HPE HELION OPENSTACK
LAB GUIDE

ČÁST DRUHÁ - POKROČILÉ

HPE Helion OpenStack 2.0

Prosinec 2015

Tomáš Kubica

Dokument verze 2.00

www.cloudsvet.cz

Obsah

1. Úvod

V první části labu jste si vyzkoušeli prakticky všechny části Helion OpenStack. Pracovali jsme s instancemi, storage volumy, sítěmi, firewally, load-balancery i objekty. Zaměřili jsme se i na orchestraci a šablony a další témata. Proč tedy druhá část labu?

V tomto labu opustíme svět postavený na krásném GUI a ukážeme si další metody ovládání OpenStack jako je příkazová řádka nebo API (SDK). Jejich výhodou je vyšší rychlost pro některé operace (je rychlejší to napsat, než naklikat), ale hlavně možnost začít si psát malé (i větší) skripty nebo dokonce skoro-aplikace třeba v Python, Ruby nebo některém z dalších jednoduše použitelných jazyků (samozřejmě si můžete psát třeba v C++, ale pro účely jednoduchých věcí, které jsou přehledné a každý jim porozumí, to nemusí být ideální).

Automatizaci infrastruktury je vhodné doplnit automatizací při konfiguraci operačního systému a aplikací. V labu si ukážeme nástroje jako jsou Ansible (velmi mocný nástroj a při tom velmi jednoduchý, ideální do začátku) nebo Chef (pokročilejší řešení s mnoho enterprise funkcemi). Také se budeme věnovat práci s diskovými obrazy a jak je dělat konzistentně, automatizovaně a pro různá prostředí s použitím Packer.

Objektové storage OpenStack Swift jsme se v první části věnovali jen velmi krátce, ale možnosti které nabízí, jsou podstatně širší. V tomto labu ze Swiftu dostaneme daleko víc.

V dalších verzích tohoto lab guide můžete očekávat další komplementární nástroje, technologie a postupy. Helion OpenStack je ideálním základním stavebním blokem vašeho IT. Stavte na něm a rozvíjejete se do ostatních směrů.

2. Příkazová řádka (CLI) a SDK

2.1. Proč příkazovou řádku

V předchozí části labu jsme se soustředili na ovládání skrze grafické rozhraní. Proč tedy vůbec má smysl znát příkazovou řádku?

- Příkazová řádka je open source a využívá OpenStack API, takže funguje stejně v různých modifikací OpenStack od různých výrobců – to co se naučíte se vám hodí v mnoha situacích
- V některých operacích je příkazová řádka rychlejší
- Ovládat infrastrukturu můžete i z minimálního systému bez grafického prostředí
- Příkazy můžete skriptovat, tedy vytvářet jednoduché „aplikace“, které dávkovým způsobem budou provádět nějaké operace nebo provádět pravidelné záležitosti (například sestavíte nějaké demo prostředí, které dáte k rozbití a po ukončení takové session skriptem vrátíte demo do původního stavu)
- Některé funkce jsou dostupné pouze tímto způsobem – běžný vývoj je, že nová funkce je jako první k dispozici v příkazové řádce a později v grafickém interface

Pokročilejší uživatel by tedy měl CLI znát, stojí to za to.

2.2. OpenStack CLI

Příkazová řádka OpenStack je implementovaná jako Python aplikace, kterou můžete nainstalovat prakticky kamkoli - v našem případě je pro vás připravena na labServer. Historicky měl každý projekt svou vlastní aplikaci. Pro práci se servery jse volali „nova“, pro storage „cinder“, pro image „glance“ a tak dál. V tuto chvíli HPE a komunita pracuje na novém moderním integrovaném projektu příkazové řádky, pro který nemusíte znát kódové názvy jednotlivých součástek OpenStack. Naprostou většinu operací už můžete provádět tímto novým způsobem (a tomu se budeme v labu věnovat), ale mohou být stále ještě speciality, pro které použijete původní příkazy (například práce s LBaaS v základním lab guide byla přes příkaz „neutron“). Dá se ovšem očekávat, že všechny chybějící možnosti (hlavně kolem networkingu) se v novém klientovi objeví v průběhu roku 2016.

Abychom nemuseli neustále dokola zadávat komunikační parametry jako je login do OpenStack, uložíme si je do proměnných prostředí. Ve vašem domovském adresáři na labServer je pro vás připraven soubor stack, který to udělá (klidně si ho prohlédněte). Načtěte ho do prostředí.

```
tomas@labserver:~$ source stack
```

Příkazy vždy začínají slovem openstack, pak následuje podstatné jméno (s čím chcete něco dělat) a sloveso (co s tím chcete dělat) a pak případné další detaily. Fungovat můžete interaktivně, tedy pouze spustit openstack a příkazy zadávat v tomto prostředí.

```
tomas@labserver:~$ openstack
(openstack) flavor list
+--------------------------------------+------------+-------+------+-----------+-------+-----------+
| ID                                   | Name       |   RAM | Disk | Ephemeral | VCPUs | Is Public |
+--------------------------------------+------------+-------+------+-----------+-------+-----------+
| 1                                    | m1.tiny    |   512 |    1 |         0 |     1 | True      |
| 13                                   | dev.medium |  4096 |   20 |         0 |     4 | True      |
| 2                                    | l1.small   |  2048 |   20 |         0 |     2 | True      |
| 3                                    | m1.medium  |  4096 |   40 |         0 |     2 | True      |
| 4                                    | m1.large   |  8192 |   80 |         0 |     4 | True      |
| 5                                    | m1.xlarge  | 16384 |  160 |         0 |     8 | True      |
| d9422513-43c2-4212-97c7-759ba0d66cd9 | m1.small   |  1024 |    3 |         0 |     1 | True      |
+--------------------------------------+------------+-------+------+-----------+-------+-----------+
(openstack) exit
```

Nebo (a to budeme v našem labu preferovat) zadáte příkaz rovnou celý. Takhle si tedy vypíšeme nám dostupné Flavors.

```
tomas@labserver:~$ openstack flavor list
+--------------------------------------+------------+-------+------+-----------+-------+-----------+
| ID                                   | Name       |   RAM | Disk | Ephemeral | VCPUs | Is Public |
+--------------------------------------+------------+-------+------+-----------+-------+-----------+
| 1                                    | m1.tiny    |   512 |    1 |         0 |     1 | True      |
| 13                                   | dev.medium |  4096 |   20 |         0 |     4 | True      |
| 2                                    | l1.small   |  2048 |   20 |         0 |     2 | True      |
| 3                                    | m1.medium  |  4096 |   40 |         0 |     2 | True      |
| 4                                    | m1.large   |  8192 |   80 |         0 |     4 | True      |
| 5                                    | m1.xlarge  | 16384 |  160 |         0 |     8 | True      |
| d9422513-43c2-4212-97c7-759ba0d66cd9 | m1.small   |  1024 |    3 |         0 |     1 | True      |
+--------------------------------------+------------+-------+------+-----------+-------+-----------+
```

A pokud nás zajímají detaily některého z nich, použijeme show.

```
tomas@labserver:~$ openstack flavor show m1.medium
+----------------------------+-----------+
| Field                      | Value     |
+----------------------------+-----------+
| OS-FLV-DISABLED:disabled   | False     |
| OS-FLV-EXT-DATA:ephemeral  | 0         |
| disk                       | 40        |
| id                         | 3         |
| name                       | m1.medium |
| os-flavor-access:is_public | True      |
| properties                 |           |
| ram                        | 4096      |
| rxtx_factor                | 1.0       |
| swap                       |           |
| vcpus                      | 2         |
+----------------------------+-----------+
```

Pokud si chcete nechat poradit, použijte help.

```
tomas@labserver:~$ openstack help
usage: openstack [--version] [-v] [--log-file LOG_FILE] [-q] [-h] [--debug]
                 [--os-cloud <cloud-config-name>]
...
  image add project  Associate project with image
  image create   Create/upload an image
  image delete   Delete image(s)
  image list     List available images
  image remove project  Disassociate project with image
  image save     Save an image locally
  image set      Set image properties
  image show     Display image details
...


tomas@labserver:~$ openstack help image list
```

```
usage: openstack image list [-h] [-f {csv,json,table,value,yaml}] [-c COLUMN]
                            [--max-width <integer>] [--noindent]
                            [--quote {all,minimal,none,nonnumeric}]
                            [--public | --private | --shared]
                            [--property <key=value>] [--long]
                            [--sort <key>[:<direction>]]

List available images

optional arguments:
  -h, --help            show this help message and exit
  --public              List only public images
  --private             List only private images
  --shared              List only shared images
  --property <key=value>
                        Filter output based on property
  --long                List additional fields in output
  --sort <key>[:<direction>]
                        Sort output by selected keys and directions(asc or
                        desc) (default: asc), multiple keys and directions can
                        be specified separated by comma

output formatters:
  output formatter options

  -f {csv,json,table,value,yaml}, --format {csv,json,table,value,yaml}
                        the output format, defaults to table
  -c COLUMN, --column COLUMN
                        specify the column(s) to include, can be repeated

table formatter:
  --max-width <integer>
                        Maximum display width, 0 to disable

json formatter:
  --noindent            whether to disable indenting the JSON

CSV Formatter:
  --quote {all,minimal,none,nonnumeric}
                        when to include quotes, defaults to nonnumeric
```

Můžete volit různé formátování výstupu, zobrazovat jen některé sloupce apod.

```
tomas@labserver:~$ openstack image list --max-width 40
+--------------------------------------+------------------------------------------+
| ID                                   | Name                                     |
+--------------------------------------+------------------------------------------+
| 9511e241-48f5-44b3-888f-3e1f4098036a | Gladinet                                 |
| 8701c68c-d628-4f6e-a6ef-dabec79b394f | ownCloud                                 |
| 7058da05-a43d-4c2a-a3af-0e4fc687326e | Windows Server 2012 R2 eval              |
| 6b837dc7-41b1-47f7-80de-088ad7f31d81 | webNode                                  |
| e68e44d2-eb89-45dc-92c7-a02f2bd15e9c | helionce-1.0.0_cloud-1.0.20151106T125623 |
|                                      | Z                                        |
| 9dd2e18b-e25a-4d6a-8c3e-58f471a847a8 | HP Helion Development Platform -         |
|                                      | Application Lifecycle Service Seed Node  |
|                                      | 2.0.0.548                                |
| a1813f56-17ad-4ceb-8be9-f613dae55e98 | HP Helion Development Platform -         |
|                                      | Application Lifecycle Service Installer  |
|                                      | 2.0.0.548                                |
| 8490e531-6c9d-4caa-8e15-61e5f749b0ff | Ubutnu 14.04                             |
| 48131042-8e36-4f12-952e-32d73b3600df | cirros-0.3.4-x86_64                      |
+--------------------------------------+------------------------------------------+

tomas@labserver:~$ openstack image list -f yaml
- ID: 9511e241-48f5-44b3-888f-3e1f4098036a
  Name: Gladinet
- ID: 8701c68c-d628-4f6e-a6ef-dabec79b394f
  Name: ownCloud
- ID: 7058da05-a43d-4c2a-a3af-0e4fc687326e
  Name: Windows Server 2012 R2 eval
- ID: 6b837dc7-41b1-47f7-80de-088ad7f31d81
  Name: webNode
- ID: e68e44d2-eb89-45dc-92c7-a02f2bd15e9c
  Name: helionce-1.0.0_cloud-1.0.20151106T125623Z
- ID: 9dd2e18b-e25a-4d6a-8c3e-58f471a847a8
```

```
  Name: HP Helion Development Platform - Application Lifecycle Service Seed Node 2.0.0.548
- ID: a1813f56-17ad-4ceb-8be9-f613dae55e98
  Name: HP Helion Development Platform - Application Lifecycle Service Installer 2.0.0.548
- ID: 8490e531-6c9d-4caa-8e15-61e5f749b0ff
  Name: Ubutnu 14.04
- ID: 48131042-8e36-4f12-952e-32d73b3600df
  Name: cirros-0.3.4-x86_64

tomas@labserver:~$ openstack image list -f csv
"ID","Name"
"9511e241-48f5-44b3-888f-3e1f4098036a","Gladinet"
"8701c68c-d628-4f6e-a6ef-dabec79b394f","ownCloud"
"7058da05-a43d-4c2a-a3af-0e4fc687326e","Windows Server 2012 R2 eval"
"6b837dc7-41b1-47f7-80de-088ad7f31d81","webNode"
"e68e44d2-eb89-45dc-92c7-a02f2bd15e9c","helionce-1.0.0_cloud-1.0.20151106T125623Z"
"9dd2e18b-e25a-4d6a-8c3e-58f471a847a8","HP Helion Development Platform - Application Lifecycle Service Seed
Node 2.0.0.548"
"a1813f56-17ad-4ceb-8be9-f613dae55e98","HP Helion Development Platform - Application Lifecycle Service
Installer 2.0.0.548"
"8490e531-6c9d-4caa-8e15-61e5f749b0ff","Ubutnu 14.04"
"48131042-8e36-4f12-952e-32d73b3600df","cirros-0.3.4-x86_64"

tomas@labserver:~$ openstack image list -f json
[
  {
    "ID": "9511e241-48f5-44b3-888f-3e1f4098036a",
    "Name": "Gladinet"
  },
  {
    "ID": "8701c68c-d628-4f6e-a6ef-dabec79b394f",
    "Name": "ownCloud"
  },
  {
    "ID": "7058da05-a43d-4c2a-a3af-0e4fc687326e",
    "Name": "Windows Server 2012 R2 eval"
  },
  {
    "ID": "6b837dc7-41b1-47f7-80de-088ad7f31d81",
    "Name": "webNode"
  },
  {
    "ID": "e68e44d2-eb89-45dc-92c7-a02f2bd15e9c",
    "Name": "helionce-1.0.0_cloud-1.0.20151106T125623Z"
  },
  {
    "ID": "9dd2e18b-e25a-4d6a-8c3e-58f471a847a8",
    "Name": "HP Helion Development Platform - Application Lifecycle Service Seed Node 2.0.0.548"
  },
  {
    "ID": "a1813f56-17ad-4ceb-8be9-f613dae55e98",
    "Name": "HP Helion Development Platform - Application Lifecycle Service Installer 2.0.0.548"
  },
  {
    "ID": "8490e531-6c9d-4caa-8e15-61e5f749b0ff",
    "Name": "Ubutnu 14.04"
  },
  {
    "ID": "48131042-8e36-4f12-952e-32d73b3600df",
    "Name": "cirros-0.3.4-x86_64"
  }
]

tomas@labserver:~$ openstack image list -f value -c Name
Gladinet
ownCloud
Windows Server 2012 R2 eval
webNode
helionce-1.0.0_cloud-1.0.20151106T125623Z
HP Helion Development Platform - Application Lifecycle Service Seed Node 2.0.0.548
HP Helion Development Platform - Application Lifecycle Service Installer 2.0.0.548
Ubutnu 14.04
cirros-0.3.4-x86_64

tomas@labserver:~$ openstack image list --property name=webNode
+--------------------------------------+----------+
```

```
| ID                                   | Name    |
+--------------------------------------+---------+
| 6b837dc7-41b1-47f7-80de-088ad7f31d81 | webNode |
+--------------------------------------+---------+
```

Zkusme nastartovat instanci. Nejprve si zjistěme názvy sítí, klíčů apod.

```
tomas@labserver:~$ openstack network list
+--------------------------------------+----------+--------------------------------------+
| ID                                   | Name     | Subnets                              |
+--------------------------------------+----------+--------------------------------------+
| 1712d425-ed71-443c-af0c-43d2f6e3c0aa | ext-net  | f0342aa1-7022-4728-85d8-ce0249d4f7a0 |
| 5cec492f-3150-44a9-9245-37d23a8c45b1 | mujNet   | 1b70d872-24f8-484c-9cf0-0e92a546069a |
| 9b1ed2c9-6bba-4b24-ac31-c7952d1392a4 | druhaSit | 2f5d2519-a05f-48ba-85d9-09f313c1fc8f |
| b830b33d-06fc-470a-aa12-ec4e709ef3c4 | mojeSit  | fab467ff-0767-4c8f-aa7d-a211f82360e3 |
+--------------------------------------+----------+--------------------------------------+

tomas@labserver:~$ openstack keypair list
+---------+-------------------------------------------------+
| Name    | Fingerprint                                     |
+---------+-------------------------------------------------+
| mujKlic | 2a:84:03:ad:ae:1b:ae:a9:01:2a:e2:4c:93:6c:20:ef |
+---------+-------------------------------------------------+

tomas@labserver:~$ openstack image list --max-width 50
+--------------------------------------+----------------------------------------------------+
| ID                                   | Name                                               |
+--------------------------------------+----------------------------------------------------+
| 9511e241-48f5-44b3-888f-3e1f4098036a | Gladinet                                           |
| 8701c68c-d628-4f6e-a6ef-dabec79b394f | ownCloud                                           |
| 7058da05-a43d-4c2a-a3af-0e4fc687326e | Windows Server 2012 R2 eval                        |
| 6b837dc7-41b1-47f7-80de-088ad7f31d81 | webNode                                            |
| e68e44d2-eb89-45dc-92c7-a02f2bd15e9c | helionce-1.0.0_cloud-1.0.20151106T125623Z          |
| 9dd2e18b-e25a-4d6a-8c3e-58f471a847a8 | HP Helion Development Platform - Application       |
|                                      | Lifecycle Service Seed Node 2.0.0.548              |
| a1813f56-17ad-4ceb-8be9-f613dae55e98 | HP Helion Development Platform - Application       |
|                                      | Lifecycle Service Installer 2.0.0.548              |
| 8490e531-6c9d-4caa-8e15-61e5f749b0ff | Ubutnu 14.04                                       |
| 48131042-8e36-4f12-952e-32d73b3600df | cirros-0.3.4-x86_64                                |
+--------------------------------------+----------------------------------------------------+

tomas@labserver:~$ openstack flavor list
+--------------------------------------+------------+-------+------+-----------+-------+-----------+
| ID                                   | Name       |   RAM | Disk | Ephemeral | VCPUs | Is Public |
+--------------------------------------+------------+-------+------+-----------+-------+-----------+
| 1                                    | m1.tiny    |   512 |    1 |         0 |     1 | True      |
| 13                                   | dev.medium |  4096 |   20 |         0 |     4 | True      |
| 2                                    | l1.small   |  2048 |   20 |         0 |     2 | True      |
| 3                                    | m1.medium  |  4096 |   40 |         0 |     2 | True      |
| 4                                    | m1.large   |  8192 |   80 |         0 |     4 | True      |
| 5                                    | m1.xlarge  | 16384 |  160 |         0 |     8 | True      |
| d9422513-43c2-4212-97c7-759ba0d66cd9 | m1.small   |  1024 |    3 |         0 |     1 | True      |
+--------------------------------------+------------+-------+------+-----------+-------+-----------+

tomas@labserver:~$ openstack security group list
+--------------------------------------+---------+------------------------+
| ID                                   | Name    | Description            |
+--------------------------------------+---------+------------------------+
| e3d9df32-a8da-4128-b5c6-a3049ad8d345 | default | Default security group |
+--------------------------------------+---------+------------------------+
```

Nastartujme instanci.

```
tomas@labserver:~$ openstack server create serverCLI --image cirros-0.3.4-x86_64 --flavor m1.small --
security-group default --key-name mujKlic --nic net-id=5cec492f-3150-44a9-9245-37d23a8c45b1 --min 2 --max 2 -
-max-width 40
+--------------------------------------+--------------------------------------+
| Field                                | Value                                |
+--------------------------------------+--------------------------------------+
| OS-EXT-AZ:availability_zone          | nova                                 |
| OS-EXT-STS:power_state               | 0                                    |
| OS-EXT-STS:task_state                | scheduling                           |
| OS-EXT-STS:vm_state                  | building                             |
| OS-SRV-USG:launched_at               | None                                 |
| OS-SRV-USG:terminated_at             | None                                 |
```

```
| accessIPv4                           |                                          |
| accessIPv6                           |                                          |
| addresses                            |                                          |
| adminPass                            | zED9pBW4x4Lm                             |
| config_drive                         |                                          |
| created                              | 2015-12-11T08:58:42Z                     |
| flavor                               | m1.small                                 |
|                                      | (d9422513-43c2-4212-97c7-759ba0d66cd9)   |
| hostId                               |                                          |
| id                                   | 0b965dbd-05e7-4866-85ff-35d5b7579d72     |
| image                                | cirros-0.3.4-x86_64 (48131042-8e36-4f12  |
|                                      | -952e-32d73b3600df)                      |
| key_name                             | mujKlic                                  |
| name                                 | serverCLI-1                              |
| os-extended-volumes:volumes_attached | []                                       |
| progress                             | 0                                        |
| project_id                           | d20fb4c9c0d645b4962484390a3be701         |
| properties                           |                                          |
| security_groups                      | [{u'name': u'default'}]                  |
| status                               | BUILD                                    |
| updated                              | 2015-12-11T08:58:42Z                     |
| user_id                              | b58279fd16304c99a064c76b98a7c01f         |
+--------------------------------------+------------------------------------------+
```

