символов и опционально точки
        # {3} – 3 раза
        print(mac_address.group())
```

```
▶ ⚡ ~access-vlan-deployer > 🐍 master ?1 python3 examples/re_example.py
e49d.73cf.f270
882f.6419.5f00
882f.641b.0000
5c17.8334.0ecc
5c17.8334.01e8
```

# Netmiko

Модуль в Python для работы с сетевым оборудованием через SSH:

- Основан на Paramiko;
- Поддерживает множество вендоров: Cisco, Juniper, Huawei и т.д;
- Готовые функции для отправки команд и парсинга промпта;
- Настройка таймаутов и обработка исключений;
- Активное сообщество, подробная документация;

```
from netmiko import ConnectHandler

device = {
    'device_type': 'cisco_ios',
    'host': 'cisco-sw',
    'username': 'cisco',
    'password': 'cisco',
}

connection = ConnectHandler(**device)
output = connection.send_command('show version')
print(output)
connection.disconnect()
# Или через менеджер контекста
with ConnectHandler(**device) as connection:
    output = connection.send_command('show version')
    print(output)
```

# Netmiko. Пример

```
▶ Apple ~ /access-vlan-deployer > master ?1 python3 examples/netmiko_example.py ✓ < py312 ✨ < 17:55:41 ⏺
Cisco IOS Software, vios_l2 Software (vios_l2-ADVENTERPRISEK9-M), Version 15.2(CML_NIGHTLY_20180510)FL0_DSGS7, EARLY DEPLOYMENT DEVELOPMENT BUILD, synced to V152_6_0_81_E
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2018 by Cisco Systems, Inc.
Compiled Thu 10-May-18 05:13 by mmem

ROM: Bootstrap program is IOSv

cisco-sw uptime is 1 week, 5 hours, 51 minutes
System returned to ROM by reload
System image file is "flash0:/vios_l2-adventerprisek9-m"
Last reload reason: Unknown reason
```

# Scrapli

Более современная библиотека для работы с сетевым оборудованием, более гибкая и производительная чем Netmiko.

- Поддерживает как синхронный так и асинхронный код;
- Так же имеет поддержку большого количества вендоров;
- Имеет поддержку разных типов транспорта;
- Имеет встроенную поддержку парсинга вывода, например через TextFSM;
- Более гибкий чем Netmiko за счет встроенного функционала;

```
from scrapli import Scrapli

device = {
    "host": "cisco-sw",
    "auth_username": "cisco",
    "auth_password": "cisco",
    "auth_strict_key": False,
    "platform": "cisco_iosxe",
    "ssh_config_file": True
}

conn = Scrapli(**device)
conn.open()
response = conn.send_command("show interface description")
print(response.result)
conn.close()

# Или через менеджер контекста
with Scrapli(**device) as conn:
    response = conn.send_command("show interface description")
    print(response.result)
```

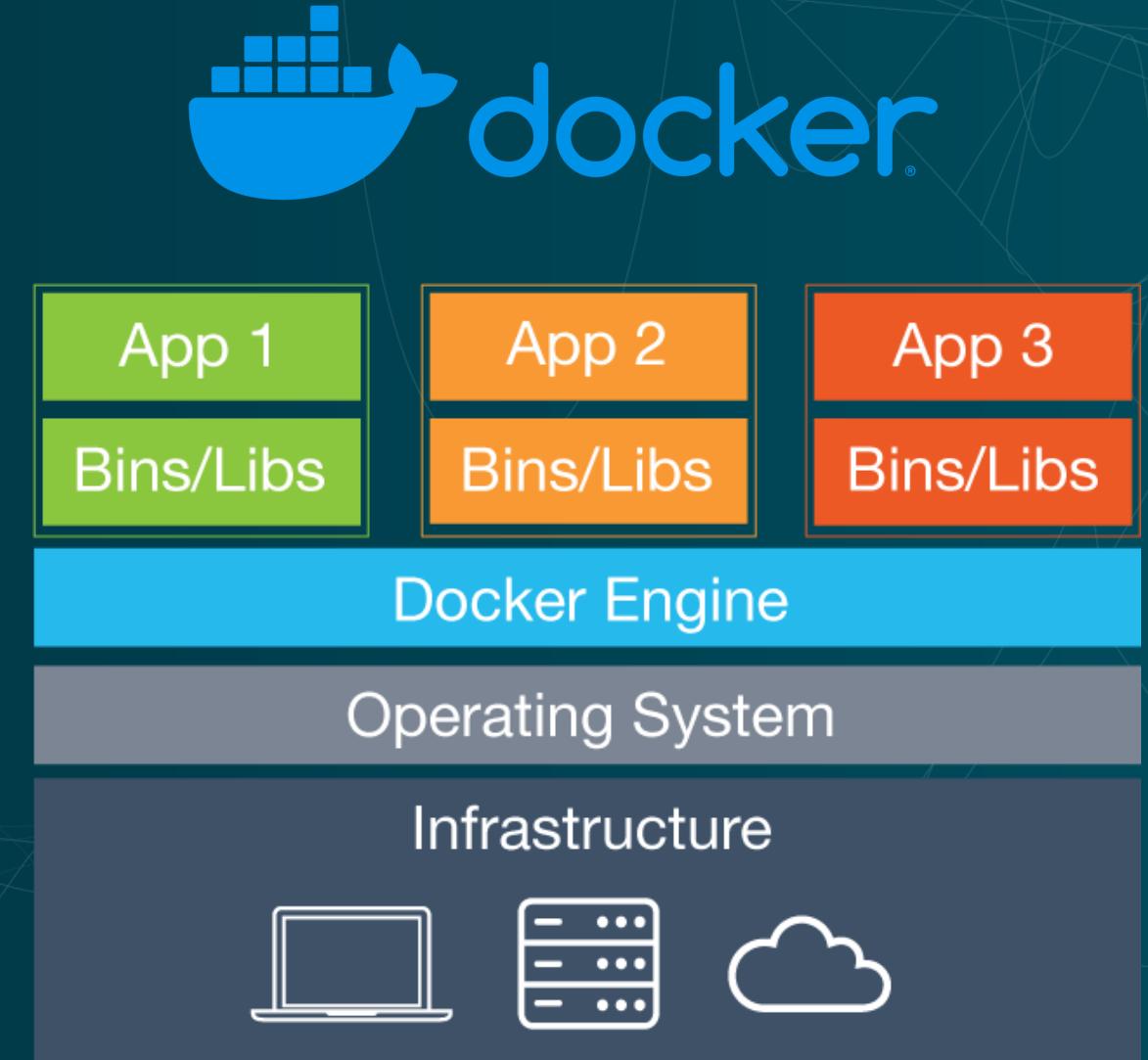
# Scrapli. Пример

