Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Hálózati Rendszerek és Szolgáltatások Tanszék

Barta Máté Zsombor

KONTÉNERIZÁLT VEZÉRLŐK SDN HÁLÓZATOKBAN

KONZULENS

Dr. Zsóka Zoltán

BUDAPEST, 2025

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Rezeda Kázmér**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot/ diplomatervet (nem kívánt törlendő) meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2025. 06. 01.

...…….

Rezeda Kázmér

Összefoglaló

Ide jön a ½-1 oldalas magyar nyelvű összefoglaló, melynek szövege a Diplomaterv Portálra külön is feltöltésre kerül.

Abstract

Ide jön a ½-1 oldalas angol nyelvű összefoglaló, amelynek szövege a Diplomaterv Portálra külön is feltöltésre kerül.

1 Bevezetés

A szoftver alapú hálózatkezelés (SDN, Software-Defined Networking) forradalmasította a hagyományos hálózati architektúrákat, lehetővé téve a hálózatok rugalmasabb és hatékonyabb kezelését. Az SDN kontrollerek központi szerepet játszanak ebben, mivel lehetővé teszik a hálózati erőforrások dinamikus konfigurálását és optimalizálását. Az SDN kontroller használata számos előnnyel jár, amelyek közül kiemelkedő a hibatűrés és a skálázódás.

A hibatűrés kulcsfontosságú a modern hálózatok megbízhatósága szempontjából. A kontrollerek képesek több példányban is működni ezáltal biztosítva a magas elérhetőséget. A működésüktől függően ez a rendelkezésre állás lehet szinkron és állapot alapú is.

A skálázódás szintén alapvető fontosságú, különösen a folyamatosan növekvő adatforgalom és a felhasználói igények fényében. A kontrollerek lehetővé teszik a hálózatok egyszerű és hatékony bővítését, mivel a hálózati erőforrások dinamikusan, központilag kezelhetők. Ez a rugalmasság nemcsak a költségek csökkentését segíti elő, hanem a hálózatok teljesítményének optimalizálását is, lehetővé téve a gyors alkalmazkodást a változó körülményekhez.

2 SDN alapok

2.1 Openflow

Az OpenFlow az SDN egyik alapvető protokollja, amely kulcsfontosságú szerepet játszik a hálózati forgalom irányításában és vezérlésében. Az OpenFlow lehetővé teszi a hálózati eszközök, például switch-ek és routerek, programozását központi vezérlők segítségével, ezáltal jelentősen növelve a hálózat rugalmasságát és hatékonyságát.

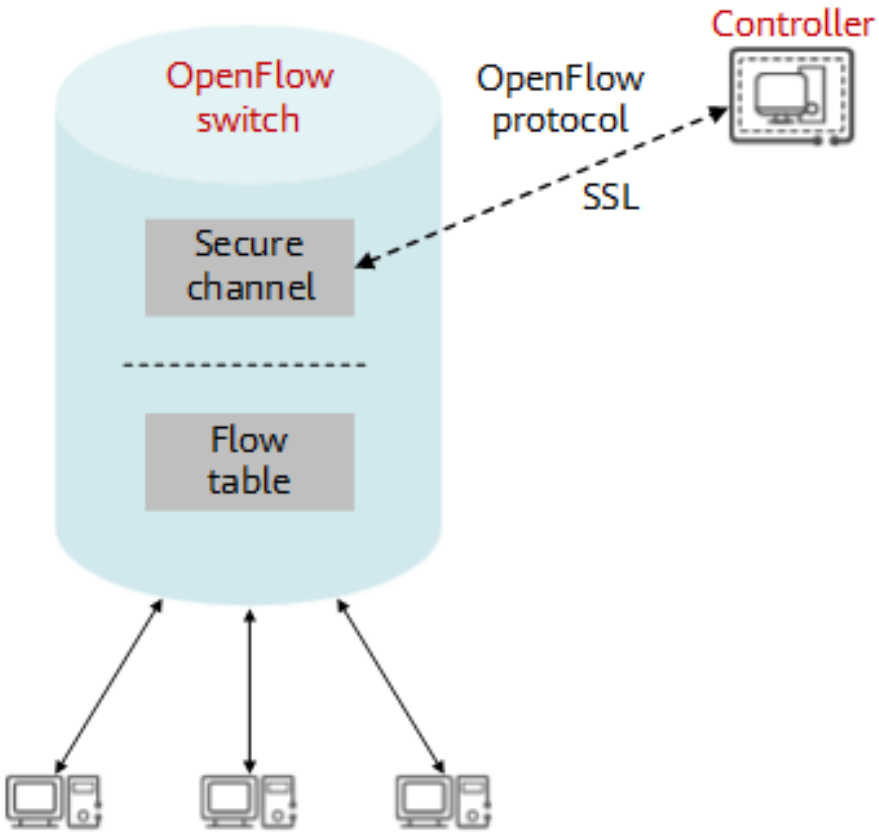
Az egyik legfontosabb tulajdonsága, ami igazából lehetővé teszi SDN kontrollerek használatát az az adat és irányítási sík szétválasztása. Ez a szeparáció lehetővé teszi, hogy a vezérlő szoftver (kontroller) központilag irányítsa a hálózati forgalmat, míg az adatforgalmi sík leköveti a vezérlő által adott utasításokat, kapcsolásokat, kapcsolatokat.

Az OpenFlow architektúrája három fő komponensből áll:

OpenFlow vezérlő: A vezérlő a hálózat központi agya, amely döntéseket hoz a forgalom irányításáról. A vezérlő legtöbbször szoftveres alkalmazás, amely globálisan rálát a hálózatra és ez alapján kezeli, és küldi el a vezérlő bejegyzéseket a lentebbi rétegekbe. A vezérlő akár mi magunk is lehetünk, amennyiben közvetlen kezeljük a hálózatban a flow bejegyzéseket és azokat mi írjuk be a közvetítő switchbe.

OpenFlow kompatibilis switch-ek: Ezek a hálózati eszközök fogadják és végrehajtják a vezérlő utasításait. Az OpenFlow switch-ek képesek feldolgozni és a bejegyzések alapján irányítani az adatforgalmat a vezérlő által meghatározott szabályok szerint.

OpenFlow protokoll: Ez a protokoll határozza meg a kommunikációt a vezérlő és a switch-ek között. Az OpenFlow protokoll segítségével a vezérlő képes utasításokat küldeni a switch-eknek, valamint információt gyűjteni a hálózati forgalomról és az eszközök állapotáról.



