NEUMANN JÁNOS INFORMATIKAI KAR

DIPLOMAMUNKA

OE-NIK HARASZTI GÁBOR
0 0

OE-NIK HARASZTI GÁBOR

Budapest
2019

Tartalomjegyzék

1. fejezet

Bevezetés

A felhő technológia mára már a mindennapok részévé vált az információ technológiát felhasználó cégeknél. Számos előnyt jelent a felhasználók számára, leginkább olcsón üzemeltethető és rugalmasan méretezhető infrastruktúrát ad a megfelelő biztonsággal. Mindezt bérleti konstrukcióban szolgáltatja a beruházási konstrukcióval szemben, mely költséghatékonyabb a cégek számára.

Több területen is fel tudják használni, az intranetes rendszereiktől kezdve az internetes megjelenést biztosító rendszereken át a partnerek felé nyitott extranetes megoldásokig. A felhő technológiák fontosságát jól jelzi, hogy nélkülük nem létezhetnének azok a mára már természetesnek vett szolgáltatások az Interneten, amiket százmilliók, milliárdok vesznek igénybe nap mint nap, nélkülük nem lehetne kiszolgálni a 3 milliárd Internet felhasználót.

Manapság az Interneten megjelenő új adatok több, mint 80%-a nem emberi kéz által létrehozott adat, a felhasznált rendszerek generálják azokat, beleértve az ipari rendszerek által létrehozott adatmennyiséget, az egyre terjedő IoT alkalmazásokon át az egyéb kiberfizikai rendszerek által kibocsátott adatokig. Fontos tényező lett a hatalmas adatvagyon – big data – elemzése, ami nem létezhetne a felhők számítási és adattárolási kapacitása nélkül. Manapság már a gyárak, ipari üzemek sem nélkülözhetik a felhő technológiák használatát, a robotrendszereken és szenzorokon keletkezett adatok alapján jobb rugalmassággal tudják az adatokat feldolgozni és vezérelni a gyártás folyamatot.

Fontos felhasználási területe lesz a felhő szolgáltatásoknak a szolgáltató robotok. Jelenleg ugyan elenyésző a számarányuk az ipari robotokhoz képest, viszont a jövőben ez jelentősen változni fog – egyes becslések szerint a 2015-ös 22%-ról 2025-re 66%-ot várnak –. Fontos viszont megkülönböztetnünk a szolgáltató robotokat az ipari robotoktól. Az ISO 13482-es szabvány szerint az ipari és szolgáltató robotoknak a definíciója:

ipari robot:. Az ipari robot egy automatikus irányítású, újraprogramozható, többcélú automatikus ipari feladatok elvégzésére használt manipulátor, három vagy több tengelye programozható, melyek lehetnek rögzítettek vagy mobilak.

szolgáltató robot:. Olyan robot, amely hasznos feladatokat hajt végre emberek vagy berendezések számára, kivéve az ipari automatizáló berendezéseket.

Mint látható a megfogalmazásból a szolgáltató robotok definíciója elég tág, ezért érdemes pár egyéb szempont szerint is megvizsgálni, ami a Robotmechanizmusok jegyzet [12] szerint a következő:

	IPARI ROBOTOK	SZOLGÁLTATÓ ROBOTOK
MUNKAKÖRNYEZET	Ellenőrzött és jól meghatározott környezet	Rendezetlenebb, nehezebben definiálható környezet
FELHASZNÁLÓK	Betanítás speciális feladatokra	A betanítás a tevékenységek széleskörű skáláját öleli fel
BIZTONSÁG	Gépfüggő	Robot és felhasználófüggő
MUNKAFILOZÓFIA	Robotok és emberek elkülönítése	Robotoknak és embereknek meg kell osztozniuk a munkaterületen, hogy szolgáltatást nyújtsanak/kapjanak
GÉP TERVEZÉSE	Megbízásra rugalmasan reagál	Igényre rugalmasan reagál

A diplomamunka feladat fő célja, hogy az ipari robotika igényei mentén feltárja a felhő technológiákat felhasználó különböző szoftver architektúra irányokat és egy kísérleti PoC[1] környezet kiépítése a megoldások mérnöki és tudományos vizsgálatához. A megvalósításnál törekedni kell az elterjedt nyílt forráskódú technológiák használatára a kiírás szerint.

A feladatnak meg kell valósítania különböző környezetekben egy példa alkalmazást, egy pályatervezési feladatot. A felépítendő rendszer része lesz egy robot a maga környezetével, helyi szerver számítógép, valamint több felhő rendszer is, úgy mint a Microsoft Azure és Amazon AWS rendszere. Külön említést érdemel a Rapyuta[15], specifikus robot felhő szolgáltatásának vizsgálata is.

[1] Proof of Concept

2. fejezet

Irodalmi áttekintés

A munka megkezdése előtt célszerű tájékozódni a probléma lehetséges megoldásairól a hazai és nemzetközi szakirodalomban, ami után a megoldási alternatívák már ki tudnak rajzolódni és lehetséges lesz szakmailag megalapozott javaslat megfogalmazására.

