

# ProjectAtomic - Tutorial

12.11.2015

Mihai Csaky  
[SysOP Consulting](#)

## Introducere

În ultimul timp auzit tot mai des despre [Docker](#), aplicații în cloud, Linux pentru Cloud ([ProjectAtomic](#), [CoreOS](#)), Kubernetes. Dar, în afară de cei ce lucrează efectiv cu ele pentru firmele din IT, pentru utilizatorul de desktop, rămân o mare necunoscută.

## Scop

1. Acest tutorial se dorește o introducere în ProjectAtomic, cu un exemplu clar de utilizare. Exemplul poate fi rulat pe orice laptop/desktop cu minim 6-8GB RAM și un procesor modern care suportă extensiile de virtualizare.
2. Partea a doua a tutorialului arată cum putem transforma o aplicație clasică web, scrisă clar pentru utilizarea pe un singur calculator, fără posibilități prea mari de scalare, într-o aplicație care scalează în cloud, folosind Docker.

## Specificații

Avem nevoie de un calculator host, care să aibă liber 6-8GB RAM, procesor cu extensii de virtualizare. Exemplele au fost create pe un laptop ce rula Linux (fedora) și KVM cu virt-manager. Se poate folosi orice combinație de sistem de operare/soluție de virtualizare (VirtualBox etc), cu adaptările de rigoare.

Acces internet, de pe host și din rețeaua în care se află mașinile virtuale.

## AtomicProject

CentOS Atomic se poate descărca de la adresa de mai jos:

<https://wiki.centos.org/SpecialInterestGroup/Atomic/Download/>

Eu am folosit imaginea ISO, pentru a o instala folosind virt-manager, evident se pot folosi și alte soluții sau se poate lucra direct în cloud, pe [Amazon](#) sau [Google Cloud Platform](#). Pentru un data-center, atomic se poate instala direct pe bare-metal. Exemplele funcționează și direct pe CentOS 7 (nu este nevoie de versiunea Atomic) sau direct pe orice altă distribuție de linux, dar versiunile pentru Cloud sunt gândite pentru o instalare rapidă, simplă și minimală, cu update incremental și posibilități de roll-back. CoreOS este un alt bun candidat.

## Concepte

Un cluster de calculatoare cu AtomicOS folosește Kubernetes pentru a orchestra și distribui automat resursele între nodurile care fac parte din cluster. Principala resursă distribuită automat se numește POD și constă într-o colecție de containere Docker. Unul dintre calculatoare trebuie desemnat master (atomic-master în exemplul nostru), iar celelalte sunt noduri (node01, node02,...) pe care rulează containerele Docker grupate în

POD-uri. Pentru orchestrare se mai folosește etcd iar pentru asigurarea comunicației între containere vom folosi flanneld (flanneld creează o rețea virtuală suprapusă pe rețeaua docker, astfel încât două containere Docker aflate pe mașini virtuale diferite să poată comunica direct între ele).

*Exemplul nu abordează partea de securitate, best-practices, criptare, redundanță, partea de izolare, limitare și control al resurselor, monitorizare sau partea de portal pentru accesul clienților. Este o abordare entry-level, dar un nivel mai sus decât o simplă rulare a unei aplicații într-un container docker.*

## Etape

Configurația la care vrem să ajungem este un host master (atomic-master), care să coordoneze două noduri (node01, node02), iar un host (frontend) să asigure expunerea serviciilor web ale aplicației în rețea (intern sau internet). Nodurile vor fi coordonate de către master, dar pe frontend nu vom configura kubernetes, va rula doar containerele Docker necesare pentru load-balancing/acces din exterior.

Pe atomic-master vom rula și o bază de date MySQL, de aceea va avea alocat 2G de RAM.

În producție vom rula baza de date pe un host dedicat, sau o putem rula ca și pod în clusterul kubernetes. Fiecare soluție are avantaje/dezavantaje, sau depinde de providerul de cloud pe care-l folosim.

### I. Creare imagine de bază cu CentOS atomic

O imagine de bază, singurele configurări vor fi cele de la instalare și asigurarea accesului cu ssh, pentru ușurința configurărilor. Opțional, se poate configura un utilizator cu drepturi sudo, pentru a mări securitatea, dar partea de securitate nu face obiectul acestui tutorial, deci nu este recomandată utilizarea acestei configurații în producție.