Vypište si servery - zatím se teprve budují.

```
tomas@labserver:~$ openstack server list
+--------------------------------------+-------------+--------+----------------------------------+
| ID                                   | Name        | Status | Networks                         |
+--------------------------------------+-------------+--------+----------------------------------+
| e6f9f85f-920f-4a83-92d8-5cbeed168755 | serverCLI-2 | BUILD  |                                  |
| 0b965dbd-05e7-4866-85ff-35d5b7579d72 | serverCLI-1 | BUILD  |                                  |
| 976ce791-f78b-4d06-a508-181172eb428b | ownCloud    | ACTIVE | mojeSit=192.168.1.43, 10.201.0.11 |
+--------------------------------------+-------------+--------+----------------------------------+
```

Po chvilce budou nahoře.

```
tomas@labserver:~$ openstack server list
+--------------------------------------+-------------+--------+----------------------------------+
| ID                                   | Name        | Status | Networks                         |
+--------------------------------------+-------------+--------+----------------------------------+
| e6f9f85f-920f-4a83-92d8-5cbeed168755 | serverCLI-2 | ACTIVE | mujNet=192.168.5.19              |
| 0b965dbd-05e7-4866-85ff-35d5b7579d72 | serverCLI-1 | ACTIVE | mujNet=192.168.5.18              |
| 976ce791-f78b-4d06-a508-181172eb428b | ownCloud    | ACTIVE | mojeSit=192.168.1.43, 10.201.0.11 |
+--------------------------------------+-------------+--------+----------------------------------+
```

Vytvořme teď dva volumy ve storage.

```
tomas@labserver:~$ openstack volume create Vol_1 --size 1
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| attachments         | []                                   |
| availability_zone   | nova                                 |
| bootable            | false                                |
| created_at          | 2015-12-11T09:02:18.242738           |
| display_description | None                                 |
| display_name        | Vol_1                                |
| encrypted           | False                                |
| id                  | c68dd71d-b5d9-4d4c-8807-56d1410e102f |
| multiattach         | false                                |
| properties          |                                      |
| size                | 1                                    |
| snapshot_id         | None                                 |
| source_volid        | None                                 |
| status              | creating                             |
| type                | vsa_thin                             |
+---------------------+--------------------------------------+

tomas@labserver:~$ openstack volume create Vol_2 --size 1
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
```

```
| attachments         | []                                   |
| availability_zone   | nova                                 |
| bootable            | false                                |
| created_at          | 2015-12-11T09:02:25.881498           |
| display_description | None                                 |
| display_name        | Vol_2                                |
| encrypted           | False                                |
| id                  | 95ff6aa6-1256-4397-b548-dceb01f702c1 |
| multiattach         | false                                |
| properties          |                                      |
| size                | 1                                    |
| snapshot_id         | None                                 |
| source_volid        | None                                 |
| status              | creating                             |
| type                | vsa_thin                             |
+---------------------+--------------------------------------+

tomas@labserver:~$ openstack volume list
+--------------------------------------+--------------+-----------+------+-------------+
| ID                                   | Display Name | Status    | Size | Attached to |
+--------------------------------------+--------------+-----------+------+-------------+
| 95ff6aa6-1256-4397-b548-dceb01f702c1 | Vol_2        | available |    1 |             |
| c68dd71d-b5d9-4d4c-8807-56d1410e102f | Vol_1        | available |    1 |             |
+--------------------------------------+--------------+-----------+------+-------------+
```

A napojme je k serverům.

```
tomas@labserver:~$ openstack server add volume serverCLI-1 Vol_1
tomas@labserver:~$ openstack server add volume serverCLI-2 Vol_2

tomas@labserver:~$ openstack volume list -c 'Display Name' -c 'Attached to'
+--------------+--------------------------------------+
| Display Name | Attached to                          |
+--------------+--------------------------------------+
| Vol_2        | Attached to serverCLI-2 on /dev/vdb  |
| Vol_1        | Attached to serverCLI-1 on /dev/vdb  |
+--------------+--------------------------------------+
```

Teď můžeme třeba přidat pravidlo do Security Group

```
tomas@labserver:~$ openstack security group rule list default
+--------------------------------------+-------------+-----------+------------+
| ID                                   | IP Protocol | IP Range  | Port Range |
+--------------------------------------+-------------+-----------+------------+
| 3959bbd9-6e09-44c1-9136-6c23e03dc9bf | tcp         | 0.0.0.0/0 | 80:80      |
| 4d7bdb8c-fe66-4a51-99a0-0e2d8c8be0c6 | tcp         | 0.0.0.0/0 | 443:443    |
| 93c785f6-53bb-49c7-ae32-bbcb3e3d7a7f | tcp         | 0.0.0.0/0 | 22:22      |
| c5ecfbc6-1536-4837-bed4-86c67170f80f | icmp        | 0.0.0.0/0 |            |
+--------------------------------------+-------------+-----------+------------+

tomas@labserver:~$ openstack security group rule create default --dst-port 8080:8080
+-----------------+--------------------------------------+
| Field           | Value                                |
+-----------------+--------------------------------------+
| group           | {}                                   |
| id              | 6572b88e-c10a-4a9b-91bf-ff18af37d449 |
| ip_protocol     | tcp                                  |
| ip_range        | 0.0.0.0/0                            |
| parent_group_id | e3d9df32-a8da-4128-b5c6-a3049ad8d345 |
| port_range      | 8080:8080                            |
+-----------------+--------------------------------------+

tomas@labserver:~$ openstack security group rule list default
+--------------------------------------+-------------+-----------+------------+
| ID                                   | IP Protocol | IP Range  | Port Range |
+--------------------------------------+-------------+-----------+------------+
| 3959bbd9-6e09-44c1-9136-6c23e03dc9bf | tcp         | 0.0.0.0/0 | 80:80      |
| 4d7bdb8c-fe66-4a51-99a0-0e2d8c8be0c6 | tcp         | 0.0.0.0/0 | 443:443    |
| 6572b88e-c10a-4a9b-91bf-ff18af37d449 | tcp         | 0.0.0.0/0 | 8080:8080  |
| 93c785f6-53bb-49c7-ae32-bbcb3e3d7a7f | tcp         | 0.0.0.0/0 | 22:22      |
| c5ecfbc6-1536-4837-bed4-86c67170f80f | icmp        | 0.0.0.0/0 |            |
+--------------------------------------+-------------+-----------+------------+
```

A na závěr pojďme instance i volume smazat

```
tomas@labserver:~$ openstack server delete serverCLI-1 serverCLI-2
tomas@labserver:~$ openstack server list
+--------------------------------------+----------+--------+----------------------------------+
| ID                                   | Name     | Status | Networks                         |
+--------------------------------------+----------+--------+----------------------------------+
| 976ce791-f78b-4d06-a508-181172eb428b | ownCloud | ACTIVE | mojeSit=192.168.1.43, 10.201.0.11 |
+--------------------------------------+----------+--------+----------------------------------+
tomas@labserver:~$ openstack volume delete Vol_1 Vol_2
tomas@labserver:~$ openstack volume list
```

2.3. Shell skriptování

Pokud pracujete s klientem v Linux prostředí můžete vytvářet jednoduché skripty pro bash shell. Na labServeru najdete dva triviální příklady.

První skript vezme slova, která mu předáte a podle toho založí příslušná počet volumů tohoto názvu.

```
tomas@labserver:~/bash-cli$ cat vytvorVolume.sh
for vol in "$@"
do
    openstack volume create $vol --size 1
done
```

Druhý skript si vypíše všechny (! opatrně, ať tam nemáte něco, co chcete zachovat) volumy vašeho tenantu a pak je promaže.

```
tomas@labserver:~/bash-cli$ cat smazVsechnyVolume.sh
for vol in $(openstack volume list -f value -c ID)
do
   openstack volume delete $vol
done

echo Smazano!
```

Jednoduché ... takhle to vypadá.

```
tomas@labserver:~$ cd bash-cli/
tomas@labserver:~/bash-cli$ chmod +x *
tomas@labserver:~/bash-cli$ ./vytvorVolume.sh volA volB volC
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| attachments         | []                                   |
| availability_zone   | nova                                 |
| bootable            | false                                |
| created_at          | 2015-12-11T11:14:14.281163           |
| display_description | None                                 |
| display_name        | volA                                 |
| encrypted           | False                                |
| id                  | 2288b737-f03b-4671-981a-0b5773caa857 |
| multiattach         | false                                |
| properties          |                                      |
| size                | 1                                    |
| snapshot_id         | None                                 |
| source_volid        | None                                 |
| status              | creating                             |
| type                | vsa_thin                             |
+---------------------+--------------------------------------+
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| attachments         | []                                   |
| availability_zone   | nova                                 |
| bootable            | false                                |
| created_at          | 2015-12-11T11:14:15.896893           |
| display_description | None                                 |
| display_name        | volB                                 |
| encrypted           | False                                |
| id                  | c62c8f11-d475-4ee7-a349-343cff111f84 |
| multiattach         | false                                |
| properties          |                                      |
```

```
| size                | 1                                    |
| snapshot_id         | None                                 |
| source_volid        | None                                 |
| status              | creating                             |
| type                | vsa_thin                             |
+---------------------+--------------------------------------+
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| attachments         | []                                   |
| availability_zone   | nova                                 |
| bootable            | false                                |
| created_at          | 2015-12-11T11:14:17.541467           |
| display_description | None                                 |
| display_name        | volC                                 |
| encrypted           | False                                |
| id                  | 3baa5b53-54de-46c9-b94c-dbb00930292c |
| multiattach         | false                                |
| properties          |                                      |
| size                | 1                                    |
| snapshot_id         | None                                 |
| source_volid        | None                                 |
| status              | creating                             |
| type                | vsa_thin                             |
+---------------------+--------------------------------------+
+--------------------------------------+--------------+-----------+------+-------------+
| ID                                   | Display Name | Status    | Size | Attached to |
+--------------------------------------+--------------+-----------+------+-------------+
| 3baa5b53-54de-46c9-b94c-dbb00930292c | volC         | available |    1 |             |
| c62c8f11-d475-4ee7-a349-343cff111f84 | volB         | available |    1 |             |
| 2288b737-f03b-4671-981a-0b5773caa857 | volA         | available |    1 |             |
+--------------------------------------+--------------+-----------+------+-------------+

tomas@labserver:~/bash-cli$ ./smazVsechnyVolume.sh
Smazano!

tomas@labserver:~/bash-cli$ openstack volume list

tomas@labserver:~/bash-cli$
```

2.4. SDK a Python

OpenStack přichází se sadou RESTful API, které můžete využít v jakékoli aplikaci. Toto velmi jednoduché rozhraní využívá principů známých z webové komunikace a nevyžaduje tak žádné speciální binární klienty či knihovny. Aby to bylo ještě jednodušší, existuje řada knihoven přímo pro různé programovací jazyky, která přinese vyšší abstrakce. Protože OpenStack je napsaný v Python, tak knihovna právě pro tento jazyk je velmi dobře zpracována. Pojďme si ji v labu vyzkoušet. Pokud nejste programátoři, nevadí - alespoň nahlédněte do kódu. I na pár řádkách se da udělat něco celkem užitečného a vytvoříte si představu, jak mohou vaši vývojaři jednoduše zakomponovat OpenStack do svých aplikací či webových portálů.

Načtěte si komunikační proměnné, skočte do adresáře python-stack (tam jsou hotové „aplikace“) a učiňte je spustitelnými.

```
tomas@labserver:~$ source stack
tomas@labserver:~$ cd python-stack/
tomas@labserver:~/python-stack$ chmod +x *
```

První náš příklad bude jednoduché vypsání serverů, imagů a flavorů do konzole. Prohledněte si obsah souboru vypsat.py. Nejprve si načteme přihlašovací údaje z prostředí a pak provedeme jednoduchý cyklus. Pro každý server, který získáme z kolece všech serverů, vypíšeme jeho jméno. Totéž pro image a flavory. Nic složitého.

Takhle vypadá kód.

```
#!/usr/bin/python

from openstack import connection
import os

auth_args = {
    'auth_url': os.environ['OS_AUTH_URL'],
    'project_name': os.environ['OS_PROJECT_NAME'],
```

```
    'username': os.environ['OS_USERNAME'],
    'password': os.environ['OS_PASSWORD'],
    'user_domain_id': os.environ['OS_USER_DOMAIN_ID'],
    'project_domain_id': os.environ['OS_PROJECT_DOMAIN_ID'],
}
conn = connection.Connection(**auth_args)

print "Servery:"
print "--------"
for server in conn.compute.servers():
    print server.name

print
print "Image:"
print "------"
for image in conn.compute.images():
    print image.name

print
print "Flavor:"
print "-------"
for flavor in conn.compute.flavors():
    print flavor.name
```

A tohle dělá po spuštění.

```
Servery:
--------
ownCloud

Image:
------
Gladinet
ownCloud
Windows Server 2012 R2 eval
webNode
helionce-1.0.0_cloud-1.0.20151106T125623Z
HP Helion Development Platform - Application Lifecycle Service Seed Node 2.0.0.548
HP Helion Development Platform - Application Lifecycle Service Installer 2.0.0.548
Ubutnu 14.04
cirros-0.3.4-x86_64

Flavor:
-------
m1.tiny
dev.medium
l1.small
m1.medium
m1.large
m1.xlarge
m1.small
```

V druhém příkladě uděláme něco podobného, ale tentokrát chceme, aby to byla webová aplikace (volíme primitivní, jen na ukázku). Chceme aplikaci se zabudovaným webovým serverem, na který se připojíme a dostaneme seznam flavorů. V rámci labu je pro něj vyhrazen port 7123 - v jeden okamžik ho může spustit jen jeden z vás (prostřídejte se).

Toto je kód v souboru web_list.py. Jediný rozdíl oproti předchozímu je, že vypisujee údaje v HTML tabulce zabudovaného Python Bottle web serveru.

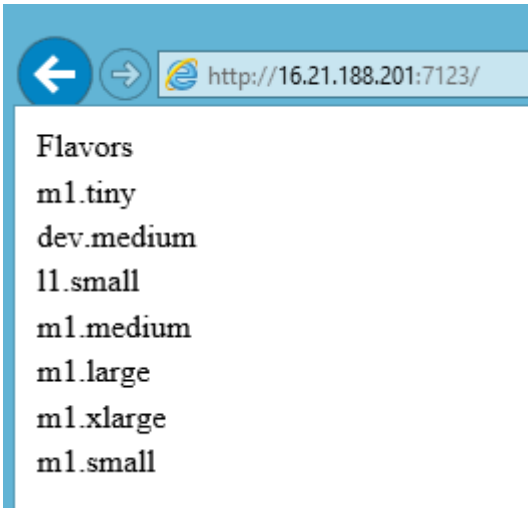
```
#!/usr/bin/python

from openstack import connection
import os
from bottle import route, run

auth_args = {
    'auth_url': os.environ['OS_AUTH_URL'],
    'project_name': os.environ['OS_PROJECT_NAME'],
    'username': os.environ['OS_USERNAME'],
    'password': os.environ['OS_PASSWORD'],
    'user_domain_id': os.environ['OS_USER_DOMAIN_ID'],
```

```
    'project_domain_id': os.environ['OS_PROJECT_DOMAIN_ID'],
}
conn = connection.Connection(**auth_args)

@route('/')
def app():
    vystup = "<table><TR><TD>Flavors</TD></TR>"
    for flavor in conn.compute.flavors():
        vystup = vystup + "<TR><TD>" + flavor.name + "</TD></TR>"
    vystup = vystup + "</table>"
    return vystup

run(host='0.0.0.0', port=7123)
```

Spusťte aplikaci.

```
tomas@labserver:~/python-stack$ ./web_vm.py
Bottle v0.12.9 server starting up (using WSGIRefServer())...
Listening on http://0.0.0.0:7123/
Hit Ctrl-C to quit.
```

Ze svého notebooku se připojte na 16.21.188.201:7123



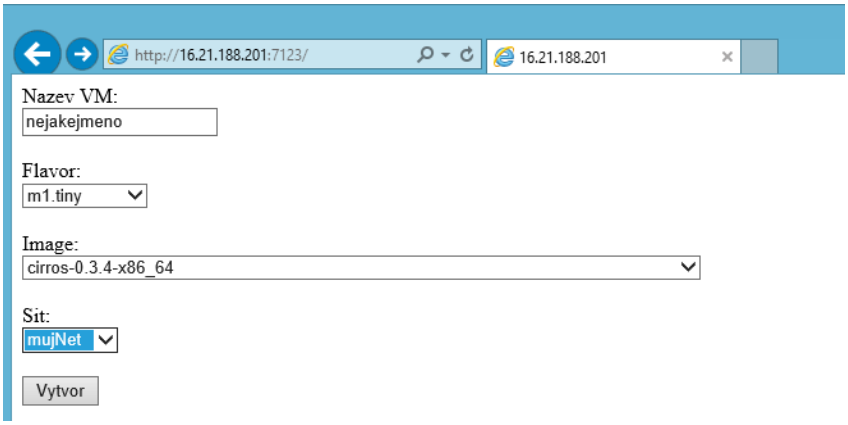
Výborně, webová aplikace nám funguje. Vraťte se do konzole labServer a aplikaci ukončete (CTRL+C).

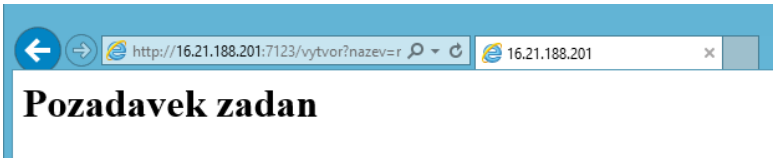
Na závěr naznačíme už něco, co by mohlo být pro někoho užitečné. Nabídneme spuštění instance z našeho vlastního webového formuláře. Prohlédněme si obsah soubor web_vm.py. V hlavní části webu neděláme nic moc nového. Načítáme si flavory, image a sítě, ale všechno balíme do HTML formuláře. Uživatel si tak vybere co by chtěl a také svůj server nějak pojmenuje. Jakmile klikne na tlačítko Vytvor zavolá se funkce vytvor(). Ta převezme vyplněné údaje a na základě nich zavolá OpenStack pro vytvoření příslušného serveru.

```
#!/usr/bin/python

from openstack import connection
import os
from bottle import route, run, request

auth_args = {
    'auth_url': os.environ['OS_AUTH_URL'],
    'project_name': os.environ['OS_PROJECT_NAME'],
    'username': os.environ['OS_USERNAME'],
    'password': os.environ['OS_PASSWORD'],
    'user_domain_id': os.environ['OS_USER_DOMAIN_ID'],
    'project_domain_id': os.environ['OS_PROJECT_DOMAIN_ID'],
}
conn = connection.Connection(**auth_args)

@route('/')
def app():
    vystup = """<!DOCTYPE html>
            <html>
```

```
            <body>
            <form action="vytvor">
            Nazev VM:<br>
            <input type="text" name="nazev" value="nejakejmeno">
            <br><br>
            Flavor:<br>
            <select name="flavor">"""

    for flavor in conn.compute.flavors():
        vystup = vystup + '<option value="{}">{}</option>'.format(flavor.name, flavor.name)
    vystup = vystup + """
            </select>
            <br><br>
            Image:<br>
            <select name="image">"""

    for image in conn.compute.images():
        vystup = vystup + '<option value="{}">{}</option>'.format(image.name, image.name)
    vystup = vystup + """
            </select>
            <br><br>

    Sit:<br>
    <select name="sit">"""

    for sit in conn.network.networks():
        vystup = vystup + '<option value="{}">{}</option>'.format(sit.name, sit.name)
    vystup = vystup + """
            </select>
            <br><br>

            <input type="submit" value="Vytvor">
            </form>

            </body>
            </html>"""
    return vystup

@route('/vytvor')
def vytvor():
    nazev = request.query.nazev
    image = conn.compute.find_image(request.query.image)
    flavor = conn.compute.find_flavor(request.query.flavor)
    network = conn.network.find_network(request.query.sit)
    server = conn.compute.create_server(
        name=nazev, image=image, flavor=flavor,
        networks=[{"uuid": network.id}])

    return "<h1>Pozadavek zadan</h1>"

run(host='0.0.0.0', port=7123)
```

Spusťte aplikaci a připojte se na web. Vyberte si a klikněte na vytvoř.





Přesvěčte se, že všechno dopadlo dle očekávání.

Instance Name	Image Name	IP Address	Size	Key Pair	Status	Availability Zone	Task	Power State	Time since created	Actions
nejakejmeno	cirros-0.3.4-x86_64	192.168.5.21	m1.tiny	-	Active	nova	None	Running	0 minutes	Create Snapshot

V této části bylo cílem představit možnosti API a SDK, nikoli ukázat jak kvalitně programovat. Příklady berte jako demonstraci možností a ne něco, co v této podobě chcete provozovat. Pokud je vaším cílem jednoduchý česky mluvící portál pro koncové uživatele, podívejte se také na hotové a velmi mocné komerční řešení HPE Cloud Service Automation.