Galambos Péter cikkéből [1] megtudhatjuk, hogy habár sokáig megfelelt a robotikában a helyi kontrollerekről vezérelni a robotokat, a manapság szükséges számítási kapacitás igényeknek ez már nem felel meg. Az új kihívások szükségessé teszik külső erőforrások bevonását is a munkába. A kapacitás bővítésnek több módja is van a lokális szerverektől a publikus felhő szolgáltatásokig bezárólag. A cikk bemutatja a hagyományos IT területtől némileg eltérő robotikai alkalmazások szükségleteit a felhő szolgáltatásokban.

Először ismertetésre kerülnek a főbb paradigmák a robotika kontextusában:

Cloud computing: azaz felhő számítástechnika, hozzáférés nagy, jól skálázható hálózati számítási kapacitásokhoz (végrahajtási, memória és disk) távolról. Ez az infrastruktúra lehet privát, publikus és hibrid.

Edge computing: A robot kontrollerek mellett, helyi erőforrásként felhasznált számítógépeket használhatunk azon feladatok elvégzésére, melyeket nem praktikus a felhőbe tenni – például a nagy és folyamatosan generált adatmennyiség miatt, amire jó példa a nagy tömegű, robot szenzorokból érkező adatok elő-feldolgozása – valamint képesek gateway-ként illeszteni a helyi rendszereket az Internet követelményeinek megfelelően a felhőhöz. Ide tartoznak még a beépített eszközök számítási kapacitásai, melyek előfeldolgozást végeznek a magasabb absztrakciós szintek számára.

A cikk továbbá kitér a felhőbe kiszervezett számítástechnika előnyeire és hátrányaira, az elfogadható késleltetés és adatforgalom tükrében. Minden esetben a tervező mérnök

kezébe adva a döntés jogát az egyes szolgáltatások helyének megtervezésére az igények szerint. Bevezeti továbbá a **puppet robot**, azaz báb-robot fogalmát, mely, mely mint fizikai egység már csak az érzékelésért és beavatkozásért felelős szemben a hagyományos gyártási forgatókönyvekkel.

A folytatásban tisztázásra kerül egy nagyon fontos aspektusa a témának, az architekturális kihívások és kulcs technológiák a felhő robotikában. a szerző élesen elkülönít kétféle szolgáltatás típust – vagy inkább képességet, robotikai értelemben az SaaS inkább Skills as a Service-t jelent – az állapot-információk szerint, mégpedig:

Context-free: azaz állapot független szolgáltatásoknak nincs szükségük több információra, mint például a pillanatnyi állapot, vagy az alkalmazás belső állapota. Működésükhöz elegendő a meghívásukhoz szükséges adat. Ezek a szolgáltatások tipikusan egyszerűen implementálhatóak, akár RESTful microservice-ben, akár felhős funkcióban – mint amilyen az AWS Lambda, vagy az Azure Functions – és könnyen skálázhatóak maradnak.

Contextful: azaz állapot függő szolgáltatásoknak szükségük van információra a tárgyrendszerről, annak fizikai struktúrájáról, pillanatnyi belső állapotáról. Ezek a szolgáltatások viszont már nem skálázódnak jól, tipikusan szükséges hozzájuk dedikált ROS [3] példány. (A ROS pillanatnyilag csak egy robot környezetet támogat, ezért kell környezetenként egy-egy ROS példányt futtatni)

Xi Vincent Wang és munkatársai cikke [2] definiálja a mindenhol jelen lévő gyártási rendszert a felhő segítségével, illetve az ICMS rendszereket – Interoperable Cloud Manufacturing System / Átjárható Felhő Gyártási Rendszer –. A felhő új üzleti modellt és lehetőségeket hoz a gyártásba, létrehozva a mindenhol jelen lévő gyártást, segítve az SME-ket – Small and Medium-sized Enterprises / Kis és közép vállalkozások – a kezdeti nagy beruházási költségekkel járó robotikai beruházásokban, csökkentve a modern gyártási környezet kialakításához szükséges költségeket.

A gyártási rendszerek új generációjának a létrehozása lehetővé válik a felhő szolgáltatások által, ami biztosítja a megfelelő számítási kapacitást, képességeket és erőforrásokat. Más szemszögből nézve, a felhőben rendelkezésre áll az a kapacitás, amit költséges lenne egyenként fenntartani és kihasználatlan is lenne, ezért okos gondolat ezen erőforrásokat megosztani több gyártó rendszer között is, ideértve nem csak a számítási kapacitást, hanem azt a tudást is, ami egy-egy feladat hatékony megoldásához szükséges és valaki már elkészítette hozzá a szoftvert.

A cikk foglalkozik a felhő lehetőségeinek azokkal az aspektusaival is, amik megkön-

nyíthetik az ember-gép együttműködést – például kollaboratív robotok –. Minden ember másmilyen, mind fizikailag, mind mozdulatait tekintve. A felhő számítási kapacitása így jobban képes alkalmazkodni az ember jelenléte miatt szükségessé vált változékony környezethez alkalmazkodni.