Pași de urmat:

- descărcați centos atomic (versiunea preferată pentru tipul de instalare folosit)
- instalați și configurați sistemul de operare (un disk de 10-20GB este suficient), setați parola pentru root, creați opțional un utilizator pentru sudo (în exemplele mele, voi folosi direct acces root)
- configurați posibilitatea de acces remote, cu ssh, de preferință cu chei
- 

*de exemplu, dacă mașina virtuală are IP-ul 192.168.122.144, de pe host executați comanda:*

```
~$ ssh-copy-id root@192.168.122.144
```

## II. Configurare Kubernetes Master (host atomic-master)

Pentru atomic-master dorim un IP fix: 192.168.122.10 (libvirt folosește implicit 192.168.122.0/24 pentru mașinile virtuale, IP-ul se alege în funcție de soluția de virtualizare folosită).

*exemplu:*

```
$vi /etc/libvirt/qemu/networks/default.xml
<dhcp>
  <range start='192.168.122.100' end='192.168.122.254' />
  <host mac="52:54:00:e3:7b:90" name="atomic-master.cloud"
ip="192.168.122.10" />
</dhcp>
```

Registry Docker Local - pentru a păstra imaginile docker la dispoziția sistemului, vom crea un registru docker local, în care putem pune direct propriile imagini. Dacă o imagine nu este în registrul local, va fi descărcată din Docker HUB.

Pe atomic-master, rulăm:

```
$ mkdir -p /var/lib/local-registry
$ chcon -Rvt svirt_sandbox_file_t /var/lib/local-registry
```

Creăm imaginea docker:

```
$ docker create -p 5000:5000 \
-v /var/lib/local-registry:/srv/registry \
-e STANDALONE=false \
-e MIRROR_SOURCE=https://registry-1.docker.io \
-e MIRROR_SOURCE_INDEX=https://index.docker.io \
-e STORAGE_PATH=/srv/registry \
--name=local-registry registry
```

Ne asigurăm că imaginea Docker creată pornește automat la pornirea mașinii virtuale, folosind un fișier pentru systemd:

```
vi /etc/systemd/system/local-registry.service
```

cu conținutul:

```
[Unit]
Description=Local Docker Mirror registry cache
Requires=docker.service
After=docker.service
[Service]
Restart=on-failure
RestartSec=10
ExecStart=/usr/bin/docker start -a %p
ExecStop=-/usr/bin/docker stop -t 2 %p
[Install]
WantedBy=multi-user.target
```

serviciul mai trebuie activat și pornit:

```
$systemctl daemon-reload
$systemctl enable local-registry
$systemctl start local-registry
```

verificăm dacă totul este în ordine:

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	
03ca68d7bc4d	registry	"docker-registry"	38 seconds ago
Up 9 seconds	0.0.0.0:5000->5000/tcp	local-registry	

*În acest moment, ar trebui să avem o mașină virtuală, cu numele atomic-master, cu IP-ul 192.168.122.10, pe care rulează un registru Docker, expunând rețelei interne portul 5000/tcp.*

Configurăm [etcd](#)

etcd este un serviciu folosit de kubernetes, dar și de scripturile din containerele Docker. etcd poate fi instalat în cluster, accesul poate fi securizat. Noi vom folosi forma cea mai simplă, etcd instalat pe master, cu acces liber din rețeaua internă.

edităm `$vi /etc/etcd/etcd.conf`

și modificăm fișierul pentru ca etcd să asculte pe toate ip-urile, înlocuind localhost cu 0.0.0.0

Un mod mai simplu de a ne atinge scopul, este rularea comenzii:

```
$sed -i 's/localhost/0.0.0.0/g' /etc/etcd/etcd.conf
```

Configurare Kubernetes pe master  
edităm fișierele din /etc/kubernetes:

```
$vi /etc/kubernetes/config
# Comma seperated list of nodes in the etcd cluster
KUBE_ETCD_SERVERS="--etcd_servers=http://192.168.122.10:2379"
# How the replication controller and scheduler find the kube-apiserver
KUBE_MASTER="--master=http://192.168.122.10:8080"
```

```
$vi /etc/kubernetes/apiserver
# The address on the local server to listen to.
KUBE_API_ADDRESS="--address=0.0.0.0"
```