3. Image a automatizace konfigurací OS

V prvním díle labu pro začátečníky jsme mluvili o strategiích pro práci s Image našich instancí. Připomeňme jaké to jsou:

- Image vznikají tak, že něco ve VM ručně kutáme a výsledek uložíme jako nový obraz
  - Výhoda: Nemusíme se nic učit, takhle se to dělalo i v prehistorii
  - Nevýhoda: Každý image je unikát, vše trvá dlouho, vzniká velké množství chyb, opakovatelnost je velmi problematická, dokumentace prakticky neexistuje nebo je špatně, závislost (více či méně) na hypervisor platformě (a s tím spojená špatná přenositelnost a problém nekonzistencí mezi vývojem, QA a produkcí)
- Máme základní diskový obraz a konfigurační nástroje, které ho dostanou do stavu požadovaného aplikací
  - Výhoda: Vše je automatizované a dopadne vždy stejně, výborná a živá dokumentace, nezávislost na hypervisor platformě a přenositelnost
  - Nevýhoda: Příprava služby (aplikace) trvá dlouho, troubleshooting vyžaduje znalosti
- Používáme immutable servery, tedy image je read only a pro každou verzi aplikace vždy vzniká nový fixní obraz
  - Výhoda: Rychlý start, vysoká bezpečnost (lze třeba i vypnout SSH přístup apod.), pro provoz jednoduché nasazení, při použití vhodných nástrojů lze jedním postupem vygenerovat funkčně identické obrazy pro různé hypervisory a platformy (přenositelnost)
  - Nevýhoda: Nutnost změnit development strategii a naučit se používat nové nástroje předávání údajů aplikacím místo tradičních konfiguračních souborů

Jaké konfigurační nástroje můžeme použít? Může jít o jednoduchý bash skript tak, jak jsme to udělali v první části labu, nebo něco sofistikovanější co je lépe předvídatelné a dobře udržitelné – Puppet, Chef, Ansible nebo SaltStack, HPE Server Automation. Puppet používá agenta a stojí hodně na deklarativním modelu a teorii slibů. Chef používá popis vycházející z Ruby syntaxe a způsobu přemýšlení a má rovněž propracovanou hierarchii typu agent/server/workstation. Oba zmínění mají silnou komunitu, jsou často používaní a díky tomu existujeme poměrně dost hotového obsahu. Ansible sice nemá některé pokročilé vlastnosti předchůdců, ale vyniká fantasticko jednoduchostí a je ideální pro začátečníky – nepotřebuje ani žádného agenta, stačí mu SSH. SaltStack je typicky nasazován v režimu master/minions a je proti ostatním docela mladý, ale velmi zajímavý. HPE Server Automation je rize komerční řešení (nemá open source variantu) a přichází nejen s intuitivním GUI, ale především obrovskou bází obsahu garantovanou přímo výrobcem zahrnující nejen současné Linux systémy, ale i Windows nebo UNIX a mnohé další OS a aplikace.

Chcete něco jednoduššího na proniknutí do světa konfiguračních nástrojů? Doporučuji Ansible.

Chcete komerční hotové řešení s kvalitním obsahem a certifikacemi? Doporučuji HPE Server Automation.

Preferujete programátorsky laděný přístup a máte rádi Ruby? Zkuste Chef.

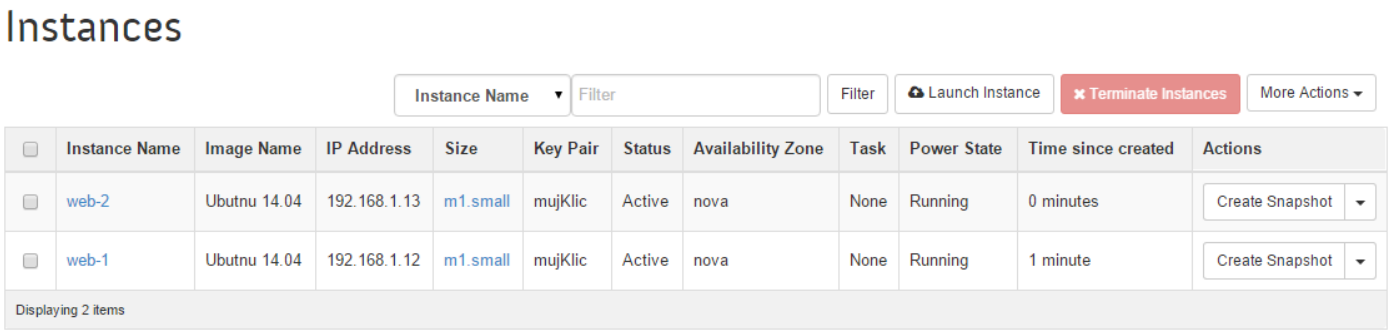
Hledáte dospělý desired-state open source přístup? Zvolte Puppet.

Chcete jet na vlně mladého systému nezatíženého dlouhou historií? To by mohl být SaltStack.

V další části této kapitole budeme kombinovat některý z těchto nástrojů s automatizací vytváření zlatých obrazů v konceptu immutable servers. Použijeme nástroj Packer. Půjde o jakousi myšlenkovou průpravu pro vaše případné pozdější zkoumání kontejnerů nebo Platform-as-a-Service, kde jsou tyto principy často používány.

3.1. Ansible

Připravte si dvě VM velikosti m1.small s image Ubuntu 14.04, dejte jim Floating IP, povolte Security Group s přístupem na SSH a web (80) a nezapomeňte přiřadit váš SSH klíč.



Připojte se na labServer, skočte do adresáře ansible

```
cd ansible
```

Použijte nějaký textový editor (nano, vi, ...) a připravte si soubor „hosts“. Jde v zásadě o seznam operačních systémů (IP adres nebo doménových jmen), které budeme chtít ovládat. V hranatých závorkách můžeme dělat skupiny – nám zatím bude stačit jen jedna.

```
[mujlab]
10.201.0.10
10.201.0.11
```

Ansible komunikuje čistě přes protocol SSH – otestujme spojení. Potřebujete mít svůj SSH klíč (viz předchozí část labu – pokud budete mít potíže, zkuste nejprve ssh ubuntu@FloatingIP, musí procházet).

```
tomas@labserver:~/ansible$ ansible mujlab -m ping -i hosts -u ubuntu --private-key=mujklic.pem
10.201.0.10 | success >> {
    "changed": false,
    "ping": "pong"
}

10.201.0.11 | success >> {
    "changed": false,
    "ping": "pong"
}
```

3.1.1. Instalace statického webového serveru

Nejprve si vyzkoušíme instalaci statického webového serveru – playbook (tedy předpis toho co se má dít v Ansible terminologii) najdete už hotový na labSeveru, ale projděte si ho.

```
---
- hosts: mujlab
  become: yes
  remote_user: ubuntu
  environment:
    http_proxy: http://web-proxy.corp.hp.com:8088
```

```
  tasks:
    - name: Nainstaluj NGINX webovy server
      apt: pkg=nginx state=installed update_cache=true
      notify:
        - start nginx

    - name: Nakopiruj index.html na server
      copy: src=index.html dest=/usr/share/nginx/html/ mode=0644

  handlers:
    - name: start nginx
      service: name=nginx state=started
```

Co tedy říkáme? Specifikujeme skupinu serveru, pro které chceme nastavení provést (mujlab ve vašem hosts souboru). Dále říkáme, že chceme operaci provadět se zvýšenými právy (become). Na začátku playbooku také definujeme environmentální proměnné, v našem případě proxy server. Pak už jedeme jednotlivé úkoly. Tím prvním je nainstalovat balíček nginx a vzít index.html s labServer a nakopírovat do cílových VM skupiny mujlab. Po instalaci serveru navíc chceme zajistit, že jeho služba je nahoře.

Spusťte tento playbook a sledujte, jak Ansible postupuje.

```
tomas@labserver:~/ansible$ ansible-playbook web1.yml -i hosts --private-key mujklic.pem

PLAY [mujlab] *****************************************************************

GATHERING FACTS ***************************************************************
ok: [10.201.0.10]
ok: [10.201.0.11]

TASK: [Nainstaluj NGINX webový server] ****************************************
changed: [10.201.0.10]
changed: [10.201.0.11]

TASK: [Nakopiruj index.html na server] ****************************************
changed: [10.201.0.11]
changed: [10.201.0.10]

NOTIFIED: [start nginx] *******************************************************
ok: [10.201.0.10]
ok: [10.201.0.11]

PLAY RECAP ********************************************************************
10.201.0.10                : ok=4    changed=2    unreachable=0    failed=0
10.201.0.11                : ok=4    changed=2    unreachable=0    failed=0
```

U tasků máte napsáno changed nebo ok, k tomu se za chvilku vrátíme.

Vyzkoušejte funkčnost webových serverů s využitím „textového prohlížeče“ curl.

```
tomas@labserver:~/ansible$ curl 10.201.0.10
Tohle je muj web

tomas@labserver:~/ansible$ curl 10.201.0.11
Tohle je muj web
```

Spusťme teď playbook ještě jednou a sledujme, co se bude dít.

```
tomas@labserver:~/ansible$ ansible-playbook web1.yml -i hosts --private-key mujklic.pem

PLAY [mujlab] *****************************************************************

GATHERING FACTS ***************************************************************
ok: [10.201.0.11]
ok: [10.201.0.10]

TASK: [Nainstaluj NGINX webovy server] ****************************************
ok: [10.201.0.11]
ok: [10.201.0.10]

TASK: [Nakopiruj index.html na server] ****************************************
```

```
ok: [10.201.0.11]
ok: [10.201.0.10]

PLAY RECAP *********************************************************************
10.201.0.10                : ok=3    changed=0    unreachable=0    failed=0
10.201.0.11                : ok=3    changed=0    unreachable=0    failed=0
```

Žádná havárie, vše funguje i nadále. Právě jste byli svědky vlastnosti, které se obvykle říká idempotence, tedy to, že můžete vše opakovat kolikrát chcete a vždy to dopadne stejně. V tomto běhu máte u všech tasků napsáno ok.
V popisu jsem použil výraz „nainstaluj“, ale to je zavádějící – správně bych měl mít poznámku ve znění „Ujisti se, že NSGIX web server je nainstalován“. To je totiž přesně to, co Ansible udělá – uvede systém do požadovaného stavu. Je krok už splněn? Fajn, odškrtnuto (ok). Není? Udělejme to (changed).

3.1.2. Přidání uživatelů do systému

Zkusme si teď něco jiného. Naše image má login ubuntu a přihlášení je povolenou pouze SSH klíčem. V následujícím playbooku vytvoříme nové uživatelské účty a povolíme login heslem. Soubor user1.yml najdete na labServer, ale pojďme okomentovat, co se v něm děje.

```
---
- hosts: mujlab
  become: yes
  remote_user: ubuntu
  environment:
    http_proxy: http://web-proxy.corp.hp.com:8088
  vars:
    users:
      - pavel
      - marek
      - tonda
      - lojza
  tasks:
    - name: Pridej uzivatele
      user:
        name: "{{ item }}"
        shell: /bin/bash
        createhome: yes
        password:
$6$rounds=656000$BnsdFwwr8iTaJIIP$UfWt9SFUU5nxZP2xkF3cmGsYD92nnhfJ86x3uAT5FQxyGgkWRxv81wz4zubkhygT/Sdk2ltMxeKZ03h5K067G.
      with_items:
        "{{ users }}"

    - name: Povol prihlaseni heslem
      replace: dest=/etc/ssh/sshd_config regexp='PasswordAuthentication no' replace='PasswordAuthentication yes'
      notify:
        - restart_ssh

  handlers:
    - name: restart_ssh
      service: name=ssh state=restarted
```

Popišme jen to, co je jiné oproti předchozí situaci. Je tady nový objekt vars, tedy proměnné a v něm users, které obsahuje výčet uživatelů. Task Pridej uzivatele zakládá účet s jménem, které je obsaženo v objektu users (tedy operace se provádí pro každého uživatele). Specifikujeme jaký základní shell má uživatel mít, že chceme vytvořit i domovský adresář a také uvádíme heslo „helion“ (to musí být v hash formátu, existují online generátory na Internetu). V druhém kroku povolíme přihlášení heslem – na to je potřeba modifikovat soubor sshd_config na cílovém serveru. Provedeme operaci replace, kde najdeme jeden řetězec a nahradíme ho jiným. Na závěr SSH restartujeme, ale jen, pokud bylo nutné soubor měnit (tedy pokud už soubor nebyl modifikovaný z předchozího běhu).

```
tomas@labserver:~/ansible$ ansible-playbook users1.yml -i hosts --private-key mujklic.pem

PLAY [mujlab] ******************************************************************

GATHERING FACTS ****************************************************************
```

```
ok: [10.201.0.11]
ok: [10.201.0.10]

TASK: [Pridej uzivatele] ******************************************************
changed: [10.201.0.11] => (item=pavel)
changed: [10.201.0.11] => (item=marek)
changed: [10.201.0.10] => (item=pavel)
changed: [10.201.0.11] => (item=tonda)
changed: [10.201.0.11] => (item=lojza)
changed: [10.201.0.10] => (item=marek)
changed: [10.201.0.10] => (item=tonda)
changed: [10.201.0.10] => (item=lojza)

TASK: [Povol prihlaseni heslem] ***********************************************
changed: [10.201.0.11]
changed: [10.201.0.10]

NOTIFIED: [restart_ssh] *******************************************************
changed: [10.201.0.10]
changed: [10.201.0.11]

PLAY RECAP ********************************************************************
10.201.0.10                : ok=4    changed=3    unreachable=0    failed=0
10.201.0.11                : ok=4    changed=3    unreachable=0    failed=0
```

Zkuste se přihlásit jako jeden z uživatelů na jeden z vašich serverů.

```
tomas@labserver:~/ansible$ ssh tonda@10.201.0.11
tonda@10.201.0.11's password:
Welcome to Ubuntu 14.04.3 LTS (GNU/Linux 3.13.0-66-generic x86_64)

...

tonda@web-1:~$
```

3.1.3. Práce se šablonami souborů

Jednou z typických operací v rámci instalace služeb do VM je modifikace konfiguračního souboru či jiná úprava nějaké výchozí šablony. Vyzkoušejme si teď obohatit náš playbook pro instalaci a konfiguraci web serveru o šablonu pro index.html. Jak samotný playbook (web2.yml) tak šablonu (index.html.j2) najdete na labServer.

Tato vypadá naše šablona:

```
{{ item.osloveni }} ctenari

Dnesnim vyhercem je {{ item.vyherce }}

Gratulujeme!
```

Políčka osloveni a vyherce budeme dosazovat až v playbooku:

```
---
- hosts: mujlab
  become: yes
  remote_user: ubuntu
  environment:
    http_proxy: http://web-proxy.corp.hp.com:8088
  vars:
    webovky:
      - osloveni: Nazdarek
        vyherce: Standa Hlozek
  tasks:
    - name: Nainstaluj NGINX webovy server
      apt: pkg=nginx state=installed update_cache=true
      notify:
        - start nginx

    - name: Nakopiruj ze sablony a modifikuj index.html
      template: src=index.html.j2 dest=/usr/share/nginx/html/index.html mode=0644
      with_items:
        "{{ webovky }}"

  handlers:
```

```
    - name: start nginx
      service: name=nginx state=started
```

V části vars jsme si definovali objekt webovky a v nem proměnné osloveni a vyherce. Tyto dosadíme na příslušná místa šablony a umístíme ji do cílové VM. Spusťte playbook.

```
tomas@labserver:~/ansible$ ansible-playbook web2.yml -i hosts --private-key mujklic.pem

PLAY [mujlab] ******************************************************************

GATHERING FACTS ****************************************************************
ok: [10.201.0.10]
ok: [10.201.0.11]

TASK: [Nainstaluj NGINX webovy server] *****************************************
ok: [10.201.0.10]
ok: [10.201.0.11]

TASK: [Nakopiruj ze sablony a modifikuj index.html] ***************************
changed: [10.201.0.11] => (item={'vyherce': 'Standa Hlozek', 'osloveni': 'Nazdarek'})
changed: [10.201.0.10] => (item={'vyherce': 'Standa Hlozek', 'osloveni': 'Nazdarek'})

PLAY RECAP *********************************************************************
10.201.0.10                : ok=3    changed=1    unreachable=0    failed=0
10.201.0.11                : ok=3    changed=1    unreachable=0    failed=0
```

Vyzkoušejte výsledek.

```
tomas@labserver:~/ansible$ curl 10.201.0.10
Nazdarek ctenari

Dnesnim vyhercem je Standa Hlozek

Gratulujeme!
```

3.1.4. Hotové role – MySQL

Některé konfigurační úkony se mohou často opakovat a Ansible nabízí institut role. Jde o kompletní seznam úloh řešící nějakou konkrétní oblast. Takovou rolí třeba bude SQL server, webový server, aplikační server apod. Takové role si můžete vytvořit sami, ale také je možné si je stáhnout z veřejného repozitáře (Ansible Galaxy). Vyzkoušejme si to.

Přidejte novou VM s Ubuntu a přiřaďte ji Floating IP.

Instance Name	Image Name	IP Address	Size	Key Pair	Status	Availability Zone	Task	Power State	Time since created	Actions
mysql	Ubutnu 14.04	192.168.1.18 Floating IPs: 10.201.0.12	m1.small	mujKlic	Active	nova	None	Running	0 minutes	Create Snapshot

Displaying 1 item

Vytvořte si podadresář pro role a stáhněte si roli ANXS.mysql

```
tomas@labserver:~/ansible$ mkdir roles
tomas@labserver:~/ansible$ source ~/proxy
tomas@labserver:~/ansible$ ansible-galaxy install ANXS.mysql -p roles/
- downloading role 'mysql', owned by ANXS
- downloading role from https://github.com/ANXS/mysql/archive/v1.1.0.tar.gz
- extracting ANXS.mysql to roles/ANXS.mysql
- ANXS.mysql was installed successfully
```

Upravte vás hosts soubor a přidejte do něj novou skupinu mysql s příslušnou adresou.

```
[mujlab]
10.201.0.10
10.201.0.11

[mysql]
```

```
10.201.0.12
```

Role mysql se postará nejen o instalaci MySQL databázového serveru, ale může vám například vytvořit database či uživatele (víc najdete v dokumentaci role na Ansible Galaxy). Playbook mysql.yml najdete take na labServer a je jednoduchý:

```
- hosts: mysql
  vars:
    mysql_users:
      - name: mujuser
        pass: mojeheslo
  become: yes
  environment:
    http_proxy: http://web-proxy.corp.hp.com:8088
    https_proxy: http://web-proxy.corp.hp.com:8088
  remote_user: ubuntu
  roles:
    - { role: ANXS.mysql }
```

Spusťte playbook.

```
tomas@labserver:~/ansible$ ansible-playbook mysql.yml -i hosts --private-key mujklic.pem

PLAY [mysql] *******************************************************************

GATHERING FACTS ****************************************************************
ok: [10.201.0.12]

TASK: [ANXS.mysql | MySQL | Load the os specific variables] ********************
ok: [10.201.0.12]

TASK: [ANXS.mysql | MySQL | Make sure the MySql packages are installed] *******
changed: [10.201.0.12] => (item=mysql-server,mysql-client,python-mysqldb)

TASK: [ANXS.mysql | MySQL | Ensure MySQL is running] ***************************
ok: [10.201.0.12]

TASK: [ANXS.mysql | MySQL | Update the my.cnf] *********************************
changed: [10.201.0.12]

TASK: [ANXS.mysql | MySQL | Set the root password.] ****************************
ok: [10.201.0.12] => (item=mysql)
ok: [10.201.0.12] => (item=127.0.0.1)
ok: [10.201.0.12] => (item=::1)
ok: [10.201.0.12] => (item=localhost)

TASK: [ANXS.mysql | MySQL | Set the root password.] ****************************
skipping: [10.201.0.12] => (item=127.0.0.1)
skipping: [10.201.0.12] => (item=::1)
skipping: [10.201.0.12] => (item=localhost)

TASK: [ANXS.mysql | MySQL | Configure MySql for easy access as root user] *****
changed: [10.201.0.12]

TASK: [ANXS.mysql | MySQL | Remove anonymous MySQL server user] ***************
ok: [10.201.0.12] => (item=mysql)
ok: [10.201.0.12] => (item=localhost)

TASK: [ANXS.mysql | MySQL | Remove the MySQL test database] ********************
ok: [10.201.0.12]

TASK: [ANXS.mysql | MySQL | Make sure the MySQL databases are present] ********
skipping: [10.201.0.12]

TASK: [ANXS.mysql | MySQL | Make sure the MySQL users are present] *************
changed: [10.201.0.12] => (item={'name': 'mujuser', 'pass': 'mojeheslo'})

TASK: [ANXS.mysql | MySQL | (Monit) Copy the mysql monit service file] ********
skipping: [10.201.0.12]

NOTIFIED: [ANXS.mysql | restart mysql] *****************************************
changed: [10.201.0.12]
```

```
PLAY RECAP *********************************************************************
10.201.0.12                : ok=12   changed=5    unreachable=0    failed=0
```

Připojte se do cílové VM a pokuste se nalogovat do MySQL vytvořeným účtem.

```
tomas@labserver:~/ansible$ ssh ubuntu@10.201.0.12 -i mujklic.pem
Welcome to Ubuntu 14.04.3 LTS (GNU/Linux 3.13.0-66-generic x86_64)

…

Last login: Mon Nov 30 10:34:18 2015 from 10.201.0.3

ubuntu@mysql:~$ mysql -umujuser -pmojeheslo
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 37
Server version: 5.5.46-0ubuntu0.14.04.2 (Ubuntu)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> exit
Bye
ubuntu@mysql:~$
```

Tímto úvod do Ansible ukončíme. Jde o nejjednodušší vstup do světa automatizace konfigurací vašich VM a ze systému rolí se dá poskládat i velmi komplexní system. Například Helion OpenStack Lifecycle Manager (instalátor a udržovač Helion OpenStack) je postaven právě na Ansible a zahruje stovky/tisíce kroků a desítky rolí.