Fontos részlet – főleg a gyártás számára –, hogy a robotrendszerek energia hatékony módon dolgozzanak, amiben szintén nélkülözhetetlen lehet a felhő szerepe, de fontos lehet olyan helyzetekben is, amikor az energiavételezés lehetőségei korlátozottak. Az energiaoptimalizált működés fontos szerepet játszik a zöld-felhasználásban, nem csak a gyártás területén.

Adarsha Kharel és munkatársai cikke [4] a ROS alapú felhő-robotikával foglalkozik. Először is fogalmakat tisztáz. Az operációs rendszer és a robotikai alkalmazások között definiál egy réteget, a "Robotic middleware"-t, mely az alkalmazások elől elfedi a hardware és OS rétegeket, így egyszerűvé téve a robotikai fejlesztéseket és olcsóbbá téve azt. A ROS [3] – Robot Operating System – egy nyílt forráskódú "meta-operációs rendszer", ami biztosítja a

- Hardware abstrakciót
- Alacsony szintű eszköz vezérlést
- Üzenet továbbítást
- Csomag menedzsmentet

A cikk leírja a ROS alapfogalmait, részeit. Alapvető egység a Node, ami reprezentálja a ROS-on belül a felületet a programok felé. Egy ROS példány több Node-ot is indíthat. Az egyes Node-ok tartalmazzák azután a kétféle ROS szolgáltatás típust és a message-eket.

message. A message-ek az üzeneteket reprezentálják ROS-on belül.

service. A service-ek message párokat jelképeznek, kérés/válasz-ként, más rendszerekbe a függvények felelnek meg nekik.

topic. A Node-ok fel tudnak iratkozni topic-okra és maguk is tudnak kiajánlani topic-okat – megvalósítva az informatikában jól ismert publisher/subscriber mintát –. Egy topic-ba több node is írhat (publish) és több node is olvashat (subscrbe) belőle.

A ROS-t számos nyelvből el lehet érni, , többek között C++, Python, lisp, Java, stb. környezetekből. Illetve rendelkezik egy csomag kezelővel, amivel finomhangolható, hogy mely szoftverkomponensek legyenek felhasználva az adott ROS példányban. A ROS elsődleges célja, hogy támogassa a kód újrafelhasználást a robotikai kutatásokban és fejlesztésekben [3]. A ROS támogatja a moduláris, "tool-based" fejlesztés filozófiáját a robotikában.

Pablo González-Nalda és munkatársai ROS és Docker alapú kiber-fizikai – CPS, Cyber-Physical Systems – architektúra tervezéssel foglalkozó cikke [5] bevezetést nyújt általánosságban az ICT – Information and Communication Technologies – rendszerek tervezési aspektusaiba, legfőképpen az ipari és CPS felhasználások tekintetében, érintve a tervező rendszerektől az Ipar 4.0-n át az IoT rendszerekig. Kitér a FOSS – Free and Open-Source Systems – rendszerek felhasználhatóságára, a ROS hasznára, mint a kód újrafelhasználhatóság és magas szintű integráció lehetőségére.

A cikk foglalkozik a könnyűsúlyú – lightweight – virtualizációval, ezen belül is a Docker konténer technológiájával. Ez a szoftver nem kívánja meg, hogy minden egyes ROS példányhoz külön operációs rendszert is installálni kelljen, hanem egyugyanazon operációs rendszeren képes futtatni több ROS példányt is, ezzel erőforrásokat megtakarítva az adott gép hardware-én.

Végül a cikk áttekinti a CPS rendszerek tervezésének hardware aspektusait, kezdve az egyszerű Raspberry Pi-től az ipari PLC-n át – Programmable Logic Controller – a PC-kig. Operációs rendszernek – a PLC kivételével – Linux-ot ajánl, valamint a ROS-t, mint middleware-t.

Christopher Crick és munkatársai Rosbridge-el fogglalkozó cikke [6] bemutatja a Rosbridge-et, mint a ROS egy köztes réteg absztrakciós szintjét, interfészt a nem ROS felhasználók felé. Lényegében a Rosbridge létrehoz a ROS szolgáltatásaihoz egy roxy-t, amit más környezetekből fel tudunk használni, ráadásul akár az internetről is, mert a tűzfalbarát WebSocket-et használja a kliensek felé.

A cikk bemutatja a ROS-t is. Legfelül vannak az egymással üzeneteket váltó Node-ok, amik szervizekkel és topic-okkal kommunikálnak egymással. A szervizeket úgy kell elképzelni, mint távoli funkció hívásokat, míg a topic-ok olyan folyamatos adatfolyamok, melyekbe "bárki" beleírhat és melyek üzeneteire "bárki" feliratkozhat.

A Rosbridge egy következő absztrakciós szint a ROS-hoz képest. Képes igény szerint elindítani és leállítani ROS node-okat, a kommunikációt a kliensekkel pedig JSON formában bonyolítja. Lényegében bármely klienssel képes kommunikálni, amelyik "megérti" a JSON formátumot, Web alkalmazásokkal, vagy például Java, .NET szoftverekkel is. Nem csak HTML5 WebSocket-ekkel képes kommunikálni a klienseivel, hanem akár hagyományos POSIX IP socketekkel is. Kiadtak hozzá egy Javascript könyvtárat is a ROSJS-t, amit könnyen integrálni tudunk a Javascript-ben megírt programokhoz.

