

**SZÁMÍTÁSI FELHŐK EGYÜTTMŰKÖDÉSÉNEK ELŐSEGÍTÉSE
ADATFELHŐK HASZNÁLATÁVAL**

Pflanzner Tamás
II. évf. programtervező informatikus MSc.

Témavezető: Dr. Kertész Attila
SZTE TTIK Szoftverfejlesztés Tanszék

TARTALOMJEGYZÉK

1.	SZÁMÍTÁSI FELHŐK.....	1
1.1.	BEVEZETÉS	1
1.2.	OPENNEBULA	2
1.3.	PROBLÉMAFELVETÉS.....	4
2.	ADATFELHŐK.....	5
2.1.	ÖSSZEHASONLÍTÁS.....	5
2.2.	DROPBOX.....	6
2.2.1.	<i>Fejlesztés</i>	6
2.2.2.	<i>Műveletek</i>	7
3.	HASZNÁLATI ESET.....	10
3.1.	ANDROID API HASZNÁLATA.....	10
3.2.	DROPBOX FIGYELÉS A FELHŐBŐL	12
3.2.1.	<i>Dropbox figyelő</i>	12
3.2.2.	<i>Web Szolgáltatás</i>	16
3.2.3.	<i>Virtuális gép</i>	17
3.2.4.	<i>Számítási felhő</i>	18
3.3.	HASZNÁLATI ESET ÖSSZEFOGLALÁSA	21
4.	ÖSSZEFOGLALÁS	23
4.1.	TOVÁBBFEJLESZTÉSI LEHETŐSÉGEK, CÉLOK	23
4.1.1.	<i>A számítási felhőket érintő tervek</i>	23
4.1.2.	<i>Az adatfelhőkkel kapcsolatos további fejlesztések</i>	24
4.1.3.	<i>A kliens alkalmazással kapcsolatos tervek</i>	24
4.1.4.	<i>Konkrét alkalmazás terve</i>	24
	IRODALOMJEGYZÉK	26
	MELLÉKLETEK.....	I
	1. MELLÉKLET: NÉHÁNY NAGYOBB ADATFELHŐ SZOLGÁLTATÓ ÖSSZEHASONLÍTÁSA	II

TÁBLÁZAT- ÉS ÁBRAJEGYZÉK

1. táblázat: Dropbox API hibakódok	7
1. ábra: Sunstone Dashboard.....	3
2. ábra: Sunstone Marketplace	3
3. ábra: A VMware virtuális gép beállításai.....	18
4. ábra: Új képfájl létrehozása, hálózati elérési út is megadható	19
5. ábra: A Sunstone felülete	20
6. ábra: Template létrehozása.....	21

1. SZÁMÍTÁSI FELHŐK

Napjainkban egyre nagyobb teret hódítanak a különböző felhőszolgáltatások (Cloud Computing). Számos nemzetközi szabványosító szervezet definiálta a különféle felhő megoldások gyakorlati alkalmazását, tipikusan nyilvános, magán és hibrid vagy közösségi felhőkről beszélhetünk a számítási felhők tekintetében. E két utóbbi megoldás több felhő együttes használatát teszi lehetővé, amit felhő federációnak is neveznek. Az egyik megoldatlan probléma az ilyen federációkban az adatok közös, együttes kezelése. A felhő szolgáltatások egy másik csoportját az adat felhők alkotják, amelyek segítségével különféle felhasználói adatot tárolhatunk a felhőben. Dolgozatom célja egy olyan módszer kidolgozása, ami az adat felhők szolgáltatásait használja fel ahhoz, hogy megoldja a különféle számítási felhők adat-szintű együttműködését.

1.1. BEVEZETÉS

Európában az Európai Bizottság által felkért szakértői csoport készített egy riportot [1] a felhő szolgáltatások jövőjéről. Ebben a cikkben a szolgáltatói felhőket 5 csoportba osztották: az első a Magán felhők (private) csoportja, amelyek olyan erőforrásokból állnak, amiket egy adott céghez vagy intézethez tartozó infrastruktúra szolgáltató üzemeltet. Tipikusan ilyen felhők nyújtanak SaaS-szintű (Software as a Service) szolgáltatásokat, mint például az eBay.

A második a nyilvános (public) felhők kategóriája, ahol a működtető cég külső felhasználók számára szolgáltat, esetlegesen több infrastruktúrát is igénybe véve. Ezen felhőket igénybe véve cégek költséghatékonyban működtethetik szolgáltatásaikat. Ide tartoznak az Amazon, a MS Azure és a Google Apps felhői. A harmadik csoportot a hibrid (hybrid) felhők alkotják, amelyek magán és nyilvános felhőket is magukba foglalhatnak összetettebb alkalmazások támogatására (pl. személyes adatok kezelésére magán felhő használata, egyébként nyilvános). A negyedik kategóriát a közösségi (community) felhők alkotják, melyekben különféle résztvevők ajánlanak be erőforrásokat, hogy egy közös infrastruktúrával hozzanak létre egy magán vagy nyilvános felhőt. Általában kisebb cégek profitálhatnak leginkább ilyen felhőkből, példa erre a kategóriára a Zimory. Végül az ötödik csoportot az egyéb speciális célú (special purpose) felhők alkotják, melyek valamilyen speciális funkcionalitással rendelkeznek, egyedi szolgáltatást nyújtanak, mint például a Google Docs.

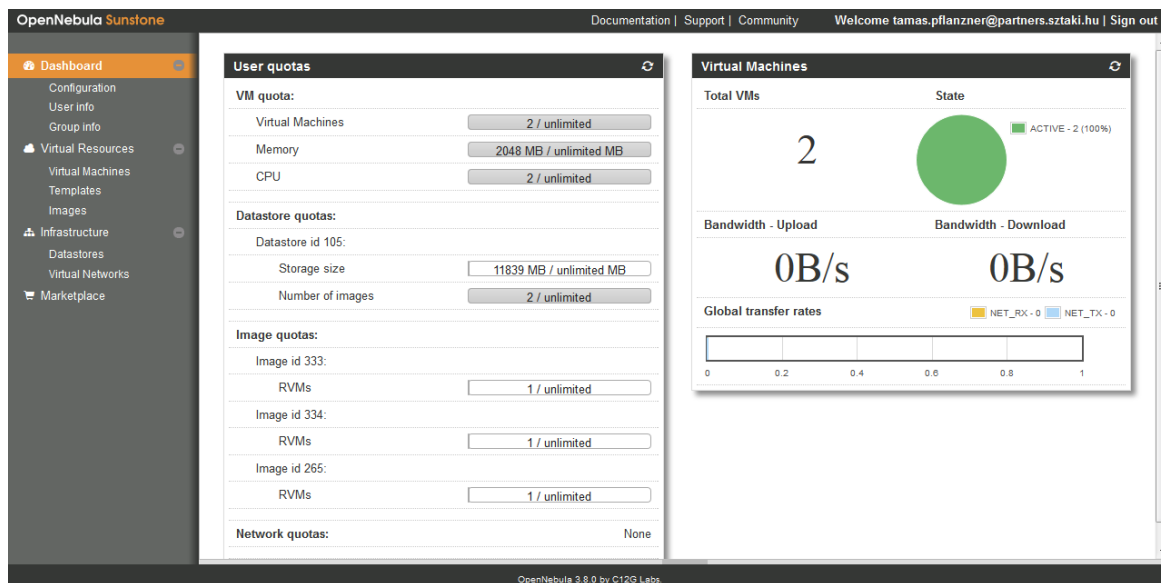
Az amerikai Szabványok és Technológiák Nemzeti Intézete (National Institute of Standards and Technology - NIST) is készített egy hasonló célú dokumentumot [2], ami-ben 4 változatot különítenek el: az első a magán (private) felhő, mely infrastruktúráját ki-fejezetten egy szervezet számára tartják fenn, és maga a szervezet vagy egy harmadik fél üzemelteti lokálisan vagy távolról. A második a közösségi (community) felhő, melyet több szervezet megosztva használ egy közös cél kiszolgálására, amit a vagy a szervezetek együttesen, vagy egy harmadik fél üzemeltet. A harmadik a nyilvános (public) felhő, amely bármely külső felhasználó vagy egy nagyobb ipari csoport számára érhető el, és tu-lajdonosa egy felhőszolgáltatásokat nyújtó cég. Végül a negyedik a hibrid (hybrid) felhő, ami két vagy több felhő együttese, ahol a résztvevő felhők különböző felek, melyek standardizációs vagy egyéb technológiai megoldásokkal teszik lehetővé összetettebb al-kalmazások futtatását.