```
▶ ⚡ ~/access-vlan-deployer > 🐍 master ?2 python3 examples/scrapli_example.py
Interface          Status      Protocol Description
Gi0/0              down       down     Tatooine
Gi0/1              down       down     Coruscant
Gi0/2              down       down     Naboo
Gi0/3              down       down     Kashyyyk
Gi1/0              down       down     Hoth
Gi1/1              down       down     Endor
Gi1/2              down       down     Mandalore
Gi1/3              up        up      Bespin
Vl10              up        up      up
```

# Docker

Платформа для сборки и запуска контейнеров.

- Обеспечивает изоляцию приложений. Каждое приложение работает в своей изолированной среде;
- Воспроизводимость среды. Образы собранные в Docker могут быть запущены на разных платформах и ОС где есть поддержка Docker;
- Образы хранятся в специальном репозитории, например Dockerhub или Github Container Registry (GCR)
- Простота запуска. Образ можно скачать и развернуть из него контейнер «по одному клику»;
- Есть возможность оркестрации через docker-compose;



# Docker. Запускаем в один клик

```
▶ Apple ➤ ~ docker run --rm -d -p 80:80 --name nginx nginx:1.13
Unable to find image 'nginx:1.13' locally
1.13: Pulling from library/nginx
f2aa67a397c4: Download complete
4a99993b8636: Download complete
3c091c23e29d: Download complete
Digest: sha256:b1d09e9718890e6ebbbd2bc319ef1611559e30ce1b6f56b2e3b479d9da51dc35
Status: Downloaded newer image for nginx:1.13
b282801e0224fe943f7718ed6c3e640c7563125fc39fc6df3e0224d74c46c990
▶ Apple ➤ ~ lynx -dump localhost
                                Welcome to nginx!

If you see this page, the nginx web server is successfully installed
and working. Further configuration is required.

For online documentation and support please refer to [1]nginx.org.
Commercial support is available at [2]nginx.com.