Ben Hu és munkatársai Cloudroid-al foglalkozó cikke [8] betekintést enged egy nyílt

forráskódú felhő keret-rendszerbe [11], mely lehetőséget biztosít a meglévő robotikai szoftverek számára, hogy felhőbe konvertálhatók legyenek, biztosítva számukra a teljes átalakítási transzparenciát az Internet-en elért felhő szolgáltatás felhasználása mellett, a megfelelő QoS – Quality of Service – biztosításával. A biztonságot növelő megoldása, hogy ha hálózati hiba keletkezik képes a kliensnek legenerált Stub visszakapcsolni lokális hívásra, hogy ne akadjon meg végzetesen az elkezdett folyamat.

A Cloudroid mögöttes technológiája a Rosbridge, annak absztrakciójára épül, hasonlóan, mint a Rapyuta publikus robotikai felhő szolgáltató rendszere, mindketten megvalósítják a PaaS – Platform as a Service – infrastruktúra szolgáltatást. Bevezet négy mehanizmust:

Öntartalmazó VM csomag:. Amikor egy ROS csomagot migrálunk a felhőbe, akkor az nem lenne képes kommunikálni a ROS Master-el, így ilyenkor automatikusan becsomagolódik egy Docker konténerbe, ami tartalmazni fogja a működéshez szükséges összes többi ROS csomagot.

Felhő bridging:. Az adoptált Rosbridge A ROS protokolját konvertálja JSON formátumúvá és WebSocket alapú protokollá..

Igény szerinti szolga példányosítás és szétosztás:. A Cloudroidban a szervizből két rész létezik: az interfész és a szolga, azaz egy Docker példány ami tartalmazza a ROS csomagot. Mivel a ROS egy robot kiszolgálására készült, ezért a Cloudroid minden kliens számára más-más Docker példányt készít.

Szolgáltatás Stub automatikus generálása:. A Cloudroid-on futó szolgáltatások elérhetőek bármely manuálisan megírt WebSocket kliensssel. Ezen túl a Cloudroid képes automatikusan legenerálni a szolgáltatás eléréséhez egy stub-ot, amivel kényelmesen elérhető lesz a szolgáltatás a kliensekről.

.

Russell Toris és munkatársai Robot Web Tools-al foglalkozó cikke [9] a nevezett, Rosbridgere épülő Javascript keretrendszerrel foglalkozik. A ROSJS mellett ez a másik feltörekvő, bár kicsit más készletet tartalmazó, Rosbridge-re épülő keretrendszer. Alapvetően három részből tevődik össze az RWT, a *roslibjs* egy kliens könyvtár, a *ros2djs* és a *ros3djs* pedig vizualizációs könyvtár.

A *roslibjs* felhasználható bármely helyen mindenféle vizualizációs függőség nélkül. Egy kliens könyvtár, ami képes kommunikálni a ROS topic-okkal és szervizekkel. Tartalmaz továbbá eszközöket az alapvető robotikai feladatok ellátására, például transzfomációk, URDF – Unified Robot Description Format – parser és alapvető mátrix, vektor számítások.

A *ros2djs* és a *ros3djs* vizualizációs könyvtárak a ROS-al összefüggő adatokhoz – az EaselJS és three.js továbbfejlesztései –, például robot modellek, vagy térképek, lézer letapogatások és pontfelhők számára. Mindkét könyvtár HTML5 `<canvas>` elemre rajzol.

Az RWT-ben külön figyelmet fordítottak a streaming adatok megfelelő kezelésére, például pontfelhők folyamatos megjelenítésére. Az RWT vizualizációs része használja a HTML5 `<video>` és `<img>` elemeit a megjelenítésükhöz VP8 codec felhasználásával.

Carla Mouradian és munkatársai cikke [10] egy nagyszabású, katasztrófa keresési és mentési feladatokat ellátó nagyon összetett rendszer tervezéséről szól, amit már csak a felhasznált felhős és egyéb technológiák száma is bizonyít. Az IoT-től – Internet of Things – kezdődően, az RFID-n át – Radio-Frequency Identification –, a különböző felhő technológiákig, SaaS, PaaS, IaaS. Az esettanulmány a már említett technológiák – úgy mint, ROS, virtualization, Rapyuta – felhasználásával épít ki egy komplex, katasztrófa környezetben használható, felhő alapú rendszert.

3. fejezet

Felhasznált technológiák bemutatása

A diplomamunka feladat által megvalósítandó PoC környezet kiépítéséhez több technológiát is meg kellett vizsgálni, mint lehetséges alternatívákat és a munkához megfelelőket ki kellett választani. A kialakítandó rendszerben használni kell majd saját szervereken futó virtualizációs környezeteket – leginkább konténer technológiával –, futtatni kell HTTP szervert, valamint ROS példányokat. A rendszer részei lesznek az Amazon AWS és Microsoft Azure felhő szolgáltatók által nyújtott virtualizációs és funkcionális szolgáltatásai is.

