Az **automatizált konfiguráció** és a **hagyományos kézi konfiguráció** között jelentős különbségek vannak hatékonyság, skálázhatóság és hibakezelés terén. Az alábbi összehasonlítás segít megérteni, mikor melyik megközelítést érdemes használni Cisco routerek esetében.

**1. Hagyományos kézi konfiguráció**

**Előnyök:**

* **Egyszerűség:** Nincs szükség szoftveres ismeretekre, csak CLI parancsokra.
* **Pontos kontroll:** Az adminisztrátor közvetlenül látja és irányítja minden konfigurációs lépést.
* **Hibák gyors felismerése:** Azonnali visszajelzés a CLI-ből, könnyű diagnosztizálni problémákat.

**Hátrányok:**

* **Időigényes:** Nagyobb hálózatoknál rengeteg manuális munkát igényel.
* **Hibalehetőség:** Emberi hibák (pl. elírás, kihagyott parancsok) könnyen előfordulhatnak.
* **Nehezen reprodukálható:** Minden konfigurációt egyenként kell végrehajtani, ami bonyolulttá teszi a folyamatok megismétlését.
* **Dokumentáció hiánya:** Nincs automatikusan naplózott konfigurációs változás.

**Példa hagyományos konfigurációra (CLI):**

Router> enable

Router# configure terminal

Router(config)# hostname MyRouter

Router(config)# interface GigabitEthernet0/0

Router(config-if)# ip address 192.168.1.1 255.255.255.0

Router(config-if)# no shutdown

Router(config-if)# exit

Router(config)# write memory

**2. Automatizált konfiguráció (pl. Netmiko, API, Ansible)**

**Előnyök:**

* **Gyorsaság:** Több eszköz egyszerre konfigurálható, időmegtakarítás nagy hálózatoknál.
* **Reprodukálhatóság:** Az automatizált szkriptek újrahasználhatók, biztosítva a következetességet.
* **Hibák csökkentése:** Automatizált logika minimalizálja az emberi hibákat.
* **Skálázhatóság:** Könnyen alkalmazható több száz vagy ezer eszközre.
* **Központi menedzsment:** Egyetlen scriptből kezelhető az egész infrastruktúra.

**Hátrányok:**

* **Tanulási görbe:** Szükség van programozási (pl. Python) és automatizálási ismeretekre.
* **Hibakeresés komplexitása:** Nehezebb lehet azonosítani a hibákat, ha az automatizálás sikertelen.
* **Kezdeti beállítási idő:** Automatizálási környezet (pl. Netmiko, Ansible, API) konfigurálása több időt vehet igénybe.

**Példa Netmiko-val (Python):**

from netmiko import ConnectHandler

device = {

'device\_type': 'cisco\_ios',

'host': '192.168.1.1',

'username': 'admin',

'password': 'password',

}

connection = ConnectHandler(\*\*device)

connection.enable()

commands = [

'hostname MyRouter',

'interface GigabitEthernet0/0',

'ip address 192.168.1.1 255.255.255.0',

'no shutdown',

]

output = connection.send\_config\_set(commands)

print(output)

connection.disconnect()

**Mikor melyiket érdemes használni?**

* **Kisebb hálózatok (1-5 eszköz):**  
  Hagyományos CLI konfiguráció egyszerű és gyors.
* **Közepes hálózatok (5-50 eszköz):**  
  Automatizálás (Netmiko, Ansible) jelentős időt takarít meg.
* **Nagyvállalati környezet (>50 eszköz):**  
  API-alapú vagy Ansible automatizálás elengedhetetlen.

**Kombinált megközelítés**

Sok szervezet **hibrid modellt** alkalmaz:

* **Automatizálás** az ismétlődő, időigényes feladatokhoz (pl. VLAN konfiguráció, IP beállítások).
* **Kézi CLI** finomhangolásokhoz és hibakereséshez.