Ștergem ServiceAccount din parametrul KUBE\_ADMISSION\_CONTROL:

```
KUBE_ADMISSION_CONTROL="--admission_control=NamespaceLifecycle,NamespaceExists,LimitRanger,SecurityContextDeny,ResourceQuota"
```

activăm serviciile:

```
$systemctl enable etcd kube-apiserver kube-controller-manager
kube-scheduler
```

pornim serviciile:

```
$systemctl start etcd kube-apiserver kube-controller-manager
kube-scheduler
```

verificăm:

```
$ ps aux | grep kube
```

```
kube      2685 8.0 1.5 46972 28288 ?      Ssl 11:03 0:01
/usr/bin/kube-apiserver --logtostderr=true --v=0
--etcd_servers=http://127.0.0.1:2379 --address=0.0.0.0
--allow_privileged=false --service-cluster-ip-range=10.254.0.0/16
--admission_control=NamespaceLifecycle,NamespaceExists,LimitRanger,SecurityContextDeny,ResourceQuota
kube      2686 0.6 0.5 26944 10452 ?      Ssl 11:03 0:00
/usr/bin/kube-controller-manager --logtostderr=true --v=0
--master=http://192.168.122.10:8080
kube      2687 0.3 0.3 16220 7108 ?      Ssl 11:03 0:00
/usr/bin/kube-scheduler --logtostderr=true --v=0
--master=http://192.168.122.10:8080
```

În acest moment ar trebui să avem un host master complet configurat, care așteaptă conexiuni de la nodurile din cluster și instrucțiuni pentru a le coordona. În etapa următoare configurăm două noduri, prin clonarea imaginii inițiale a mașinii virtuale.

### III. Configurare Nod Kubernetes

Clonăm imaginile pentru noduri și le dăm câte un IP fix:

```
$ vi /etc/libvirt/qemu/networks/default.xml
<dhcp>
  <range start='192.168.122.100' end='192.168.122.254' />
  <host mac="52:54:00:e3:7b:90" name="atomic-master.cloud"
ip="192.168.122.10" />
  <host mac="52:54:00:e3:7b:92" name="frontend.cloud"
ip="192.168.122.20" />
  <host mac="52:54:00:91:ec:15" name="node01.cloud"
ip="192.168.122.101" />
  <host mac="52:54:00:e1:ef:92" name="node01.cloud" ip="192.168.122.102"
/>
</dhcp>
```

în exemplu am alocat un IP fix și pentru host-ul frontend: 192.168.122.20

### IV. Configurăm docker, kubernetes și flanneld pe fiecare nod

Docker trebuie să utilizeze docker-registry cache:

```
$vi /etc/sysconfig/docker
OPTIONS='--registry-mirror=http://192.168.122.10:5000 --selinux-enabled'
```

Flannel overlay:

```
$vi /etc/sysconfig/flanneld
# etcd url location. Point this to the server where etcd runs
FLANNEL_ETCD="http://192.168.122.10:2379"
# etcd config key. This is the configuration key that flannel queries
# For address range assignment
FLANNEL_ETCD_KEY="/atomic01/network"
```

activăm serviciul:

```
$systemctl enable flanneld
```

setăm numele hostname:

```
$vi /etc/hostname
kube01.cloud
```

configurăm nodul kubelet:

```
$vi /etc/kubernetes/config
# How the controller-manager, scheduler, and proxy find the apiserver
KUBE_MASTER="--master=http://192.168.122.10:8080"
$vi /etc/kubernetes/kubelet
# kubernetes kubelet (minion) config
# The address for the info server to serve on (set to 0.0.0.0 or "" for all
interfaces)
KUBELET_ADDRESS="--address=192.168.122.101"
# The port for the info server to serve on
# KUBELET_PORT="--port=10250"
# You may leave this blank to use the actual hostname
KUBELET_HOSTNAME="--hostname_override=192.168.122.101"
# location of the api-server
KUBELET_API_SERVER="--api_servers=http://192.168.122.10:8080"
# Add your own!
KUBELET_ARGS=""
```

activăm serviciile:

```
$systemctl enable flanneld kubelet kube-proxy
```

reboot:

```
$systemctl reboot
```



Repetăm pentru fiecare nod din rețea. Folosim doar două noduri, deci mai facem un host prin clonare, numit node02, cu setările respective.