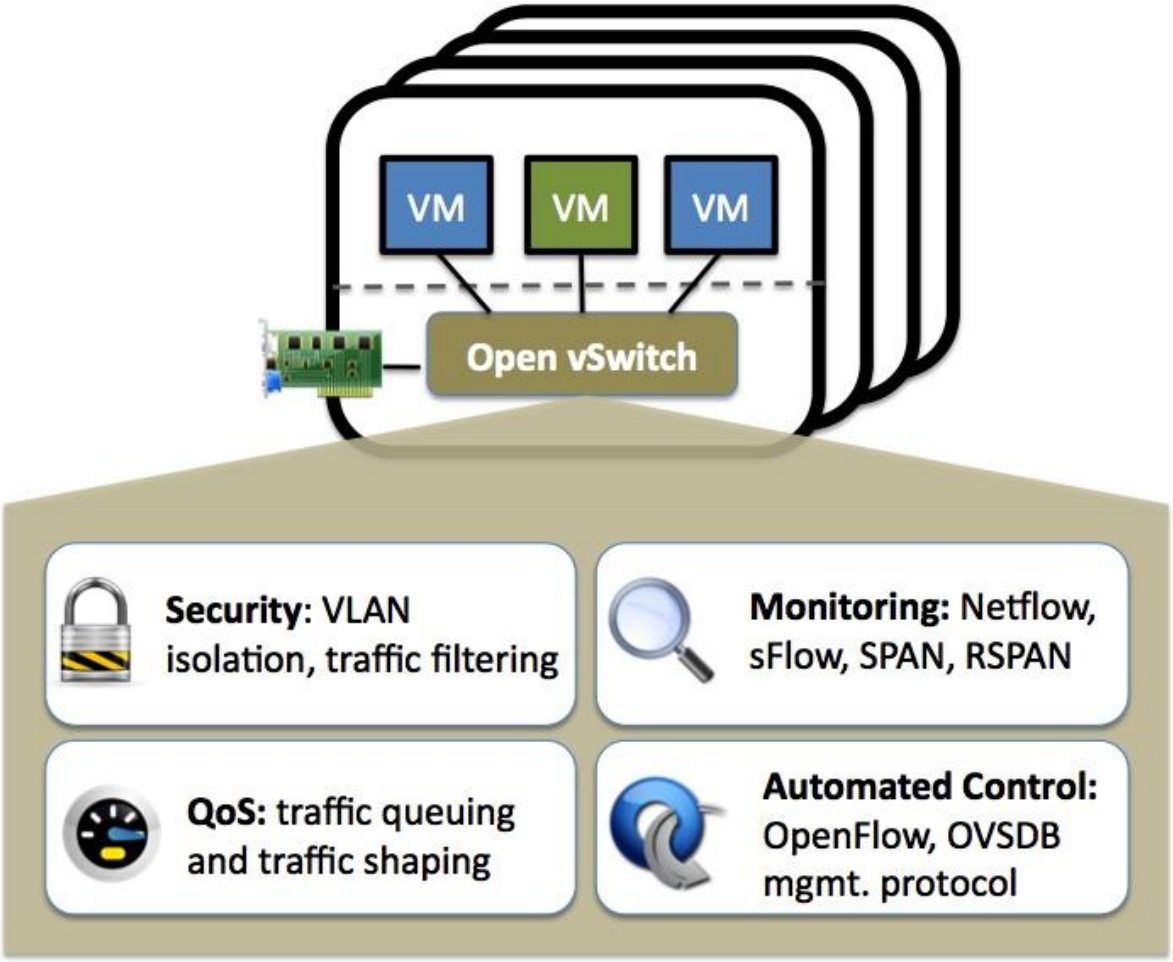
2.1. ábra:

Az OpenFlow működésének alapja a flow tábla, amely a switch-ekben található. A flow tábla tartalmazza azokat a szabályokat, amelyek alapján a switch-ek eldöntik, hogyan kezeljék az érkező adatcsomagokat. A vezérlő dinamikusan frissíti a flow táblákat, hogy optimalizálja a hálózati forgalmat.

Amikor egy switch kap egy csomagot, ellenőrzi a flow tábláját, hogy talál-e megfelelő szabályt a csomag kezelésére. Ha talál, alkalmazza azt, és továbbítja a csomagot a megfelelő kimeneti porton. Ha nincs megfelelő szabály, a switch a vezérlőhöz fordul egy új szabályért. A vezérlő elemzi a csomagot és frissíti a switch flow tábláját a szükséges szabállyal.

2.2 Open vSwitch

Az Open vSwitch (OVS) egy nyílt forráskódú virtuális switch, amelyet kifejezetten virtuális hálózatokhoz és adatközpontokhoz fejlesztettek ki. Az OVS célja, hogy egy szoftveres megoldást biztosítson a fizikai switch-ek funkcionalitásának emulálására a virtualizált környezetekben. Képes támogatni a standard protokollokat, mint például az OpenFlow-t, és integrálható különféle virtualizációs platformokkal, például az OpenStackkel.



2.2. ábra:

A kód nagy része platformfüggetlen C nyelven íródott, így könnyen átültethető más környezetekbe. Az Open vSwitch jelenlegi kiadása a következő funkciókat tartalmazza:

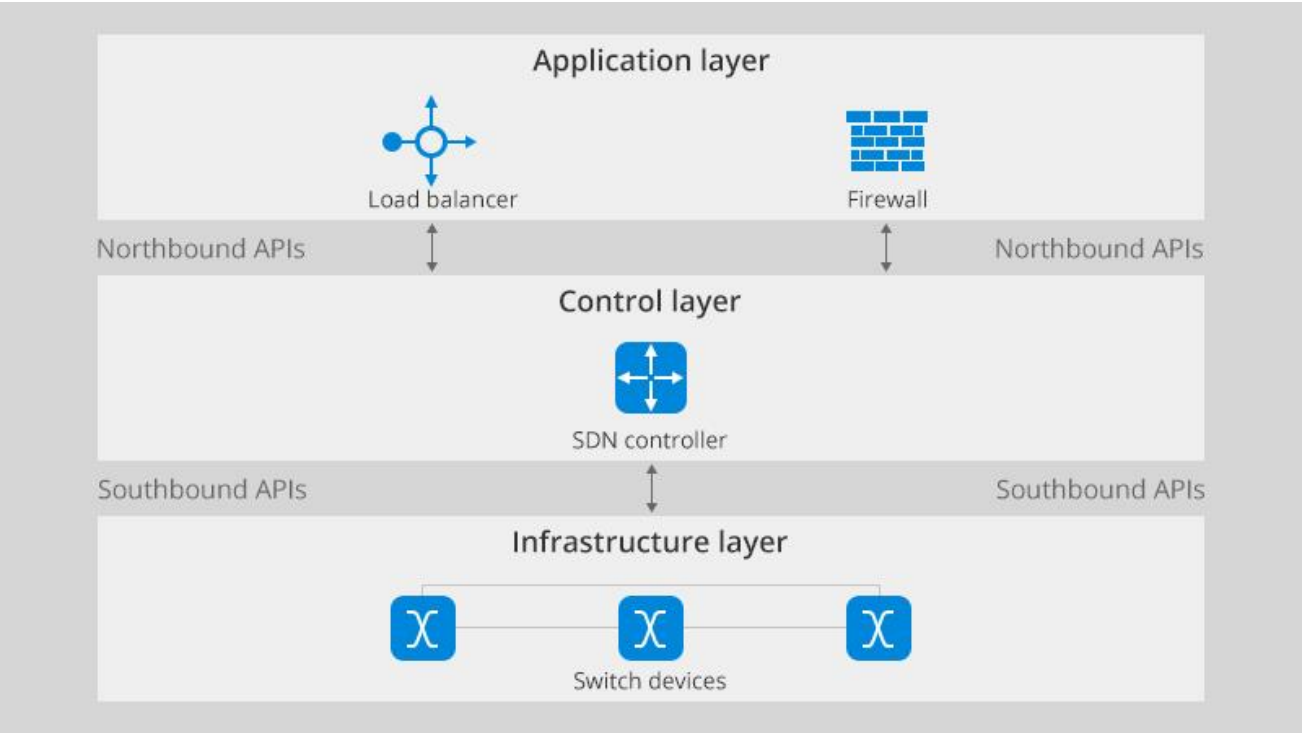
- Szabványos 802.1Q VLAN modell trunk és access portokkal
- NIC bonding, akár LACP-vel, akár anélkül
- NetFlow, sFlow(R) és mirroring a jobb láthatóság érdekében
- QoS (Quality of Service) konfiguráció, valamint forgalomkorlátozás
- Geneve, GRE, VXLAN, STT, ERSPAN, GTP-U, SRv6, Bareudp és LISP tunneling protokollok támogatása
- 802.1ag kapcsolat hibakezelés
- OpenFlow 1.0 és számos kiterjesztés

- Tranzakciós konfigurációs adatbázis C és Python interfészekkel
- Nagy teljesítményű adatforgalom irányítás Linux kernel modullal

Az Open vSwitch egy hatékony eszköz a virtuális hálózatok kezeléséhez, különösen nagy skálájú adatközponti környezetekben. Támogatja az összes főbb virtualizációs platformot, és lehetővé teszi a bonyolult hálózati topológiák egyszerű megvalósítását és kezelését. Telepítése és konfigurálása egyszerű, és számos haladó funkcióval rendelkezik, amelyek révén a hálózati adminisztrátorok teljes körű ellenőrzést gyakorolhatnak a virtuális hálózatok felett.

2.3 Architektúra

Az előbb ismertetett technológiák teszik számunkra lehetővé az SDN kontrollerek hatékony használatát. Az OpenFlow lehetővé teszi a logikai szétválasztás, ami az SDN megközelítés alapja, az Open vSwitch pedig, hogy egy azonos szabványos köztes médiumon keresztül történjen minden fajta hálózati kommunikáció. Így könnyen lehet adaptálni különböző gyártók között is a konfigurációt.



2.3. ábra:

A SDN felépítése egyszerű. Két fő interfészt definiál, egy délit és egy északit. A déli interfész a kontroller és a hálózati infrastruktúra közötti kommunikáció

megvalósításáért felel. Az északi pedig számunkra ad lehetőséget a kontroller konfiguráció módosításához.

Északi interfész:

Az északi interfész az SDN kontroller és az alkalmazások, valamint a felsőbb szintű szolgáltatások között helyezkedik el. Segítségével a kontroller szabványos kommunikációs interfészeket nyújt a felette lévő applikációk vagy akár a felhasználók felé. Ezen keresztül tudja a felhasználó módosítani a kontroller beállításait, a hálózati topológiát, a különböző szolgáltatásait, például QoS. Általában a kontrollerek nagy része REST és gRPC technológiát támogat, de előfordulnak más megoldások is.

Déli interfész:

A déli interfész az SDN kontroller és az adatforgalmat kezelő réteg (data plane) között helyezkedik el. Ez felel a kontroller ás a hálózatot irányító eszközök közötti kommunikációért. Ezen keresztül képes a kontroller leküldeni a kontroll üzeneteket az eszközöknek, ami meghatározza, hogy a flow bejegyzések hogyan érvényesüljenek. Az SBI-n (Southbound interface) keresztül információt tud gyűjteni a kontroller a hálózati eszközök állapotáról, például a forgalmi statisztikákról, hibákról és egyéb hálózati eseményekről. Több különböző protokollt támogat, a legelterjedtebb az Openflow, de ezen kívül általában képes NETCONF vagy RESTCONF-ra is.

3 SDN kontrollerek

Továbbiakban szeretném ismertetni az általam megismert SDN kontrollereket.