1.2. OPENNEBULA

A számítási felhők esetében a továbbiakban az infrastruktúra felhő szolgáltatásokat értjük (IaaS – Infrastructure as a Service). Ezen kategórián belül a gyakorlatban számos megva-lósítás született mind az iparban (pl. Amazon EC2), mind az akadémiában (pl. OpenStack és OpenNebula [4]). Ebben a dolgozatban bemutatott megoldáshoz az OpenNebula felhő megvalósítást használtam, így ezt mutatom be ebben a fejezetben. Egy számítási felhő há-rom fő részből épül fel, a Front-End-ből, a Tároló(k)ból, és a Hostokból. A Front-End Li-nux számítógépére van telepítve az OpenNebula [4], itt történik a menedzselés, az üteme-zés és a monitorozás és a webes interfész szolgáltatása. Hálózaton SSL kapcsolattal kom-munikál a Tárolóval és a Hostokkal. A Tárolókon találhatóan a virtuális gépek képfájljai, melyeket a Hostokon futtathatunk. A hálózat segítségével a Hostokon futtatott virtuális gépek is hálózatokba kapcsolhatóak. A Hostok rendelkeznek egy úgynevezett Hypervisorral, mely végrehajtja a Front-End utasításokat az adott hoszton. Az OpenNebula többféle Hypervisort támogat, ilyenek a KVM [10], a VMware [11] és a Xen [12]. Egy számítási felhőn belül lévő Hostok más-más Hypervisorral is rendelkezhetnek, az ütemezés és monitorozás ilyenkor is működik, azonban esetleg problémát okozhat a speciálisabb feladatoknál. A Front-End és a Hostok között megosztott fájlrendszer is lehet, ilyenkor lehetőség van futásban lévő virtuális gépek mozgására is.

Az OpenNebulát CLI-n (Command Line Interface) keresztül is vezérelhetjük, de rendelkezésre áll egy sokkal kényelmesebb mód, a Sunstone nevű webes grafikus felület.

A Dashboard részen az általános, felhasználóhoz tartozó adatokat találhatunk, mint például a virtuális gép és tárhely valamint hálózat használat mértéke.



1. ábra: Sunstone Dashboard

A 3.6-os verziótól kezdve teljesen integrálták a rendszerbe a Marketplace-t, mely előre elkészített, beállított virtuális gépek template-eit kínálja az OpenNebula felhasználóknak, így pár kattintással, könnyedén egy kész rendszer áll a rendelkezésünkre. A Marketplace nem csak a felhasználóknak hasznos, de a fejlesztőknek is fontos lehet, hiszen saját rendszerük elterjedését gyorsíthatják, ha előre elkészített template segítségével közzéteszik azt.

The screenshot shows the OpenNebula Sunstone Marketplace. It features a search bar and a table of available templates. The table has columns for Name, Publisher, Hypervisor, Arch, and Format. Below the table, it indicates 'Showing 1 to 10 of 13 entries' and provides pagination controls (First, Previous, 1, 2, Next, Last).

Name	Publisher	Hypervisor	Arch	Format
OpenNebula Sandbox KVM 3.4.1	C12G Labs	KVM	x86_64	raw
CentOS Server 6.2 - kvm	OpenNebula.org	KVM	x86_64	raw
ttlinux - kvm	OpenNebula.org	KVM	x86_64	raw
Ubuntu Server 12.04 (Precise Pangolin) - kvm	OpenNebula.org	KVM	x86_64	raw
CentOS Server 6.2 - VMware	C12G Labs	VMWARE	x86_64	vmdk
ttlinux - VMware	C12G Labs	VMWARE	i686	vmdk
CentOS Server 5.8 - xen	C12G Labs	XEN	x86_64	raw
Debian Squeeze - xen	C12G Labs	XEN	x86_64	raw
openSUSE Server 12.1	C12G Labs	XEN	x86_64	raw
Ubuntu Server 12.04 (Precise Pangolin) - xen	C12G Labs	XEN	x86_64	raw

2. ábra: Sunstone Marketplace

Az Infrastructure rész az adattárolással és a virtuális gépek közti hálózatok menedzselésével foglalkozik. A Virtual Resources szekcióban a képfájlokat, template-eket és a virtuális gépeket kezelhetjük.

1.3. PROBLÉMAFELVETÉS

Összességében elmondható hogy a legtöbb nemzetközi szervezet elkülöníti a magán és publikus felhőket, valamint egy külön kategóriát használ a több felhőből álló hibrid vagy federációs megoldásokra. Akár különálló felhők esetében akár hibrid felhők esetében problémát jelent az egyik típusú felhőben felhasznált ill. keletkezett adat átmozgatása és feldolgozása egy másik típusú felhőben (sőt a hasonló szoftverréteggel felállított felhők között sem egyértelmű az adatok együttes feldolgozása). Dolgozatom célja az eltérő technológiai megvalósítással rendelkező felhők federációjának adatkezelési problémájának egy lehetséges megoldása, amit a későbbiekben bemutatandó adattároló felhő szolgáltatások alkalmazásával valósítottam meg.

2. ADATFELHŐK

A felhőszolgáltatások egyik fajtája az adatfelhő szolgáltatás [3], melynek fő célja az adatok tárolása. A felhasználó adatokat tölthet fel vagy automatikus szinkronizációt állíthat be bizonyos mappákra, melyekhez más eszközökről is hozzáférhet az interneten keresztül. Nem csak az adathordozást könnyíti meg, de hasznos lehet különböző meghibásodások esetén biztonsági másolatként is. Szolgáltatótól függően egyéb hasznos funkciók is elérhetőek lehetnek, ilyen például a verziókövetés, a titkosítás, a publikus megosztás, a csapatmunka támogatása. Fejlesztők számára rendelkezésre állhat API és esetleg SDK is.

2.1. ÖSSZEHASONLÍTÁS

Az adatfelhő szolgáltatók száma rohamosan nő. A felhasználók számára is egyre nehezebb a választás. A szolgáltatást használók száma befolyásolja az alkalmazások fejlesztését is, mivel a legnépszerűbb szolgáltatásokat érdemes támogatni első sorban. Kivétel lehet, ha olyan speciális funkcióra van szükség, mely nem minden szolgáltatónál adott, ilyen lehet a verziókövetés vagy a titkosítás. Népszerűsége ellenére még viszonylag új technológia, ezért a felhasználók még csak ismerkednek vele, így az ingyenes szolgáltatások nagy előnyt élveznek. Elterjedt üzleti stratégia az ilyen szolgáltatóknál, hogy ingyenesen igénybe vehetőek, azonban bizonyos extra tulajdonságokért már fizetni kell. Ilyen extra lehet a nagyobb tárhely vagy a verziókövetés.

Az összehasonlítási szempontok között érdemes figyelembe venni az ingyenesen kapott tárhely méretét. A legtöbb szolgáltató felhasználóbázisuk növelése érdekében plusz tárhelyet ad, ha a felhasználó meghívja ismerőseit is. Az így szerorzhető maximális ingyenes tárhely mérete is fontos szempont lehet. Nem elhanyagolható az sem, hogy mely operációs rendszereket támogat a kliensprogram, beleértve a mobil platformokat is. A támogatott eszközök száma, a verziókövetés és a titkosítás terén is lehetnek különbségek. Fontos megkülönböztetni két titkosítási fajtát, az egyiknél a titkosítást feloldó kulcs a szolgáltató is birtokolja, míg a másik esetben egyedül a felhasználó jelszavával lehet feloldani a titkosítást. Az első fajta előnye, hogy ha a felhasználó elfelejti a jelszavát, akkor is visszanyerhetőek az adatok. A második fajta titkosítás előnye abban rejlik, hogy az esetleges szolgáltatást érő támadások során sem férhet más az adatokhoz, illetve az esetleges jogi végzés ellenére sem férhet hozzá az adatokhoz a hatóság sem. Fejlesztői szempontból fontos az API, illetve a fejlesztést nagyban megkönnyítő SDK megléte. Az általam kidolgozott részletes összehasonlító táblázat az 1. Mellékletben olvasható.