**3. Az automatizált hálózatkonfiguráció megvalósítási lehetőségei**

Az automatizált konfiguráció Cisco eszközökön többféleképpen is megvalósítható. A két leggyakoribb módszer:

1. **API-alapú megközelítés (pl. RESTCONF, NETCONF, Cisco DNAC API)**
2. **SSH-alapú automatizálás (pl. Netmiko könyvtár használatával)**

Nézzük meg a részletes összehasonlítást:

**3.1. API-alapú konfiguráció (pl. RESTCONF, NETCONF, Cisco DNAC API)**

Az API-alapú megközelítés a Cisco eszközök automatizált konfigurálásában azt jelenti, hogy a hálózati eszközöket közvetlenül egy **programozható felületen (API-n, Application Programming Interface)** keresztül irányítjuk. Ez egy modern módszer, amely lehetővé teszi, hogy alkalmazások, szkriptek vagy automatizálási rendszerek kommunikáljanak az eszközökkel anélkül, hogy hagyományos parancssoros interfészt (CLI-t) kellene használni.

**Mi az az API pontosan?**

* Az API egy szabályrendszer, amely meghatározza, hogyan kommunikálhatnak egymással különböző szoftverek.
* Cisco eszközök esetében gyakran REST API-t használnak, ami HTTP protokollon keresztül működik, hasonlóan a webes alkalmazásokhoz.
* Az eszközök API-hívások formájában fogadnak kéréseket (pl. GET, POST, PUT, DELETE), és JSON formátumban küldenek választ.

**Előnyök:**

* **Strukturált adatkezelés:** JSON, XML vagy YAML formátumú adatátvitel, ami könnyen feldolgozható.
* **Gyorsaság:** Az API-k általában gyorsabbak, mivel közvetlenül az eszköz szoftveres rétegével kommunikálnak.
* **Biztonság:** HTTPS (TLS) alapú kommunikáció, erős hitelesítési lehetőségekkel.
* **Modern automatizáció:** Könnyen integrálható DevOps eszközökkel (Ansible, Terraform, Jenkins).
* **Párhuzamos műveletek:** Nagy számú eszköz egyszerre történő konfigurálása hatékonyabban végezhető.

**Hátrányok:**

* **Komplexitás:** Több beállítást igényel (API engedélyezése, tokenek kezelése stb.).
* **Támogatottság:** Csak az újabb Cisco eszközök és IOS verziók támogatják (pl. IOS XE).
* **Tanulási görbe:** Az API-k használata (pl. RESTCONF/NETCONF) kezdetben bonyolultabb lehet a hagyományos CLI-hez képest.

**API típusok Cisco eszközökön:**

1. **RESTCONF API:**
   * Modern Cisco IOS XE eszközökön elérhető.
   * HTTP/HTTPS protokollt és JSON/XML formátumot használ.
2. **NETCONF API:**
   * XML-alapú protokoll, YANG modellekkel együtt.
   * Nagyon részletes konfigurációs lehetőségeket kínál.
3. **Cisco DNA Center API:**
   * Nagyvállalati hálózatok központi menedzsmentjére.
   * Felhőalapú és hibrid hálózatok automatizálásához.
4. **Meraki Dashboard API:**
   * Cisco Meraki eszközök felhőalapú menedzselésére.
   * RESTful API JSON formátummal.

**API-alapú automatizálás működése:**

1. **Hitelesítés:**  
   API-token, felhasználónév/jelszó vagy OAuth 2.0 alapú hitelesítés.
2. **Kérés küldése:**  
   A kliens (pl. Python szkript) HTTP kérést küld az eszköz API-jához:
   * **GET** – adatok lekérdezése
   * **POST** – új konfiguráció létrehozása
   * **PUT/PATCH** – meglévő konfiguráció módosítása
   * **DELETE** – konfiguráció törlése
3. **Válasz fogadása:**  
   Az eszköz JSON-formátumban válaszol, amely tartalmazza az eredményeket vagy hibakódokat.