3.1. Operációs rendszer

A részletes feladatkiírásnak megfelelően elsősorban a nyílt forrású operációs rendszereket vettem számításba. A kereskedelmi Windows változatok teljesen kizáródtak a kiválasztásból, mert rajtuk nem fut a ROS [3]. Az open source operációs rendszerek közül a ROS által támogatottak tipikusan azok, melyek a Debian[13] platform csomagkezelő szolgáltatásait használják. Pillanatnyilag a három támogatott – Linux alapú, debian rendszerű – operációs rendszer az Ubuntu Wily (15.10-es verzió), az Ubuntu Xenial (16.04-es verzió) és Debian Jessie (8-as verzió). Ezek már mind elavult verziók, az Ubuntu LTS[1] a 18.10-es verziónál tart, míg a Debian a 9.4.0 verziónál.

A választás másik aspektusa, hogy az említett felhő szolgáltatóknál mely Linux verziók támogatottak a háromból. A Microsoft Azure az Ubuntu Server 16.04 LTS verzióját, valamint a Debian 8 Jessie verziót is, azaz kettőt is a ROS platformjai közül. Az Amazon AWS szintén támogatja a Debian 8 Jessie verzióját, valamint az Ubuntu 16.04 LTS-t is.

Az operációs rendszer kiválasztásánál a követelményeknek megfelel mind az Ubuntu

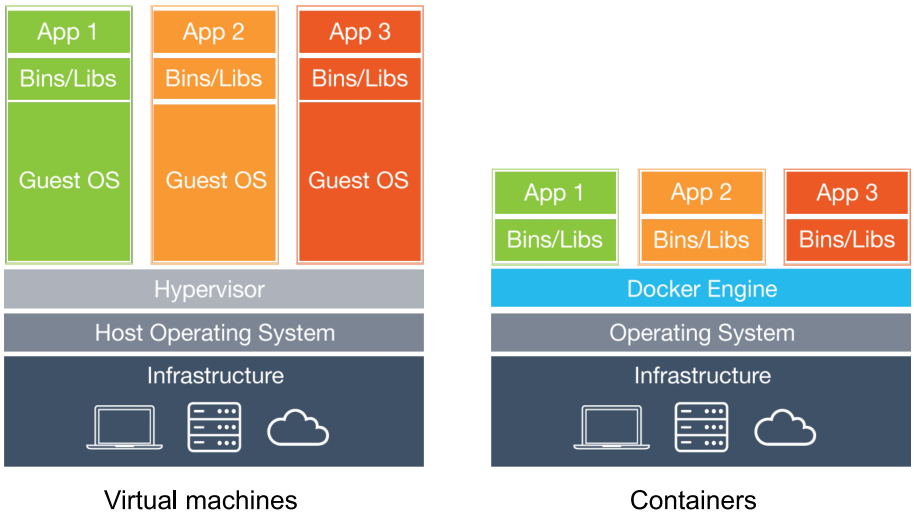
[1] LTS - Long Term Support (hosszútávú támogatás)

16.04 Xenial, mind a Debian 8 Jessie, így a döntést a Robot laborban meglévő szokások és tudás dönti el. A kollégák, akik majd a PoC rendszeren fognak dolgozni és a későbbiekben fel tudják építeni belőle a saját magukra hangolt rendszert, inkább az Ubuntu szoftveréhez értenek, így a választásom is erre esett, azaz **Ubuntu 16.04 LTS Server**

3.2. Virtualizációs technológia

A virtualizáció alatt általában azt értjük, ha egy fizikai számítógépen egyszerre több, egymástól független, izolált operációs rendszert tudunk futtatni, feléjük emulált virtuális számítógépet mutatva. Az izolációval el tudjuk érni, hogy az egyes operációs rendszerek ne tudjanak még végzetes hiba esetén se egymásra hatni, illetve a virtualizáció miatt a fizikai számítógép erőforrásait – processzor, RAM, disk, egyéb IO – is optimálisabban, a szükségleteknek megfelelően tudjuk szétosztani az operációs rendszerek között.

Manapság már a virtualizáció jelentése kibővült a konténer technológiákkal is. Az alapötlet, hogy nincs szükség a teljes operációs rendszer virtualizációra a költséges memória és processzor többlet terheléssel, hanem elegendő lenne a szoftvereket egymástól elkülöníteni azok eltérő környezetével együtt, a közösen használt operációs rendszer elegendő védelmet nyújt a processzek[2] elválasztásához, elegendő lenne az operációs rendszer kernelében[3] elkülöníteni a folyamatot.