2.2. DROPBOX

Drew Houston azt a hétköznapi problémáját szeretne volna megoldani, hogy rendszeresen elfelejtette magával vinni a pendrive-ját. Mivel nem talált akkoriban megfelelően gyors, nagy fájlokat is kezelni tudó, megbízhatóan működő szolgáltatást, így elkezdett egy saját megoldást gyártani a problémájára, amikor rájött, hogy ez bizonyára sok más hasonló cipőben járó embernek hasznára válna. [8] 2007-ben megalapította a Dropbox Inc. nevű céget, majd 2008-ban meg is jelent a megoldást ígérő szolgáltatás. 2011-re már a piac 14.14%-át tudhatta magának. [9] Sikeréhez az is hozzájárult, hogy a Softbank és a Sony Ericson mobilszolgáltatókkal kötött szerződés miatt, a felhasználók telefonjára előre telepítve volt a Dropbox [5]. 2011 végére már több mint 50 millióan használták a szolgáltatást. Népszerűségét talán az bizonyítja legjobban, hogy a Kína kormány is felfigyelt rá, és blokkolta a kínai felhasználókat a hozzáféréstől. 2012 áprilisában hadat üzent a két legnagyobb ellenfelének a Google Drive-nak és a SkyDrive-nak, mivel az összes felhasználónak az eddigi 2 GB felett további 3 GB tárhelyet adott, ha a felhasználó automatikusan feltöltötte az általa készített képeket és videókat. 2012 júliusában egy alkalmazott tárhelyét feltörték, így sok Dropbox felhasználó adataihoz jutottak hozzá, melyeket spam küldésére használtak. [7] Az incidens hatására vezette be a Dropbox a két lépéses azonosítást, mely ugyan nem alapértelmezett, de a felhasználó bekapcsolhatja. 2012 szeptemberében a Facebook-kal társult, és a Facebook csoportokon belül közvetlenül a Dropbox tárhelyre tölthetnek fel adatokat a felhasználók.

2.2.1. Fejlesztés

A Dropbox jó dokumentációval rendelkezik a fejlesztők számára [6]. A részletes leírások mellé fejlesztőeszközök, úgynevezett SDK-k (Software development kit) letöltésére is lehetőség van. A hivatalos fejlesztői honlapról letölthető iOS, Android, Python, Ruby, Java és OS X SDK is, melyekhez dokumentációt és tutoriált is mellékelnek. A fejlesztők életét szintén megkönnyíti, hogy az egyes verzióváltásokhoz a változásokat is közzéteszik. Az SDK-t könnyen importálható a fejlesztői környezetbe, mely után a platform-specifikus könyvtárak segítségével elkerülhetővé válik nehézkes HTTP hívások intézése a Dropbox API felé.

Ahhoz, hogy egy alkalmazás a Dropbox API-t használhassa, előbb a hivatalos fejlesztői honlapon regisztrálni kell azt. A regisztráció során meg kell adni az alkalmazás nevét, rövid leírását, és ami a legfontosabb a típusát. Itt meg kell említenem, hogy a kapcsolódó útmutató (Branding guide) az elnevezéssel kapcsolatos ajánlásában leírja, hogy az al-

kalmazásunk neve lehetőleg ne tartalmazza a Dropbox szót, valamint a hozzá hangzásban vagy írásban hasonló szavakat, valamint a „Drop” szó ne szerepeljen prefixként. Kétféle alkalmazástípus létezik, melyeket a hozzáférési szintjük különböztet meg. Az úgynevezett App Folder hozzáférési szintű alkalmazás csak a felhasználó gyökérkönyvtárában található Apps speciális mappán belüli, az alkalmazás nevével azonos mappának a tartalmához férhet hozzá. A Full Dropbox tulajdonságú alkalmazás ezzel szemben a felhasználói fiókban található összes adathoz hozzáfér, ezért ennek szükségességére jó indokot kell adni a felhasználónak. Ezek után a fejlesztő két kódot kap, egy alkalmazás kulcsot (app key) és egy alkalmazás titkos kulcsot (app secret). Az alkalmazás kulcs segítségével azonosítható az alkalmazás, a titkos kulcs pedig arra szolgál, hogy az API hívásokat hitelesítse, aláírja.

2.2.2. Műveletek

A REST API segítségével kezelhetjük a Dropbox tárhelyet. Segítségével olyan platformokon is elérhetővé válik a tárhely és az összes funkcionalitás, melyre nincs SDK. A hívások csak és kizárólag biztonságos kapcsolaton (SSL) keresztül történhetnek. Azokban az URL-ekben, melyekben a fájl helyét is meg kell adni, meg kell különböztetni a teljes hozzáférésű és az alkalmazás mappa hozzáférésű alkalmazásokat, hiszen az előző esetben a tárhely gyökerétől, míg az utóbbiban az Apps speciális mappában lévő, az alkalmazással azonos nevű mappa a kiindulási pont. A megkülönböztetésnél a „dropbox” és a „sandbox” gyökérkönyvtár nevek használhatóak. A Dropbox Api kommunikáció UTF-8 kódolású szöveggel történik. A dátumok formátuma egységesen „%a, %d %b %Y %H:%M:%S %z” (például: „Sat, 21 Aug 2010 22:31:20 +0000”). Jelenleg az 1-es verzió az aktív, 2012 decemberétől az előző verzió már nem használható.

A hibakezelés a szabványos HTTP hibakód segítségével történik, míg az esetleges plusz információk a válasz tartalmában, JSON formában találhatók meg.

1. táblázat: Dropbox API hibakódok

400	Rossz bemeneti paraméter.
401	Rossz vagy lejárt azonosító. Újraazonosítás megoldhatja a problémát.
403	Rossz OAuth kérés.
404	Fájl vagy könyvtár nem található az adott helyen.
405	Váratlan kérés metódus (általában GET vagy POST használatos)
503	Túl sok kérés. (alkalmazásonként vagy felhasználónként)
507	Megtelt a tárhely.
5xx	Szerver hiba.

A hívások három nagyobb csoportra bonthatóak. Az első csoport az azonosítással foglalkozik, a második a fájlokkal és a metaadatokkal, a harmadik pedig azon speciális fájlműveleteket foglalja magába, melyek a tárhelyen belüli módosításokkal foglalkoznak.

Az azonosításnál az első lépés a `/request_token` hívás intézése, mely a további azonosítási folyamatban lesz használatos. Ezután az `/authorize` hívás hatására a felhasználó engedélyezheti az alkalmazásnak a tárhelyhez való hozzáférést. Ezt a hivatalos honlapon, vagy a hivatalos alkalmazásban teheti meg. Paraméterben megadható egy callback URL, melyre az elfogadás után át lesz irányítva a felhasználó. Abban az esetben, ha nem történt meg az engedélyezés, semmilyen értesítést nem kap az alkalmazás. Ez az egyetlen API hívás, mely a `www.dropbox.com` végpontot használja, az összes többi az `api.dropbox.com` vagy az `api-content.dropbox.com` URL-en keresztül történik. A sikeres azonosítás után az `/access_token` API hívással kaphatjuk meg a kulcsunkat.

Miután az azonosítás megtörtént, a felhasználói fiókról az `/account/info` hívással kérhetünk információkat. Egy JSON válaszban megkapjuk a megjeleníthető nevet, azonosítót, régiót, a használt és az elérhető tárhely méretét. A tárhelyen tárol fájlok és könyvtárak tulajdonságait a `/metadata` hívással kérhetjük el, mely támogatja a verziókezelést is, tehát régebbi fájlok metaadatait is le tudjuk kérni. Az adatok között szerepel a méret, a verziószám, az utolsó módosítás időpontja, könyvtárak esetében a tartalmazott elemek és még sok más információ. A rendszer fő hívásai a feltöltéshez használt `/files_put`, `/files`(POST) valamint a `/chunked_upload` és a letöltéshez használható `/files` (GET) hívások. A `/chunked_upload` nagy fájlok esetén vagy nem megbízható internetkapcsolat esetén ajánlott. A `/delta` API hívás is sűrűn használt, a helyi és a tárhely fájlrendszerét hivatott szinkronban tartani. Az ajánlott hívások közt eltelt idő körülbelül 5 perc, minden hívásnál megadja az előző hívás óta történt fájlváltozásokat. Speciális esetben, ha túl sok módosítás történt, akkor szükség lehet azonnali újrahívásra is. A verziókezelés a `/restore` hívással is használható, amennyiben egy régebbi változatot szeretnénk aktívra állítani. A megosztást a `/shares` és a `/media` hívásokkal intézhetjük, a `/media` esetében lehetőség van előnézet vagy stream nyújtására. Képek esetén is elérhetőek előnézetek a `/thumbnails` hívással, paraméterében megadhatjuk, hogy mekkora előnézeti képre van szükségünk. Keresési lehetőséget is biztosít az API a `/search` hívással, mely a fájlok és könyvtárak neveiben keres egy adott sztringre.