Thank you for using nginx.
```

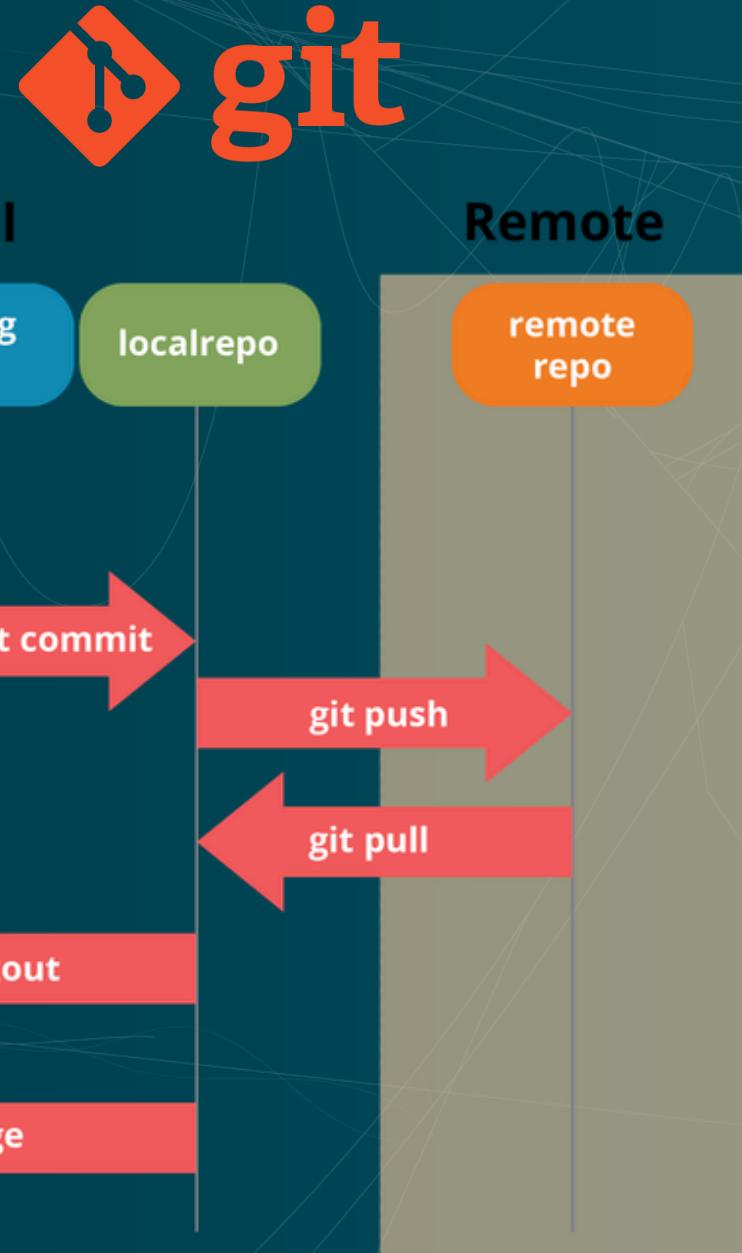
# Docker. CheetSheet

---

- docker ps – посмотреть запущенные контейнеры (--all, отобразить все);
- docker search <image> – поиск образа в репозитории (registry);
- docker pull – скачать образ (имя или имя:тег);
- docker images – список скаченных образов;
- docker rmi <image> - удалить образ;
- docker start/stop <container> – запустить/остановить контейнер;
- docker logs <container> - посмотреть его логи;
- docker exec –it <container> bash – зайти внутрь контейнера;
- docker build –t <new\_image> . – собрать свой образ из Dockerfile;
- docker network ls – посмотреть существующие сети;
- docker run – скачать образ и/или запустить контейнер.  
--name (имя контейнера), --rm (удалить по завершению)  
-d (запустить в фоне), -p 80:80 (пробросить 80 порт)  
-e VAR=123 (передать переменную окружения)  
--net (запуск в определенной сети, bridge, host, none);
- docker inspect <image/container/net/e.t.c> - подробная информация об объекте в Docker;

# Git

- Хранение истории изменений; кода/конфигураций/и т.д;
- Возможность сравнения версий;
- Возможность откатываться на предыдущие версии;
- Совместная одновременная работа над проектами;
- Возможность изолировать свои изменения в отдельной ветке;
- Поддержка слияний изменений между ветками;



# Git. CheetSheet

---

- git init – создает новый репозиторий;
- git clone <url> - клонировать существующий репозиторий;
- git add <file> - индексирует изменения (добавляет в стейдж);
- git status – текущий статус изменений;
- git diff – показать дифф между текущими изменениями и последним коммитом;
- git commit –m “My commit” - делает коммит проиндексированных изменений;
- git pull – забирает изменения из remote репозитория (fetch & merge);
- git push [<remote> <branch>] – отправляем свои изменения в репозиторий\*
- git switch –c <branch> - создать новую ветку;
- git switch <branch> - перейти в существующую;
- git checkout <commit|tag|file> - переключение между хешами коммитов, тегов, откат файлов;
- git reset <file|commit> - откат коммитов и индексированных изменений;
- git merge <branch> - мерджим удаленную ветку в текущую;
- git rebase <branch> - переписываем изменения своей ветки поверх <branch>;
- git stash – “спрятать” не проиндексированные изменения, например перед выполнением git pull;
- git stash pop – вернуть их обратно;

# Access VLAN Deployer. Из чего состоит?

---

- `templates/index.html` – Jinja шаблон HTML страницы проекта;
- `app.py` – основной файл для запуска WSGI приложения;
- `common.py` – общие вспомогательные функции;
- `netmiko_switch.py` - модуль с взаимодействием с оборудованием через Netmiko;
- `scrapli_switch.py` - модуль с взаимодействием с оборудованием через Scrapli;
- `Dockerfile` – специальный файл для сборки Docker образа;
- `requirements.txt` – файл с необходимыми зависимостями и модулями для проекта;
- `.gitignore` – особый файл для исключения «видимости» гитом файлов и каталогов;
- `examples/*` – Python скрипты рассмотренные в этой презентации;

# app.py - /<switch>

- Инициализируем экземпляр Flask приложения с именем файла (`__name__`);
- Объявляем view route который будет обрабатывать запрос. Значение переданное после корня – переменная, это fqdn или IP-адрес свича;
- Получаем со свича необходимые данные и при помощи Jinja рендерим страницу `index.html` с теми данными, что мы получили на предыдущем шаге;
- Если данных нет, считаем что до свича мы не дошли, возвращаем ошибку 404;

```
app = Flask(__name__)

@app.route("/<switch>")
def root(switch):
    ....
    Обрабатываем запрос на http://localhost:5000/cisco-sw
    ....
    data = get_switch_info(switch)
    if data:
        return render_template("index.html", **data)
    else:
        return "Can't get data from switch", 404
```

# app.py - /deploy-vlan

- Делаем еще один route, который будет обслуживать запрос на нажатие кнопки в web-форме. Данные с формы приходят в POST, поэтому определяем что route будет поддерживать именно этот режим;
- Берем данные и достаем оттуда switch, interfaces (приходят в формате JSON), vlans;
- Передаем данные функции deploy\_vlans которая выполняет конфигурацию свича;
- После делаем редирект обратно на /<switch>;
- Для целей разработки и отладки пишем app.run() для запуска встроенного в Flask сервера (только для отладки);

```
@app.route("/deploy-vlan", methods=["POST"])
def deploy():
    """
    Обрабатываем нажатие кнопки
    (POST запрос в http://localhost:5000/deploy-vlan)
    """

    form_data = request.form.copy()
    switch = form_data.pop("switch")
    # данные берутся из формы {{ interfaces | toJSON | safe }}
    # если не преобразовывать в json, то вернется строка
    # и ее придется прогонять через eval, что не рекомендуется
    interfaces = json.loads(form_data.pop("interfaces"))
    vlans = form_data
    deploy_vlan(switch, interfaces, vlans)
    return redirect(f"/{switch}")

if __name__ == "__main__":
    app.run(debug=True, host="0.0.0.0")
```

# common.py – получаем креды

- Функция `get_auth` достает переменные окружения из командной оболочки (хранить креды в коде – моветон 😊);
- Пытаемся загрузить их из `auth.env` в папке где лежит `common.py`;
- `Path(__file__).parent` буквально означает возьми путь от `common.py` возьми директорию где он лежит и добавь `auth.py`. Если файла нет то просто игнорируем и идем дальше;
- Теперь пытаемся достать переменную окружения из `value`, и если все не получилось – генерим ошибку, если все ок – возвращаем результат;