3.2. Chef

Chef je velmi mocný nástroj pro konfiguraci a automatizace OS a aplikací. Ve srovnání s Ansible je o poznání komplikovanější na používání, ale za to velmi robustní, bezpečný a výkonný. V labu si vyzkoušímenasazení v režimu Chef-server (zdroj pravdy, místo kde jsou uloženy předpisy pro cílové stanice, je tam orgaizační členění i účty uživatelů, GUI i API), Chef-workstation (pracovní stanice administrátora) a Chef-client (agent uvnitř cílové VM, který si stahuje z Chef-serveru předpisy, které vynucuje). Na hraní můžete použít také Chef-solo (ten, stejně jako Ansible s volbou local, dokáže nastavit přímo VM, ve které běží bez potřeby dalších komponent).

Primární model fungování Chefu je pull, tedy Chef-client si vyžádá informace z Chef-serveru a lokálně je vynutí (tento proces je spuštěn se startem chef-client, což lze řešit i jako démona a nastavit pravidelné spuštění třeba každých 10 minut). To má pozitivní vliv na výkon i přehlednost. Přesto, podobně jako to řeší Ansible, podporuje i push metodu (knife ssh) a samotného chef-client můžete také instalovat push metodou (knife bootstrap).

labServer je připraven jako chef-server a současně chef-workstation.

3.2.1. Instalace webového serveru

Připojte se na labServer a vygenerujte nový repozitář.

```
tomas@labserver:~$ chef generate repo chef-repo
[2015-12-01T12:59:44+00:00] WARN: ExpandNodeObject#load_node is deprecated. Please use
Chef::PolicyBuilder::Dynamic instead of using ExpandNodeObject directly at
/opt/chefdk/embedded/lib/ruby/gems/2.1.0/gems/chef-dk-0.9.0/lib/chef-dk/chef_runner.rb:54:in `policy'
Compiling Cookbooks...
...
Recipe: code_generator::repo
  * cookbook_file[/home/tomas/chef-repo/cookbooks/README.md] action create
    - create new file /home/tomas/chef-repo/cookbooks/README.md
    - update content in file /home/tomas/chef-repo/cookbooks/README.md from none to 86e9ef
    (diff output suppressed by config)
  * execute[initialize-git] action run
    - execute git init .
```

```
  * template[/home/tomas/chef-repo/.gitignore] action create
    - create new file /home/tomas/chef-repo/.gitignore
    - update content in file /home/tomas/chef-repo/.gitignore from none to 3523c4
    (diff output suppressed by config)
```

Workstation musí být shopna bezpečným způsobem (šifrování a ověřování certifikáty) komunikovat s chef-server (i v našem případě, kdy je oboce na stejném systému).

Skočte do repozitáře, stáhněte si SSL certifikáty a ověřte jejich pravost.

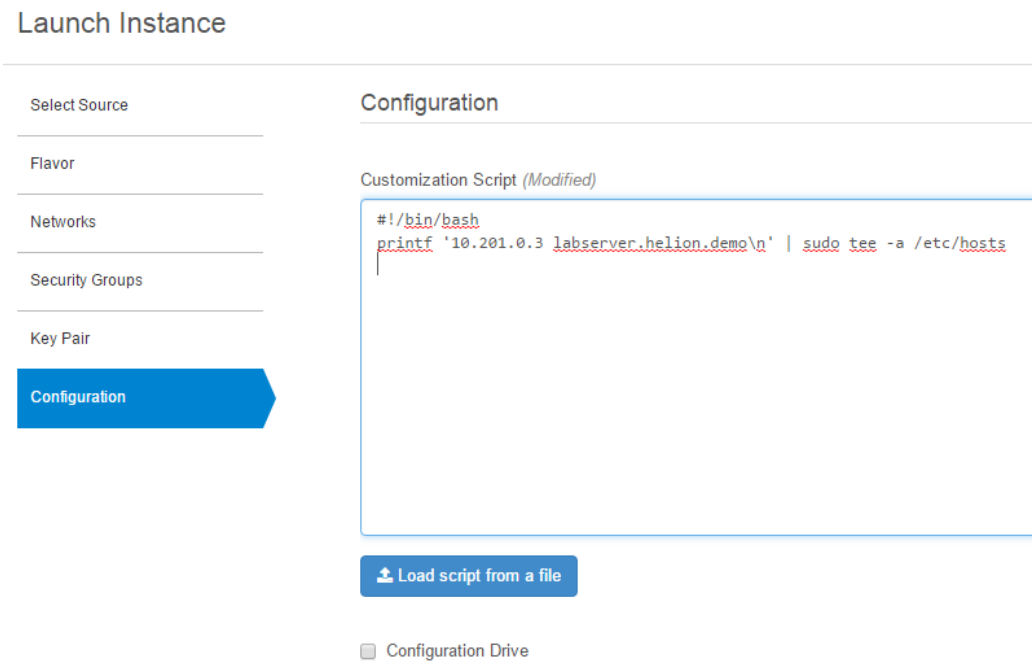
```
tomas@labserver:~/$ cd chef-repo/

tomas@labserver:~/chef-repo$ knife ssl fetch
WARNING: Certificates from labserver will be fetched and placed in your trusted_cert
directory (/home/tomas/chef-repo/.chef/trusted_certs).

Knife has no means to verify these are the correct certificates. You should
verify the authenticity of these certificates after downloading.

Adding certificate for labserver.helion.demo in /home/tomas/chef-
repo/.chef/trusted_certs/labserver_helion_demo.crt

tomas@labserver:~/chef-repo$ knife ssl check
Connecting to host labserver.helion.demo:443
Successfully verified certificates from `labserver.helion.demo'
```

Vaše workstation vám může pomoci s instalací chef klientů - vypište si aktuální seznam.

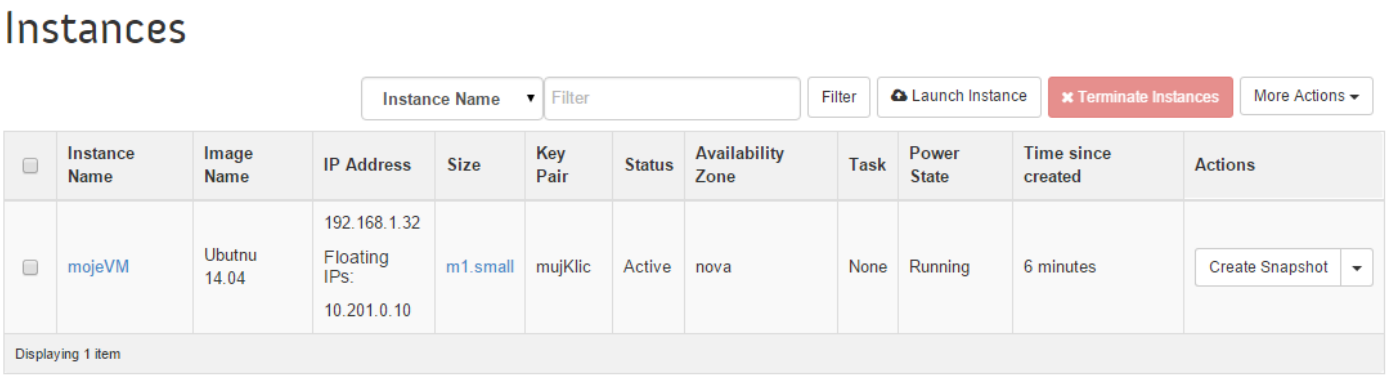
```
tomas@labserver:~/chef-repo$ knife client list
helion-validator
```

Pojďme vytvořit prázdnou Ubuntu VM, s kterou si budeme hrát. Chef vyžaduje přítomnost plných doménových jmen (FQDN), takže v praxi je žádoucí mít enterprise DNS server. Ten v labu nemáme, takže potřebujeme ručně do VM přidat záznam do hosts souboru s překladem labserver.helion.demo na jeho IP adresu. To udělejte zadáním následujícího skriptu (současně s nastavení proxy pro apt). Pokud se vám nechce opisovat, najdete ho na labServeru v adresáři chef-repo pod názvem nodnschef.sh.

```
#!/bin/bash
printf '10.201.0.3 labserver.helion.demo\n' | sudo tee -a /etc/hosts
```



Tohle je naše výsledná VM.



Na VM teď není nic. V případě Ansible by to nevadilo (tomu stačí SSH přístup), ale Chef potřebuje agenta (chef-client). Pokud jsme jeho instalaci nezařídili nějak jinak (může být už součástí image, dá se nainstalovat iniciačním skriptem nebo přes OpenStack Heat), lze použít push metodu instalace. Ta se provádí z workstation (nikoli ze chef serveru - v našem labu je to totéž, ale myslete na to v praxi z pohledu síťové konektivity). Použijte příkaz knife bootstrap, namiřte na Floating IP vaší VM, zadejte přístupové údaje (-x jako login, -i odkaz na SSH klíč), pojmenujte tento node (-N, takhle bude vidět v chef-server), povolte sudo (--sudo) a nastavte http_proxy (chef klient se bude stahovat z Internetu).

```
tomas@labserver:~/chef-repo$ knife bootstrap 10.201.0.10 -i ~/mujklic.pem -N prvniPokus -x ubuntu --sudo --
bootstrap-proxy http://web-proxy.corp.hp.com:8088 --bootstrap-no-proxy labserver.helion.demo
Doing old-style registration with the validation key at ~/chef-repo/.chef/chefhelion.pem...
Delete your validation key in order to use your user credentials instead

Connecting to 10.201.0.10
10.201.0.10 sudo: unable to resolve host mojevm
10.201.0.10 -----> Installing Chef Omnibus (-v 12)
10.201.0.10 downloading https://www.opscode.com/chef/install.sh
10.201.0.10   to file /tmp/install.sh.1493/install.sh
10.201.0.10 trying wget...
10.201.0.10 Downloading Chef 12 for ubuntu...
...
10.201.0.10 Installing Chef 12
...
10.201.0.10 Thank you for installing Chef!
10.201.0.10 Starting the first Chef Client run...
10.201.0.10 Starting Chef Client, version 12.5.1
10.201.0.10 Creating a new client identity for c using the validator key.
10.201.0.10 resolving cookbooks for run list: []
10.201.0.10 Synchronizing Cookbooks:
10.201.0.10 Compiling Cookbooks...
10.201.0.10 [2015-12-04T13:35:44+00:00] WARN: Node prvniPokus has an empty run list.
10.201.0.10 Converging 0 resources
10.201.0.10
10.201.0.10 Running handlers:
10.201.0.10 Running handlers complete
10.201.0.10 Chef Client finished, 0/0 resources updated in 04 seconds
```

Zdá se, že se chef-client stáhnul, nainstaloval i připojil. Vidíte ho v knife client?

```
tomas@labserver:~/chef-repo$ knife client list
helion-validator
prvniPokus
```

Chef přichází s řadou hotových veřejně dostupných kuchařek, stáhneme si jednu, která se nám bude hodit (provede apt-get update, což potřebujeme pro pozdější instalace balíčků, a zařídí jeho pravidelné opakování, takže se server sám patchuje - někdy to může být riskantní, ale pro některé situace se to hodí).

```
tomas@labserver:~/chef-repo$ cd cookbooks/
tomas@labserver:~/chef-repo/cookbooks$ knife cookbook site download apt
Downloading apt from Supermarket at version 2.9.2 to /home/tomas/chef-repo/cookbooks/apt-2.9.2.tar.gz
Cookbook saved: /home/tomas/chef-repo/cookbooks/apt-2.9.2.tar.gz
tomas@labserver:~/chef-repo/cookbooks$ tar zxf apt-2.9.2.tar.gz
```

Vytvoříme teď také naší novou kuchařku a necháme si vygenerovat výchozí adresářovou strukturu.

```
tomas@labserver:~/chef-repo/cookbooks$ chef generate cookbook mujWebServer
[2015-12-07T09:42:18+00:00] WARN: ExpandNodeObject#load_node is deprecated. Please use
Chef::PolicyBuilder::Dynamic instead of using ExpandNodeObject directly at
/opt/chefdk/embedded/lib/ruby/gems/2.1.0/gems/chef-dk-0.9.0/lib/chef-dk/chef_runner.rb:54:in `policy'
Compiling Cookbooks...
Recipe: code_generator::cookbook
  * directory[/home/tomas/chef-repo/cookbooks/mujWebServer] action create
...
  * cookbook_file[/home/tomas/chef-repo/cookbooks/mujWebServer/.gitignore] action create
    - create new file /home/tomas/chef-repo/cookbooks/mujWebServer/.gitignore
    - update content in file /home/tomas/chef-repo/cookbooks/mujWebServer/.gitignore from none to dd37b2
    (diff output suppressed by config)
tomas@labserver:~/chef-repo/cookbooks$
```

Skočte do adresáře kuchařky a podívejte se na její strukturu.

```
tomas@labserver:~/chef-repo/cookbooks$ cd mujWebServer/
tomas@labserver:~/chef-repo/cookbooks/mujWebServer$ tree
.
├── chefignore
├── metadata.rb
├── Policyfile.rb
├── README.md
├── recipes
│   └── default.rb
├── spec
│   ├── spec_helper.rb
│   └── unit
│       └── recipes
│           └── default_spec.rb
└── test
    └── integration
        ├── default
        │   └── serverspec
        │       └── default_spec.rb
        └── helpers
            └── serverspec
                └── spec_helper.rb

10 directories, 9 files
```

Nás teď bude recept, tedy konkrétní sada kroků. Výchozí (prázdný) byl vytvořen a ten my teď změníme.

```
tomas@labserver:~/chef-repo/cookbooks/mujWebServer$ nano recipes/default.rb
```

Takhle bude vypadat

```
#
# Cookbook Name:: mujWebServer
# Recipe:: default
#
# Copyright (c) 2015 The Authors, All Rights Reserved.

package 'apache2'

service 'apache2' do
  supports :status => true
  action [:enable, :start]
end

file '/var/www/html/index.html' do
  content 'Tohle je muj web!
'
end
```

Teď musíme uploadnout (v našem případě je to stejný server, ale i tak) obě kuchařky z chef workstation na chef server.

```
tomas@labserver:~/chef-repo/cookbooks$ knife cookbook upload mujWebServer
Uploading mujWebServer [0.1.0]
Uploaded 1 cookbook.
```

```
tomas@labserver:~/chef-repo/cookbooks$ knife cookbook upload apt
Uploading apt            [2.9.2]
Uploaded 1 cookbook.
```

Otevřte si v jiném okně další SSH sessin na labServer a z ní se zkuste napojit do cílové VM.

```
tomas@labserver:~$ ssh ubuntu@10.201.0.10 -i ~/mujklic.pem
```

Spusťte chef-client a sledujte co se děje.

```
ubuntu@mojevm:~$ sudo chef-client
sudo: unable to resolve host mojevm
Starting Chef Client, version 12.5.1
resolving cookbooks for run list: []
Synchronizing Cookbooks:
Compiling Cookbooks...
[2015-12-07T10:42:00+00:00] WARN: Node prvniPokus has an empty run list.
Converging 0 resources

Running handlers:
Running handlers complete
Chef Client finished, 0/0 resources updated in 02 seconds
```

Klient v pořádku vidí Chef server, ale nemá co dělat - nodu prvniPokus nebyla přiřazená žádná kuchařka. To můžeme udělat v jednom kroku přímo v příkazu knife bootstrap použitím volby --run-list apt,mujWebServer. My jsme to ale schválně neudělali a přidáme si kuchařku až teď.

```
tomas@labserver:~/chef-repo$ knife node run_list add prvniPokus apt
prvniPokus:
  run_list: recipe[apt]

tomas@labserver:~/chef-repo$ knife node run_list add prvniPokus mujWebServer
prvniPokus:
  run_list:
    recipe[apt]
    recipe[mujWebServer]
```

Spusťte v cílové VM chef-client znova.

```
ubuntu@mojevm:~$ sudo chef-client
sudo: unable to resolve host mojevm
Starting Chef Client, version 12.5.1
resolving cookbooks for run list: ["mujWebServer"]
Synchronizing Cookbooks:
  - mujWebServer (0.1.0)
Compiling Cookbooks...
...
...
Recipe: mujWebServer::default
  * apt_package[apache2] action install
    - install version 2.4.7-1ubuntu4.8 of package apache2
  * service[apache2] action enable (up to date)
  * service[apache2] action start (up to date)
  * file[/var/www/html/index.html] action create
    - update content in file /var/www/html/index.html from 538f31 to 25c6a1
    --- /var/www/html/index.html        2015-12-07 12:12:07.604603000 +0000
    +++ /var/www/html/.index.html20151207-4015-1wakkxb  2015-12-07 12:12:11.720603000 +0000
    @@ -1,379 +1,4 @@
    -
    -<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
...
    -  </body>
    -</html>
    -
    +Tohle je muj web!
    +
    +
  * apt_package[apache2] action install (up to date)
  * service[apache2] action enable (up to date)
  * service[apache2] action start (up to date)
  * file[/var/www/html/index.html] action create (up to date)
```

```
Running handlers:
Running handlers complete
Chef Client finished, 2/8 resources updated in 12 seconds
ubuntu@mojevm:~$
```

Chef vám říká co dělá a také co změnil a jak (u souboru index.html odstranil řádky s „-“ a přidal s „+“. Vyzkoušejme výsledek - z labServer zkuste kouknout na náš nový web.

```
tomas@labserver:~/chef-repo$ curl 10.201.0.10
Tohle je muj web!
```

3.2.2. Periodický update

Chef klient nemusí hned po akci skončit Je možné nechat jej běžet a on bude v pravidelných intervalech kontaktovat Chef server a spouštět co se mu řekne. To je velmi příjemná vlastnost, kdy jednak klient sám opraví cokoli, co se v mezičase pokazilo (ale pozor - možná včetně vašeho manuálního zásahu do VM, takže s tím procesně počítejte a ideálně nepřistupujte do VM přímo, spíše skrz modifikaci receptů), ale také stáhne nejnovější „zadání“. Má to i pozitivní vliv na výkon celého řešení (pull a lokální spuštění je rychlejší, než push krok za krokem v případě Ansible). Obvykle je vhodné interval nastavit třeba na 30 minut. V našem labu ale chceme výsledky dřív, takže použijeme (pro praxi nevhodných) 20 vteřin.

Zapněte ve VM chef-client s intervalem opakování 20 vteřin (spustíme jako normální proces, ať můžeme pohodlně sledovat výstup - dá se ale použít volba -d a spustit chef-client jako démona, typicky jako službu po startu v /init.d). Před tím můžete modifikovat kuchařku (nezapomeňte ji také uploadovat na Chef server).

```
ubuntu@mojevm:~$ sudo chef-client -i 20
```

```
Recipe: mujWebServer::default
  * apt_package[apache2] action install (up to date)
  * service[apache2] action enable (up to date)
  * service[apache2] action start (up to date)
  * file[/var/www/html/index.html] action create
    - update content in file /var/www/html/index.html from 25c6a1 to 5d96a8
    --- /var/www/html/index.html        2015-12-07 12:19:19.480603000 +0000
    +++ /var/www/html/.index.html20151207-6961-145rh8c  2015-12-07 12:21:00.964603000 +0000
    @@ -1,4 +1,4 @@
    -Tohle je muj web!
    +Tohle je muj web2!
```

Vyzkoušejte i z labServer.

```
tomas@labserver:~/chef-repo$ curl 10.201.0.10
Tohle je muj web2!
```

Upravte kuchařku znovu a sledujte co se děje v okně s cílovou VM.

```
Recipe: mujWebServer::default
  * apt_package[apache2] action install (up to date)
  * service[apache2] action enable (up to date)
  * service[apache2] action start (up to date)
  * file[/var/www/html/index.html] action create
    - update content in file /var/www/html/index.html from 5d96a8 to 057a5f
    --- /var/www/html/index.html        2015-12-07 12:21:00.964603000 +0000
    +++ /var/www/html/.index.html20151207-7951-quv8zl   2015-12-07 12:22:33.984603000 +0000
    @@ -1,4 +1,4 @@
    -Tohle je muj web2!
    +Tohle je muj web3!
```

Ověřte změnu také z labServer.

```
tomas@labserver:~/chef-repo$ curl 10.201.0.10
Tohle je muj web3!
```

3.2.3. VM za NATem aneb Chef bez použití Floating IP

Výhodou Chef řešení v porovnání s Ansible je, že v cílové VM běží chef-client, který se může pravidelně hlásit na chef-server a spouštět případné úpravy a aktualizace. Tuto komunikaci tedy iniciuje samotná VM, takže stavový firewall (mikrosegmentace, Security Group) ji může pustit ven a při tom není potřeba otevírat porty. Floating IP, kterou jsme v předhozí části labu používali pro bootstrap proces, tak můžeme z VM zase odebrat (například proto, že se nejedná o front end systém, ke kterému mají přistupovat uživatelé, ale je to třeba aplikační či databázový server, se kterým komunikuje jen aplikace). Jak už víme, instance bez Floating IP může přes virtuální router přistupovat do vnějšího světa přes source NAT podobně, jako vaše domácí počítače přistupují na Internet přes váš router. Chef-client se tedy do serveru dobouchá i když je za NAT.