A tárhelyen belül másolhatunk a `/fileops/copy`, áthelyezhetünk a `/fileops/move` és törölhetünk a `/fileops/delete` hívásokkal. Létrehozhatunk könyvtárat is a `/fileops/create_folder` utasítással.

3. HASZNÁLATI ESET

A számítási felhők adatszintű együttműködését szeretném szemléltetni adatfelhők segítségével, ráadásképpen mobileszközök bevonásával. A rendszer célja, hogy a felhasználók egyszerűen ki tudják használni a számítási felhők adta lehetőségeket saját adataik feldolgozására, a fejlesztőknek pedig egységes környezetet biztosítson az ebben segítséget nyújtó alkalmazások fejlesztéséhez. Az általam mintaalkalmazásként bemutatandó megoldás egy felhasználó mobiltelefonján keletkezett adatot juttatja el egy adatfelhő segítségével egy számítási felhőbe, ahol azt feldolgozza. Az előállított adat szintén az adatfelhőn keresztül juthat vissza a felhasználó telefonjára, egyéb eszközére, vagy más felhőkbe további feldolgozásra.

3.1. ANDROID API HASZNÁLATA

A Dropbox az Android platformra való fejlesztés megkönnyítésére SDKt (Software Development Kit) ad a fejlesztők kezébe, melyet importálva a fejlesztőkörnyezetbe, könynyítik a Dropbox API használatát. Az Android SDKhoz dokumentáció, tutorial és példa-program is tartozik. A tutorial feltételezi, hogy az Eclipse környezet az Android fejlesztéshez szükséges beállításokat már tartalmazza. Az SDK JAR fájlja miután bekerült az Eclipse Android projekt könyvtárába, rendelkezésünkre áll az Eclipse-en belül a hozzá tartozó JavaDoc, illetve a fejlesztői munkát nagyban megkönnyítő auto-complete funkció. Az SDK-hoz tartozó példa-program, a DBRoulett, hasznos információkat ad az alapvető funkciók működéséről.

Egy Androidos Dropboxot használó program pár dologban tér el csak egy szerveroldali Java programtól, ezeket a különbségeket azonban érdemes észben tartani. Egy fontos dolog az alkalmazás titkos kulcsának tárolása. A telepített alkalmazáshoz bárki hozzáférhet, és kiolvashatja a titkos kulcsot, ha nem megfelelően tároljuk azt. A másik fontos eltérés az azonosítás során, hogy lehetőséget kell adni, hogy a felhasználó a dropbox honlapon engedélyezze az alkalmazást. Ez általában a dropbox honlapján történik, egy előre kidolgozott eseménysorozat révén.

```
// MyActivity below should be your activity class name
mDBApi.getSession().startAuthentication(MyActivity.this);
```

Ezután a hívás után a felhasználó a telefon böngészőjén bejelentkezhet, engedélyezheti az alkalmazást, majd automatikusan visszatér a Dropbox alkalmazásunkhoz, melynek az `onResume` metódusa hívódik meg. A Dropbox ajánlása szerint, ezt a metódust a tutorialban leírtak szerint egy az egyben másoljuk be az alkalmazásunkba:

```
protected void onResume() {
    super.onResume();

    // ...

    if (mDBApi.getSession().authenticationSuccessful()) {
        try {
            // MANDATORY call to complete auth.
            // Sets the access token on the session
            mDBApi.getSession().finishAuthentication();

            AccessTokenPair tokens = mDBApi.getSession().getAccessTokenPair();

            // Provide your own storeKeys to persist the access token pair
            // A typical way to store tokens is using SharedPreferences
            storeKeys(tokens.key, tokens.secret);
        } catch (IllegalStateException e) {
            Log.i("DbAuthLog", "Error authenticating", e);
        }
    }

    // ...
}
```

Az azonosítás után előfordulhat olyan hiba, hogy több program is feliratkozott a Dropbox callback függvényére. Ilyen esetben megoldás lehet az alkalmazás kulcs lecserélése.

A főbb API funkciók, mint a feltöltés, letöltés és mentaadatok kinyerése megtalálható a tutorialban. A többi hívásnál nincs Android specifikus rész, megegyezik a Java alkalmazásokban használhatókkal, melyet a következő fejezetben fogok bemutatni, a szerver oldali alkalmazásnál.

Számtalan olyan Android program tölthető le a Google Play áruházából, melyek a Dropboxot használják tárolásra. A jelen használati esetben azért tartottam fontosnak bemutatni az Androidon való fejlesztés különbségeit, mert a további tervekben egyedi szabályrendszert készítő, módosító és feldolgozó alkalmazásra lesz szükségem.

3.2. DROPBOX FIGYELÉS A FELHŐBŐL

3.2.1. Dropbox figyelő

Első lépésként egy olyan Java osztályra volt szükségem, mely egy könnyen használható vázat ad a Dropbox használatához. Ez azt jelenti, hogy úgy valósítja meg a szükséges funkciókat, hogy az alkalmazás és felhasználó függő adatokat a példányosításnál kelljen csak megadni, példányosítás után pedig egyszerűen metódushívásokkal történjenek a műveletek. Egy ilyen osztály implementálásánál nagy segítséget nyújt a Dropbox SDK-ban definiált osztályok. Ennek az alkalmazás és Dropbox közti kommunikációt és adatátvitelt megvalósító osztálynak a `CloudCooperationCore` nevet választottam. Nem csak a web szolgáltatásban hasznos, de egyéb, adatfelhőt vagy adatfelhőket használó alkalmazás is felhasználhatja. Adattagként tárolni kell az alkalmazás kulcsot (app key) és az alkalmazás titkos kulcsot (app secret), ezeken felül pedig legalább a megerősítés időtartalmáig a felhasználónevet és jelszót is. Ha elvárt viselkedés az esetleges újraazonosítás automatikus kezelése, azaz, hogy ne kelljen újra bekérni az adatokat ilyen esetben, akkor szükség van a Dropbox felhasználónevet és jelszót adattagként tárolni. Ez képesség a webszolgáltatásokba való integráláskor nagyon hasznos tulajdonság, hiszen nem mindig van mód újra bekérni az adatokat. Itt szeretném megjegyezni, hogy bár a hivatalos tutorialban az app key és az app secret egy-egy statikus `private String` adattagja az osztálynak, ez a tárolási módszer nem mindig ajánlott. Az ilyen tárolás megfelelő lehet egy szerveroldali program esetében, de a kliensprogramban való használat esetén, nem könnyen, de mégis lehetséges kinyerni az alkalmazás titkos kulcsát. Egy lehetséges biztonságos megoldás, ha az alkalmazás ilyenkor először egy szervernél azonosítja magát és nem fér hozzá közvetlenül a dropbox tárhelyhez. További adattagok még egy `DropboxAPI` generikus osztály `WebAuthSession` osztállyal való példánya, mely az API hívásokat hitelesíti és egyszerűsíti azáltal, hogy a HTTP protokoll használatát egyszerű metódushívásokkal válthatjuk ki. A későbbiek során, ha a delta hívást szeretnénk használni a fájlváltozások detektálására, szükség lehet egy `DeltaPage` generikus osztály `Entry` osztállyal való példányára is, mely segítségével az egyes hívások közti változásokat tudjuk lekérdezni.

3.2.1.1. Azonosítás

Mielőtt bármilyen műveletet végre lehetne hajtani, szükség van az azonosításra. A rendelkezésre álló alkalmazás kulcsot és titkos kulcsot egy `AppKeyPair` osztálypéldányba kell csomagolni, majd átadni a `WebAuthSession` konstruktorának az `AccessType` azon kons-

tansának kíséretében, mely az alkalmazás hozzáférési szintjét jelzi. Ezt a WebAuthSession példányt kell a DropboxAPI konstruktorának adni.

```
AppKeyPair appKeys = new AppKeyPair(APP_KEY, APP_SECRET);  
WebAuthSession session = new WebAuthSession(appKeys, ACCESS_TYPE);  
mDBApi = new DropboxAPI<WebAuthSession>(session);
```

Ezek után a felhasználónak a www.dropbox.com oldalon belépve vagy a hivatalos dropbox mobil alkalmazásban engedélyeznie kell az alkalmazás hozzáférését. Erre 5 perces határidő van, így a webszolgáltatások esetében ez sem feltétlenül megoldható. Ennek a problémának a megoldására egy CloudCooperationAllow nevű statikus osztályt definiáltam, mi a HtmlUnit nevű GUI nélküli Java böngészőt használja. Az osztály allow nevű statikus metódusa megkapja a felhasználónevet, a jelszót és a WebAuthSession példány getAuthInfo() metódusából kinyert példány url nevű adattagját, majd a felhasználót szimulálva a HtmlUnit böngészője segítségével engedélyezi az alkalmazás hozzáférését.