```
def get_auth(value):
    """
    Сначала пробуем выгрузить переменную окружения
    из файла auth.env, потом берет ее из переменной
    окружения (если файл отсутствует – игнорируется)
    """
    load_dotenv(Path(__file__).parent / "auth.env")
    var = getenv(value)
    if not var:
        raise Exception(f"Missing environment variable: {value}")
    return var
```

# common.py – парсеры для данных

- Нам для нашего проекта нужны descriptions с порта и список access вланов;
- Но в чистом виде из cli их не получить, мы берем нужных команд и прогоняем через соответствующие регулярки;
- Потом через re.findall находим соответствующие паттерну данные и далее возвращаем.
- Так как это многострочный вывод используем re.MULTILINE для корректной работы.

```
def parse_interface_description(interfaces):
    """
    Парсит description интерфейсов и возвращает словарь с информацией о них
    в виде {"Gi0/0": "Tatooine", "Gi0/1": "Coruscant"...}
    """

    regex = re.compile(r"(Gi[0-9]/[0-9])\s+\S+\s+\S+\s+(\S+)", re.MULTILINE)
    interfaces = regex.findall(interfaces)
    return dict(interfaces)

def parse_vlan_brief(vlans):
    """
    Парсит VLANs и возвращает список кортежей с информацией о них в виде
    [("10", "management", "Gi1/3"), ("30", "luke_skywalker", "Gi0/1, Gi0/3")]
    """

    regex = re.compile(
        r"(\d+)\s+(\S+)\s+\S+\s+((?:Gi[0-9]/[0-9](?:, )?)+)", re.MULTILINE
    )
    vlans = regex.findall(vlans)
    return vlans
```

# common.py – подготавливаем данные

- Функция нужна нам для того, чтобы объединить получившиеся данные в один словарь вида:

```
{  
    "Gi0/0": {  
        "description": "Imperial",  
        "current_vlan": "123"  
    },  
    "Gi0/1": {  
        "description": "Rebels",  
        "current_vlan": "321"  
    },  
    ...  
}
```

```
def prepare_data(data):  
    """  
    Подготавливает данные для вывода на страницу  
    """  
  
    for iface, description in data["interfaces"].items():  
        for vlan in data["vlans"]:  
            if iface in vlan[2]:  
                current_vlan = vlan[0]  
                break  
        data["interfaces"][iface] = {  
            "description": description,  
            "current_vlan": current_vlan,  
        }  
    return data
```

# netmiko\_switch.py – параметры

- Подготавливаем данные для подключения к оборудованию через Netmiko;
- Тут нужно определить тип устройства fqdn/ip, и значения переменных окружения которые мы достаем при помощи функции get\_auth из common.py;

```
def device_params(switch):
    """
    Возвращает информацию о устройстве необходимую
    для подключения к свичу через Netmiko с типом
    cisco_ios и хостом из переменной switch
    """
    return {
        "device_type": "cisco_ios",
        "host": switch,
        "username": get_auth("USERNAME"),
        "password": get_auth("PASSWORD"),
    }
```

# netmiko\_switch.py – получение вывода

- Тут мы выполняем подключение к сетевому оборудованию при помощи контекстного менеджера (`with`), это избавляет нас от необходимости следить за открытием и закрытием сессии;
- Получаем вывод нужных команд и формируем итоговый словарь `data` с fqdn/ip свича, спаршенных данных по выводу дескрипшенов интерфейсов, подготавливаем и возвращаем данные с помощью `prepare_data`;
- Если мы не смогли подключиться выкидываем исключение с описанием ошибки;

```
def get_switch_info(switch):
    """
    Возвращает результат выполнения команд выполненных на устройстве
    Если что-то ломается в процессе – то сообщаем об ошибке
    """

    try:
        # Подключаемся к устройству
        with ConnectHandler(**device_params(switch)) as conn:
            # Выполняем команды
            interfaces = conn.send_command("show interface description")
            vlans = conn.send_command("show vlan brief | exclude unsup")
            data = {
                "switch": switch,
                "interfaces": parse_interface_description(interfaces),
                "vlans": parse_vlan_brief(vlans),
            }
        return prepare_data(data)
    except NetmikoBaseException as error:
        print(f"Ошибка подключения: {error}")
```

# netmiko\_switch.py – выкатываем вланы

- Прожигаем конфиг свича новыми вланами которые получили от пользователя из формы;
- При этом смотрим только на те вланы которые изменились, чтобы не выкатывать уже имеющуюся конфигурацию;
- Формируем из данных набор команд, которые будут выкачены на оборудование;
- Далее при помощи netmiko функции send\_config\_set Выкатываем получившуюся конфигурацию на свич;

```
def deploy_vlan(switch, interfaces, vlans):  
    """  
    Деплоим VLANы на порты свича  
    """  
  
    cmd = []  
    # Формируем команды  
    for iface, new_vlan in vlans.items():  
        # добавляем в список команд, если текущий VLAN не равен новому  
        # чтобы не применять уже примененную конфигурацию  
        if interfaces[iface]["current_vlan"] != new_vlan:  
            cmd.append(f"interface {iface}")  
            cmd.append(f"switchport access vlan {new_vlan}")  
    # Выполняем команды  
    with ConnectHandler(**device_params(switch)) as conn:  
        conn.send_config_set(cmd)
```

# scrapli\_switch.py – параметры

- Данные для подключения при помощи scrapli похожи, но есть отличия  
Так как мы идем по паролю, то  
отключим жесткую привязку логина  
по ключу и скажем явно читать  
конфиг OpenSSH клиента в системе  
(~/.ssh/config);
- Так же обратите внимание что профиль  
оборудования для Cisco IOS называется  
cisco\_iosxe;