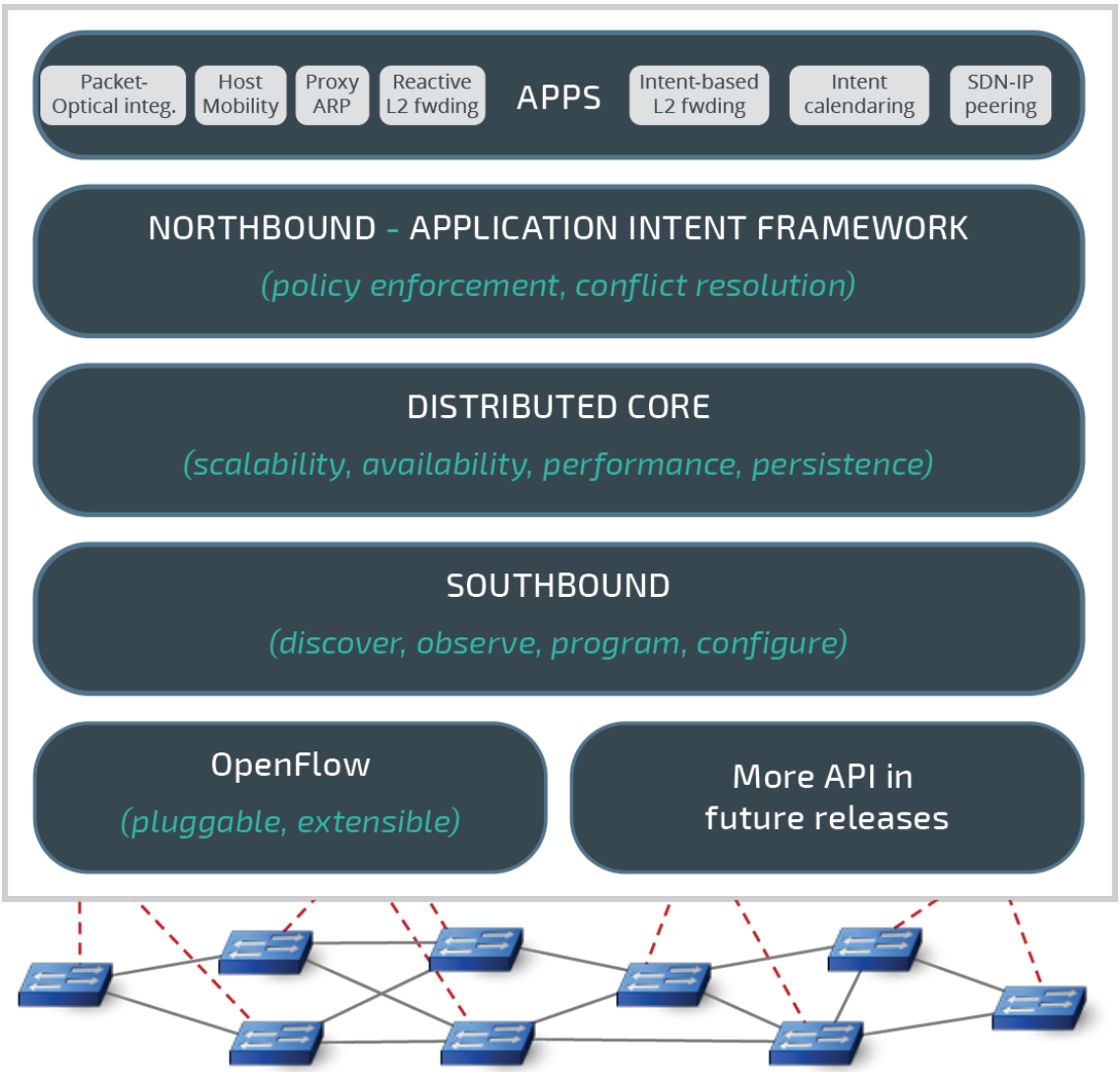
3.1 ONOS

Ezt a kontrollert kifejezetten az SD-WAN hálózatokban való használatra találták ki, tehát a célja, hogy olyan kapcsolatokat működtessen, amik lehet, hogy földrajzilag több kilométeren átívelők. Ezen célból tud georedundanciát biztosítani, ezért több példányt is lehet egyszerre üzemeltetni akár másik földrészen is és képes mindegyik példányban valós képet tartani az egész hálózatról.

Főbb jellemzői:

- Eszközöket valós időben, „runtime” lehet ki/becsatlakoztatni
- Skálázható – több példány is működtethető egyszerre
- Atomi datastore konzisztenciát biztosít a kontrollerek között
- Natívan támogatja a BGP-t
- Hamar leterhelődik – nem sok hostra tervezték
- Déli interfésze támogatja többek között: OpenFlow, P4, NETCONF, TL1, SNMP, BGP, RESTCONF and PCEP
- Északi interfész: gRPC és REST API
- High avaliability átállást biztosít
- JAVA

Jelenleg a Linux Foundation Networking támogatása alatt van.



3.1. ábra: az ONOS architektúrája

3.2 Opendaylight

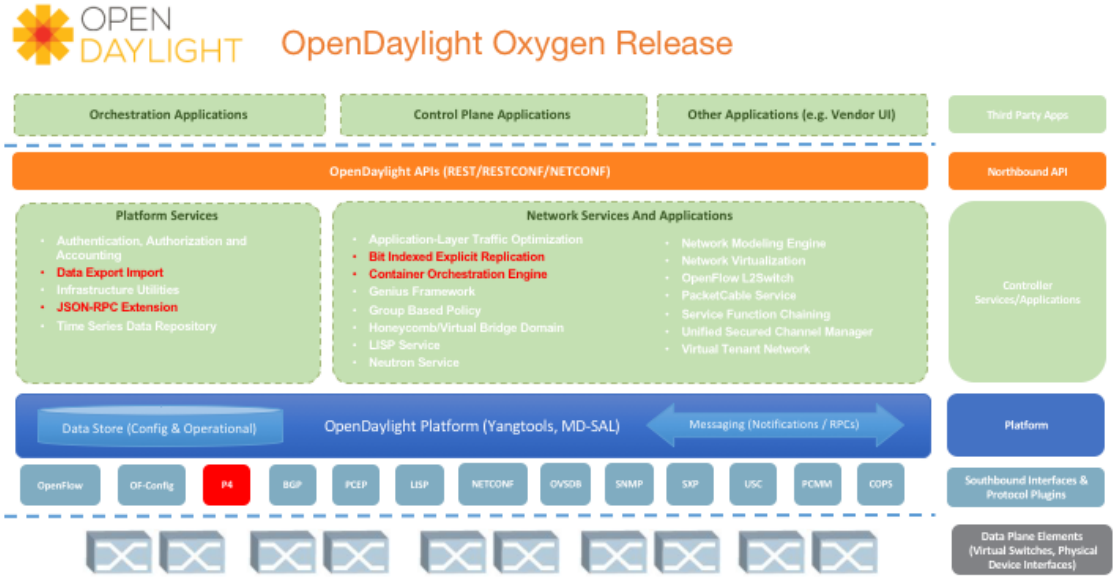
Ezen disztribúció arra törekedett, hogy minél könnyebben adaptálható legyen különböző környezetekben, ezért technológiailag egy Java-s konténerizációs alapon nyugszik, ezzel garantálva, hogy különböző operációs rendszereken is ugyanúgy lehessen futtatni.

Főbb jellemzők:

- Egyszerű felhő integráció – pl.: Openstack
- Nagy skálázhatóság és flexibilitás
- In-memory valós kép a hálózatról

- Natív BGP támogatás
- Déli interfész: OpenFlow, P4, NETCONF, SNMP, BGP, RESTCONF and PCEP
- Északi interfész: gRPC, REST API

Ez szintén a Linux Foundation Networking kezei alatt fut.