Vytvoření instance s Floating IP, bootstrap Chef klienta a následné odebrání IP není zrovna praktické. Místo toho můžeme použít SSH gateway. Stačí mít ve vašem virtuálním světě (tenant/project) jednu obyčejnou VM s Floating IP, ze které je přístup na ostatní VM bez Floating IP. Chef knife dokáže bootstrap proces tunelovat skrz tuto gateway (na ní není žádný speciální software; jak uvidíte stačí třeba holý Ubuntu Server).

Vytvořte si další instanci Ubuntu, ale tentokrát bez Floating IP. Z GUI si zjistěte přiřazenou privátní adresu. Proveďme bootstrap přes gateway (poslouží nám Ubuntu VM, kterou už máme) a rovnou nainstalujeme web server s naší stránkou.

```
tomas@labserver:~/chef-repo$ knife bootstrap 192.168.5.14 -i ~/mujklic.pem -N backendvm -x ubuntu --sudo --
bootstrap-proxy http://web-proxy.corp.hp.com:8088 --bootstrap-no-proxy labserver.helion.demo -G
ubuntu@10.201.0.10 --run-list apt,mujWebServer
Doing old-style registration with the validation key at ~/chef-repo/.chef/chefhelion.pem...
Delete your validation key in order to use your user credentials instead

Connecting to 192.168.5.14
192.168.5.14 sudo: unable to resolve host backendvm
192.168.5.14 -----> Installing Chef Omnibus (-v 12)
192.168.5.14 downloading https://www.opscode.com/chef/install.sh
..
192.168.5.14 Setting up chef (12.5.1-1) ...
192.168.5.14 Thank you for installing Chef!
192.168.5.14 Starting the first Chef Client run...
192.168.5.14 Starting Chef Client, version 12.5.1
192.168.5.14 Creating a new client identity for backendvm using the validator key.
192.168.5.14 resolving cookbooks for run list: ["apt", "mujWebServer"]
192.168.5.14 Synchronizing Cookbooks:
192.168.5.14   - apt (2.9.2)
192.168.5.14   - mujWebServer (0.1.0)
192.168.5.14 Compiling Cookbooks...
192.168.5.14 Converging 14 resources
192.168.5.14 Recipe: apt::default
...
192.168.5.14   * apt_package[apt-transport-https] action install (up to date)
192.168.5.14 Recipe: mujWebServer::default
192.168.5.14   * apt_package[apache2] action install
192.168.5.14     - install version 2.4.7-1ubuntu4.8 of package apache2
192.168.5.14   * service[apache2] action enable (up to date)
192.168.5.14   * service[apache2] action start (up to date)
192.168.5.14   * file[/var/www/html/index.html] action create
...
192.168.5.14     -
192.168.5.14     +Hele, jsem zaNATovanej!
192.168.5.14     +
192.168.5.14     +
192.168.5.14
192.168.5.14 Running handlers:
192.168.5.14 Running handlers complete
192.168.5.14 Chef Client finished, 7/16 resources updated in 28 seconds
```

Ověřme, že vše dopadlo dobře. Protože cílová VM je náš backend, tedy nemá Floating IP a dostane se k ní pouze jiná VM v našem tenantu, skočíme nejprve do první VM a z té vyzkoušíme stáhnout úvodní stránku.

```
tomas@labserver:~/chef-repo$ ssh ubuntu@10.201.0.10 -i ~/mujklic.pem

ubuntu@mojevm:~$ wget -qO- 192.168.5.14
Hele, jsem zaNATovanej!
```

Právě jsme si ověřili, že je možné používat i instalovat Chef za NATem.

3.2.4. Ovládejte OpenStack z Chef Knife

Utilitka knife je, jak jsme si už mohli všimnout, docela šikovná. Dokonce do ní existuje plugin pro Openstack. Díky tomu nemusíme připravovat VM dopředu, ale přímo jedním knife příkazem se nám VM vytvoří, nainstaluje se na ni Chef klient, ten se zaregistruje a spustí naše kuchařky. Pojďme si ho stáhnout a vyzkoušet.

```
tomas@labserver:~$ source proxy

tomas@labserver:~$ chef gem install knife-openstack
Fetching: knife-cloud-1.2.0.gem (100%)
WARNING:  You don't have /home/tomas/.chefdk/gem/ruby/2.1.0/bin in your PATH,
          gem executables will not run.
Successfully installed knife-cloud-1.2.0
Fetching: knife-openstack-1.3.2.gem (100%)
Successfully installed knife-openstack-1.3.2
2 gems installed

tomas@labserver:~$ unset http_proxy
```

Nejprve si poznamenáme některé vstupy, které budeme pro volání potřebovat - například ID favor, image, sítě apod.

```
tomas@labserver:~$ source stack
tomas@labserver:~$ cd chef-repo/
tomas@labserver:~/chef-repo$ openstack flavor list
+--------------------------------------+------------+-------+------+-----------+-------+-----------+
| ID                                   | Name       |   RAM | Disk | Ephemeral | VCPUs | Is Public |
+--------------------------------------+------------+-------+------+-----------+-------+-----------+
| 1                                    | m1.tiny    |   512 |    1 |         0 |     1 | True      |
| 13                                   | dev.medium |  4096 |   20 |         0 |     4 | True      |
| 2                                    | l1.small   |  2048 |   20 |         0 |     2 | True      |
| 3                                    | m1.medium  |  4096 |   40 |         0 |     2 | True      |
| 4                                    | m1.large   |  8192 |   80 |         0 |     4 | True      |
| 5                                    | m1.xlarge  | 16384 |  160 |         0 |     8 | True      |
| d9422513-43c2-4212-97c7-759ba0d66cd9 | m1.small   |  1024 |    3 |         0 |     1 | True      |
+--------------------------------------+------------+-------+------+-----------+-------+-----------+

tomas@labserver:~/chef-repo$ openstack image list --max-width 50
+--------------------------------------+----------------------------------------------------+
| ID                                   | Name                                               |
+--------------------------------------+----------------------------------------------------+
| 9511e241-48f5-44b3-888f-3e1f4098036a | Gladinet                                           |
| 8701c68c-d628-4f6e-a6ef-dabec79b394f | ownCloud                                           |
| 7058da05-a43d-4c2a-a3af-0e4fc687326e | Windows Server 2012 R2 eval                        |
| 6b837dc7-41b1-47f7-80de-088ad7f31d81 | webNode                                            |
| e68e44d2-eb89-45dc-92c7-a02f2bd15e9c | helionce-1.0.0_cloud-1.0.20151106T125623Z          |
| 9dd2e18b-e25a-4d6a-8c3e-58f471a847a8 | HP Helion Development Platform - Application       |
|                                      | Lifecycle Service Seed Node 2.0.0.548              |
| a1813f56-17ad-4ceb-8be9-f613dae55e98 | HP Helion Development Platform - Application       |
|                                      | Lifecycle Service Installer 2.0.0.548              |
| 8490e531-6c9d-4caa-8e15-61e5f749b0ff | Ubutnu 14.04                                       |
| 48131042-8e36-4f12-952e-32d73b3600df | cirros-0.3.4-x86_64                                |
+--------------------------------------+----------------------------------------------------+

tomas@labserver:~/chef-repo$ openstack network list
+--------------------------------------+----------+--------------------------------------+
| ID                                   | Name     | Subnets                              |
+--------------------------------------+----------+--------------------------------------+
| 1712d425-ed71-443c-af0c-43d2f6e3c0aa | ext-net  | f0342aa1-7022-4728-85d8-ce0249d4f7a0 |
| 5cec492f-3150-44a9-9245-37d23a8c45b1 | mujNet   | 1b70d872-24f8-484c-9cf0-0e92a546069a |
| 9b1ed2c9-6bba-4b24-ac31-c7952d1392a4 | druhaSit | 2f5d2519-a05f-48ba-85d9-09f313c1fc8f |
| b830b33d-06fc-470a-aa12-ec4e709ef3c4 | mojeSit  | fab467ff-0767-4c8f-aa7d-a211f82360e3 |
+--------------------------------------+----------+--------------------------------------+

tomas@labserver:~/chef-repo$ openstack keypair list
+---------+-------------------------------------------------+
| Name    | Fingerprint                                     |
+---------+-------------------------------------------------+
| mujKlic | 2a:84:03:ad:ae:1b:ae:a9:01:2a:e2:4c:93:6c:20:ef |
+---------+-------------------------------------------------+
```

Teď už stačí jen vyplnit příkaz knife openstack. Je tam toho docela hodně (v praxi můžete některé z těchto údajů uložit do konfiguračního souboru pro zjednodušení - i v našem případě jsou přihlašovací údaje pro Openstack zaznamenány v chef-repo/.chef/knife.rb). Bereme složitější scénář, tedy instance bude bez Floating IP a bootstrap proběhne přes SSH gateway. Říkáme jak se bude server jmenovat, informuje jej o proxy, určujeme flavor, image a sítě, SSH gateway a příslušé klíče, run-list pro spuštění kuchařek a také předáváme konfigurační skript v části user-data (jak si jistě vzpomínáte potřebujeme udělat statický záznam na labserver.helion.demo, protože v labu nemáme DNS službu). Knife a Chef automaticky provedou vše potřebné - nastartují server, nainstalují Chef klienta, spustí kuchařku, kterou tam natáhnou web server a naší stránku.

```
tomas@labserver:~/chef-repo$ knife openstack server create -N stackTest --sudo --bootstrap-proxy http://web-proxy.corp.hp.com:8088 --bootstrap-no-proxy labserver.helion.demo --flavor d9422513-43c2-4212-97c7-759ba0d66cd9  --image 8490e531-6c9d-4caa-8e15-61e5f749b0ff --network-ids 5cec492f-3150-44a9-9245-37d23a8c45b1 --openstack-private-network --bootstrap-network mujNet  --ssh-gateway ubuntu@10.201.0.10 --ssh-gateway-identity ~/mujklic.pem --ssh-user ubuntu --openstack-ssh-key-id mujKlic -i ~/mujklic.pem --run-list apt,mujWebServer --user-data nodnschef.sh

Waiting for server [wait time = 600].......
Instance ID        f602bcb1-1891-44be-a1e3-136a6bea9fe4
Name               stackTest
Flavor             d9422513-43c2-4212-97c7-759ba0d66cd9
Image              8490e531-6c9d-4caa-8e15-61e5f749b0ff
Keypair            mujKlic
State              ACTIVE
Availability Zone  nova
Bootstrapping the server by using bootstrap_protocol: ssh and image_os_type: linux

Waiting for sshd to host (192.168.5.15)done
Doing old-style registration with the validation key at ~/chef-repo/.chef/chefhelion.pem...
Delete your validation key in order to use your user credentials instead

Connecting to 192.168.5.15
..
192.168.5.15 Downloading Chef 12 for ubuntu...
..
192.168.5.15 Installing Chef 12
...
192.168.5.15 Starting the first Chef Client run...
192.168.5.15 Starting Chef Client, version 12.5.1
192.168.5.15 Creating a new client identity for stackTest using the validator key.
192.168.5.15 resolving cookbooks for run list: ["apt", "mujWebServer"]
192.168.5.15 Synchronizing Cookbooks:
192.168.5.15   - apt (2.9.2)
192.168.5.15   - mujWebServer (0.1.0)
192.168.5.15 Compiling Cookbooks...
192.168.5.15 Converging 14 resources
192.168.5.15 Recipe: apt::default
..
192.168.5.15 Recipe: mujWebServer::default
192.168.5.15   * apt_package[apache2] action install
192.168.5.15     - install version 2.4.7-1ubuntu4.8 of package apache2
192.168.5.15   * service[apache2] action enable (up to date)
192.168.5.15   * service[apache2] action start (up to date)
192.168.5.15   * file[/var/www/html/index.html] action create
192.168.5.15     - update content in file /var/www/html/index.html from 538f31 to a3bc70
192.168.5.15     --- /var/www/html/index.html   2015-12-11 05:31:13.840548000 +0000
192.168.5.15     +++ /var/www/html/.index.html20151211-1418-2rgdcz      2015-12-11 05:31:18.428548000 +0000
192.168.5.15     @@ -1,379 +1,4 @@
192.168.5.15     -
192.168.5.15     -<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
...
192.168.5.15     +Hele, jsem zaNATovanej!
192.168.5.15     +
192.168.5.15     +
192.168.5.15
192.168.5.15 Running handlers:
192.168.5.15 Running handlers complete
192.168.5.15 Chef Client finished, 7/16 resources updated in 39 seconds
Instance ID        f602bcb1-1891-44be-a1e3-136a6bea9fe4
Name               stackTest
Flavor             d9422513-43c2-4212-97c7-759ba0d66cd9
```

```
Image              8490e531-6c9d-4caa-8e15-61e5f749b0ff
Keypair            mujKlic
State              ACTIVE
Availability Zone  nova
```

Pokud je Knife vaším oblíbeným nástrojem, dává smysl používat Openstack plugin do něj.

3.2.5. Uvolněte zdroje v labu

Projděte si seznam nodů a klientů a odmazejte ty vaše, prosím. Současně take ukončete vaše běžící VM, ať šetříme zdroje labu.

```
tomas@labserver:~/chef-repo$ knife node list
prvniPokus

tomas@labserver:~/chef-repo$ knife node delete prvniPokus
Do you really want to delete prvniPokus? (Y/N) y
Deleted node[prvniPokus]

tomas@labserver:~/chef-repo$ knife client list
helion-validator
prvniPokus

tomas@labserver:~/chef-repo$ knife client delete prvniPokus
Do you really want to delete prvniPokus? (Y/N) y
Deleted client[prvniPokus]
```

3.3. Packer

Packer (packer.io) je zajímavý nástroj pro automatické vytváření hotových image jedním procesem pro různé platformy. Nejdříve si vyzkoušíme jednoduchý příklad přímo na našem OpenStack a později popíšeme další možnosti.

Nejprve budeme potřebovat znát UUID startovacího image (v našem labu to bude Ubuntu) a identifikátor vaší sítě.

```
tomas@labserver:~$ source stack
tomas@labserver:~$ openstack image list --max-width 50
+--------------------------------------+----------------------------------------------------+
| ID                                   | Name                                               |
+--------------------------------------+----------------------------------------------------+
| 6b837dc7-41b1-47f7-80de-088ad7f31d81 | webNode                                            |
| e68e44d2-eb89-45dc-92c7-a02f2bd15e9c | helionce-1.0.0_cloud-1.0.20151106T125623Z          |
| 9dd2e18b-e25a-4d6a-8c3e-58f471a847a8 | HP Helion Development Platform - Application       |
|                                      | Lifecycle Service Seed Node 2.0.0.548              |
| a1813f56-17ad-4ceb-8be9-f613dae55e98 | HP Helion Development Platform - Application       |
|                                      | Lifecycle Service Installer 2.0.0.548              |
| 8490e531-6c9d-4caa-8e15-61e5f749b0ff | Ubutnu 14.04                                       |
| 48131042-8e36-4f12-952e-32d73b3600df | cirros-0.3.4-x86_64                                |
+--------------------------------------+----------------------------------------------------+

tomas@labserver:~$ openstack network list -c ID -c Name
+--------------------------------------+----------+
| ID                                   | Name     |
+--------------------------------------+----------+
| 1712d425-ed71-443c-af0c-43d2f6e3c0aa | ext-net  |
| 9b1ed2c9-6bba-4b24-ac31-c7952d1392a4 | druhaSit |
| b830b33d-06fc-470a-aa12-ec4e709ef3c4 | mojeSit  |
+--------------------------------------+----------+
```

Na labServer v adresáři packer najdete příklad hotové Packer šablony. Bude potřeba v ní editovat některé údaje – zejména UUID vaší sítě. Pokud máte v projektu již přidělené a volné Floating IP, vyplňte je prosím do části “floating_ip”, abychom šetřili zdroje labu (tento řádek můžete take odstranit a Packer si sám řekne o novou Floating IP). Dále změňte položku “networks”, ověřte, že i ostatní parametry jsou v pořádku případně změňte název výsledného obrazu (image_name).

```
{
  "builders": [{
```

```
    "type": "openstack",
    "ssh_username": "ubuntu",
    "image_name": "mujPackerImage",
    "source_image": "8490e531-6c9d-4caa-8e15-61e5f749b0ff",
    "flavor": "m1.small",
    "networks": ["b830b33d-06fc-470a-aa12-ec4e709ef3c4"],
    "use_floating_ip": true,
    "floating_ip": "10.201.0.10",
    "floating_ip_pool": "ext-net"
  }],
  "provisioners": [{
    "type": "shell",
    "inline": [
      "export http_proxy=http://web-proxy.corp.hp.com:8088",
      "sudo -E apt-get update",
      "sudo -E apt-get install nginx -y",
      "printf 'Web server zabaleny Packerem\n\n' | sudo tee /usr/share/nginx/html/index.html"
      ]
  }]
}
```

Přihlašovací údaje do OpenStack si Packer vezme z prostředí (ENV), takže je nemusíme zadávat do šablony. V první části (builder) pracujeme s OpenStack a v druhé (provisioning) děláme požadované operace. Pro jednoduchost jsme zvolili jednoduché spuštění skriptu přímo uvnitř VM při vytváření image – tento script stáhne webový server a modifikuje hlavní stránku. Pokročilejší možnosti si popíšeme později.

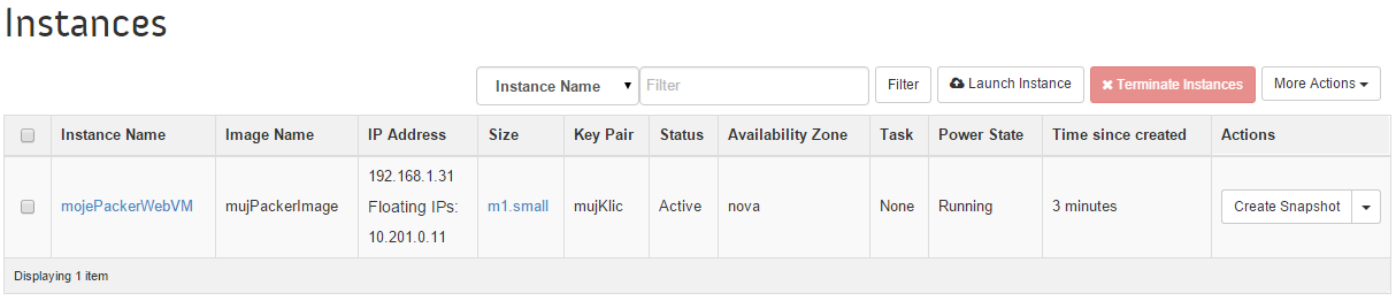
Nejprve přidejte do prostředí informaci o doméně a proveďte validaci šablony co do syntaxe.

```
tomas@labserver:~/packer$ export OS_DOMAIN_NAME=Default
tomas@labserver:~/packer$ packer validate packer_web.json
Template validated successfully.
```

Pokud je vše v pořádku, spusťte budování nového image. Sledujte textový výpis i GUI OpenStack – uvidíte co se děje.

```
tomas@labserver:~/packer$ packer build packer_web.json
openstack output will be in this color.

==> openstack: Discovering enabled extensions...
==> openstack: Loading flavor: m1.small
    openstack: Verified flavor. ID: d9422513-43c2-4212-97c7-759ba0d66cd9
==> openstack: Creating temporary keypair for this instance...
==> openstack: Launching server...
    openstack: Server ID: 150862d4-91ae-48ac-a02b-8c1890a4e690
==> openstack: Waiting for server to become ready...
==> openstack: Associating floating IP with server...
    openstack: IP: 10.201.0.10
    openstack: Added floating IP 10.201.0.10 to instance!
==> openstack: Waiting for SSH to become available...
==> openstack: Connected to SSH!
==> openstack: Provisioning with shell script: /tmp/packer-shell133305692
    openstack: sudo: unable to resolve host mujpackerimage
    openstack: Ign http://nova.clouds.archive.ubuntu.com trusty InRelease
    openstack: Get:1 http://nova.clouds.archive.ubuntu.com trusty-updates InRelease [64.4 kB]
    openstack: Get:2 http://security.ubuntu.com trusty-security InRelease [64.4 kB]
...
    openstack: Processing triggers for libc-bin (2.19-0ubuntu6.6) ...
    openstack: sudo: unable to resolve host mujpackerimage
    openstack: Web server zabaleny Packerem
    openstack:
==> openstack: Stopping server...
    openstack: Waiting for server to stop...
==> openstack: Creating the image: mujPackerImage
    openstack: Image: 802e0ee9-58fb-4a38-843f-7866b2db95a6
==> openstack: Waiting for image to become ready...
==> openstack: Terminating the source server...
==> openstack: Deleting temporary keypair...
Build 'openstack' finished.

==> Builds finished. The artifacts of successful builds are:
--> openstack: An image was created: 802e0ee9-58fb-4a38-843f-7866b2db95a6
```

Vytvořte si instanci ze svého nového Instance Snapshot a vyzkoušejte.




```
tomas@labserver:~/packer$ curl 10.201.0.11
Web server zabaleny Packerem
```

Na závěr odstraňme proměnnou OS_DOMAIN_NAME (OpenStackClient s ní má problém a CLI by nám nefungovalo)

```
tomas@labserver:~/packer$ unset OS_DOMAIN_NAME
```

Instanci i Image prosím zrušte, abychom šetřili zdroje v labu.