```
def device_params(switch):
    """
    Возвращает информацию о устройстве необходимую
    для подключения к свичу через Scrapli с типом
    cisco_iosxe и хостом из переменной switch
    Так как у нас логин/пароль, отключаем
    жестко заданную аутентификацию по ключу
    И говорим что нужно читать ~/.ssh/config
    перед подключением
    """

    return {
        "platform": "cisco_iosxe",
        "host": switch,
        "auth_username": get_auth("USERNAME"),
        "auth_password": get_auth("PASSWORD"),
        "auth_strict_key": False,
        "ssh_config_file": True,
    }
```

# scrapli\_switch.py – получение вывода

- За небольшим исключением процесс получения данных практически идентичный из примера netmiko;
- Для удобства все функции из scrapli\_switch.py и netmiko\_switch.py имеют одинаковые названия что позволяет просто управлять транспортом, комментируя нужную строчку в app.py;

```
def get_switch_info(switch):
    """
    Возвращает результат выполнения команд выполненных на устройстве
    Если что-то ломается в процессе – то сообщаем об ошибке
    """

    try:
        # Подключаемся к устройству
        with Scrapli(**device_params(switch)) as conn:
            # Выполняем команды
            interfaces = conn.send_command("show interface description").result
            vlans = conn.send_command("show vlan brief | exclude unsup").result
            data = {
                "switch": switch,
                "interfaces": parse_interface_description(interfaces),
                "vlans": parse_vlan_brief(vlans),
            }
            return prepare_data(data)
    except ScrapliException as error:
        print(f"Ошибка подключения: {error}")
```

# scrapli\_switch.py – выкатываем вланы

- Таким же образом катим новые вланы на свитч – почти один в один с функцией в netmiko\_switch.py;

```
def deploy_vlan(switch, interfaces, vlans):  
    """  
    Деплоим VLANы на порты свича  
    """  
  
    cmd = []  
    # Формируем команды  
    for iface, new_vlan in vlans.items():  
        # добавляем в список команд, если текущий VLAN не равен новому  
        # чтобы не применять уже примененную конфигурацию  
        if interfaces[iface]["current_vlan"] != new_vlan:  
            cmd.append(f"interface {iface}")  
            cmd.append(f"switchport access vlan {new_vlan}")  
    # Выполняем команды  
    with Scrapli(**device_params(switch)) as conn:  
        conn.send_configs(cmd).result
```

# index.html – наша страничка

- Приведена наиболее интересная часть html странички где мы используем jinja шаблон для рендеринга страницы;
- Как мы и говорили ранее у нас в форму попадает switch, interfaces и vlans;
- Тут мы проходимся по ним и генерим табличку при помощи HTML тегов;
- Теги довольно простые просто зайдите на сайт htmlbook и ознакомьтесь про интересующий тег (или попросите ИИ 😊);

```
<form action="/deploy-vlan" method="post">
    <h3>Access VLAN Deployment for {{ switch }}</h3>
    <table border="1" align="left" cellpadding="2" cellspacing="2" width="30%">
        <tr><th> Port </th><th> Description </th><th> Current VLAN </th><th> New VLAN </th></tr>
        {% for iface, iface_data in interfaces.items() %}
        <tr>
            <td> {{ iface }} </td>
            <td> {{ iface_data.description }} </td>
            <td> {{ iface_data.current_vlan }} </td>
            <td>
                <select name="{{ iface }}" style="margin-right: 5px;">
                    {% for vlan_id, vlan_name, _ in vlans %}
                    <option value="{{ vlan_id }}" {% if vlan_id == iface_data.current_vlan %}selected{% endif %}>
                        {{ vlan_id }} ({{ vlan_name }})
                    </option>
                    {% endfor %}
                </select>
            </td>
        </tr>
        {% endfor %}
    </table>
    <div style="clear: both;"></div>
    <br>
    <input type="hidden" name="switch" value="{{ switch }}">
    <textarea name="interfaces" style="display: none;">{{ interfaces | tojson | safe }}</textarea>
    <input type="submit" value="Катим" />
</form>
```

# index.html – Инициировано python3 app.py....

- Запускаем flask на встроенным сервере (Flask любезно предупреждает не использовать этот сервер для прода 😊);
- Заходим на адрес localhost:5000/cisco-sw;