Verificăm setările, pe atomic-master, nodurile trebuie să fie vizibile:

```
$ssh root@192.168.122.10
$kubectl get no
```

NAME	LABELS	STATUS
192.168.122.101	kubernetes.io/hostname=192.168.122.101	Ready
192.168.122.102	kubernetes.io/hostname=192.168.122.102	Ready

Pe fiecare nod, inclusiv frontend, trebuie să configurăm docker astfel încât să folosească flanneld:

```
$ mkdir -p /etc/systemd/system/docker.service.d/
$ vi /etc/systemd/system/docker.service.d/10-flanneld-network.conf
```

```
[Unit]
After=flanneld.service
Requires=flanneld.service
[Service]
EnvironmentFile=/run/flannel/subnet.env
ExecStartPre=-/usr/sbin/ip link del docker0
ExecStart=
ExecStart=/usr/bin/docker -d \
    --bip=${FLANNEL_SUBNET} \
    --mtu=${FLANNEL_MTU} \
    $OPTIONS \
    $DOCKER_STORAGE_OPTIONS \
    $DOCKER_NETWORK_OPTIONS \
    $INSECURE_REGISTRY
```

## V. Configurare frontend

Acest host va avea flanneld și docker configurat să-l folosească. Clonați imaginea originală, asigurați-vă că folosește IP-ul desemnat: 192.168.122.20

Docker trebuie să folosească docker-registry cache:

```
$vi /etc/sysconfig/docker
```

```
OPTIONS='--registry-mirror=http://192.168.122.10:5000 --selinux-enabled'
```

Flannel overlay:

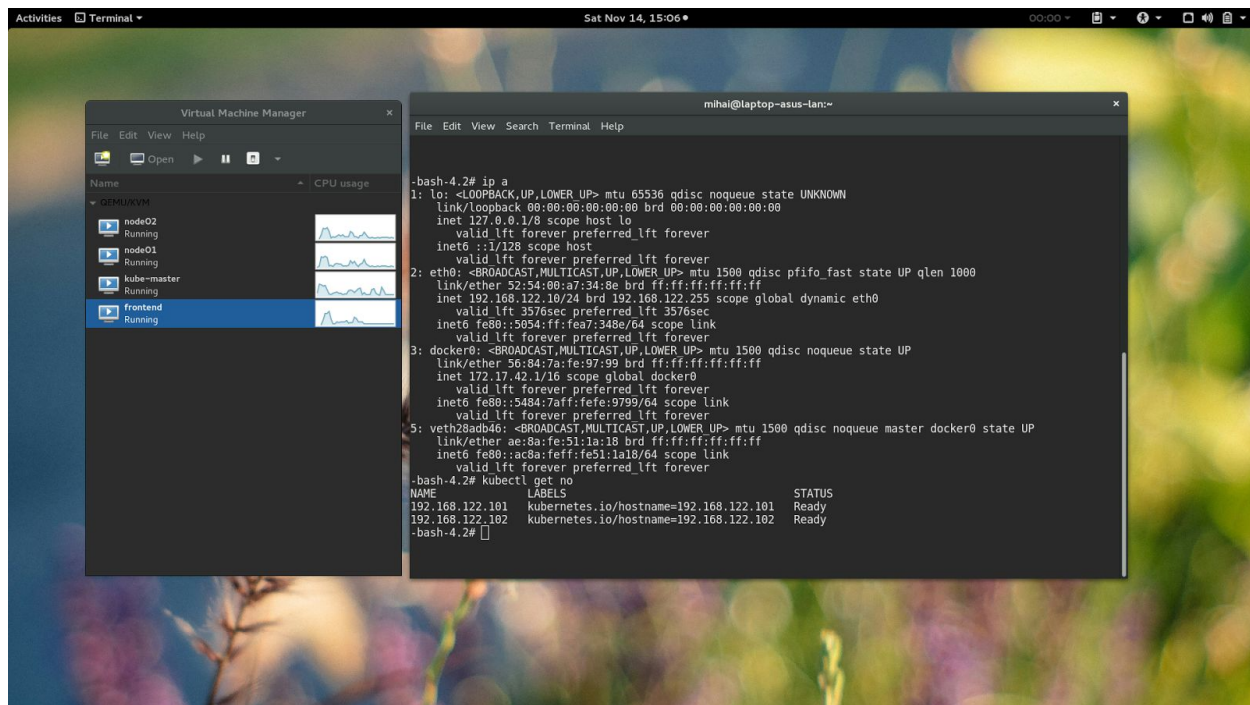
```
$vi /etc/sysconfig/flanneld
# etcd url location. Point this to the server where etcd runs
FLANNEL_ETCD="http://192.168.122.10:2379"
# etcd config key. This is the configuration key that flannel queries
# For address range assignment
FLANNEL_ETCD_KEY="/atomic01/network"
```