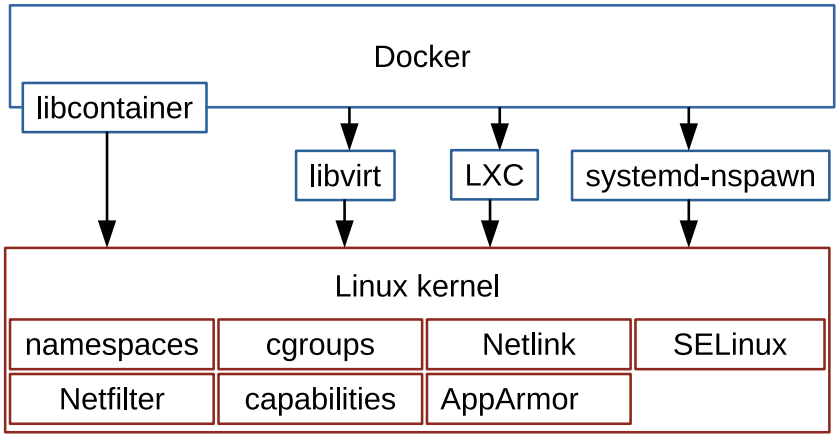
3.1. ábra. VM és Konténer különbsége

[2] Egymástól elkülönített folyamatok védett memóriával
[3] Az operációs rendszer magja, mely a processzeket futtatja

A 3.1 ábrán a 12. oldalon erre látunk példát. A fizikai gép a gazda[4] operációs rendszert futtatja, ami fölött fut egy virtualizációs[5] réteg, ami futtatja az egyes vendég[6] operációs rendszereket. Ezeken a vendég operációs rendszereken futnak a nekünk szükséges alkalmazások a nekik szükséges környezettel – például kódkönyvtárak, egyéb szükséges szoftverek –.

A másik példa a konténer technológiát mutatja be, jelen esetben a Dockerrel. Itt a fizikai gépen az az operációs rendszer fut, amin a virtualizált alkalmazásokat akarjuk futtatni. A Docker engine, beépülve az operációs rendszer kernelébe jelenti a következő réteget, mely már képes a konténereket futtatására, azaz a konténerizált processzek és környezetük izolációjára.

Jól látszanak az előnyök, megspóroltuk az izolált disk területnek a vendég operációs rendszerekhez szükséges mennyiségét, megspóroltuk a RAM-nak a vendég operációs rendszerek futtatásához szükséges többlet memória területét, megspóroltuk a vendég operációs rendszerek processz kezeléséhez szükséges többlet processzor kapacitását. Viszont nem menjünk el a nyilvánvaló hátrányok mögött se. A konténer technológia nem képes többféle operációs rendszer futtatására, se többféle frissítési szintű, verziójú, de azonos operációs rendszer kezelésére se.



3.2. ábra. Docker kapcsolata a Linux kernellel

A Docker elődje a 2008-ban megjelent LXC [7] volt, amely ugyan konténer alapú, de teljes virtuális gépeket kezel. Ezzel szemben a Docker a konténer formátum egységesítése mel-

[4] Host Operating System
[5] Hypervisor
[6] Guest Operating System
[7] Linux containers

lett komoly erőfeszítéseket tett az egyetlen program kedvéért létező konténerek működéséért.

Architektúráját a 3.2 ábra a 12. oldalon szemlélteti. Bemutatja, hogy a Docker milyen kapcsolatban áll a Linux kernellel. Látható, hogy a namespace-k és az erőforrások kiosztásáért, limitációjáért felelős cgroups is a kernel részegysége. A libcontainer a jelenlegi, az LXC driver-t váltó megoldás, gyakorlatilag nem más, mint egy egységes interfész konténerek létrehozására. A libvirt egy API, egy menedzsment eszköz a virtualizációhoz. A systemd-nspawn egy chroot-hoz hasonló megoldás, azonban ez utóbbi teljesen virtualizálja a fájlrendszer hierarchiáját, a folyamatokat, illetve az IPC alrendszereket.[16]

A PoC-hoz szükséges lesz virtualizációs technológia használatára, ahol az előző pontban megvizsgált operációs rendszereken virtualizálva tudjuk majd futtatni a ROS példányokat. Alapvetően kétféle virtualizáció lehetséges. Vagy minden egyes ROS példánynak allokálunk egy gazda-gépen futó, komplett vendég operációs rendszert, ami megfelel a hagyományos virtualizációs technológiának, vagy pedig valami konténer technológiát használunk. A ROS követelményeit áttekintve a feladat nem igényel feltétlenül nehézsúlyú megoldást, nincs szükség sokféle operációs rendszerre és a megvizsgált felhő technológiák is támogatják, ezért a Docker konténer technológiáját választottam.

3.3. Felhő szolgáltatások

A feladat részét képezi több felhő technológia kipróbálása, rendszerbe integrálása. Alapvetően két univerzális felhő szolgáltatót fogok kiértékelni. Az egyik az Amazon AWS-e a másik a Microsoft Azure szolgáltatása. Ezen kívül kiértékelésre kerül még egy robotikára specializált, ROS alapú felhő szolgáltató is, a Rapyuta[15].

Az Amazon Docker alapú felhő szolgáltatása az Amazon ECS[8], ami támogatja a kiválasztott Ubuntu 16.04 verziójú Linux operációs rendszert és a ROS-t is. A Docker image elkészítése után az feltölthető az Amazon ECR[9]-be, majd elindítható akár több példányban is.

A Microsoft Azure szolgáltatása a Docker Enterprise Edition-t használ. Az Amazon-hoz képes sokkal több platformot támogat, beleértve a számunkra szükséges Ubuntu 16.04-es verziót is.