```
▶ Apple-MacBook-Pro:~/access-vlan-deployer master python3 app.py
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://100.123.0.70:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 881-699-666
```

# Тадааам!

- Отображается табличка с нашими портами, дескрипшеном текущим вланом и селектором для выбора нового влана (при этом селектор принимает значение текущего VLAN на порту);

← → ⌂ http://localhost:5000/cisco-sw

### Access VLAN Deployment for cisco-sw

Port	Description	Current VLAN	New VLAN
Gi0/0	Tatooine	30	30 (luke_skywalker) ▾
Gi0/1	Coruscant	30	30 (luke_skywalker) ▾
Gi0/2	Naboo	40	40 (ioda) ▾
Gi0/3	Kashyyyk	20	20 (obi_wan) ▾
Gi1/0	Hoth	40	40 (ioda) ▾
Gi1/1	Endor	10	10 (management) ▾
Gi1/2	Mandalore	50	50 (dart_waider) ▾
Gi1/3	Bespin	10	10 (management) ▾

Катим

# Тадааам!

- В селекторе только те вланы которые мы спарсили при помощи show vlan brief (исключив всякое легаси для token ring, fddi при помощи exclude unsup);

← → ⌂ http://localhost:5000/cisco-sw

### Access VLAN Deployment for cisco-sw

Port	Description	Current VLAN
Gi0/0	Tatooine	30
Gi0/1	Coruscant	30
Gi0/2	Naboo	40
Gi0/3	Kashyyyk	20
Gi1/0	Hoth	40
Gi1/1	Endor	10
Gi1/2	Mandalore	50
Gi1/3	Bespin	10

1 (default)  
10 (management)  
20 (obi\_wan)  
30 (luke\_skywalker)  
 40 (ioda)  
50 (dart\_waider)

40 (ioda) ▾  
10 (management) ▾  
50 (dart\_waider) ▾  
10 (management) ▾

Катим

# Тадааам!

- Давайте выселим Люка Скайвокера из Татуина и Корусанта и отправим туда Дарта Вейдера (Gi0/0, Gi0/1) – наведем шухера в этой галактике!
- И прожимаем кнопку «Катим!»
- Ждем пару секунд...
- Звезда смерти уже в пути...
- Империя празднует победу!  
(Йода с облегчением выдохнул)

← → ⌂ http://localhost:5000/cisco-sw

### Access VLAN Deployment for cisco-sw

Port	Description	Current VLAN
Gi0/0	Tatooine	30
Gi0/1	Coruscant	30
Gi0/2	Naboo	40

1 (default)  
10 (management)  
20 (obi\_wan)  
**✓ 30 (luke\_skywalker)**  
40 (ioda)  
**50 (dart\_waider)**

Port	Description	Current VLAN	New VLAN
Gi0/0	Tatooine	50	50 (dart_waider) ▾
Gi0/1	Coruscant	50	50 (dart_waider) ▾
Gi0/2	Naboo	40	40 (ioda) ▾

# ЧТО-ТО СМУЩАЕТ....

- Давайте взглянем еще раз на страничку...
- Что-то смущает...
- Ах да...дизайн времен выхода первых частей звездных войн!
- Пойдем наводить красоту!

← → ⌂ http://localhost:5000/cisco-sw

### Access VLAN Deployment for cisco-sw

Port	Description	Current VLAN	New VLAN
Gi0/0	Tatooine	50	50 (dart_waider) ▾
Gi0/1	Coruscant	50	50 (dart_waider) ▾
Gi0/2	Naboo	40	40 (ioda) ▾
Gi0/3	Kashyyyk	20	20 (obi_wan) ▾
Gi1/0	Hoth	40	40 (ioda) ▾
Gi1/1	Endor	10	10 (management) ▾
Gi1/2	Mandalore	50	50 (dart_waider) ▾
Gi1/3	Bespin	10	10 (management) ▾

Катим

# Шаг #1: CTRL+C

```
templates > index.html
You, 15 minutes ago | 1 author (You)
1 <!DOCTYPE HTML>          Add to chat (Cmd+L) | Edit highlighted code (Cmd+I).
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>Access VLAN Deployment</title>
6   </head>
7   <body>
8     <form action="/deploy-vlan" method="post">
9       <h3>Access VLAN Deployment for {{ switch }}</h3>
10      <table border="1" align="left" cellpadding="2" cellspacing="2" width="30%">
11        <tr><th> Port </th><th> Description </th><th> Current VLAN </th><th> New VLAN </th></tr>
12        {% for iface, iface_data in interfaces.items() %}<br>
13          <tr>
14            <td> {{ iface }} </td>
15            <td> {{ iface_data.description }} </td>
16            <td> {{ iface_data.current_vlan }} </td>
17            <td>
18              <select name="{{ iface }}" style="margin-right: 5px;">
19                {% for vlan_id, vlan_name, _ in vlans %}<br>
20                  <option value="{{ vlan_id }}" {% if vlan_id == iface_data.current_vlan %}selected{% endif %}>
21                    {{ vlan_id }} ({{ vlan_name }})
22                  </option>
23                {% endfor %}
24              </select>
25            </td>
26          </tr>
27        {% endfor %}
28      </table>
29      <div style="clear: both;"></div>
30      <br>
31      <input type="hidden" name="switch" value="{{ switch }}>
32      <textarea name="interfaces" style="display: none;">{{ interfaces | tojson | safe }}</textarea>
33      <input type="submit" value="Катим" />
34    </form>
35  </body>
36 </html>
```

Add to Chat ⌂ Quick Edit ⌂

# Шаг #2: CTRL+V плюс prompt

```
<textarea name="interfaces" style="display: none;">{{ interfaces | toJson | safe }}</textarea>
<input type="submit" value="Катим" />
</form>
</body>
</html>
```

Братан, сделай по красоте!

 DeepThink (R1)

 Search

0

↑

AI-generated, for reference only

# Шаг #3: ...

templates >pretty-index.html

You, last week | 1 author (You)

```
1  <!DOCTYPE HTML>
2  <html lang="ru">
3  <head>
4      <meta charset="utf-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1">
6      <title>Deploy VLAN | {{ switch }}</title>
7      <style>
8          :root {
9              --primary: #4361ee;
10             --secondary: #3f37c9;
11             --accent: #4895ef;
12             --light: #f8f9fa;
13             --dark: #212529;
14             --success: #4cc9f0;
15             --border-radius: 8px;
16             --shadow: 0 4px 12px rgba(0, 0, 0, 0.1);
17             --transition: all 0.3s cubic-bezier(0.25, 0.8, 0.25, 1);
```

# Шаг #4: PROFIT!

Настройка VLAN на cisco-sw

Порт	Описание	Текущий VLAN	Новый VLAN
Gi0/0	Tatooine	10	10 (management)
Gi0/1	Coruscant	30	30 (luke_skywalker)
Gi0/2	Naboo	10	10 (management)
Gi0/3	Kashyyyk	40	40 (ioda)
Gi1/0	Hoth	20	20 (obi_wan)
Gi1/1	Endor	10	10 (management)
Gi1/2	Mandalore	50	50 (dart_waider)
Gi1/3	Bespin	10	10 (management)

**ПРИМЕНить измЕнЕНИЯ**

# Что дальше? А дальше – больше!

---



# Dockerfile – описываем будущий образ

- Образы пишутся не с нуля а собираются на основе других имеющихся образов;
- В нашем случае это легковесный образ `python:3.11-slim`, где `python` – образ, а `3.11-slim` – его тег;
- Так как это «голый» образ качаем туда `openssh` и формируем конфиг (старые циски не дружат с современным `openssh`, и нужно явно указывать устаревшие алгоритмы);
- Устанавливаем права;