Packer je velmi mocný nástroj – dotkněme se některých dalších možností. Podstatné je, že můžete v jedné šabloně kombinovat víc providerů a postupů. Z jednoho místa jedním konzistentním procesem tak můžete vytvořit zlaté diskové obrazy pro OpenStack, vSphere, čisté KVM/QEMU, Vagrant nebo třeba Amazon a dokonce i Docker kontejner (což může být zajímavá varianta v postupném přechodu na kontejnerové technologie). Z jakých částí se skládá?

- Builders – prostředí, pro která má Packer vytvářet hotový obraz. Možnosti zahrnují Amazon EC2, OpenStack, VMware, QEMU, VirtualBox, Google Compute Engine nebo třeba i Docker kontejner
- Provisioners – nástroje, které ze základního obrazu „dotáhnou“ VM do stavu, který chcete zmrazit do zlatého obrazu. Mezi ně patří jednoduchý i složitější bash shell, PowerShell, uploadování souborů, lokální Ansible, Chef Solo, Puppet Masterless nebo Salt
- Post-processors – po ukončení všech operací můžete ještě výsledek zabalit nebo kolem něj vytvořit nějaký předpis. Sem patří příprava Vagrant prostředí, upload do vSphere, komprese do archivu nebo push do Docker registru

Potřebujete konzistentním způsobem vytvářet obrazy pro velmi různorodá prostředí? Packer je dobrý způsob, jak to vyřešit. Začněte ze základních image s OS, automatizovaně přidejte cokoli potřebujete a získejte výsledek pro různé platformy od klasických hypervisorů přes IaaS u vás až po public cloud nebo přenositelný Docker kontejner.

4. Objektová storage: OpenStack Swift

4.1. Objektová storage

Vyťukejte v prohlížeči www.cloudsvet.cz. Co se stane? Browser kontaktuje tuto URL (a zajistí si informace o všem pod tím, tedy z DNS sežene IP, z ARP zjistí cílovou MAC nebo MAC výchozí brány) a co dostane? Přistane mu proud bitů, kterým bude HTML soubor a ten browser interpretuje. Pokud si na našem webu dáte stáhnout PDF ze sekce Dokumenty, pošle web server do prohlížeče jiný proud reprezentující PDF soubor (a metadata, která informují právě o tom, co je to za typ souboru). Nebo to bude obrázek a ten se vám začne vykreslovat tak, jak k vám bity přichází. Možná to bude podcast nebo video, které začnete poslouchat a neřešíte, že ještě nedorazil poslední bit ze závěrečných titulků. Možná to nebudou data přímo pro browser, ale půjde o RESTful API (webové programovací rozhraní) využívané mobilní aplikací na vašem telefonu nebo se takto propojí vaše ERP s ERP dodavatelů. Takhle funguje web a HTTP.

Objektová storage je vlastně web - jak nahrávání tak čtení je stejné, jako pohyb po webu. Každý nahraný objekt zahrnuje metadata (popis o co jde apod.) a URL, kde ho najdete. K jeho získání nepotřebujete žádný nízkoúrovňový protokol - neřešíte iSCSI, nastavení FC adaptéru, stačí IP konektivita, která funguje vždy stejně - z Ethernetu, WiFi, 3G nebo LTE. Nemusíte znát žádný protokol - FTP, NFS, SMB, SCSI - stačí HTTP, které umí nejen vaše servery a počítače, ale i telefon, televize, lednička a vaše auto. To je obrovská síla objektové storage.

Tento typ ukládání dat za sebou nemá dlouhou historii, existující trhy, které by bylo třeba bránit, patenty, zákazníky a jejich proinvestované prostředky. To je jeho velká výhoda - přímo ve svých pricipech má decentralizované distribuované řešení, je tedy nativně scale-out. To je především díky eventuálně konzistentnímu řešení (nemusí na rozdíl třeba od souborových systémů dodržovat POSIX).

Objektová storage obvykle funguje tak, že vezmete jednoduché nody (server s lokálními disky) a nahrajete tam chytrý software. Řešení obsahuje směrovací mechanismus, který udržuje přehled o tom, kde se který objekt fyzicky nachází - tedy má mapování URL (což je jednoznačný identifikátor objektu) na fyzické server a konkrétní disk, kde objekt je (objekt se uloží celý na disk a většinou i na několik dalších pro redundanci). Směrovač je schopen identifikovat odumřelé nody a požadavky uživatelů směrovat na ty, co fungují. Další komponentou je out-of-band kontroler a také nějaký proces, který udržuje přehled integrity. Tak například vytvoříte pravidlo, že jistý typ dat bude uložen minimálně na třech nodech, které jsou rozprostřené přes minimálně dvě zóny dostupnosti (o komplexnějších politikách později). Pokud dojde k roztržení clusteru, havárii nodu nebo odpodanutí celé zóny, zajistí mechanismus potřebnou "doreplikaci" tak, aby bylo všem politikám učiněno zadost. Všechny uvedené části, které by se daly označit za řídící, fungují na principu scale-out clusteru. S každým nodem tak zvyšujete jak celkovou kapacitu, tak celkový výkon a to plynule a za provozu (už žádné situace, kdy vám dojdou pozice pro disky v poli nebo kdy musíte vyhodit stávající kontroler a nahradit silnějším).

Některé objektové storage replikují objekty na nastavený počet kopií průběžně, tedy s eventuální konzistencí (AP systém dle CAP teorému) - Swift typicky vrací odpověď o úspěšném zápisu v okamžiku, kdy zapíše polovinu předepsaných kopií (většinou se nastavují 3 kopie v algoritmu „co nejméně podobné“, tedy ne na stejném serveru, ale ideálně v jiných zónách dostupnosti. V poslední době se pracuje i na erasure coding (místo kopie celých replik rozsekání na dílky a kontrolní součty - lepší využití místa za cenu trochu horšího výkonu).

4.2. Co je OpenStack Swift

Před rokem 2010 vytvořil public cloud poskytovatel Rackspace svůj vlastní object storage systém jako reakci na Amazon S3. V roce 2010 americká NASA uvolnila do open source svůj kód pro nadstavbu nad virtualizací a Rackspace udělal pro tento společný projekt totéž se svou object storage. Vznikl tak OpenStack a projekty Nova a Swift (a takhle ve dvou projektech to celé začalo). Swift je tedy plně otevřený s Apache2 licencí a je nasaditelný jak samostatně, tak jako součást OpenStack distribuce. Přináší tak naprostou otevřenost, žádný vendor lock-in a možnost nasazení na komoditním železe.

Swift je objektová storage, obvykle namíchaná jako AP systém v rámci CAP teorému. O tom proč je tento typ storage tak zajímavý jsem psal v článku na cloudsvet.cz. Nabízí tedy masivní škálovatelnost ve výkonu i kapacitě a řadu zajímavých funkcí.

V rámci samotného OpenStack je Swift používán třeba k ukládání image (Glance modulem) nebo záloh volumů (Cinder modul). Výhodou toho také je, že různé instalátory OpenStack umí rychle a snadno instalovat i Swift nody, takže se svou object storage můžete pracovat velmi cloudovým způsobem. Typické využití v dalších systémech je ukládání multimédií (audio, video, záznamy z kamer), snímků ve vysokém rozlišení (zdravotnictví, průmysl, umění), při renderingu a modelování, k zálohování a diskovým obrazům nebo pro cloudové ukládání fotek a dokumentů. To všechno můžete mít nejen jako službu od veřejného poskytovatele, ale Swift si jednoduše postavíte interně u sebe - bezpečně, spolehlivě a bez závazků.

Přestavme si nejzajímavější funkce:

- Samozřejmě přirozené vlastnosti scale-out object storage - velká horizontální škálovatelnost, použití komoditního hardware, vstup a výstup HTTP protokolem atd.
- Erasure-coding (v nejnovější verzi Kilo) umožňuje místo ukládání celých objektů na nodu a jejich replikaci k několika dalším využít rozsekání velkého objektu na dílky, ty rozprostřít po nodech a přidat k nim kontrolní součty. Za cenu snížení výkonu a zvýšení latence lze snížit cenu za úložný prostor (srovnání obou metod se budeme věnovat v samostatném článku)
- Řízení přístupu k vašim objektům (kdo smí?)
- Streamované vkládání, tedy schopnost vkládat objekt aniž by byla dopředu známa jeho délka

- Verzování zápisů, takže URL vede vždy na nejnovější verzi, ale systém drží a umožní získat ty předchozí
- Přímý upload z klienta přes HTTP POST, tedy webová aplikace dovoluje prohlížeči vzít lokální soubor a poslat přímo do object storage, nemusí se jít přes web server
- Časově omezené URL (dočasné odkazy)
- Časově omezené objekty (sami se odmažou)
- Velmi flexibilní storage politiky - dá se například podle uživatele nebo typu dat preferovat SSD, omezit fyzikální umístění dat na určitý region nebo naopak vynutit celoplanetární repliku

4.3. Jak se data ukládají

Připomeňme si, že adresa objektu je ve formátu „země, město, ulice“, tedy /account/container/object. Když Swift požádáte o objekt na nějaké adrese, ten musí vědět na kterém nodu a kterém disku ho najít. Klasický centralizovaný file systém by se s problémem vypořádal tak, že by si držel tabulku se seznamem souborů, metadaty a informací o jeho fyzickém umístění na disku. Scale-out distribuovaná object storage by ale s takovým přístupem měla problém.

Ve swift existuje seznam prostorů, kontejnerů a objektů a Swift ho svými procesy udržuje, nicméně ten nijak nesouvisí s vlastním umísťováním objektů a jejich nalezením - jde jednoduše o seznamy, kde uživatel nejde název svých objektů a jejich URL. K nalezení objektu dojde až po zadání této URL. Jak na to?

Udržovat stavový seznam objektů a jejich umístění špatně škáluje, proto se používá HASH funkce. Jde o matematickou operaci, která z libovolného vstupu vytvoří jeho otisk o stále stejné velikosti. Používá se dnes prakticky všude - kryptografie, ukládání dat, integrita, certifikáty - typickými zástupci jsou třeba MD5 nebo SHA. Vezmete tedy URL ve formátu /account/container/object a udělate nad tímto řetězcem HASH, které vede na nějaké číslo (je docela velké). Protože počet disků bude určitě menší mohli bychom použít modulo operaci a převést ho třeba na číslici 0-7 pro našich 8 disků a podle toho víme, kam máme zapsat (a kde najít) náš objekt. Čas vyhledávání by byl velmi rychlý a nepotřebujeme složitou stavovou tabulku. Nicméně změna počtu disků by znamenala použít modulo na jiný počet - v ten okamžik prakticky veškerá dosavadní umístění budou špatně a museli bychom celý cluster přeskládat, což je extrémně náročné. Navíc přidávání disků tímto způsobem způsobí, že jejich kompetence ve smyslu velikosti výseče budou nevyrovnané. Chce to nějaké vylepšení.

Tím je použití Consistent Hash Ring. Představte si možné výsledky HASH jako kolečko s číslicemi (HASH vrací výsledek ve stejném rozmezí bez ohledu na délku vstupu, takže kolečko bude stále stejně veliké). Zanaste vaše disky/nody na toto kolečko jako body. Disk bude "vlastnit" všechny objekty, jejichž hash je v rozmezí daném bodem disku na kružnici a vším po směru hodinových ručiček až do bodu, kde sedí další disk. V čem je to lepší, než obyčejné modulo? Když přidáte disk, ovlivníte pouze bezprostředního souseda - usurpujete si nárok na část jeho dat, protože jejich domovem jste teď vy. Všichni ostatní měnit nemusí.

Už je to lepší, ale přidání disku vytvoří stres pro jednoho souseda, což také není ideální. Swift tedy funguje ještě o něco chytřeji. Celá kružnice se rozseká na relativně velký počet výsečí, třeba 32K. Tyto se přidělí diskům, každý tedy vlastní výsečí hned několik a to napříč celým kolečkem. Nově přidaný disk si tak převezme některé výseče a rovnoměrně vycucne po kousíčkách dat z ostatních disků. Připomínám, že tato výseč (partition) je jen rozsah možných výsledků HASH.

Pojďme teď kroužků použít víc. Swift tak vytvoří jeden kroužek pro informace o záznamech o accountech (prostorech). Také jeden o kontejnerech a dále pro každou storage policy.

Jak se výseče rozdělí mezi disky v rámci různých storage policy? A jak se budou držet repliky? Každý prstenec obsahuje dvě základní tabulky - tou první je seznam všech nodů a tou druhou tabulka přidělených partition k jednotlivým diskům a replikám. Představte si jednoduchou tabulku, jejíž sloupce jsou čísla partition a řádky repliky (typicky tak bude mít tabulka 3 řádky). Obsahem je mapování partition na fyzické umístění. Při rozdělování výsečí mezi fyzické disky se berou v úvahu dva faktory. Tím prvním je "váha" disku, tedy administrátorem nastavitelné preference disku (například v reflexi na jeho velikost či rychlost). Druhou je pravidlo maximální diverzity. Pakliže jedna výseč má mít tři repliky, pro jejich fyzické umístění se zvolí co nejodlišnější disk - v jiné zóně nebo alespoň v jiném serveru.

4.4. Použití CLI

Vyzkoušejme si příkazovou řádku pro Swift (úvod do problematiky najdete v části OpenStack CLI tohoto lab guide).

Načtěte komunikační parametry.

```
tomas@labserver:~$ source stack
```

Vytvoříme nový kontejner

```
tomas@labserver:~$ openstack container create novyKontejner
+---------------------------------------+---------------+------------------------------------+
| account                               | container     | x-trans-id                         |
+---------------------------------------+---------------+------------------------------------+
| AUTH_d20fb4c9c0d645b4962484390a3be701 | novyKontejner | tx5bd654d1ad8e4b0e9d853-00566be483 |
+---------------------------------------+---------------+------------------------------------+
```

Vypíšeme si svoje kontejnery.

```
tomas@labserver:~$ openstack container list
+---------------+
| Name          |
+---------------+
| mujKontejner  |
| novyKontejner |
+---------------+
```

Vytvořme nový soubor a uploadněme jako objekt.

```
tomas@labserver:~$ echo "Muj text" > mujsoubor.txt

tomas@labserver:~$ openstack object create novyKontejner mujsoubor.txt
+---------------+---------------+----------------------------------+
| object        | container     | etag                             |
+---------------+---------------+----------------------------------+
| mujsoubor.txt | novyKontejner | b56b9d21d9b5a09869ae2c45ab74e178 |
+---------------+---------------+----------------------------------+
```

Vypišme obsah kontejneru

```
tomas@labserver:~$ openstack object list novyKontejner
+---------------+
| Name          |
+---------------+
| mujsoubor.txt |
+---------------+
```

Smažeme objekt i kontejner

```
tomas@labserver:~$ openstack object delete novyKontejner mujsoubor.txt
tomas@labserver:~$ openstack container delete novyKontejner
tomas@labserver:~$ openstack container list
+--------------+
| Name         |
+--------------+
| mujKontejner |
+--------------+
```

4.5. Držení předchozích verzí objektů

První pokročilá funkce, kterou si vyzkoušíme, je schopnost Swiftu držet všechny předchozí verze objektu. Funguje to tak, že vytvoříme jeden kontejner pro archivaci starších verzí a také jiný kontejner, z kterého budou starší verze propadávat do archivu. Výsledkem bude, že v kontejneru budeme mít vždy nejposlednější verze objektu, ale můžeme se přímo podívat do archivu na ty předchozí. Pokud v kontejneru objekt smažeme a při tom existuje jeho starší verze, Swift automaticky na jeho místo dosadí nejnovější předchozí verzi.

Vytvořme dva kontejnery - archiv a aktualni (můžete použít i GUI).

```
tomas@labserver:~$ openstack container create archiv
+---------------------------------------+-----------+------------------------------------+
| account                               | container | x-trans-id                         |
+---------------------------------------+-----------+------------------------------------+
```

```
| AUTH_d20fb4c9c0d645b4962484390a3be701 | archiv    | txd872e970893a439d999af-00566e7d84 |
+---------------------------------------+-----------+------------------------------------+

tomas@labserver:~$ openstack container create aktualni
+---------------------------------------+-----------+------------------------------------+
| account                               | container | x-trans-id                         |
+---------------------------------------+-----------+------------------------------------+
| AUTH_d20fb4c9c0d645b4962484390a3be701 | aktualni  | tx9281bc4d2a9c4aff8eab8-00566e7d90 |
+---------------------------------------+-----------+------------------------------------+
```

V dalším kroku potřebujeme zapnout funkci držení verzí. K tomu zatím neexistuje GUI ani příkaz v rámci sjednoceného openstack CLI, takže použijeme starší příkaz (dostaneme varování, ale to můžeme ignorovat), kterým pošleme kontejneru nastavení hlavičky. Tím zapneme verzování. U kontejneru aktualni tedy nastavujeme, že starší verze objektů se mají ukládat do kontejneru archiv.

```
tomas@labserver:~$ swift post aktualni -H 'X-Versions-Location: archiv'
/usr/local/lib/python2.7/dist-packages/keystoneclient/service_catalog.py:198: UserWarning: Providing attr
without filter_value to get_urls() is deprecated as of the 1.7.0 release and may be removed in the 2.0.0
release. Either both should be provided or neither should be provided.
  'Providing attr without filter_value to get_urls() is '

tomas@labserver:~$ swift stat aktualni
/usr/local/lib/python2.7/dist-packages/keystoneclient/service_catalog.py:198: UserWarning: Providing attr
without filter_value to get_urls() is deprecated as of the 1.7.0 release and may be removed in the 2.0.0
release. Either both should be provided or neither should be provided.
  'Providing attr without filter_value to get_urls() is '
           Account: AUTH_d20fb4c9c0d645b4962484390a3be701
         Container: aktualni
           Objects: 0
             Bytes: 0
          Read ACL:
         Write ACL:
           Sync To:
          Sync Key:
      Accept-Ranges: bytes
   X-Storage-Policy: General
        X-Timestamp: 1449911788.96884
         X-Trans-Id: tx9b89d36fe3eb456c9dbb7-00566e7dfa
       Content-Type: text/plain; charset=utf-8
X-Versions-Location: archiv
```

Teď už si to můžeme vyzkoušet. Vytvořte v kontejeru aktualni nový objekt z našeho textového souboru.

```
tomas@labserver:~$ openstack object create aktualni mujsoubor.txt
+---------------+-----------+----------------------------------+
| object        | container | etag                             |
+---------------+-----------+----------------------------------+
| mujsoubor.txt | aktualni  | b56b9d21d9b5a09869ae2c45ab74e178 |
+---------------+-----------+----------------------------------+
```

Přidejme do souboru lokálně řádek a uploadujme novou verzi

```
tomas@labserver:~$ echo "dalsi radek" >> mujsoubor.txt

tomas@labserver:~$ cat mujsoubor.txt
Muj text
dalsi radek

tomas@labserver:~$ openstack object create aktualni mujsoubor.txt
+---------------+-----------+----------------------------------+
| object        | container | etag                             |
+---------------+-----------+----------------------------------+
| mujsoubor.txt | aktualni  | 96ad90095dad7b3b2529a05b4e3a2529 |
+---------------+-----------+----------------------------------+
```

Swift aktualizoval objekt, ale jeho starší verzi najdete s časovou značkou v kontejneru archiv.

```
tomas@labserver:~$ openstack object list archiv
+----------------------------------+
| Name                             |
+----------------------------------+
| 00dmujsoubor.txt/1450081895.84895 |
```

```
+----------------------------------+
```

Udělejme ještě jednu verzi.

```
tomas@labserver:~$ echo "jeste jeden radek" >> mujsoubor.txt

tomas@labserver:~$ openstack object create aktualni mujsoubor.txt
+---------------+-----------+----------------------------------+
| object        | container | etag                             |
+---------------+-----------+----------------------------------+
| mujsoubor.txt | aktualni  | 1d84480e95d3e31222e8d0e4a1f2da72 |
+---------------+-----------+----------------------------------+
```

V archivu už máme dvě předchozí verze.

```
tomas@labserver:~$ openstack object list archiv
+----------------------------------+
| Name                             |
+----------------------------------+
| 00dmujsoubor.txt/1450081895.84895 |
| 00dmujsoubor.txt/1450081912.84630 |
+----------------------------------+
```

Teď pojďme smazat objekt z kontejneru aktualni. Co se stane? Z archivu se do kontejneru aktualni přesune jeho historická (ale nejmladší) verze.

```
tomas@labserver:~$ openstack object delete aktualni mujsoubor.txt

tomas@labserver:~$ openstack object list aktualni
+---------------+
| Name          |
+---------------+
| mujsoubor.txt |
+---------------+

tomas@labserver:~$ openstack object list archiv
+----------------------------------+
| Name                             |
+----------------------------------+
| 00dmujsoubor.txt/1450081895.84895 |
+----------------------------------+
```

Máme hotovo - verzování nám funguje. Jak vidíte všechno se nastavuje na úrovni kontejnerů, takže se můžete takto chovat pouze k vybraným datům.