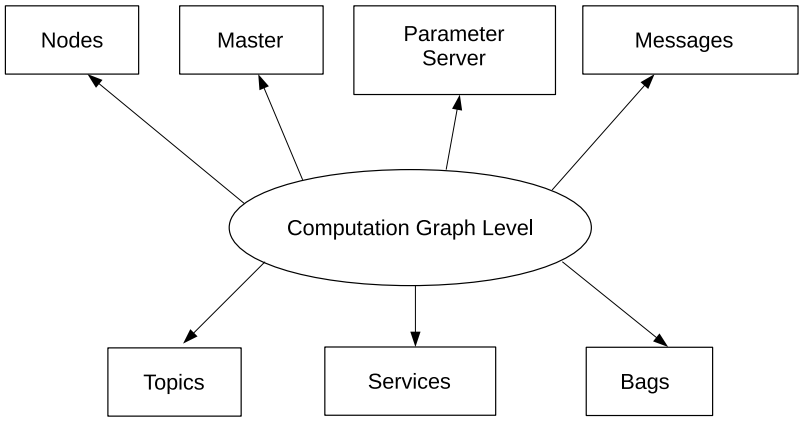
[8] Elastic Container Service
[9] Elastic Container Registry

A Rapyuta szolgáltatása már eleve azt a réteget valósítja meg, amit létrehozunk lokális és univerzális felhő szolgáltatásokkal a feladat során. Az alapvetően robot kontrollerre tervezett ROS-t a Rosbridge segítségével teszi elérhetővé a IP hálózaton keresztül, valamint a Cloudroid segítségével hoz létre egy menedzselt szolgáltatást.

3.4. ROS – Robot Operating System

A feladathoz felhasznált alap, a ROS [3], melynek szolgáltatásai kívánjuk a TCP/IP hálózaton kerewsztül elérni valamint Internet-en keresztül távolról szolgáltatni. Alapvetően arra tervezték, hogy robot kontrolleren fusson és helyi, egyedi protokollokon kommunikáljon a többi ROS példánnyal, Node-al.

A megnövekedett feladatok és új kihívások miatt egyre sürgetőbbé vált, hogy elérhető legyen a megszokott Internetes protokollokon keresztül, funkcionalitása felkerüljön a felhőbe is. Sajnos eredeti tervezése szerint nagyon kötődik a fizikai robothoz, az alap struktúra az egy robot egy ROS példány, a kontextusfüggő paraméterek mindenképpen ROS szinten találhatók meg. Itt tudjuk megadni a konkrét robot paramétereit, valamint a környezetének a leírását is. Viszont van számos olyan funkcionalitás a ROS-hoz – a hozzá kapcsolódó kiterjedt modul könyvtárban –, melyek nem kontextus függőek és így könnyen leválaszthatóak szolgáltatásként.



3.3. ábra. ROS Architektúra

Alapvetően a ROS egy middleware, mely bizonyos Linux disztribúciókra épül rá. Tartalmaz számos kódkönyvtárat, valamint több szolgáltatást is nyújt. Architektúráját a 3.3 ábra szemlélteti. A ROS számítási háló szinten a következő elemeket tartalmazza:

Nodes. A csomópont az operációs rendszeren futó processzt reprezentálja, valamennyi egyéb szolgáltatásnak hozzá kell tartoznia egy Node-hoz. Tipikusan egy rendszer több Node-ot tartalmaz, funkcionalitás szerint szétbontva a feladatokat.

Master. A Master a Node-ok számára szolgáltat egy regisztrációs szolgáltatást, ahol a Node-ok meg tudják "hirdetni" szolgáltatáskészletüket. Ha a Master nem működik a rendszerben, akkor a Node-ok szolgáltatásai sem érhetők el. Egy elosztott rendszerben elegendő egy Master egy számítógépen, míg a Node-ok futhatnak több gépen is.

Parameter Server. Gyakorlatilag egy Key-Value store, NoSQL adatbázis szerver egy központi helyen. A Node-ok tudják felhasználni a benne tárolt központi paraméterek értékeit a konfigurációjukhoz, működésükhöz.

Messages. A Node-ok üzenetekkel kommunikálnak egymás között. Az üzenet olyan adatokat tartalmaz, amelyek információkat ad más csomópontoknak. A ROS-nak sokféle üzenete van, és a szokásos üzenettípusok segítségével akár saját típusú üzenetet is létrehozhatunk.

Topic. (Téma) Minden üzenetnek nevének kell lennie, amelyet a ROS hálózat elindít. Ha egy csomópont adatokat küld, azt mondjuk, hogy a csomópont közzétesz egy üzenetet a Topic-ban. A csomópontok más csomópontok témáit is megkaphatják, egyszerűen feliratkozva a témára. A csomópont előfizethet egy témára, még akkor is, ha nincsenek olyan más csomópontok, amelyek erre a konkrét témára vonatkoznak, ami lehetővé teszi számunkra, hogy elválasszuk a termelést /publish/ a fogyasztástól /subscribe/. Ez egy hagyományos, úgynevezett publisher / subscriber modell. Fontos, hogy a topic-ok neve egyedi legyen !