```
# используем легковесный образ python 3.11
FROM python:3.11-slim

# устанавливаем системные зависимости для SSH
# и чистим кэш пакетов для оптимизации размера образа
RUN apt-get update && apt-get install -y openssh-client && rm -rf /var/lib/apt/lists/*

RUN mkdir -p ~/.ssh
RUN cat <<EOF > ~/.ssh/config
Host cisco-sw
    HostName cisco-sw
    User cisco
    KexAlgorithms diffie-hellman-group-exchange-sha1
    HostKeyAlgorithms ssh-rsa
EOF

RUN chmod 600 ~/.ssh/config
```

# Dockerfile – описываем будущий образ

- Устанавливаем рабочий каталог /app (внутри контейнера);
- Копируем файлы проекта в /app из нашего проекта;
- Запускаем установку зависимостей для проекта;
- Пробрасываем 5000 порт «наружу» чтобы контейнер мог слушать входящие подключения на нем;
- Выполняем команду для запуска нашего приложения через gunicorn на 5000 порту (так же можете протестировать локально);

```
# устанавливаем рабочую директорию в /app
WORKDIR /app

# копируем все файлы из корневой директории в /app
COPY . .

# устанавливаем зависимости из requirements.txt
RUN pip install -r requirements.txt

# пробрасываем порт 5000 из контейнера наружу
EXPOSE 5000

# запускаем приложение при старте контейнера
CMD ["gunicorn", "--log-level", "debug", "app:app", "--bind", "0.0.0.0:5000"]
```

# Dockerfile – пробуем собрать образ

```
▶ ~ ~/access-vlan-deployer > 🐳 ⚒ master docker build -t access-vlan-deployer .
[+] Building 79.6s (13/13) FINISHED                                            docker:desktop-linux
=> [internal] load build definition from Dockerfile                         0.0s
=> => transferring dockerfile: 1.36kB                                       0.0s
=> [internal] load metadata for docker.io/library/python:3.11-slim        1.2s
=> [internal] load .dockerignore                                         0.0s
=> => transferring context: 2B                                           0.0s
=> [1/8] FROM docker.io/library/python:3.11-slim@sha256:dbf1de478a55d6763afaa39c2f3d7b54b252306149802 0.0s
=> => resolve docker.io/library/python:3.11-slim@sha256:dbf1de478a55d6763afaa39c2f3d7b54b252306149802 0.0s
=> [internal] load build context                                         0.4s
=> => transferring context: 1.28MB                                      0.4s
=> CACHED [2/8] RUN apt-get update && apt-get install -y openssh-client && rm -rf /var/lib/apt/lists/ 0.0s
=> CACHED [3/8] RUN mkdir -p ~/.ssh                                       0.0s
=> CACHED [4/8] RUN cat <<EOF > ~/.ssh/config                           0.0s
=> CACHED [5/8] RUN chmod 600 ~/.ssh/config                             0.0s
=> CACHED [6/8] WORKDIR /app                                              0.0s
=> [7/8] COPY . .                                                       0.4s
=> [8/8] RUN pip install -r requirements.txt                            75.1s
=> exporting to image                                                 2.3s
=> => exporting layers                                              2.3s
=> => exporting manifest sha256:0922e411288ddc10f83299631caa2bf6c2cc74c7b9a16b9631caa8d37efbdd85 0.0s
=> => exporting config sha256:bbd767e8d8367ad28c056eed022f36711bf9465dfe334d0298f5d65965c93d8e 0.0s
=> => exporting attestation manifest sha256:b1f7bc36241559152b48facc05795d07e263b6ba7e683050587c8cd0f 0.0s
=> => exporting manifest list sha256:1f6dd38fade3d6ba62322477fc48ab658b970b0e1cbf6192230dd229e57aef36 0.0s
=> => naming to docker.io/library/access-vlan-deployer:latest           0.0s
```

# Dockerfile – запускаем контейнер