Ezek után a session tartalmazza a felhasználó hozzáférési kulcsot, mely a DropboxAPI példány getSession metódusától visszkapott session getAccessTokenPair() metódussal nyerhető ki, AccessTokenPair típusú változóként. Ezt érdemes eltárolni, így nem kell minden programindításkor újra elvégezni az azonosítást, elég csak a setAccessTokenPair metódussal átadni őket a session-nek.

```
AccessTokenPair access = getStoredKeys();  
mDBApi.getSession().setAccessTokenPair(access);
```

3.2.1.2. Fájl feltöltés

A REST API-ról szóló részben leírtak alapján többféle módszert választhatunk egy fájl feltöltésére. Mindenképpen szükség van a feltöltendő fájl elérési útjára, a tárhelyen kívánt helyének az elérési útjára és a teendőre az esetleges ütközés esetén. A választás a fájl méretétől, az internetkapcsolat megbízhatóságától és az esetleges felhasználói igényektől függ. Egyik lehetőség a /files hívás, azonban egyszerűsége miatt ajánlott helyette a /files_put hívást használni. A másik lehetőség a /chunked_upload használata, ami a nagyobb fájlknál illetve akkor ajánlott, mikor az internetkapcsolat nem megbízható és gyakran megszakad. Ilyenkor lehetőség van a feltöltés későbbi folytatására, ha az egyes darabok feltöltése közti idő nem nagyobb, mint 24 óra.

```
public void upload(String fromPath, String toPath, int overwrite){
    File tf = new File(fromPath);
    InputStream tfis=null;
    try {
        tfis = new FileInputStream(tf);
    } catch (FileNotFoundException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
        logger.log(Level.WARNING, "File not found!");
    }
    Entry upRet=null;
    try {
        upRet = mDBApi.putFileOverwrite(toPath, tfis, tf.length(), null);
    } catch (DropboxException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
        logger.log(Level.WARNING, "Upload error!");
    }
    logger.log(Level.INFO, "Upload Done!");
}
```

3.2.1.3. Metaadatok lekérése

Egy fájl vagy könyvtár adatait a metadata nevű metódussal tudjuk lekérni, melynek öt paramétere van. Meg kell adni a kívánt fájl elérési útját és könyvtár esetén hány elemét kérjük le. Ez alapértelmezetten 25000, ha nullára van állítva az érték. Ha túl sok elemet kaptunk akkor 406-os hibakódot kaphatunk. Fájllok esetén 1 az ajánlott érték. Továbbá lehetőség van megadni egy hash-t, mely akkor hasznos, ha egy könyvtár metaadatait már elmentettük, ilyen esetben ha egyezik a hash, tehát nem változott a könyvtár, akkor 304-es hibakódot fogunk kapni, és nem kell újra lekérdezni a metaadatokat. Egy boolean változó-

val igaz értékével megadhatjuk, hogy könyvtár esetén a leszármazottainak metaadatait is kérjük-e, hamis érték esetén csak a könyvtár metaadatait küldi a szerver. Fájlok esetén figyelmen kívül hagyja a rendszer a paramétert. Az utolsó paraméter egy verziószámot tartalmazhat, ha egy régebbi fájl metaadataihoz szeretnénk hozzájutni. Ha az érték null, akkor a legfrissebb metaadatokat kapjuk meg.

A metaadatok a fájl vagy könyvtár méretét, útvonalát, utolsó módosítás dátumát, a hash-ét és a verziószámát tartalmazzák.

```
public Entry getMetaData(String path){
    Entry metaRet=null;
    try {
        metaRet = mDBApi.metadata(path, 1, null, false, null);
    } catch (DropboxException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
        logger.log(Level.WARNING, "Metadata error!");
    }
    return metaRet;
}
```

3.2.1.4. Fájl letöltés

A fájl letöltése nagyon hasonló a feltöltés módjához. A `getFile` metódust használhatjuk, mely szintén a fájl elérési útját, ha régebbi verziót szeretnénk letölteni, akkor a verziószámát (null esetén a legfrissebb verziót kapjuk) egy `OutputStream` objektumot, és egy `ProgressListener`-t, mely segítségével nyomon követhetjük a letöltés állását.

```
public void download(String fromPath, String toPath, int overwrite){
    FileOutputStream outputStream = null;
    DropboxFileInfo info=null;
    try {
        File file = new File(toPath);
        outputStream = new FileOutputStream(file);
        info = mDBApi.getFile(fromPath, null, outputStream, null);
    } catch (Exception e) {
        logger.log(Level.SEVERE, "Fájl letöltési hiba!");
    } finally {
        if (outputStream != null) {
            try {
                outputStream.close();
            } catch (IOException e) {
                logger.log(Level.SEVERE, "Fájl lezársi hiba!");
            }
        }
    }
    logger.log(Level.INFO, "Download done!");
}
```

3.2.1.5. Delta

A delta metódus fő célja, hogy a dropbox tárhelyen lévő fájlok és könyvtárak állapotát figyelve a helyi fájlstruktúrát annak megfelelően módosítsuk. A javasolt hívási időköz 5 perc, de speciális esetekben több azonnali hívásra is szükség lehet. Paraméterben egy kurzort vár, mely ha null, akkor a tárhelyen található összes elem információját letölti, és visszatér egy DeltaPage objektummal, mely rendelkezik egy kurzorral. Amennyiben már rendelkezünk egy előző hívásból egy kurzorral, és azt paraméterben megadjuk, úgy a hívásnál csak az azóta változott adatokat kapjuk meg.

A visszaadott DeltaPage típusú objektum rendelkezik egy kurzorral, melyet a későbbi hívások során használható fel. Megtalálható benne az előző hívás óta változott fájlok és könyvtárak elérési útvai. A hasMore adattagja egy bool érték, mely ha igaz, akkor még további információk várnak a szerveren, ilyen esetben javasolt az azonnali újrahívás, hogy az összes adatot feldolgozhassuk. Egy rest bool értékkel is rendelkezik, mely speciális esetekben felvehet igaz értéket, ezzel azt jelezve, hogy mielőtt feldolgoznánk a visszakapott adatokat, érdemes alaphelyzetbe hozni a helyi dropbox könyvtárat, azaz az összes elemet kitörölni belőle.

A delta hívás elég egyszerű, az általam implementált változat két paramétert vár, melyben megadható, hogy az összes információt kérjük-e, vagy csak a legutóbbi hívás után történt változásokat, illetve, lehetőség van csak a fájlok vagy csak a könyvtárak listázására is.

3.2.2. Web Szolgáltatás

A fájlok változását figyelő program megvalósítására egy Java web service-t hoztam létre az eclipse fejlesztőkörnyezet segítségével. A webszolgáltatáson belül a konkrét figyelés megvalósítását egy ServletContextListener interfészt megvalósító Listener-ben oldottam meg. Az interfész, a contextIntialized, a contextDestroyed metódusokat tartalmazza, így követi a webszolgáltatás életciklusát. a contextIntialized függvény akkor hívódik meg, amikor a webszolgáltatás elindítása után készen áll a használatra. A másik függvény a webszolgáltatás szándékos vagy nem várt leállásakor hívódik meg. A contextIntialized metódusban egy ScheduledExecutorService osztályt példányosítottam, melynek a scheduleAtFixedRate metódus segítségével könnyen lehet ütemezett, külön szálakon végrehajtandó feladatokat végrehajtani. A függvény paraméterként vár egy Runnable osztályt, mely a végrehajtandó feladatot valósítja meg, egy kezdeti késleltetést, mellyel meg-

oldható, hogy ne azonnal kezdődjön az ütemezett végrehajtás első futása. Meg kell adni a feladat végrehajtásai közt kívánt időintervallum méretét, valamint egy `TimeUnit` konstans is, mely meghatározza, hogy a kezdeti késleltetési idő és a végrehajtások elindítása közt eltelt idő másodpercben, percben, órában vagy egyéb időegységben értelmezendő.