3.2. ábra:

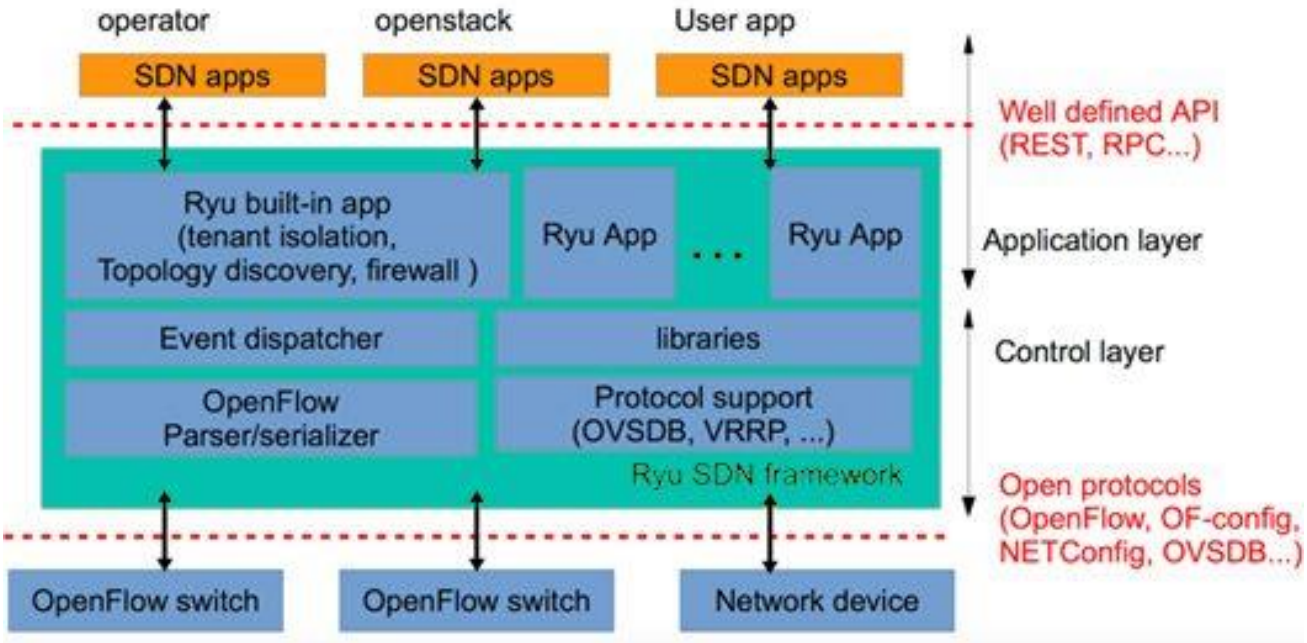
3.3 Ryu

Neve Japán eredetű, lefordítva angolra a ryu flow-t jelent. Ez az eddigiektől logikában eltérő konténer. A hálózatot python nyelven lehet leírni és azt az interpreter fordítja és indítja el. Ebből kifolyólag ha bármi változást akarunk eszközölni a hálózatban mindig újra kell indítani a kontrollert, de viszont ezt kevesebb, mint egy szekundum alatt képes megtenni, ezért a felhasználók számára ez a kiesés észrevehetetlen.

Főbb jellemzők:

- Nagy flexibilitás
- Nincs cooperatív cluster kezelés
- Jól definiált API-k
- Külső eszközök lehetővé teszik új példányok használatát (pl.: Zookeeper)
- Déli interfész: Openflow, NETCONF, OF-config… (mint az eddigiek)
- Északi interfész: REST API

Teljes mértékben az érdeklődők közösségének a szabad fejlesztése alatt van.



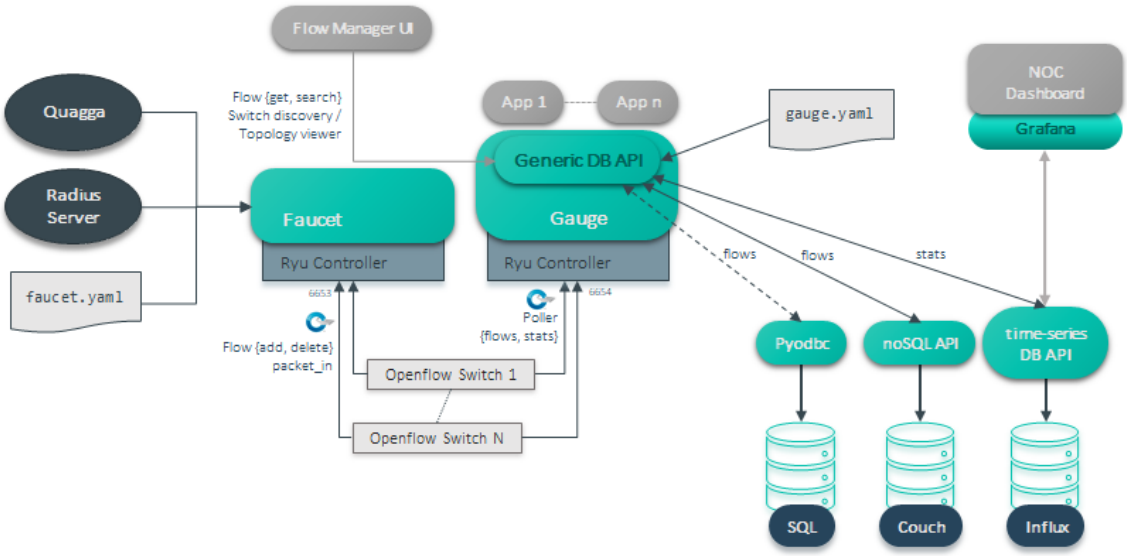
3.3. ábra:

3.4 Faucet

Alapjaiban a Ryu kontrollerre épít. Fentebbi absztrakciós szintekbe viszi a hálózat kezelését, ezért könnyebben le lehet írni a hálózat állapotát, a kapcsolatokat.

Főbb jellemzők:

- Akár bare-metal vagy konténerben is telepíthető
- Idempotens cluster
- Nincs cooperatív cluster támogatás, csak külső eszközökkel
- Gyors újraindulás
- Déli interfész: Openflow, natív BGP
- Északi interfész: YAML



3.4. ábra:

Mivel ezzel a kontrollerrel dolgoztam szeretném ezt részletesebben is bemutatni.

A bemutatáshoz a fentebbi ábrát fogom segítségül venni és ennek segítségével végig megyünk a kontroller pontos működésén és a komponenseinek a felelősségén.

Faucet: A kontrollernek ez a része az agy. Itt történik minden hálózati forgalom módosítás. Egy ryu kontrollerre épül rá. Ezen egy applikáció helyezkedik el, ami segítségével egyszerűen egy YAML fájlban leírva a hálózatot tudjuk leírni a kívánt kapcsolatokat, ez az északi interfésze. Ennek a segítségével tudunk még a kontrollerhez csatlakoztatni külső eszközöket is, mint például egy radius szervert.

Gauge: Felépítésében nem sokban különbözik a faucet-tól, majdnem megegyeznek. A különbség az, hogy míg a faucet kezeli a hálózat irányítását a gauge semmi változtatást nem visz végbe, csak egy azonos állapotot tart fent a hálózatról. Jól definiált API-k segítségével ez az elem arra szolgál, hogy a hálózatunkból információt nyerhessünk ki. Egy real-time adatbázisban el tudjuk tárolni a belőle kinyert adatokat így ennek segítségével erre monitoring rendszert is képesek vagyunk felépíteni. Ezeket mi is külön beállíthatjuk, de a telepítés során a kontroller ezzel egybe építve telepítődik.