4.6. Omezení platnosti objektu

Swift umožňuje u objektu nastavit časově omezenou platnost a ten se ve stanovené datum a čas stane nepřístupným. Můžete tak zabránit k přístupu k dokumentům, které už v daný okamžik nebudou aktuální (standardy, výkresy). Ve Swift implementaci může rovněž běžet čistící proces, který takto vypršené objekty fyzicky smaže a šetří tak místo v objektové storage. Často firmy dávají data na FTP nebo NFS server se záměrem dočasnosti, ale pak se na to zapomene a zbytečná data pak trvale zabírají místo. Pokud dopředu víte, že objekt má časové mezení, Swift vám jeho vypršení ohlídá sám.

Funguje to nastavením hodnot v metadatech a to jak při nahrávání objektu přes API nebo modifikací metadat již nahraného objektu. Vyzkoušíme si to. Použitý atribut v metadatech se jmenuje X-Delete-At a obsahuje datum a čas v Unix formátu (tedy počet vteřin od 1.1.1970). Abychom nemuseli složitě počítat, použijeme variantu X-Delete-After, kam zapíšeme počet sekund, po které má být objekt platný.

```
tomas@labserver:~$ openstack container create docasne
+---------------------------------------+-----------+-------------------------------------+
| account                               | container | x-trans-id                          |
+---------------------------------------+-----------+-------------------------------------+
| AUTH_d20fb4c9c0d645b4962484390a3be701 | docasne   | tx187397150c2a4e4e8eb77-00566e91f1 |
+---------------------------------------+-----------+-------------------------------------+

tomas@labserver:~$ openstack object create docasne mujsoubor.txt
+---------------+-----------+----------------------------------+
| object        | container | etag                             |
```

```
+---------------+-----------+----------------------------------+
| mujsoubor.txt | docasne   | 1d84480e95d3e31222e8d0e4a1f2da72 |
+---------------+-----------+----------------------------------+
```

Máme tedy kontejner a objekt. Pojďme teď u něj nastavit expiraci 120 vteřin.

```
tomas@labserver:~$ swift post docasne mujsoubor.txt -H 'X-Delete-After: 120'
/usr/local/lib/python2.7/dist-packages/keystoneclient/service_catalog.py:198: UserWarning: Providing attr
without filter_value to get_urls() is deprecated as of the 1.7.0 release and may be removed in the 2.0.0
release. Either both should be provided or neither should be provided.
  'Providing attr without filter_value to get_urls() is '
```

Zkuste objekt stáhnou - půjde to.

```
tomas@labserver:~$ openstack object save docasne mujsoubor.txt
```

Počkejte dvě minuty a zkuste to znovu. Objekt už není nalezen, není dostupný.

```
tomas@labserver:~$ openstack object save docasne mujsoubor.txt
Not Found (HTTP 404)
```

Pokud si ale teď necháte vypsat obsah kontejneru, možná tam ještě chvilku bude. Nikdo už se k němu nedostane, ale fyzicky ještě není smazaný. Jedná se tedy o nedostupný, ale ještě nesmazaný objekt. Swift v pravidelných intervalech (obvykle kolem pěti minut) prochází svoje nody a neplatné objekty fyzicky odstraňuje.

```
tomas@labserver:~$ openstack object list docasne
+---------------+
| Name          |
+---------------+
| mujsoubor.txt |
+---------------+
```

Zkuste to později - už bude pryč.

```
tomas@labserver:~$ openstack object list docasne
```

4.7. Dočasné veřejné URL

Swift kotejner můžete označit jako Private nebo Public. Může ovšem být situace, kdy obsah kontejneru nechcete volně přístupný všem, ale současně vznikne potřeba někomu dát velmi rychle a jednoduše možnost objekt stáhnout, možná i modifikovat. Například máte soubor ve Swiftu a na email je moc velký. Swift umožňuje vygenerovat dočasnou URL, pod kterou si lze objekt vyzvednout a můžete s přesností na vteřiny nastavit její platnost. Tato URL nebude vyžadovat žádné přihlášení, ale po vypršení času už nebude fungovat (na samotný objekt to pro přihlášené uživatele samozřejmě nemá vliv). Tato funkce nemá nativní grafickou nadstavbu (je ovšem velmi jednoduché si ji vytvořit a integrovat do nějakého z vašich systémů), použijeme CLI.

Nejprve musíte ke svému účtu přiřadit náhodný řetězec, který se bude používat při kryptografickém generování URL (v praxi použijte něco sofistikovanější, než mojeheslo).

```
tomas@labserver:~$ swift post -m "Temp-URL-Key:mojeheslo"
/usr/local/lib/python2.7/dist-packages/keystoneclient/service_catalog.py:198: UserWarning: Providing attr
without filter_value to get_urls() is deprecated as of the 1.7.0 release and may be removed in the 2.0.0
release. Either both should be provided or neither should be provided.
  'Providing attr without filter_value to get_urls() is '
```

Vytvořte kontejner a uploadujte objekt.

```
tomas@labserver:~$ openstack container create dalsiKontejner
+---------------------------------------+----------------+------------------------------------+
| account                               | container      | x-trans-id                         |
+---------------------------------------+----------------+------------------------------------+
| AUTH_d20fb4c9c0d645b4962484390a3be701 | dalsiKontejner | tx4bd30a477ad7408bbe338-00566e9b21 |
+---------------------------------------+----------------+------------------------------------+

tomas@labserver:~$ openstack object create dalsiKontejner mujsoubor.txt
+---------------+----------------+----------------------------------+
| object        | container      | etag                             |
```

```
+--------------+---------------+----------------------------------+
| mujsoubor.txt | dalsiKontejner | 1d84480e95d3e31222e8d0e4a1f2da72 |
+--------------+---------------+----------------------------------+
```

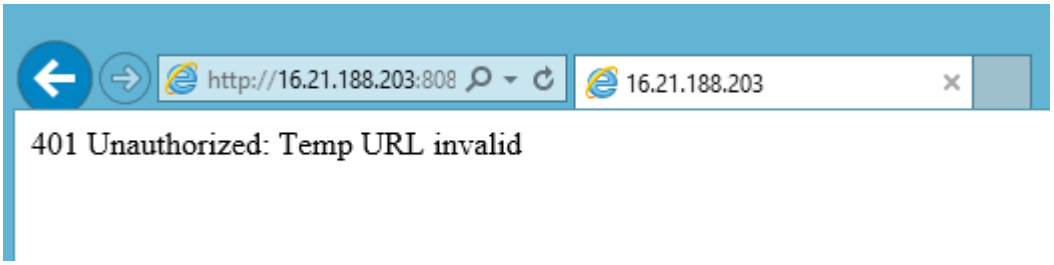
Teď vygenerujeme URL. Říkáme, zda bude pouze pro čtení (GET) nebo povolíme modifikaci objektu (PUT). Možnost PUT se často využívá v kombinaci s funkcí uploadování z webového formuláře (bezpečný způsob, jak můžete ve webové aplikaci nechat někoho uploadovat objekt do privátního kontejneru - webová aplikace vygeneruje tempurl, na kterou formulář objekt přímo pošle - uživatel tak neukládá skrz webový server samotný, což by ho zatěžovalo, ale přímo do object store). Dále říkáme počet vteřin, po které bude dočasná URL platná (přepínačem --absolute můžete také definovat cílové datum a čas v UNIX formátu). Pak specifikujete adresu k objektu (to je váš account/kontejner/objekt).

```
tomas@labserver:~$ swift tempurl GET 240
/v1/AUTH_d20fb4c9c0d645b4962484390a3be701/dalsiKontejner/mujsoubor.txt mojeheslo
/v1/AUTH_d20fb4c9c0d645b4962484390a3be701/dalsiKontejner/mujsoubor.txt?temp_url_sig=632a0f1702fbc4ba2b0f88486
7732c0b082d5e5d&temp_url_expires=1450090240
```

Na svém počítači vyzkoušejte, že jste schopni si stáhnout objekt z dočasné URL (adresa je 16.21.188.203:8080 následovaná výše uvedenou cestou.

```
http://16.21.188.203:8080/v1/AUTH_d20fb4c9c0d645b4962484390a3be701/dalsiKontejner/mujsoubor.txt?temp_url_sig=
632a0f1702fbc4ba2b0f884867732c0b082d5e5d&temp_url_expires=1450090240
```

Po nějaké době to zkuste znovu - URL už vypršela.



4.8. Řízení přístupu aneb od kolegů po veřejnost

Zatím jsme vytvářeli kontejnery, které byly dostupné pouze uživatelům našeho tenantu (tedy kromě vygenerování dočasné URL). OpenStack Swift ale disponuje možností vytvářet přístupová pravidla (ACL) jak pro čtení, tak pro zápis. Ve výchozím stavu nebudou žádná (tedy ke kontejneru mohou jen uživatelé tenantu).

```
tomas@labserver:~$ swift stat dalsiKontejner
/usr/local/lib/python2.7/dist-packages/keystoneclient/service_catalog.py:198: UserWarning: Providing attr
without filter_value to get_urls() is deprecated as of the 1.7.0 release and may be removed in the 2.0.0
release. Either both should be provided or neither should be provided.
  'Providing attr without filter_value to get_urls() is '
        Account: AUTH_d20fb4c9c0d645b4962484390a3be701
      Container: dalsiKontejner
        Objects: 1
          Bytes: 39
       Read ACL:
      Write ACL:
        Sync To:
       Sync Key:
   Accept-Ranges: bytes
X-Storage-Policy: General
     X-Timestamp: 1450089249.14773
      X-Trans-Id: tx377fa57478f34b76a641a-00566f017b
    Content-Type: text/plain; charset=utf-8
```

Můžete povolit přístup jinému tenantu (v takovém případě zadáváte jména tenantů oddělená čárkou) nebo jen konkrétním osobám nějakých tenantů (pak zadáváte tenant:jméno a pokud je takových záznamů víc, oddělujete je čárkou). Můžete také vytvořit kontejner, který je volně přístupný ke čtení odkudkoli bez jakéhokoli ověření (váš object store pak může třeba přímo servírovat obrázky na web přímo bez „průtoku“ web serverem a tak podobně). Pro volný přístup se používá řetězec .r:*,.rlistings.

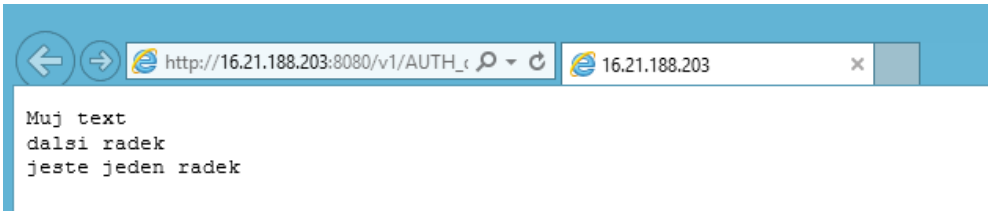
Povolte tomuto kontejneru veřejný přístup.

```
tomas@labserver:~$ swift post dalsiKontejner -r '.r:*,.rlistings'
/usr/local/lib/python2.7/dist-packages/keystoneclient/service_catalog.py:198: UserWarning: Providing attr
without filter_value to get_urls() is deprecated as of the 1.7.0 release and may be removed in the 2.0.0
release. Either both should be provided or neither should be provided.
  'Providing attr without filter_value to get_urls() is '

tomas@labserver:~$ swift stat dalsiKontejner
/usr/local/lib/python2.7/dist-packages/keystoneclient/service_catalog.py:198: UserWarning: Providing attr
without filter_value to get_urls() is deprecated as of the 1.7.0 release and may be removed in the 2.0.0
release. Either both should be provided or neither should be provided.
  'Providing attr without filter_value to get_urls() is '
         Account: AUTH_d20fb4c9c0d645b4962484390a3be701
       Container: dalsiKontejner
         Objects: 1
           Bytes: 39
        Read ACL: .r:*,.rlistings
       Write ACL:
         Sync To:
        Sync Key:
   Accept-Ranges: bytes
      X-Trans-Id: txc4733acddd3f4a758f245-00566f08e2
X-Storage-Policy: General
     X-Timestamp: 1450089249.14773
    Content-Type: text/plain; charset=utf-8
```

Ze svého počítače si nechte vypsat obsah kontejneru zcela bez přihlášení na adrese

http://16.21.188.203:8080/v1/AUTH_d20fb4c9c0d645b4962484390a3be701/dalsiKontejner

Funguje - můžete zadat přímo URL vašeho objektu (tedy přidat mujsoubor.txt na konec předchozí).



4.9. Swift jako statický web

OpenStack Swift může sloužit také jako primitivní statický web. Výhodou je, že nemusíte nic moc řešit a máte jednoduchý webový přístup k objektům některého z kontejnerů a využít tak všech výhod object storage (tzn. vysoce redundantní řešení, data efektivně distribuovaná přes mnoho nodů/disků, možnost inteligentní replikace napříč planetou, programovatelný přístup z aplikací přes jednoduché API apod.).

Vytvořme si ještě další kontejner, uploadujme do něj nějaký objekt a zpřístupněme kontejner všem.

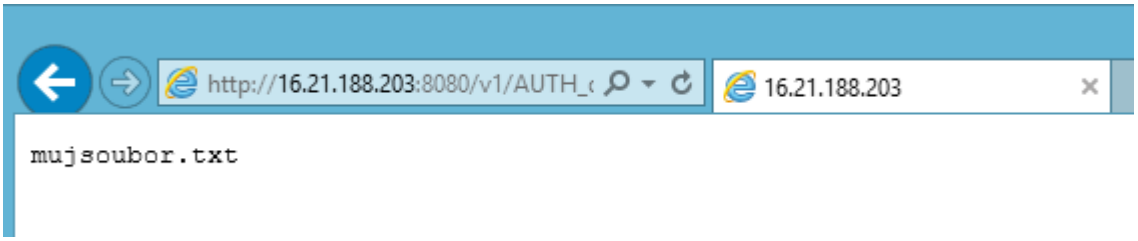
```
tomas@labserver:~$ openstack container create staticweb
+---------------------------------------+-----------+------------------------------------+
| account                               | container | x-trans-id                         |
+---------------------------------------+-----------+------------------------------------+
| AUTH_d20fb4c9c0d645b4962484390a3be701 | staticweb | tx74b698ec077246e7a1c41-00566f0c5d |
+---------------------------------------+-----------+------------------------------------+

tomas@labserver:~$ openstack object create staticweb mujsoubor.txt
+---------------+-----------+----------------------------------+
| object        | container | etag                             |
+---------------+-----------+----------------------------------+
| mujsoubor.txt | staticweb | 1d84480e95d3e31222e8d0e4a1f2da72 |
+---------------+-----------+----------------------------------+

tomas@labserver:~$ swift post staticweb -r '.r:*,.rlistings'
/usr/local/lib/python2.7/dist-packages/keystoneclient/service_catalog.py:198: UserWarning: Providing attr
without filter_value to get_urls() is deprecated as of the 1.7.0 release and may be removed in the 2.0.0
release. Either both should be provided or neither should be provided.
```

```
  'Providing attr without filter_value to get_urls() is '
```

Podobně jako v předchozí kapitolce se prohlížečem svého počítač připojte na URL kontejneru (IP adresa vašeho OpenStack Swift a jeho portu, potom v1 a pak account a jméno kontejneru), tedy v mém případě

http://16.21.188.203:8080/v1/AUTH_d20fb4c9c0d645b4962484390a3be701/staticweb/

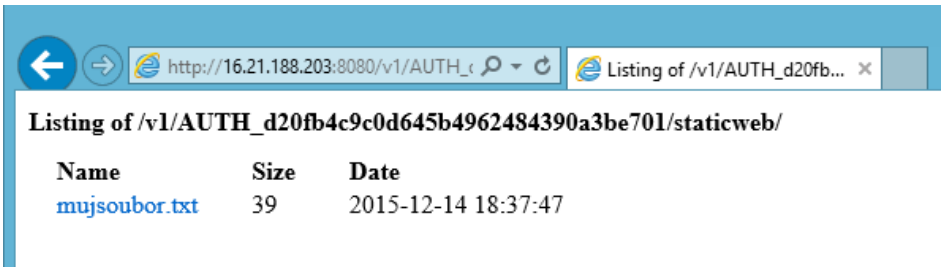
Co uvidíme? Výpis našich objektů, ale pouze jako text.



Pojďme teď říci Swiftu, že pro tento kontejner má fungovat jako mini-web-server a na hlavní URL zobrazit HTML se seznamem všech objektů, tentokrát ale ve formě odkazů, takže si jednoduše můžete stáhnout (nebo přečíst, pokud jde o HTML objekty, poslechnout pokud jde o MP3 objekty apod.).

```
tomas@labserver:~$ openstack container set --property 'Web-Listings=true' staticweb
```

Refreshujte stránku v prohlížeči.



Graficky si můžeme stránku upravit použitím CSS. Vytvořme tedy jednoduchý CSS soubor, uploadujme ho do kontejneru a nastavme Swift tak, že jej použije jako kaskádový styl.

```
tomas@labserver:~$ echo "body {background-color: lightblue;}" > styl.css

tomas@labserver:~$ openstack object create staticweb styl.css
+----------+-----------+----------------------------------+
| object   | container | etag                             |
+----------+-----------+----------------------------------+
| styl.css | staticweb | dc63ed634268ac2e4c5aa19338aa3069 |
+----------+-----------+----------------------------------+

tomas@labserver:~$ openstack container set --property 'Web-Listings-CSS=styl.css' staticweb
```

Znovu obnovte stránku.

Místo výpisu obsahu můžete take zobrazit nějaký výchozí HTML soubor. Vytvořme jej, uploadněme a řekněme to Swiftu.

```
tomas@labserver:~$ echo "<H1>Muj staticky web</H1>" > index.html

tomas@labserver:~$ openstack object create staticweb index.html
+------------+-----------+----------------------------------+
| object     | container | etag                             |
+------------+-----------+----------------------------------+
| index.html | staticweb | 542a2a044483c3b834bd25a28d9768de |
+------------+-----------+----------------------------------+

tomas@labserver:~$ openstack container set --property 'Web-Index=index.html' staticweb
```

4.10. Vnořování objektů do stránek na jiném web serveru

Vaše webová aplikace může přímo referencovat objekty ve Swiftu. Jednou z možností například je vytvořit si kontejner typu public, takže jeho dokumenty jsou běžně dostupné. Pokud by šlo o download nějakých souborů "na kliknutí" (něco ve smyslu "diskový obraz si stáhnete ZDE"), nemusíte podnikat nic dalšího, bude to fungovat. Pakliže by ovšem šlo o nějaké objekty vnořené do stránky (obrázek, video), browser může ukazovat varování o cross-origin. V rámci Swift můžete kontejneru nastavit CORS, což je standardizované HTML řešení, jak korektně cross-origin řešit. Nastavení se provádí na úrovni kontejneru, kde zadáte stránku, která může přistupovat (mimochod lze použít i wildcard a povolit všechny).

```
tomas@labserver:~$ openstack container set --property Access-Control-Allow-Origin=http://mojesite.demo
dalsiKontejner
```

Následujícím způsobem si můžeme z CURL otestovat práva při simulaci správného a špatného origin.

```
tomas@labserver:~$ curl -k -i -X OPTIONS -H "Origin: http://mojesite.demo" -H "Access-Control-Request-Method:
GET" http://16.21.188.203:8080/v1/AUTH_d20fb4c9c0d645b4962484390a3be701/dalsiKontejner/mujsoubor.txt
HTTP/1.1 200 OK
access-control-allow-origin: http://mojesite.demo
access-control-allow-methods: HEAD, GET, PUT, POST, COPY, OPTIONS, DELETE
Allow: HEAD, GET, PUT, POST, COPY, OPTIONS, DELETE
Content-Length: 0
X-Trans-Id: tx4791ce90e5c84d2699a01-00566f0b07
Date: Mon, 14 Dec 2015 18:31:35 GMT

tomas@labserver:~$ curl -k -i -X OPTIONS -H "Origin: http://jinasite.demo" -H "Access-Control-Request-Method:
GET" http://16.21.188.203:8080/v1/AUTH_d20fb4c9c0d645b4962484390a3be701/dalsiKontejner/mujsoubor.txt
HTTP/1.1 401 Unauthorized
Content-Length: 131
Content-Type: text/html; charset=UTF-8
Allow: HEAD, GET, PUT, POST, COPY, OPTIONS, DELETE
WWW-Authenticate: Keystone uri='http://helion-cp1-vip-KEY-API-mgmt:5000'
X-Trans-Id: tx3b20fc7a386d4545b8994-00566f0b15
Date: Mon, 14 Dec 2015 18:31:49 GMT

<html><h1>Unauthorized</h1><p>This server could not verify that you are authorized to access the document you
requested.</p></html>
```

Takto tedy nemusíte ukládat velké soubory (typicky obrázky ve vysokém rozlišení, soubory, videa) prostředky vašeho webového serveru, ale odkážete se přímo na vaší instalaci Swift se všemi výhodami, které přináší – vysokou míru spolehlivosti díky replikaci, regionální politiky (replikace napříč planetou a upřednostnění blízkých nodů s objektem), přehled, multitenancy a velkou škálovatelnost.