Services. (Szolgáltatások) A témák közzétételekor sokfelé küldhetünk üzenetet, több feliratkozónak is, de ha választ is akarunk akkor ezt nem tehetjük meg a témákkal. A szolgáltatások lehetővé teszik számunkra a csomópontokkal való interakciót. A szolgáltatásoknak egyedi névvel kell rendelkezniük. Ha egy csomópontnak van egy szolgáltatása, az összes csomópont kommunikálhat vele, köszönhetően a ROS ügyfélkönyvtárainak.

Bags. A táskák egy formátum ROS üzenetek elmentésére és visszajátszására, fontos mechanizmus az adatok tárolására, – például a szenzora adatok eltárolására a teszteléshez, fejlesztéshez és ezek szükség szerinti visszajátszása –.

4. fejezet

A megvalósítandó rendszer specifikációja

A megvalósítandó rendszer egy PoC, ami nem feltétlenül egy feladatra optimalizált konkrét megvalósítás, sokkal inkább a technológiák kipróbálására szolgáló megoldás, így minden kipróbálandó technológia része lesz a kialakított architektúrának. Alapvetően célja, hogy a robotikai kutatások, alkalmazások számára ki lehessen választani a legjobb megoldást a rendelkezésre álló eszközkészletből. A rendszer által megvalósított példa feladat a pályatervezés lesz, amit a rendszerben különféle pontokra kihelyezve tesztelünk sebességre és válaszidőre.

A rendszerhez felhasználjuk a Cloudroid[8] szoftvert a ROS szolgáltatásainak kiajánlására az Internet alapú hálózat felé.

Irodalomjegyzék

[1] Péter Galambos: Cloud-, Fog-, and Mist Computing in Service of Advanced Robot Applications, *ide be kell irni majd, hogy GP cikke hol jelent meg !*, 1969, pp.111-222.

[2] Xi Vincent Wang, Lihui Wang, Abdullah Mohammed, Mohammad Givehci (Department of Production Engineering, KTH Royal Institute of Technology, Stockholm, Sweden): Ubiquitous manufacturing system based on Cloud: Robotics, em ELSEVIER, Robotics and Computer-Integrated Manufacturing, 45.szám, 2017, pp.116-125

[3] Robot Operating System (http://www.ros.org/about-ros/), utoljára megtekintve: 2018.11.02.

[4] Adarsha Kharel, Dorjee Bhutia, Sunita Rai, Dhruba Ningombam: Cloud Robotics using ROS, *International Journal of Computer Applications* ® (IJCA) (0975 ? 8887), *National Conference cum Workshop on Bioinformatics and Computational Biology*, NCWBCB- 2014, pp.18-21.

[5] Pablo González-Nalda, Ismael Etxeberria-Agiriano, Isidro Calvo: A modular CPS architecture design based on ROS and Docker, em ©Springer-Verlag France, 2016, pp.950-955.

[6] Christopher Crick, Graylin Jay, Sarah Osentoski, Benjamin Pitzer, Odest Chadwicke Jenkins: Rosbridge: ROS for Non-ROS Users ©*Springer International Publishing Switzerland*, 2017, pp.493-503

[7] Docker lightweight container engine (https://www.docker.com/products/docker-engine), utoljára megtekintve: 2018.11.02.

[8] Ben Hu, Huaimin Wang, Pengfei Zhang, Bo Ding, Huimin Che: Cloudroid: A Cloud Framework for Transparent and QoS-aware Robotic Computation Outsourcing, *IEEE 10th International Conference on Cloud Computing* 25-30 June 2017, pp.114–121

[9] Russell Toris, Julius Kammerl, David V. Lu, Jihoon Lee, Odest Chadwicke Jenkins, SarahOsentoski, Mitchell Wills, Sonia Chernova: Robot Web Tools: Efficient Messaging for Cloud Robotics, *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems* 2015-09, pp.4530–4537

[10] Carla Mouradian, Sami Yangui, Roch H. Glitho: Robots as-a-Service in Cloud Computing: Search and Rescue in Large-scale Disasters Case Study, 15^{th} *IEEE Consumer Communications and Networking Conference, Las Vegas, USA* 12-15 January 2018, pp.1–7

[11] Cloudriod cloud robotic platform Website (https://github.com/cyberdb/Cloudroid), utoljára megtekintve: 2018.11.02.

[12] Dr. Szabó Zsolt, Budai Csaba, Dr. Kovács László, Dr. Lipovszki György: Robotmechanizmusok Web site tananyag, robotok csoportosítása fejezet (http://www.mogi.bme.hu/TAMOP/robotmechanizmusok/ch14.html), utoljára megtekintve: 2018.12.12.

[13] Debian operációs rendszer (https://www.debian.org/), utoljára megtekintve: 2018.12.12.

[14] Ubuntu operációs rendszer (https://www.ubuntu.com/), utoljára megtekintve: 2018.12.12.

[15] ROS-t és Cloudroid-ot felhasználó specifikus robot-felhő szolgáltató (https://www.rapyuta-robotics.com/), utoljára megtekintve: 2018.12.12.

[16] Ákos, Kovács: Docker, a konténer alapú virtualizáció em Refaktor Magazin 2018.12.13. (https://www.refaktor.hu/docker-a-kontener-alapu-virtualizacio-1-resz/), utoljára megtekintve: 2018.12.12.