**Egyszerű API példa (Python + REST API):**

import requests

from requests.auth import HTTPBasicAuth

# Eszköz IP-címe és hitelesítési adatok

url = "https://192.168.1.1/restconf/data/Cisco-IOS-XE-native:native/hostname"

username = "admin"

password = "password"

# API kérés fejléce (JSON használata)

headers = {

"Accept": "application/yang-data+json",

"Content-Type": "application/yang-data+json"

}

# Új hostname beállítása

payload = {

"Cisco-IOS-XE-native:hostname": "NewRouter"

}

# PUT kérés küldése az API-hoz

response = requests.put(url, auth=HTTPBasicAuth(username, password), headers=headers, json=payload, verify=False)

# Eredmény kiíratása

if response.status\_code == 204:

print("Sikeres konfiguráció!")

else:

print(f"Hiba történt: {response.status\_code} - {response.text}")

**3.2. SSH-alapú automatizálás (Netmiko használatával)**

**Előnyök:**

* **Egyszerű használat:** CLI-parancsokat küld közvetlenül, ami ismerős a hagyományos hálózati adminisztrátoroknak.
* **Kompatibilitás:** Működik régebbi eszközökön is, ahol nincs API-támogatás.
* **Gyors beállítás:** Nem kell API-t engedélyezni az eszközön, elég az SSH kapcsolat.

**Hátrányok:**

* **Sebesség:** Lassabb, mivel szimulálja az emberi CLI-műveleteket (parancs beírás, válasz olvasás).
* **Hibakezelés:** Nehezebb azonosítani a pontos hibákat, mivel CLI output alapján kell dolgozni.
* **Skálázhatóság:** Nagy számú eszköz esetén kevésbé hatékony, mivel minden parancs külön SSH-kapcsolatot igényel.

**Példa Netmiko használatra:**

from netmiko import ConnectHandler

device = {

'device\_type': 'cisco\_ios',

'host': '192.168.1.1',

'username': 'admin',

'password': 'password',

}

connection = ConnectHandler(\*\*device)

connection.enable()

# Hostname beállítása

commands = [

'configure terminal',

'hostname NewRouterName',

'end',

'write memory'

]

output = connection.send\_config\_set(commands)

print(output)

connection.disconnect()

**Mikor melyiket érdemes használni?**

* **Kisebb hálózatokhoz / régebbi eszközökhöz:**  
  **Netmiko** a gyors, egyszerű megoldás.
* **Nagyvállalati környezetben / modern eszközökkel:**  
  **API (RESTCONF/NETCONF)** hatékonyabb és biztonságosabb.
* **Vegyes környezetben:**  
  Akár **mindkettőt kombinálhatjuk** a rugalmasság érdekében.

**4. Netmiko függvénykönyvtár használata:**

A **Netmiko** egy Python könyvtár, amely megkönnyíti a hálózati eszközök (például Cisco IOS routerek) automatizált vezérlését SSH protokollon keresztül. A **Paramiko** könyvtárra épül, és kifejezetten hálózati automatizáláshoz optimalizált.

**4.1. Netmiko telepítése**

Először telepítsük a Netmiko-t Python környezetbe (Windows parancssorban vagy Linux terminálban):

pip install netmiko

**4.2. Cisco IOS router vezérlése alapvető Netmiko szkripttel**

**Példa: Egyszerű parancs futtatása Cisco routeren**

from netmiko import ConnectHandler

# Eszköz adatai

cisco\_device = {

'device\_type': 'cisco\_ios', # Cisco IOS eszköz

'host': '192.168.1.1', # Router IP-címe

'username': 'admin', # Felhasználónév

'password': 'password123', # Jelszó

'secret': 'enable\_password', # Enable jelszó (ha szükséges)

'port': 22, # SSH port (alapértelmezés: 22)

'verbose': True # Részletes kimenet

}

# Csatlakozás az eszközhöz

try:

connection = ConnectHandler(\*\*cisco\_device)

connection.enable() # Belépés az enable módba

# Parancs futtatása

output = connection.send\_command('show ip interface brief')

print(output)

# Konfigurációs parancsok futtatása

config\_commands = [

'interface GigabitEthernet0/1',

'description Configured via Netmiko',

'no shutdown'

]

config\_output = connection.send\_config\_set(config\_commands)

print(config\_output)

# Kapcsolat bontása

connection.disconnect()

except Exception as e:

print(f"Hiba történt: {e}")

**4.3. Fontosabb paraméterek:**

* **device\_type**: Cisco IOS esetén cisco\_ios.
* **host**: Az eszköz IP-címe.
* **username / password**: Az SSH bejelentkezési adatok.
* **secret**: Az enable módhoz szükséges jelszó (ha van).
* **port**: Az SSH port, alapértelmezetten 22.