Swift podporuje i možnost přímého uploadování objektů z webového formuláře klienta. Pakliže tedy máte webovou aplikaci, implementace nevyžaduje, že klientský browser nejprve uploaduje soubor na váš web server, který pak provede upload do Swift (což ovšem ale samozřejmě možné a může to mít i nějaké výhody). Místo toho lze povolit přímý kontakt Swift s klientem z HTML formuláře (tedy data neprotečou přes webový server a šetří jeho výkon).

Zabezpečeno je to trochu podobně jako u TempURL, tedy definujete parametry (tentokrát zahrnují třeba i maximální velikost objektu) a ty potom zašifrujete – tímto způsobem nelze řešení zneužít k neoprávněnému odesílání dat mimo vaší webovou aplikaci. Víc informací najdete v dokumentaci (http://docs.openstack.org/developer/swift/api/form_post_middleware.html)

4.11. Hromadné operace

OpenStack Swift umožňuje programátorům zefektivnit některé operace při uploadování a mazání objektů. První z funkcí je možnost uploadovat archiv ve formátu tar, gz bebo bz2 s tím, že Swift si archiv sám rozbalí a jeho soubory uloží v požadovaném kontejneru. Dále je k dispozici API, které v jednou požadavku na smazání objektu umožňuje jich uvést až 10 000, což má pozitivní vliv na výkon v případě nutnosti smazat velké množství objektů najednou.

Protože je tato funkce aktuálně zaměřena na vývojáře používající API, nebudeme ji v našem labu zkoušet (nicméně funguje dobře, na detaily se kliedně zeptejte nebo prostudujte dokumentaci).

4.12. Velké soubory

Swift je ve výchozím stavu omezen na maximální velikost objektu 5GB. Hlavním cílem je zabránit nerovnoměrnému rozložení dat v systému. Hodnotu můžete zvětšit, ale Swift má možnost rozdělení vetších objektů na menší kousky. Funguje to tak, že při uploadování si soubor rozdělíte na bloky o velikosti do 5GB, ale soubor uložíte tak, že využijete funkce Static Large Objects. Ta umožní, že v jednom kontejneru máte segmenty, ale v jiném je virtuální reprezentace celého objektu. Při jeho stahování tedy nemusí uživatelé dělat nic dalšího, jednoduše stahují (ukládání tedy může vyžadovat nějakou péči, alse stahování ne.

Klasické administrátorské GUI toto neumožňuje, nicméně existují tři příjemné možnosti:

- Použijete API například v situaci, kdy objekt je vytvářen jinou aplikací (pro programátory je to velmi snadné) a ne přímo uživatelem (ten ho případně pouze stahuje, což nevyžaduje nic dalšího)
- Použijete nadstavbové GUI, které to udělá za vás (uvidíte v další části labu)
- Použijte klasické Swift CLI, které to umožňuje

Právě CLI si teď vyzkoušíme, ale abychom šetřili zdroje v labu, uděláme segmenty podstatně menší.

Vytvořte soubor o velikosti 5 MB.

```
tomas@labserver:~$ dd if=/dev/zero of=vetsisoubor  bs=5M  count=1
1+0 records in
1+0 records out
5242880 bytes (5.2 MB) copied, 0.0128545 s, 408 MB/s
```

Uploadujte do kontejneru s využitím klasického Swift CLI klienta a specifikujte použití segmentace (SLO) a velikost segment dejte na 1 MB.

```
tomas@labserver:~$ swift upload mujKontejner vetsisoubor --use-slo --segment-size 1048576
/usr/local/lib/python2.7/dist-packages/keystoneclient/service_catalog.py:198: UserWarning: Providing attr
without filter_value to get_urls() is deprecated as of the 1.7.0 release and may be removed in the 2.0.0
release. Either both should be provided or neither should be provided.
  'Providing attr without filter_value to get_urls() is '
vetsisoubor segment 3
vetsisoubor segment 2
vetsisoubor segment 4
vetsisoubor segment 1
vetsisoubor segment 0
vetsisoubor
```

Zkuste si object stáhnout - z CLI nebo GUI, nic zvláštního to po vás chtít nebude.

```
tomas@labserver:~$ openstack object save mujKontejner vetsisoubor
```

Podívejme na seznam kontejnerů - najdete něco nového?

```
tomas@labserver:~$ openstack container list
+----------------------+
| Name                 |
+----------------------+
```

```
| aktualni              |
| archiv                |
| dalsiKontejner        |
| docasne               |
| mujKontejner          |
| mujKontejner_segments |
| staticweb             |
+-----------------------+
```

Byl vytvořen kontejner se segmenty pro mujKontejner (výchozí jméno je možné při uploadování zvolit jiné). Podívejme co obsahuje ten originální a ten se segmenty.

```
tomas@labserver:~$ openstack object list mujKontejner
+---------------------------+
| Name                      |
+---------------------------+
| infrastructure as code.pdf |
| vetsisoubor               |
+---------------------------+

tomas@labserver:~$ openstack object list mujKontejner_segments
+-----------------------------------------------------------+
| Name                                                      |
+-----------------------------------------------------------+
| vetsisoubor/slo/1450122096.583428/5242880/1048576/00000000 |
| vetsisoubor/slo/1450122096.583428/5242880/1048576/00000001 |
| vetsisoubor/slo/1450122096.583428/5242880/1048576/00000002 |
| vetsisoubor/slo/1450122096.583428/5242880/1048576/00000003 |
| vetsisoubor/slo/1450122096.583428/5242880/1048576/00000004 |
+-----------------------------------------------------------+
tomas@labserver:~$
```

4.13. Kvóty na počet objektů nebo velikost

Swift podporuje nastavení kvóty pro každý kontejner zvlášť co do počtu objektů nebo celkové velikosti kontejneru. Toto nastavení smí provádět pouze uživatel v roli Swift administrátora a omezit tak možnosti jiným členům tenantu, kteří nemají taková práva. Velmi užitečné je to zejména v případě veřejných kontejnerů například s uploadem přes webový formulář ve vaší aplikaci nebo pro ochranu před nepovedenou aplikací, která by nějakou chybou zabrala veškerý prostor. Vyzkoušejme si to.

Vytořte nový kontejner.

```
tomas@labserver:~$ openstack container create omezeni
+---------------------------------------+-----------+------------------------------------+
| account                               | container | x-trans-id                         |
+---------------------------------------+-----------+------------------------------------+
| AUTH_d20fb4c9c0d645b4962484390a3be701 | omezeni   | tx69262081437246dca2ec0-00566f1fa5 |
+---------------------------------------+-----------+------------------------------------+
```

Nastavte kvótu na počet objektů (nebo Quota-Bytes na velikost kontejneru).

```
tomas@labserver:~$ openstack container set --property Quota-Count=2 omezeni
```

Vytvořte si tři soubory

```
tomas@labserver:~$ touch soubor1
tomas@labserver:~$ touch soubor2
tomas@labserver:~$ touch soubor3
```

Uploadujte první dva.

```
tomas@labserver:~$ openstack object create omezeni soubor1
+---------+-----------+----------------------------------+
| object  | container | etag                             |
+---------+-----------+----------------------------------+
| soubor1 | omezeni   | d41d8cd98f00b204e9800998ecf8427e |
+---------+-----------+----------------------------------+

tomas@labserver:~$ openstack object create omezeni soubor2
+---------+-----------+----------------------------------+
| object  | container | etag                             |
+---------+-----------+----------------------------------+
```

```
| soubor2 | omezeni    | d41d8cd98f00b204e9800998ecf8427e |
+---------+-----------+----------------------------------+
```

Teď bude nutné na chvilku počkat (typicky 60 vteřin). Swift je masivně škálovatelný, má obrovskou kapacitu a může se rozprostřít přes celou planetu. Z toho důvodu stojí na eventuální konzistenci (na rozdíl třeba od souborových systémů). Pokud byste uploadovali třetí objekt rychle, pravděpodobně se to podaří - informace o spotřebě se v clusteru replikuje nějakou dobu. Po chvilce zkuste nahrát třetí objekt.

```
tomas@labserver:~$ openstack object create omezeni soubor3
Request Entity Too Large (HTTP 413)
```

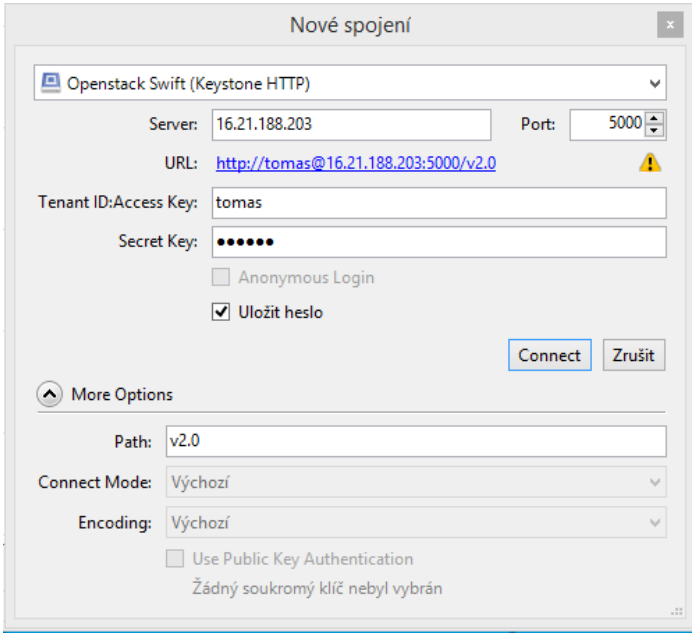
Pro běžné účely je takové chování dostatečné - odměnou za toleranci je pro vás masivní celoplanetární škálovatelnost současně s lokální dostupností, nízkou latencí a velkým výkonem. Tam, kde je nutné striktně dodržovat podobná pravidla doporučuji použít nějakou nadstavbu, která vynucení (a s ním spojené distribuované zámečky nebo centralizaci přístupů) provede o úroveň víš.

4.14. Klienstké GUI s open source CyberDuck

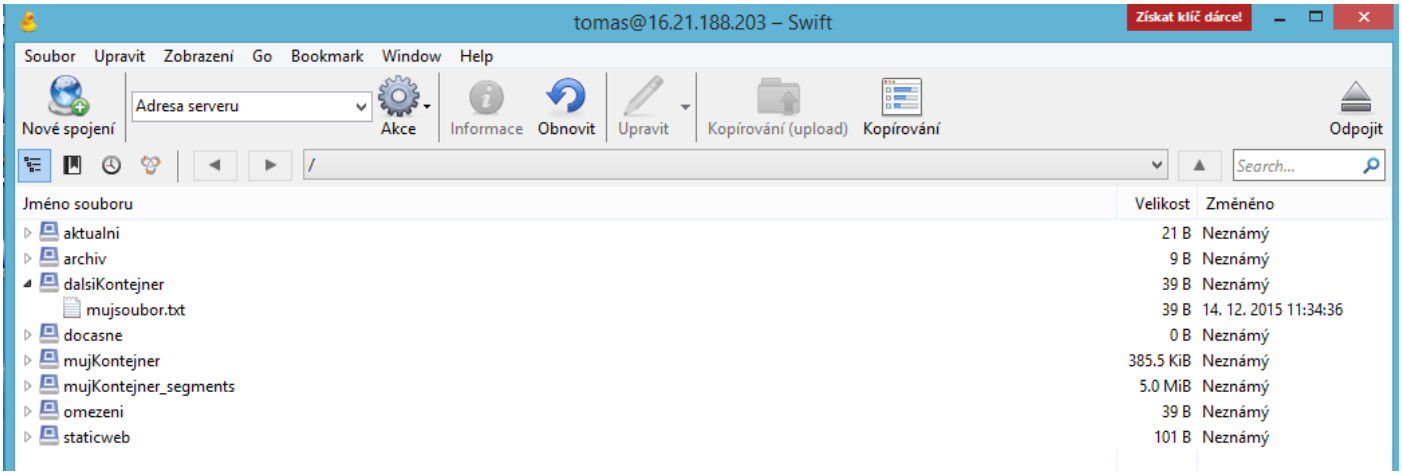
V rámci labu si můžete vyzkoušet nainstalovat klienta na váš počítač (a to dokonce s podporou češtiny). Jde o open source řešení podporující přístup do OpenStack Swift. Stáhněte si na https://cyberduck.io/

V našem labu je instalace OpenStack bez použití TLS šifrování a běžně dostupné profily CyberDuck po instalaci vyžadují HTTPS (což je pro produkci více než rozumné). Abyste mohli vyzkoušet řešení s HTTP stáhněte si nový profil na adrese https://trac.cyberduck.io/wiki/help/en/howto/openstack v části Connect without SSL.

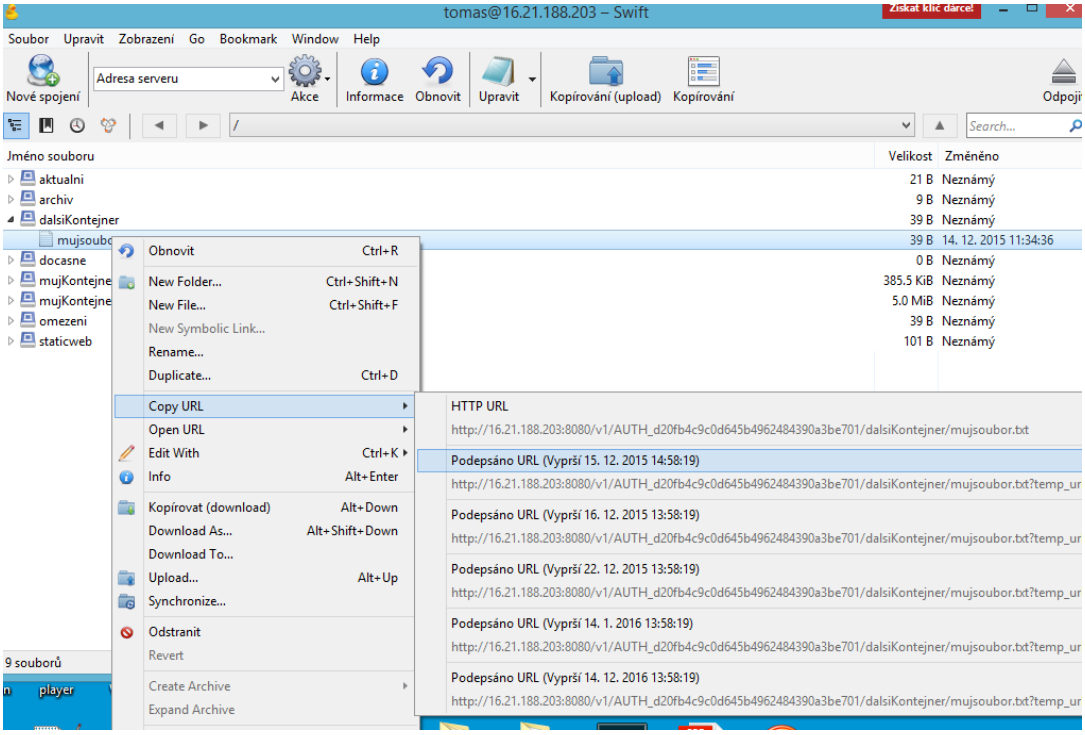
Vytvořte profil.



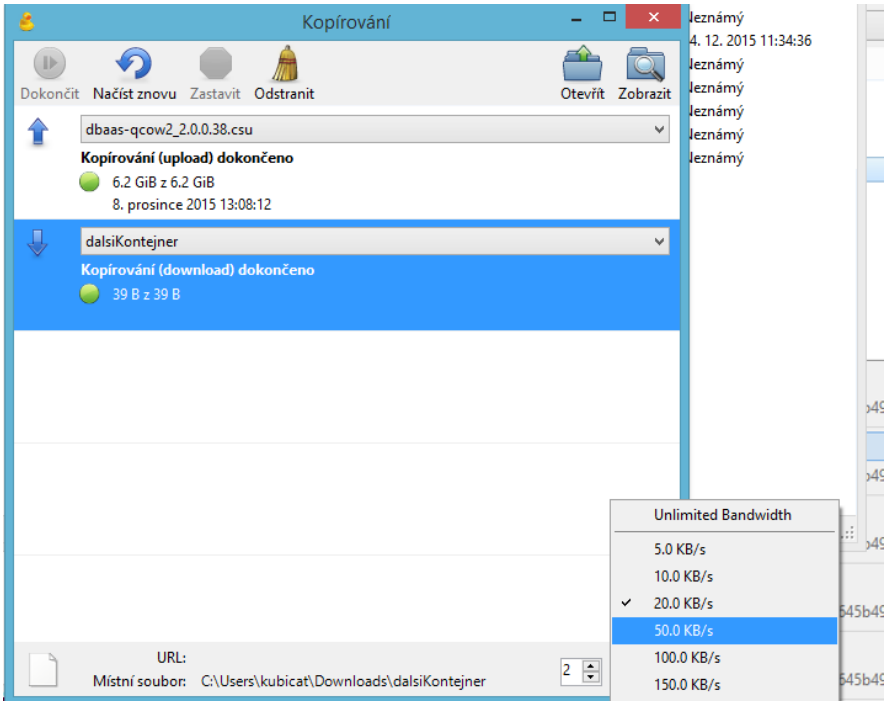
Po zadání tenantu (v labu má stejný název jako vaše přihlašovací jméno) se vám načtou vaše kontejnery.



Pokud máte u svého účtu nastaven TempURL klíč (viz příslušná kapitola) můžete například generovat dočasné URL pro přístup bez přihlášení.



Dále můžete synchronizovat adresář na vašem počítači, stahovat a uploadovat objekty (a to včetně podpory libovolných velikostí) a to velmi spolehlivě (s možností navázání spojení v případě přerušení). Dá se také omezit rychlost stahování.



4.15. Komerční nadstavby

Na trhu existují komerčně dostupné systémy pro enterprise sdílení souborů a jejich synchronizaci, které podporují OpenStack Swift jako svůj backend. Mezi ně patří:

- Gladinet
- Storage Made Simple
- Owncloud

4.16. Využití Swift ve vašich aplikacích a webech

Jak jsme viděli, Swift používá jednoduché RESTful API, s kterým se velmi dobře pracuje. Z libovolného programovacího jazyka je pak můžete volat přes jejich HTTP knihovny. Pokud to chcete ještě o něco jednoduší, existují knihovny (SDK) pro programovací jazyky:

- C (https://github.com/ukyg9e5r6k7gubiekd6)
- C++ (https://github.com/bshafiee/Swift_CPP_SDK)
- Go (https://launchpad.net/goose)
- Java (http://joss.javaswift.org/)
- Andriod (https://github.com/bluebossa63/OpenStackIntegration)
- Node.js (https://github.com/nodejitsu/pkgcloud)
- Python (http://docs.openstack.org/developer/swift/)
- .NET (https://wiki.openstack.org/wiki/OpenStack-SDK-DotNet)
- PHP (https://github.com/rackspace/php-opencloud)

5. Shrnutí a závěr

V prvním lab guide jsme si ukázali co Helion OpenStack dokáže, ale z automatizace dostanete ještě víc, když ji použijete na více frontách. V tomto lab guide jsme se pustili do náznaku některých složitějších nebo příbuzných témat, která jdou právě tímto směrem. Vyzkoušeli jsme si opustit grafické rozhranní a objevit příkazovou řádku a dokonce si naprogramovali aplikaci využívající API a SDK knihovny. Zaměřili jsme se také na automatizaci na úrovni konfigurace OS, aplikací a diskových obrazů. Velmi mocnou objektovou storage Swift, která je součástí Helion OpenStack, jsme podrobili detailnějšímu zkoumání a demonstrovali její vlastnosti.

Tady cesta nekončí. Uvedená témata je vhodné prohlubovat a současně můžete očekávat, že se průběžně v tomto lab guide objeví ještě další komplementární nástroje.

Připravujeme pro vás ještě nový lab guide zaměřený na Helion PaaS, kde již opustíme hranice IaaS a nahlédneme do světa vývoje moderních aplikací, kontejnerů, PaaS a Cloud Foundry, DBaaS a cloud-native konceptů. Vraťte se pro něj na cloudsvet.cz.

Tomáš Kubica, Enterprise architect

www.cloudsvet.cz

6. Další zdroje

Blog o novém IT a nejnovější verze těchto lab guide:

www.cloudsvet.cz

Helion dokumentace je webová a často aktulizovaná, ideální primární zdroj informací:

http://docs.hpcloud.com/

HP nadstavby jako je Cloud Service Automation, Server Automation nebo Executive Scorecard:

http://www.hpe.com/software

Open source projekty:

http://www.openstack.org/

http://cloudfoundry.org/