activați serviciul:

```
$systemctl enable flanneld
$ mkdir -p /etc/systemd/system/docker.service.d/
$ vi /etc/systemd/system/docker.service.d/10-flanneld-network.conf
[Unit]
After=flanneld.service
Requires=flanneld.service
[Service]
EnvironmentFile=/run/flannel/subnet.env
ExecStartPre=-/usr/sbin/ip link del docker0
ExecStart=
ExecStart=/usr/bin/docker -d \
    --bip=${FLANNEL_SUBNET} \
    --mtu=${FLANNEL_MTU} \
    $OPTIONS \
    $DOCKER_STORAGE_OPTIONS \
    $DOCKER_NETWORK_OPTIONS \
    $INSECURE_REGISTRY
```

*În acest punct trebuie să putem rula clusterul ce constă în 4 imagini virtuale, un host master (192.168.122.10), două noduri pentru pod-uri (192.168.122.101 și 192.168.122.102) și un host pentru expunerea aplicației către exterior (frontend, 192.168.122.20). Pe frontend nu configurăm kubernetes, dar configurăm flanneld pentru a permite comunicația între contaienrele Docker de pe frontend și cele de pe nodurile clusterului.*

## VI. Instalare Mysql (mariadb)



Clusterul nostru virtual arată astfel: un master (192.168.122.10), două noduri (192.168.122.101, 192.168.122.102) și un frontend (192.168.122.20).

Verificăm dacă masterul și nodurile sunt conectate, rulând pe master comanda:

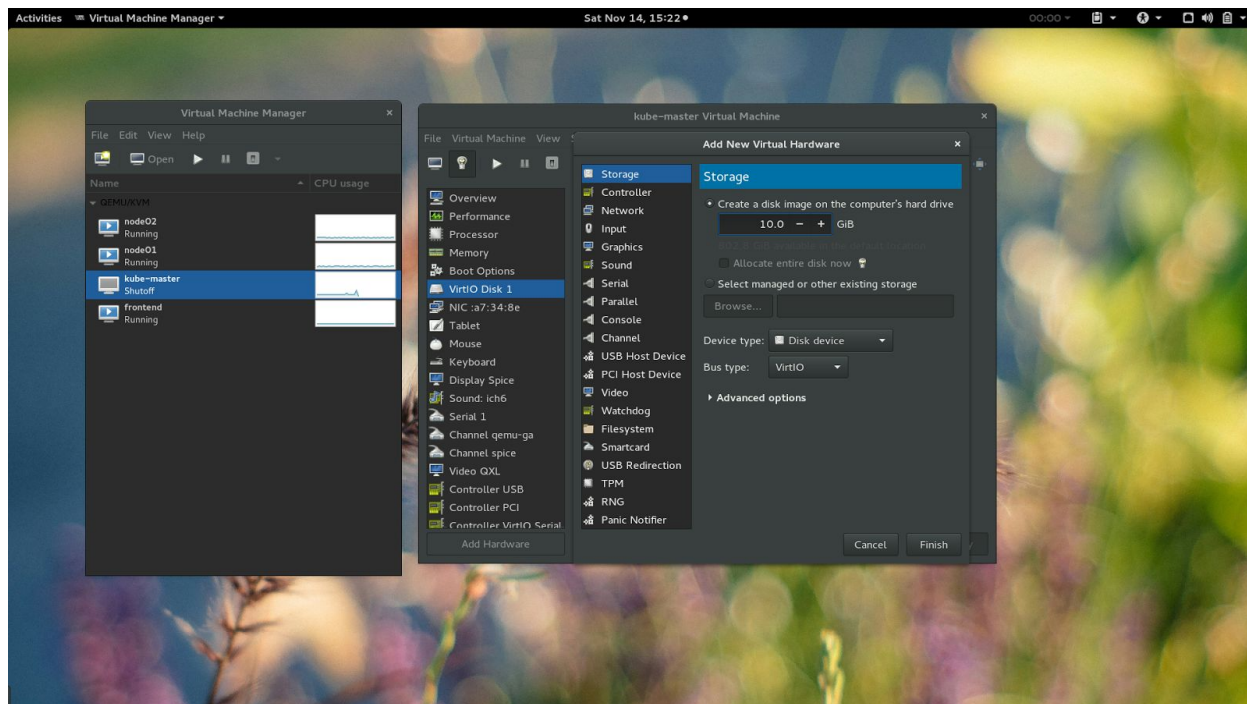
```
-bash-4.2# kubectl get no
```

NAME	LABELS	STATUS
192.168.122.101	kubernetes.io/hostname=192.168.122.101	Ready
192.168.122.102	kubernetes.io/hostname=192.168.122.102	Ready

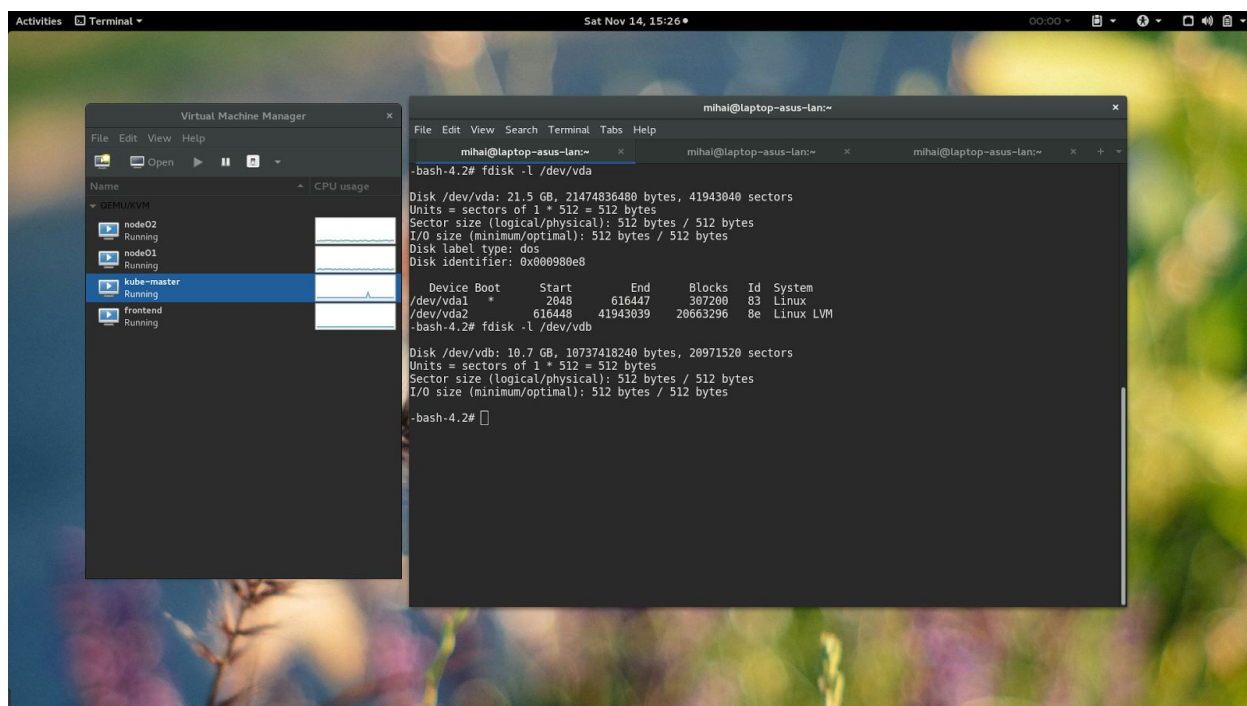
Avem nevoie de persistență, deci prima aplicație pe care o vom instala este o bază de date MySQL. Avem multe variante de instalare la dispoziție, de la un host complet separat de cluster până la o integrare totală, în care baza de date să fie controlată de kubernetes. Fiind doar un demo, am ales o soluție de compromis, baza de date va rula pe master, nu va fi controlată de kubernetes, dar va expune portul 3306 ca și serviciu disponibil aplicațiilor din cloud.