A `contextDestroyed` webszolgáltatás leállításakor meghívódó metódusban a `ScheduledExecutorService` példány `shutdownNow` függvényét meghívva állítottam le az ütemezett végrehajtások futtatását.

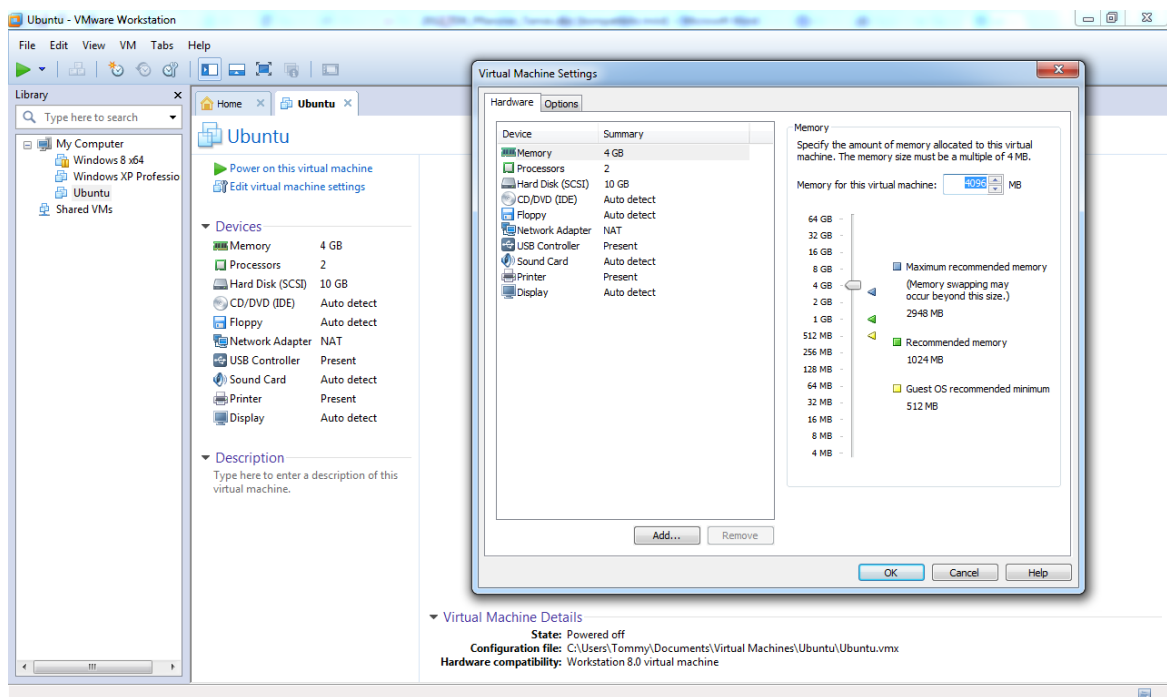
A tárhely új fájlokat figyelő `Runnable` interfészt megvalósító osztályt, melyet az ütemezőnek adok át, a fejezet elején bemutatott `CloudCooperationCore` osztály segítségével valósítottam meg. Az interfész miatt megvalósítandó `run` metódusban, ha még nincs `dropbox session`, azaz még nem vagyunk bejelentkezve, akkor a `login` függvénnyel ezt megteszi a program, majd a `logDelta` metódust hívom meg. Ez a metódus a `CloudCooperationCore` `delta` függvényét használja a „-f” paraméterrel, mely az előző `delta` lekérdezés óta a tárhelyre feltöltött új fájlok listáját adja vissza.

A webszolgáltatásban megvalósítottam egy logolást megvalósító részt is, mely tárol minden fontosabb eseményt a bekövetkezés dátumával együtt. Ilyen fontosabb esemény az webszolgáltatás életciklusában bekövetkező változás, a tárhelyre való bejelentkezés, a tárhelyre feltöltött új fájl észlelése. Ezeket az adatokat egy `Servlet` segítségével meg is jeleníthetjük.

3.2.3. Virtuális gép

Az OpenNebula rendszer rugalmassága miatt sokféleképpen megoldható, hogy az alkalmazást tartalmazó rendszert a felhőben futtassuk. Vannak olyan módszerek, melyekkel a virtuális gép előre történő elkészítése teljes mértékben kihagyható, például egy teljes operációs rendszert tartalmazó képfájl telepítése, és az alkalmazás utólagos bemásolása a felhőbe. Azonban mivel a jövőbeni célok között az is szerepel, hogy minél több számítási felhő szolgáltatót támogasson a rendszer, bár az OpenNebula nagyon elterjedt akadémiai körökben, ennek ellenére könnyen előfordulhat, hogy egy másik támogatni kívánt rendszer esetében az utólagos telepítés az interfészek különbözősége miatt problémás lehet, ezért bemutatom egy saját képfájl készítésének főbb lépéseit, ami már a teljes alkalmazást is tartalmazza. A két legelterjedtebb virtualizációval foglalkozó program a `VirtualBox` [13] és a `VMware`. Mindkét szoftver megfelel a feladatnak. A `VMware` és a `VirtualBox` képfájl formátumát közvetlenül támogatja az OpenNebula, de az egyéb képfájlok is könnyen konvertálhatóak valamilyen támogatott formátumra.

A VMware program könnyen kezelhető varázslója segítségével könnyen a szükségünknek megfelelő virtuális gépet hozhatunk létre. Vannak beépített operációs rendszerek is, de lehetőség van telepítő képfájl megadására is. A telepítés menetét nem részletezném, hiszen ez teljes mértékben az operációs rendszertől függ, az alkalmazásom pedig csak annyi követelményt támaszt vele szemben, hogy futtatható legyen rajta egy java webszolgáltatás. A virtuális gép létrehozásakor a hardver beállításoknál adható meg a processzorok és a processzoronkénti magok száma, a memória nagysága, a merevlemez mérete, a hálózati beállítások és még sok más egyéb.



3. ábra A VMware virtuális gép beállításai

3.2.4. Számítási felhő

Az alkalmazásom kipróbálásához az MTA SZTAKI (Magyar Tudományos Akadémia Számítástechnikai és Automatizálási Kutatóintézet) Párhuzamos és Elosztott Rendszerek Kutatólaboratórium (LPDS - Laboratory of Parallel and Distributed Systems) [14] OpenNebulára épülő felhőszolgáltatását használtam. Az első fejezetben bemutatott OpenNebula eszköztár segítségével egy helyen kezelhető az összes számítási felhővel kapcsolatban felmerülő feladat.

Az általam most bemutatott módszer csak egy a sok közül, de az általam választott módszerrel szeretném szemléltetni az OpenNebula képességeit. A cél, hogy a már meglévő webszolgáltatás a felhőben fusson. Ehhez létrehozhatnánk egy virtuális gépet, melyet feltölthetnénk a felhőbe, azonban erre nincs feltétlenül szükség. Lehetőség van teljesen az alapoktól, az operációs rendszer telepítésétől, az egyéb végső beállításokig mindent a fel-

hőben elvégezni. Ezt megtehetjük az OpenNebula API-ja segítségével, de sokkal egyszerűbb a grafikus webes felületet, a Sunstone-t használni. A továbbiakban bemutatom, hogy a Sunstone segítségével, hogyan telepítettem egy Ubuntu operációs rendszert egy virtuális gépre a felhőben, majd hoztam létre egy template-et, melyből ezek után már könnyedén tetszőleges számú virtuális gépet példányosíthattam.

Az operációs rendszer telepítőlemezének létrehozásakor nem kell a telepítőlemez sem feltölteni, megadható hálózati elérési út, http vagy ftp hely, melyet automatikus letölt a rendszer, így nem függ a folyamat gyorsasága az internetünk sávszélességétől. A név, elérési út és típus, mely jelen esetben CD-ROM, megadása után egyből elkezdődik a letöltés. A folyamat végén a képfájl „LOCKED” állapotából „READY” állapotra változik, innen tudjuk, hogy befejeződött a letöltés. Hasonló módon kell létrehozni azt az üres Datablock típusú képfájlt is, melyre az operációs rendszert települhet.