4 Kubernetes

A Kubernetes egy nyílt forráskódú konténer orkesztrátor rendszer, amelyet a Google fejlesztett ki, és ma már a Cloud Native Computing Foundation (CNCF) irányítása alatt áll. A Kubernetes célja, hogy automatizálja a konténerek telepítését, skálázását és kezelését, lehetővé téve a fejlesztők és üzemeltetők számára, hogy könnyebben kezeljék a konténerizált alkalmazásokat.

Főbb jellemzői:

- **Automatikus skálázás:** A Kubernetes képes automatikusan skálázni az alkalmazásokat a terhelés függvényében, így biztosítva a megfelelő erőforrásokat a felhasználói igényekhez.
- **Öngyógyítás:** Ha egy konténer leáll vagy hibát jelez, a Kubernetes automatikusan újraindítja vagy új példányokat indít, hogy biztosítsa az alkalmazás folyamatos működését.
- **Terheléselosztás:** A Kubernetes automatikusan felfedezi a konténereket, és terheléselosztást végez közöttük, így a felhasználók mindig a legjobban teljesítő erőforráshoz csatlakoznak.
- **Tárolás orkesztráció:** A Kubernetes lehetővé teszi, hogy automatikusan csatlakoztass egy általad választott tárolórendszert, például helyi tárolókat, nyilvános felhőszolgáltatókat és még sok mást.
- **Automatizált telepítés és rollback:** Leírhatod a kívánt állapotot a telepített konténereid számára a Kubernetes segítségével, és az képes a tényleges állapotot a kívánt állapotra változtatni egy ellenőrzött ütemben. Például automatizálhatod a Kubernetes-t, hogy új konténereket hozzon létre a telepítésedhez, eltávolítsa a meglévő konténereket, és átvegye az összes erőforrásukat az új konténerhez.
- **Automatikus konténer áthelyezés:** A Kubernetes-nek egy node-okból álló klasztert adsz. Leírod a konténereidnek, hogy mennyi CPU-ra és memóriára (RAM) van szüksége. A Kubernetes képes a konténereket a node-okra úgy elhelyezni, hogy a legjobban kihasználja az erőforrásaidat.

5 Motiváció

A fenti technológiák ismertetése után jól látható, hogy könnyen párosítható az SDN kontrollerek és a Kubernetes, mint a kontrollerek orkesztrátora, ami biztosítja a kontrollerek magas rendelkezésére állását futási környezetét.

A konténerek csak úgy, mint egy átlag program könnyen konténeresíthető és a legtöbbjük olyan nyelveken is van írva, amikhez könnyen tudunk konténer alapokat illeszteni. Kontroller függő, de a legtöbbjük nem is robosztus, ezért könnyen indulnak, állnak le, ezzel is elősegítve a konténer környezet tulajdonságait.

Mivel a legtöbbet nem úgy tervezték, hogy kooperatívan cluster-ként tudjanak működni ezért felmerül a kérdés, hogy mégis mi történik akkor, ha a kontroller leáll. Ilyenkor a kapcsolat az alatta kezelt hálózattal megszűnik és a hálózat immár nem tudja, hogy a forgalmat hogyan irányítsa, mivel elveszett a kapcsolat a központi vezérlővel, ami az egyetlen vezérlő volt a hálózatunkban.

A Kubernetes egy kézenfekvő rendszer ezen problémák orvoslására. Az öngyógyítással és a skálázással könnyen tudjuk szabályozni a fent említett problémát. Egyszerre több kontroller életciklusát is tudja kezelni a klaszteren belül és a hibás példányok újraindítását és átcsoportosítását is – probléma esetén.

Ám ezen keretrendszerben nem lehet akármelyik SDN vezérlőt behelyezni. Ahhoz, hogy több példány is egyszerre tudjon létezni a hálózatunk kezelésére valamilyen szinten egy szinkron kommunikációnak léteznie kell a példányok között, annak érdekében, hogy ne lépjen fel inkonzisztens állapot – és emiatt hiba. Könnyen belátható, hogy ez egyértelmű és adódik is, hogyha mi változtatásokat eszközölünk a hálózatunkon, akkor azt szeretnénk, ha arról mindenki értesüljön, ne legyen probléma, hogy az egyik része a hálózatunknak nem tud a változtatásokról (amelyiket az a kontroller kezeli, aki nem értesült a változtatásokról). A másik lehetőség, ha a kontrollerek állapotokat őriznek és ezen állapot minden példányban azonos. Leírhatjuk a hálózatunkat egy megadott fájlban például, és úgy indítjuk a konténereket, hogy ezt az állapotot őrizzék meg, és csak úgy tudjunk változtatni a hálózat állapotán, ha a kontrollerünket újraindítjuk a már új konfigurációval.

Mivel egy szinkron kommunikációs rendszer nagy tervezést igényél és munkám során már meglévő megoldásokkal szerettem volna dolgozni és azoknak a hibáira megoldást kínálni így az utóbbi megvalósítást választottam.

A Faucet kontroller pontosan ezen az alapon működik. Belső felépítése a Ryu kontrolleren alapul, de ahelyett, hogy a felhasználó python nyelven tudná beprogramozni a hálózatot, csak egy yaml leíró fájlban kell a felépítését megadnia. Ez a fájl adja a konzisztens állapotot. Ebből adódóan minden változtatáskor a kontrollert újra kell indítani, hogy érvényesítsük azt. Ezt szem előtt tartva tervezték a kontrollert, ezért maga a kontroller is csak pár ezer sor kód, így egy újraindítás ideje kevesebb, mint egy másodperc.

6 Megvalósítás

A probléma összefoglalva és röviden az, hogy hiába központi vezérlést valósítanak meg az SDN kontrollerek, nem kínálnak megoldást a hibakezelésükre így könnyen a hálózatunk single point of failure-e lehetnek. Ezen problémára szeretnék megoldást találni a Kubernetes orkesztrációs rendszert és a Faucet SDN kontrollert használva.

6.1 SDN és „legacy” hálózatok összekötése

Ahhoz, hogy az SDN kontrollerünk által vezérelt hálózat a külvilággal is összeköttetésben álljon szükségünk van egy protokollra, amin keresztül a két különböző hálózat meg tudja egymással osztani az útjaikat. Erre az összes általam ismert kontroller a BGP protokollt használja.