Pe master vom crea un disk suplimentar, de 10GB, unde vor fi stocate fișierele bazei de date. Fiindcă folosim Atomic ca și distribuție, nu putem instala aplicații noi pe host, MySQL

va rula ca și instanță docker.



În qemu, discul 2 se numește vdb:



Creăm o partiție primară și o formatăm XFS.

```
-bash-4.2# mkfs.xfs -L data /dev/vdb1
```

```
meta-data=/dev/vdb1      isize=256  agcount=4, agsize=655296 blks
                =                  sectsz=512  attr=2, projid32bit=1
                =                  crc=0      finobt=0
data        =                  bsize=4096  blocks=2621184, imaxpct=25
                =                  sunit=0    swidth=0 blks
naming      =version 2      bsize=4096  ascii-ci=0 ftype=0
log         =internal log   bsize=4096  blocks=2560, version=2
                =                  sectsz=512  sunit=0 blks, lazy-count=1
realtime    =none          extsz=4096  blocks=0, rtextents=0
```

Creăm un director /var/lib/mysql accesibil din containerele docker (de aceea modificăm contextul), și montăm permanent partiția nou creată în acest director:

```
-bash-4.2# mkdir /var/lib/mysql
```

```
-bash-4.2# vi /etc/fstab
```

```
/dev/vdb1  /var/lib/mysql      xfs  nobarrier,noatime,nodiratime 1 1
```

```
-bash-4.2# mount -a
```

verificăm:

```
-bash-4.2# df -h
```

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/mapper/cah-root	3.0G	957M	2.1G	32%	/
devtmpfs	904M	0	904M	0%	/dev
tmpfs	921M	0	921M	0%	/dev/shm
tmpfs	921M	424K	920M	1%	/run
tmpfs	921M	0	921M	0%	/sys/fs/cgroup
/dev/vda1	283M	70M	195M	27%	/boot
/dev/vdb1	10G	33M	10G	1%	/var/lib/mysql

```
-bash-4.2# chcon -Rvt svirt sandbox file t /var/lib/mysql/
```

changing security context of '/var/lib/mysql/'

Am folosit ultima variantă de MariaDB, a cărei documentație spune că se rulează astfel:

```
docker run --name some-mariadb -v /my/own/datadir:/var/lib/mysql -e
MYSQL_ROOT_PASSWORD=my-secret-pw -d mariadb:tag
```

Creăm o imagine docker:

```
-bash-4.2# docker create -p 3306:3306 -v /var/lib/mysql:/var/lib/mysql -e
MYSQL_ROOT_PASSWORD=zcGYLCqF --name=server-mariadb mariadb
Unable to find image 'mariadb:latest' locally
latest: Pulling from docker.io/mariadb
```

Imaginea este descărcată, dar nu rulează:

```

mihai@laptop-asus-lan:~$ docker create -p 3306:3306 -v /var/lib/mysql:/var/lib/mysql -e
MYSQL_ROOT_PASSWORD=zcGYLCqF --name=server-mariadb mariadb
bash: $: command not found
-bash-4.2# docker create -p 3306:3306 -v /var/lib/mysql:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=zcGYLCqF --name=server-mariadb mariadb
Unable to find image 'mariadb:latest' locally
latest: Pulling from docker.io/mariadb
ef2704e74ecc: Pull complete
1d6f63d023f5: Pull complete
f45839e43904: Pull complete
45534ee3011a: Pull complete
bfff60e4df2d: Pull complete
b401dff48567: Pull complete
f743137a1da7: Pull complete
f2e69ee6a6c6: Pull complete
2a5e39fe3417: Pull complete
e310f7cbd3e5: Pull complete
efalaa0d59b0: Pull complete
fclcffb5d70c: Pull complete
a7b1252b1908: Pull complete
b8e208c58634: Pull complete
5b3c17c1ef71: Pull complete
docker.io/mariadb:latest: The image you are pulling has been verified. Important: image verification is a tech preview feature and should not be relied on to provide security.
Digest: sha256:73693ccfb1477b1989968f7651bc7efc837a74d206077b0aea6d396c09f06e7
Status: Downloaded newer image for docker.io/mariadb:latest
7bcb2be87b1f93d7dce5789276694ffd7aac5ab6c56846cc5eedcf8368d7669f
-bash-4.2#
-bash-4.2# docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS                    NAMES
03ca68d7bc4d   registry   "docker-registry"       3 days ago    Up 28 minutes    0.0.0.0:5000->5000/tcp    local-registry
-bash-4.2# docker ps -a
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS                    NAMES
7bcb2be87b1f   mariadb    "/docker-entrypoint.    14 seconds ago    Up 29 minutes    0.0.0.0:5000->5000/tcp    server-mariadb
03ca68d7bc4d   registry   "docker-registry"       3 days ago    Up 29 minutes    0.0.0.0:5000->5000/tcp    local-registry
-bash-4.2#