```
▶ Apple ➜ ~/access-vlan-deployer ➜ 🐳 master docker run --rm -d --name avd01 \
-e USERNAME=cisco -e PASSWORD=cisco -p 8080:5000 access-vlan-deployer
473fcf41a10f50bf69be4c64bf5329ce6ae65a93a65f7e8665cf86eddc8f04b0
▶ Apple ➜ ~/access-vlan-deployer ➜ 🐳 master
▶ Apple ➜ ~/access-vlan-deployer ➜ 🐳 master docker ps
CONTAINER ID   IMAGE          COMMAND           CREATED        STATUS       PORTS          NAMES
473fcf41a10f   access-vlan-deployer "gunicorn --log-leve..." 9 seconds ago  Up 8 seconds  0.0.0.0:8080->5000/tcp   avd01
▶ Apple ➜ ~/access-vlan-deployer ➜ 🐳 master
▶ Apple ➜ ~/access-vlan-deployer ➜ 🐳 master docker logs -f avd01
[2025-05-31 12:59:21 +0000] [1] [DEBUG] Current configuration:
  config: ./gunicorn.conf.py
  wsgi_app: None
  bind: ['0.0.0.0:5000']
  backlog: 2048
  workers: 1
  worker_class: sync
  threads: 1
  worker_connections: 1000
  max_requests: 0
  max_requests_jitter: 0
  timeout: 30
  graceful_timeout: 30
  ✓ < py312 ✨ < 15:58:40 ⚡
  ✓ < py312 ✨ < 15:59:20 ⚡
  ✓ < py312 ✨ < 15:59:21 ⚡
  ✓ < py312 ✨ < 15:59:29 ⚡
  ✓ < py312 ✨ < 15:59:31 ⚡
```

# Dockerfile – проверяем

localhost:8080/cisco-sw

### Настройка VLAN на cisco-sw

Порт	Описание	Текущий VLAN	Новый VLAN
Gi0/0	Tatooine	50	50 (dart_waider)
Gi0/1	Coruscant	50	50 (dart_waider)
Gi0/2	Naboo	40	40 (ioda)
Gi0/3	Kashyyyk	20	20 (obi_wan)
Gi1/0	Hoth	40	40 (ioda)
Gi1/1	Endor	10	10 (management)
Gi1/2	Mandalore	50	50 (dart_waider)
Gi1/3	Bespin	10	10 (management)

ПРИМЕНИТЬ ИЗМЕНЕНИЯ

# Бонус для Netbox: делаем кастом линк

**access-vlan-deployer**  
Created 2025-05-31 13:14 · Updated 2025-05-31 13:15

Custom Link   Changelog

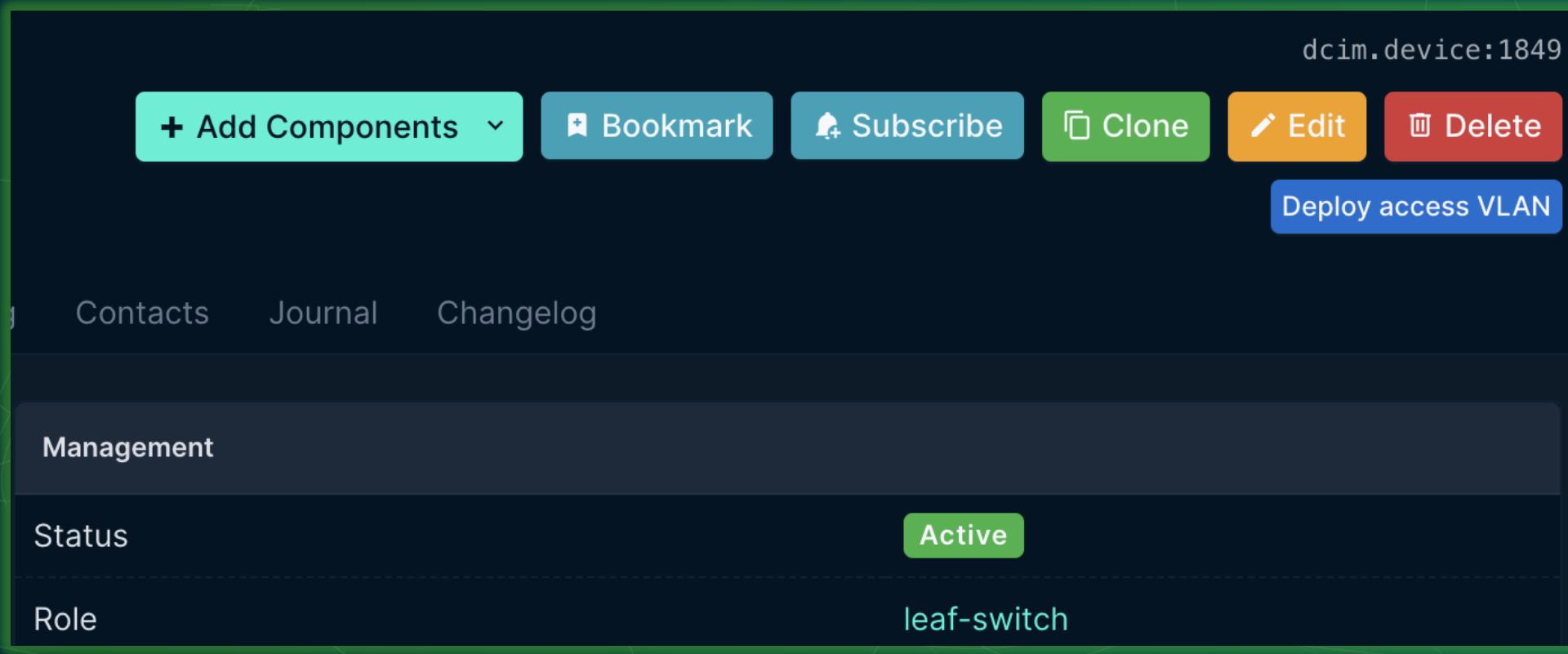
Custom Link	
Name	access-vlan-deployer
Enabled	✓
Group Name	—
Weight	100
Button Class	Blue
New Window	✓

Assigned Models  
DCIM | device

Link Text  
Deploy access VLAN

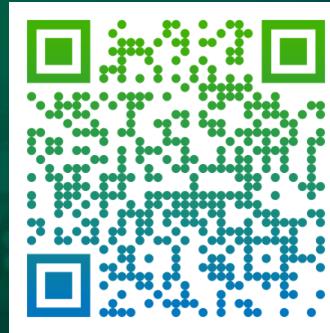
Link URL  
`http://localhost:8080/{{ object.name }}`

# Бонус для Netbox: делаем кастом линк



# Спасибо за внимание!

Буду рад ответить на все ваши вопросы сейчас или свяжитесь со мной в будущем:



**Роман Карапланов**

[ebgp@yandex.ru](mailto:ebgp@yandex.ru)

+7 977 266 55 16

[@AnteronWrk](https://www.instagram.com/anteronwrk)