Create Image

Fields marked with are mandatory

Name:

Description:

Datastore:

Type:

Persistent: ☐

Device prefix:

Driver:

Target:

Image location: ☒ Provide a path ☐ Upload ☐ Create an empty datablock

Path:

4. ábra: Új képfájl létrehozása, hálózati elérési út is megadható

Itt érdemes az operációs rendszer követelményei és a futtatni kívánt alkalmazások igényei szerint megválasztani a méretet. Az elkészülés után a típust OS-re kell állítani, így jelezve, hogy egy bootolható operációsrendszert tartalmaz (majd) az image. Ahhoz, hogy ez így is legyen be kell pipálni, hogy a képfájl perzisztens legyen.

ID	Owner	Group	Name	Datastore	Type	Persistent	Status	#VMS
219	oneadmin	oneadmin	[APP] Condor-Debian6	iscsi-izabel	OS	☐	READY	0
230	oneadmin	oneadmin	[APP] Condor-Debian6 OK	iscsi-izabel	OS	☐	READY	0
231	oneadmin	oneadmin	[BLANK]16G_ext4	iscsi-izabel	DATABLOC	☐	USED	1
241	oneadmin	oneadmin	[APP] Boinc GBAC 3GB	iscsi-izabel	OS	☐	READY	0
259	mgergely@sztaki.hu	oneadmin	[BASE] SL 5.8	iscsi-izabel	OS	☐	USED	1
260	mgergely@sztaki.hu	oneadmin	[BASE] Debian 6.0.6	iscsi-izabel	OS	☐	READY	0
261	mgergely@sztaki.hu	oneadmin	[BASE] SL 6.3	iscsi-izabel	OS	☐	READY	0
265	acs@sztaki.hu	oneadmin	[BASE] Ubuntu 12.04 Server	iscsi-izabel	OS	☐	USED	2
281	mgergely@sztaki.hu	oneadmin	[APP] guse 3.4.7	iscsi-izabel	OS	☐	USED	1
333	tamas.pflanzner@partners.sztaki.hu	LPDS-partners	[INSTALL]Ubuntu 12.04 LTS desktop amd64	iscsi-izabel	CDROM	☐	READY	0
334	tamas.pflanzner@partners.sztaki.hu	LPDS-partners	[BASE]Ubuntu 12.04 LTS desktop amd64	iscsi-izabel	OS	☒	READY	0

5. ábra: A Sunstone felülete

Ezek után egy template-et kell létrehozni, melyhez a telepítő és az üres image-eket is hozzá kell adni. Fontos, hogy a template architektúrája egyezzen a telepíteni kívánt operációs rendszerével, valamint, hogy a VNC elérést be legyen állítva, hogy később kapcsolatba lehessen lépni a virtuális géppel vagy gépekkel. Itt adható meg a virtuális gép hardver specifikációja is. A telepítéshez a CD-ROM image-et kell a bootolás forrásának beállítani. A template elkészülte után példányosíthatóvá válik, majd ha az állapota futóra változik, VNC-n keresztül grafikus felületen telepíthető az operációs rendszer, konfigurálható és az alkalmazás környezetének beállítására is lehetőség van.

6. ábra: Template létrehozása

Ha megfelelően be lett állítva a virtuális gép, ki lehet kapcsolni a perzisztenciát, de érdemes legalább egy nem perzisztens másolatot eltárolni. Egy újabb template elkészítésére is szükség van, ami nagyjából megegyezik az előzővel, azonban a CD-ROM image-re már nincs szükség, illetve az üres datablock helyett az imént elkészített képfájlt fogja tartalmazni. Ezután tetszőleges számban példányosíthatjuk.

Példányosítás után, a futó virtuális gépekhez VNC-n keresztül kapcsolódhatunk. Ezt megtehetjük a Sunstone felületébe beépített kis ikon segítségével a böngészőn belül, de használhatunk egyéb VNC klienseket is.

3.3. HASZNÁLATI ESET ÖSSZEFOGLALÁSA

Az előző fejezetek bemutatták, hogy hogyan kezelhető egyszerűen az adatfelhő API-n keresztül, illetve a már létező mobilalkalmazásokon kívül, hogyan fejleszthető saját adatfelhő API-t használó program és, hogy hogyan építhető fel lépésről-lépésre egy olyan számítási felhőben lévő virtuális gépen futó alkalmazás, mely automatikusan érzékeli, ha az adatfelhőbe új fájl kerül. Az ezek alapján előállított prototípus alkalmazás képes fájlokat feltölteni egy felhasználó mobiljáról a Dropbox API-n keresztül egy felhasználói fiók alá, majd automatikusan észlelni és feldolgozni azt az OpenNebulában indított virtuális gépben

futó webszolgáltatás segítségével. A feldolgozás után az új fájl is bekerül a Dropboxos fiókba, amit a felhasználó megtekinthet a mobiltelefonján keresztül.

A két részfolyamatot összefogva egy olyan rendszert kapunk, mely jó kiindulási alap az első fejezetben felvetett probléma megoldására, sőt nem csak a számítási felhők együttes adatfeldolgozása oldható meg ilyen módon, de egyszerűen összekapcsolható más felhasználói eszközzel. A használati esetben bemutatott módszer a mobil eszközök funkcionalitását is bővíti, a következő példán ezt szeretném szemléltetni:

A mobileszközöket manapság nem csak telefonálásra, hanem a beépített nagyfelbontású kamera segítségével képek és videók rögzítésére is használatosak. Mivel feltehető, hogy egy ilyen mobil eszköz is rendelkezik megfelelő internetkapcsolattal, lehetőség nyílik az adatok felhőben való tárolására. Így nem csak a kis tárhely problémája oldható meg, de egyúttal a mobileszközünk képességeit is bővíthetjük, hiszen amint feltöltődtek az adatok az adatfelhőbe, a számítási felhő vagy felhők is hozzáférhetnek, így nagy mennyiségű számítási erő birtokába kerülünk. A gyorsaság és egyszerűség kedvéért előre beállított szabályok alapján, a szokásos műveleteket automatikusan végrehajtják a számítási felhők és az eredményt visszatöltve az adatfelhőbe, azonnal elérhetővé teszik akár a mobileszközünk, akár egyéb eszközök számára is. Olyan műveleteket végezhetők el a gyenge teljesítményű mobileszköz és a számítási felhők segítségével, melyekre egyébként nem lenne lehetőség. Ilyen lehet egy tájról elkészített képek alapján összeillesztett panoráma kép létrehozása, vagy egy videó konvertálása, illetve egyéb speciális művelet futtatása.

4. ÖSSZEFOGLALÁS

Az eddigi fejezetekben áttekintettük a számítási felhőkkel kapcsolatos tudnivalókat és problémákat. A különböző típusú számítási felhők közti adatmegosztás még nincs megoldva, ezért ez problémát okoz a használóknak. A probléma lényege, hogy egy virtuális gép, mely a számítási felhőben fut, bizonyos adatokat feldolgoz, de a végeredmény kinyeréséhez az egész képfájlt le kell tölteni és kinyerni belőle az adatokat. Jelen dolgozat erre a problémára ad egy lehetséges megoldási módszert.

Bemutatásra került egy akadémiai körökben elterjedt számítási felhő rendszer működését megvalósító OpenNebula rendszer, melyet a használati esetben használtam. A másik nagyobb rész az adatfelhőkkel foglalkozik, azon belül konkrétan a Dropbox szolgáltatást írja le. A rendszer az adatfelhő segítségével osztja meg a módosítani kívánt adatokat a számítási felhők között, valamint a végeredményt a felhasználókkal. Egy mobilkliens alkalmazás fejlesztését is bemutattam, mely képes a mobil eszközön keletkezett adat feltöltésére az adatfelhőbe, és a fájlok letöltésére.

4.1. TOVÁBBFEJLESZTÉSI LEHETŐSÉGEK, CÉLOK

A dolgozatban bemutatott mintaalkalmazás bár képes a számítási és adat felhők közötti együttműködés szemléltetésére, még csak prototípusnak tekintendő. Ebben a fejezetben a jövőbeli továbbfejlesztési lehetőségeket és terveimet foglalom össze.