**4.4. További Netmiko funkciók**

**Konfigurációs módba lépés:**

connection.config\_mode() # Belépés a konfigurációs módba

connection.exit\_config\_mode() # Kilépés a konfigurációs módból

**Mentés (write memory):**

connection.save\_config()

**Több parancs futtatása:**

commands = ['show version', 'show running-config']

for cmd in commands:

output = connection.send\_command(cmd)

print(output)

**4.5. Tipikus hibák és megoldások**

* **SSH elutasítva:** Győződjünk meg róla, hogy az SSH engedélyezve van a routeren
* **Timeout hiba:** Növeljük az időtúllépés maximális értékét
* **Authentication failure:** Ellenőrizzük a felhasználónevet és jelszót

**4.6. Példa Python script:**

Csatlakozás a megadott IP-című Cisco IOS eszközökhöz, majd azok futó konfigurációjának lementése a 192.168.99.99 IP-című TFTP-szerverre.

**Működés leírása:**

1. **Felhasználói adatok bekérése:**  
   A szkript bekéri az SSH-felhasználónevet, jelszót és (ha szükséges) az enable jelszót.
2. **IP-címek bekérése:**  
   Az IP-címeket vesszővel elválasztva kell megadni.
3. **Kapcsolódás az eszközökhöz:**  
   Minden IP-cím esetén SSH-kapcsolatot hoz létre.
4. **Futó konfiguráció mentése TFTP-re:**  
   A copy running-config tftp://192.168.99.99/filename parancsot futtatja.
5. **Fájlnév formázása:**  
   Az IP-cím alapján generál fájlnevet (pl. 192\_168\_1\_1\_running\_config).
6. **Hibakezelés:**  
   Ha nem sikerül csatlakozni vagy menteni, részletes hibajelentést ad.

**A példa program forráskódja:**

from netmiko import ConnectHandler

# TFTP szerver IP-címe

tftp\_server = '192.168.99.99'

# Felhasználói adatok bekérése

username = input("Felhasználónév: ")

password = input("Jelszó: ")

secret = input("Enable jelszó (ha van, ha nincs, nyomj Enter-t): ")

# IP-címek bekérése

ip\_list = input("Add meg az IP-címeket vesszővel elválasztva: ").split(',')

# Eszközök konfigurációjának mentése TFTP-re

for ip in ip\_list:

ip = ip.strip() # Szóközök eltávolítása

print(f"\nKapcsolódás az eszközhöz: {ip}")

device = {

'device\_type': 'cisco\_ios',

'host': ip,

'username': username,

'password': password,

'secret': secret,

'port': 22,

'verbose': False

}

try:

connection = ConnectHandler(\*\*device)

connection.enable() # Enable módba lépés

# Mentési parancs futtatása

filename = f"{ip.replace('.', '\_')}\_running\_config" # Fájlnév az IP- cím alapján

backup\_command = f"copy running-config tftp://{tftp\_server}/{filename}"

print(f"Futó konfiguráció mentése {filename} néven...")

output = connection.send\_command\_timing(backup\_command)

# Válasz kezelése (Enter megnyomása a megerősítéshez)

if 'Address or name of remote host' in output:

output += connection.send\_command\_timing('') # TFTP cím megerősítése

if 'Destination filename' in output:

output += connection.send\_command\_timing('') # Fájlnév megerősítése

print(output)

print(f"Mentés sikeres az IP-címről: {ip}")

connection.disconnect()

except Exception as e:

print(f"Hiba az {ip} IP-című eszköznél: {e}")

**Példa futtatás:**

Felhasználónév: admin

Jelszó: \*\*\*\*\*\*

Enable jelszó (ha van, ha nincs, nyomj Enter-t): enablepass

Add meg az IP-címeket vesszővel elválasztva (pl. 192.168.1.1,192.168.1.2): 192.168.1.1,192.168.1.2

**Hibaelhárítás:**

* **TFTP probléma:** Ellenőrizzük, hogy a TFTP-szerver fut-e: sudo systemctl status tftpd-hpa
* **Tűzfal:** Győződjünk meg róla, hogy a TFTP port (69/UDP) nyitva van-e: sudo ufw allow 69/udp