6.2 Orkesztráció

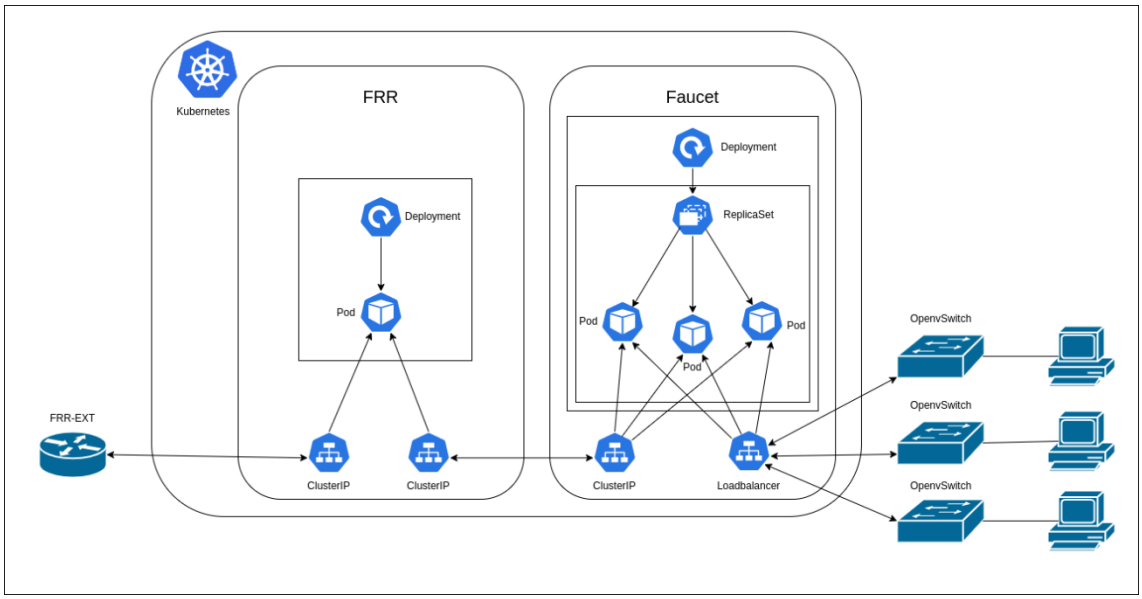
Miután már tudjuk, hogy a kontrollerünket miképp kötjük össze a régi legacy eszközeinkkel megtervezhetjük az orkesztrátoron belüli felépítést. Két fajta megközelítést tudunk egymással szemben állítani.

6.3 Köztes router a cluster és legacy network között

Az első megközelítésben egy köztes eszközt helyezünk az SDN kontroller és a külvilágot összekötő router közé. Így a kontrollerünk közvetlen a Kubernetes cluster-en belül fog csak kommunikálni, így nem kell bajlódni az IP cím/port átfordításokkal.

Az összeköttetést egy konténerizált legacy router végzi a Kubernetes cluster-en belül. Így a külső router nem lát csak egyetlen BGP endpoint-ot (szomszédot) az SDN vezérlő felé, ami megkönnyíti a dolgunkat akkor, ha ez a külső kapcsolati eszköz nem a mi hatásunk alatt van. Mivel a cluster-en belül a kontrollerek száma tud változni és valószinűsíthető, hogy változni is fog, ezért a köztes router-t valamilyen automatizációval kell konfigurálni a kontroller példányok függvényében.

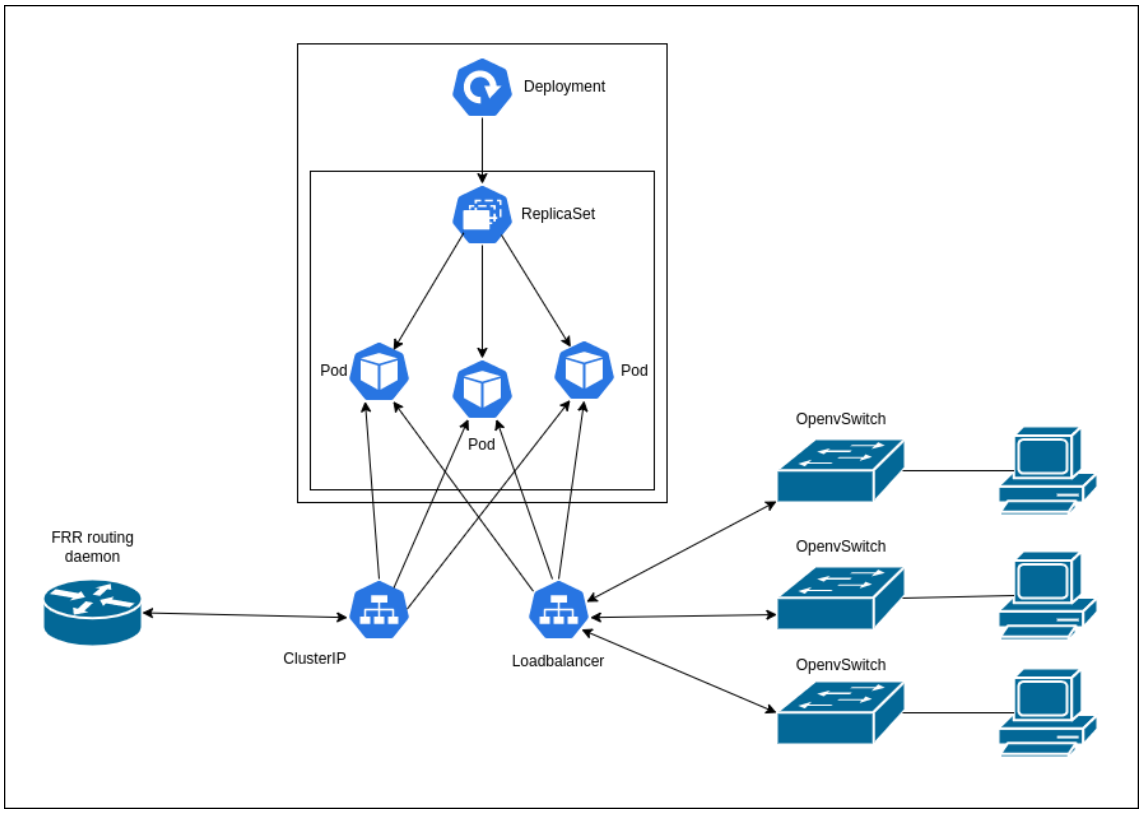
Ezenkívűl így közvetlen egy ponton tudjuk szűrni vagy akár a szabályokat beállítani az SDN felől ki- és bemenő forgalomnak.



6.3. ábra: Köztes router az SDN és legacy kapcsolatban

6.4 Közvetlen kapcsolat a cluster és legacy router között

A második megközelítésben nincsen köztes elem a kapcsolatok között, ezért a külső hálózat edge router-je közvetlen kommunikál a SDN kontrollerekkel.



6.4. ábra: Közvetlen kapcsolat az edge router és a vezérlők között

Ezt a megoldást választottam, mivel az előbbi jelen esetben csak plusz konfigurációs terheket adott volna számomra. Az edge router-t is én kezelem, ezért annak a konfiguráció automatizációja is az én kezemben van.