4.1.1. A számítási felhőket érintő tervek

A számítási felhővel kapcsolatban jelenleg nagy hiányossága a programnak, hogy csak egy felhasználó adatfelhőjét képes kezelni. Persze megoldható lenne minden felhasználóhoz külön virtuális gépet példányosítani, de nagyobb méreteknél ez menedzselési problémákhoz vezetne, könnyebb és átláthatóbb lenne egy alkalmazáson belül lekezelni a többfelhasználós esetet. Természetesen fontos szempont még, hogy minél több felhőszolgáltatót támogasson a rendszer. Ez elsősorban, amíg IaaS számítási felhőkről beszélünk, nem az alkalmazás módosításával, inkább a virtuális géppel kapcsolatos módosításokat vet fel. Jelenleg fő célom a nagyobb számítási felhő szolgáltatókkal való kompatibilitás megteremtése, ilyenek az Amazon és a Google felhői. Mivel az alkalmazás erőssége, hogy több felhőn is futhat egy-egy példányban, így a feladatok elvégzését ütemezni kellene. Elképzelhető, hogy működhetne egy olyan rendszer, melyben egy kinevezett virtuális gép ütemezné a többi munkáját, de talán célszerűbb lenne egy, az adattárolóban tárolt, közösen

szerkeszthető XML fájl segítségével megoldani a problémát. A fájl tartalmazná, hogy jelenleg mely virtuális gépek figyelik az adott tárhelyet és milyen szolgáltatásokat tudnak nyújtani. Továbbá az alkalmazást bővíteni lehetne, hogy képes legyen a fájlok mellé feltöltött konfigurációs fájlban leírt egyedi szabályokat értelmezni és az utasítások szerint elvégezni a feladatot.

4.1.2. Az adatfelhőkkel kapcsolatos további fejlesztések

Az adatfelhők terén a fő cél, hogy minél több szolgáltatóval tudjon közreműködni a rendszer. Elsősorban az elterjedtebb és API-val rendelkező adatfelhők bevonása a cél. A jelenlegi rendszer mellett vagy vele együtt szükség lehet a kompatibilis adatfelhők közti adatmenedzselésre. Egy középréteg gondoskodna az adatok elosztásáról az adatfelhők között, így az esetleges hibák is könnyebben orvosolhatók lennének, lehetőség lenne biztonsági mentések létrehozására és a meglévő tárterületek hasznos kihasználására. A rendszer mellett szeretnék egy olyan általános adatfelhő nyelvet megvalósítani, melyek az alapvető műveleteket képesek lennének minél több adatfelhőn elvégezni. Ehhez kapcsolódhat az adatok automatikus elosztását lebonyolító menedzser rész is. A közös nyelv egy közös interfészt jelentene, melyet más alkalmazások is használhatnának, így egyszerűsítve a fejlesztést. Nem csak segítheti a fejlesztést, de önmagában is hasznos program lenne a felhasználók számára, természetesen megfelelő grafikus felület mellett.

4.1.3. A kliens alkalmazással kapcsolatos tervek

A kliensalkalmazás fejlesztésével kapcsolatban a fő cél, hogy minél több platformon elérhető legyen. A jelenlegi Android mellett iOS és Windows Phone alkalmazások fejlesztése is indokolt. Természetesen nem csak külön alkalmazás fejlesztése a cél, hanem hogy minél jobban beépüljön az adott operációs rendszerbe, így segítve a többi alkalmazást. Emellett a kliens oldalon is meg kell valósítani az egyéni és automatikus szabályok kezelését. Itt praktikus, hogy az adatfelhőben lévő számítási felhő konfigurációs fájlból kiolvashatók az elérhető szolgáltatások, melyek közül választhat a felhasználó.

4.1.4. Konkrét alkalmazás terve

Mivel a most leírt rendszer tekinthető egy keretrendszernek, mely segítségével könnyen megoldható a fájlok mozgatása az adatfelhőbe, majd a számítási felhő értesülése után tesztolges alkalmazás futtatható a szabályrendszer feldolgozásának segítségével. Így a fejlesztőnek nem kell csak a konkrét alkalmazással foglalkozni, az adatfeltöltés, ennek számítási felhőben való érzékelése és a végeredmény visszatöltése már meg van oldva.

Ezt a rendszert kihasználó alkalmazás az átlag felhasználók számára a mobiltelefonon készített kép vagy videó manipulálását jelenthetné. Azonban ezen túl speciálisabb, akár belső céges használhatósága is van a rendszernek. Például hasznos lenne egy olyan rendszer kiépítése, mely támogatná a mobileszközökön való nagyobb projektek fejlesztését és tesztelését. A kliens alkalmazás segítségével bővíthető vagy módosítható lenne a Dropbox tárhelyén található projekt forráskódja. A módosítások után automatikusan újrafordítaná az alkalmazást a számítási felhőben futó builder. Ezután automatikus lenne az újabb verzió tesztelése is, sőt a kliensalkalmazással új teszteseteket is be lehetne vonni.

IRODALOMJEGYZÉK

- [1] K. Jeffery, and B. Neidecker-Lutz, The Future of Cloud Computing, Opportunities for European Cloud Computing beyond 2010, Expert Group Report, January 2010. (Online: <http://cordis.europa.eu/fp7/ict/ssai/docs/cloud-report-final.pdf>)
- [2] P. Mell and T. Grance. The NIST Definition of Cloud Computing, NIST Special Publication 800-145. Online: <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>, September 2011.
- [3] Fraunhofer Institute for Secure Information Technology, On THE Security of Cloud Storage Services, SIT Technical reports, 2012 Március. Web: http://www.sit.fraunhofer.de/content/dam/sit/en/documents/Cloud-Storage-Security_a4.pdf

Internetes források

- [4] OpenNebula weboldala: <http://opennebula.org/>
- [5] Dropbox fejlesztőknek: <https://www.dropbox.com/developers>
- [6] Dropbox API weboldala: <https://www.dropbox.com/developers/reference/api>
- [7] Dropbox hack: <http://www.informationweek.com/security/application-security/dropbox-two-factor-authentication-has-ki/240006269>
- [8] Drew Houston blog: <https://blog.dropbox.com/?p=23>
- [9] OPSWAT market share report: <http://www.opswat.com/sites/default/files/OPSWAT-market-share-report-december-2011.pdf>
- [10] KVM weboldalal: http://www.linux-kvm.org/page/Main_Page
- [11] VMware weboldala: <http://www.vmware.com/>
- [12] Xen weboldala: <http://www.xen.org/>
- [13] VirtualBox weboldala: <https://www.virtualbox.org/>
- [14] MTA SZTAKI felhő weboldala: <http://cloud.sztaki.hu/>

MELLÉKLETEK

1. MELLÉKLET: NÉHÁNY NAGYOBB ADATFELHŐ SZOLGÁLTATÓ ÖSSZEHASONLÍTÁSA

Szolgáltató	Kezdeti tárhely	Bónusz ismerősönként	Maximális tárhely	Támogatott OSeK	Támogatott mobil platformok	Verziókezelés	Titkosítás	Eszközök száma	API	SDK
Google Drive	5GB +1GB Picasa	-	5GB	Win, Mac	iOS, Android	van	nincs	nincs korlát	van	Java, Python, PHP, .NET, Ruby
Amazon Cloud Drive	5GB	-	5GB*	Win, Mac	iOS, Android	nincs	nincs	8	nincs	nincs
SkyDrive	7GB	-	7GB	Win, Mac, Linux	iOS, Android	van	nincs	nincs korlát	van	Android, .NET, iOS
Spideroak	2GB	1GB	10GB	Win, Mac, Linux	iOS, Android	van	van	nincs korlát	van	nincs
Dropbox	2GB	500MB	16GB**	Win, Mac, Linux	iOS, Android	van	van****	nincs korlát	van	iOS, Android, Python, Ruby, Java, OS X
IDrive	5GB	1GB ***	50GB	Win, Mac	iOS, Android	van	van	1	van	nincs
Sugarsync	5GB	500MB	32GB	Win, Mac	iOS, Android	van	van****	1	van	nincs
Glide	30GB	-	30GB	Win, Mac, Linux	iOS, Android	van	nincs	6	nincs	nincs
CX (Cloud Experience)	10GB	300MB	16GB	Win, Mac	iOS, Android, Kindle Fire	van	van****	nincs korlát	van	nincs
Memopal	3GB	500MB	13GM	Win, Mac, Linux	iOS, Android, Blackberry	van	van****	10	van	nincs

* elérhető több is Amazonon történő zene vásárlással

** a SpaceRace keretében a diákok akár +25GBot is kaphatnak

*** 10GB minden névjegyzékbeli ismerőssel való megosztásért

**** nem egyéni titkosítás