6.4.1 Megvalósítás

Először elkészítjük a kubernetes deployment-et, ami biztosítani fogja a kontrollerünk futását. A Faucet kontrollerhez már van egy előre elkészített docker konténer, ezt fogjuk használni a Pod-unkban. A Gauge részét a kontrollernek mi most nem fogjuk használni, de hogy bebizonyosodjunk a stabil futásról ezt becsomagoljuk a Pod-ba. A konfigurációs fájl-t a a kubernetes/deployment.yaml-ben találjuk. Alapvető sztenderd konfiguráció, amit fontos kiemelni, hogy a kontroller-hez a 9179 és a 6653 portokat kötjük, ezen keresztül fog tudni csatlakozni a switch openflow-val és az edge router BGP-vel. A pod-ba felcsatoljuk az operációs rendszeren található könyvtárból a faucet konfigurációs fájl-ját, ezt fogja olvasni, amikor elindul.

A deployment-et Service-ekkel tudjuk a külvilág felé láthatóvá tenni. A kubernetes-ben három fajta service definiált. Ez a NodePort, ClusterIP és a LoadBalancer. A Nodeport lényegében egy Port Addres TRanslation-t csinál a pod-hoz a 30000 feletti portokat használva. A ClusterIP kubernetes-en belüli kommunikációra szolgál, kívülről nem érhető el. Mi LoadBalancert fogunk használni az OpenFlow protokollhoz. Ezen keresztül el tudjuk fedni, ha több példány is fut a kontrollerek között és ezek között dinamikusan lesznek elosztva a kapcsolatok. Így minden switch ugyanazon a címen próbál meg kapcsolódni a kontrollerhez és számára láthatatlan lesz, hogy pontosan melyikkel létesít kapcsolatot. Valamint, ha megszakad a kapcsolata a kontrollerel, mert az valami hiba miatt leáll, automatikusan ugyanazon a címen tud újra próbálkozni és csatlakozhat egy másik példányhoz.

6.5 Fallback teszt

Miután elindítottuk a kontrollereinket le tudjuk tesztelni, hogy sikeresen működik-e az átállás, ha a switch leszakad az éppen aktív kontrolleréről. Ehhez elindítjuk a deployment-et, majd felskálázzuk a pod-ok számát kettőre, így két kontroller fog futni a cluster-ben ugyanazzal a konfigurációval. Valamint elindítunk két switchet is, akik fel fognak csatlakozni a kontrollerekre. Mivel LoadBalancer-el fedjük el a pod-okat, ezért a

switchek külön egy-egy kontrollerre fognak kapcsolódni, mert a LoadBalancer figyeli a kapcsolatokat és dinamikusan osztja ki azonos pod-ok között.

Miután mindkettő Switch felcsatlakozott leállítjuk a kontrollerek közül az egyiket. Alább látható, ahogy leállítjuk a pod-ot:

```
~
❯ k get all -n faucet
NAME                           READY   STATUS        RESTARTS   AGE
pod/faucet-765d77b58f-hk6bw    2/2     Terminating   0          12d
pod/faucet-765d77b58f-mmfxm    2/2     Running       0          12d

NAME                       TYPE           CLUSTER-IP    EXTERNAL-IP     PORT(S)          AGE
service/faucet-openflow    LoadBalancer   10.43.19.20   192.168.10.65   6653:32739/TCP   12d

NAME                     READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/faucet   1/1     1            1           12d

NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/faucet-765d77b58f   1         1         1       12d
```

6.5.1. ábra: Pod leállítás

Itt pedig látható, ahogy a két switch-ből az egyik elvesztette a kapcsolatot és utána újra csatlakozott a másik kontrollerre:

```
external_ids        : {}
inactivity_probe    : []
is_connected        : true
local_gateway       : []
local_ip            : []
local_netmask       : []
max_backoff         : []
other_config        : {}
role                : other
status              : {last_error="Connection timed out", sec_since_connect="100", sec_since_disconnect="103", state=ACTIVE}
target              : "tcp:192.168.10.65:6653"
type                : []
```

6.5.2. ábra: a lecsatlakozott switch

```
role                : other
status              : {sec_since_connect="48739", state=IDLE}
target              : "tcp:192.168.10.65:6653"
type                : []
```

6.5.3. ábra: a switch, aminek megmaradt a kapcsolata

A 6.5.2.-es ábrán látható, ahogy a switch elvesztette a kapcsolatot, Timed Out erroral. Utána pedig írja, hogy 100 másodperce már megvan a jelenlegi kapcsolata és 103 másodperccel ezelőtt vesztette el utoljára a kapcsolatot. Látható, hogy körülbelül 3 másodperc alatt a switch képes volt újra csatlakozni a másik kontrollerre.

7 További teendők

Jelen pillanatban a kontrollerünk átállása működik és a teszt alapján látható, hogy ha kiesik egy, akkor gyorsan át tud állni a switch egy másik vezérlőre. Viszont nincs még összekötve a külvilággal. Ehhez a félév során próbáltam felkonfigurálni már a router(-eket), de sikertelenül. A címfordítások és szegmentáció jelenleg bonyolult, ezért nehéz még pontosan átlátni, hogy hogyan lehet kapcsolatot létesíteni a kontrollerek és az edge router között.

A Kubernetes automata skálázódását még nem használtuk ki. Ehhez készíteni kell egy stressztesztet, amihez még egy HorizontalAutoScaler-t is kell készíteni.

Majd ha már sikerült felkonfigurálni a legacy és SDN vezérelt hálózatunk között a kapcsolatot akkor különböző való életbeli forgatókönyveket le lehet tesztelni, hogy megbizonyosodjunk arról, hogy mégis produktív környezetben mennyire lehet hasznosítani a megoldást. Ilyen forgató könyvek lehetnek:

- Mi történik akkor, ha egy hálózat az SDN vezérlő alatt leszakad? A switch átállása befolyásolja a hálózati forgalmat, vagy addig képes megszakadás nékül folytatódni a kommunikáció?
- Mivel a kontrollerek száma dinamikusan változik, ezért az edge router konfigurációját is automatikusan kell változtassuk, ha azt szeretnénk, hogy minden kontroller megfelelően csatlakozzon a külső hálózathoz.
- Ha szeretnénk „legacy” hálózatról SDN vezérleltre átállni, akkor miután felépítettük az architektúránkat a hostjainkat át kell mozgatni. Le lehet tesztelni, hogy hogyan tudjuk áthelyezni a hostjainkat a legjobb módon, anélkül, hogy szakadás lenne.

Irodalomjegyzék

[1] ProofIT – Mi az az SPOF?
https://proofit.tech/blog/hu/mi-az-az-spof/

[2]

Függelék