```

Facem setările pentru a porni imaginea la pornirea calculatorului:

```
vi /etc/systemd/system/server-mariadb.service
```

```
[Unit]
Description=Local Docker MariaDB server
Requires=docker.service
After=docker.service
[Service]
Restart=on-failure
RestartSec=10
ExecStart=/usr/bin/docker start -a %p
ExecStop=-/usr/bin/docker stop -t 2 %p
```

[Install]

```
WantedBy=multi-user.target
```

serviciul mai trebuie activat și pornit:

```
$systemctl daemon-reload
$systemctl enable local-registry
$systemctl start local-registry
```

verificăm dacă totul este în ordine:

```
-bash-4.2# docker logs server-mariadb
```



```

mihai@laptop-asus-lan:~$
Version: '10.0.22-MariaDB-1-jessie' socket: '/var/run/mysqld/mysqld.sock' port: 0 mariadb.org binary distribution
Warning: Unable to load '/usr/share/zoneinfo/leap-seconds.list' as time zone. Skipping it.
151114 14:01:16 [Warning] 'proxies_priv' entry '@% root@7bcb2be87b1f' ignored in --skip-name-resolve mode.

/docker-entrypoint.sh: ignoring /docker-entrypoint-initdb.d/*

151114 14:01:16 [Note] mysqld: Normal shutdown
151114 14:01:16 [Note] Event Scheduler: Purging the queue, 0 events
151114 14:01:16 [Note] InnoDB: FTS optimize thread exiting.
151114 14:01:16 [Note] InnoDB: Starting shutdown...
151114 14:01:18 [Note] InnoDB: Shutdown completed; log sequence number 1616717
151114 14:01:18 [Note] mysqld: Shutdown complete

MySQL init process done. Ready for start up.

151114 14:01:18 [Note] mysqld (mysqld 10.0.22-MariaDB-1-jessie) starting as process 1 ...
151114 14:01:18 [Note] InnoDB: Using mutexes to ref count buffer pool pages
151114 14:01:18 [Note] InnoDB: The InnoDB memory heap is disabled
151114 14:01:18 [Note] InnoDB: Mutexes and rw locks use GCC atomic builtins
151114 14:01:18 [Note] InnoDB: Memory barrier is not used
151114 14:01:18 [Note] InnoDB: Compressed tables use zlib 1.2.8
151114 14:01:18 [Note] InnoDB: Using Linux native AIO
151114 14:01:18 [Note] InnoDB: Using CPU crc32 instructions
151114 14:01:18 [Note] InnoDB: Initializing buffer pool, size = 256.0M
151114 14:01:18 [Note] InnoDB: Completed initialization of buffer pool
151114 14:01:18 [Note] InnoDB: Highest supported file format is Barracuda.
151114 14:01:18 [Note] InnoDB: 128 rollback segment(s) are active.
151114 14:01:18 [Note] InnoDB: Waiting for purge to start
151114 14:01:18 [Note] InnoDB: Percona XtraDB (http://www.percona.com) 5.6.26-74.0 started; log sequence number 1616717
151114 14:01:18 [Note] Plugin 'FEEDBACK' is disabled.
151114 14:01:18 [Note] Server socket created on IP: '::'.
151114 14:01:18 [Warning] 'proxies_priv' entry '@% root@7bcb2be87b1f' ignored in --skip-name-resolve mode.
151114 14:01:18 [Note] mysqld: ready for connections.
Version: '10.0.22-MariaDB-1-jessie' socket: '/var/run/mysqld/mysqld.sock' port: 3306 mariadb.org binary distribution
-bash-4.2# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED            STATUS              PORTS                               NAMES
7bcb2be87b1f      mariadb            "/docker-entrypoint.   10 minutes ago    Up 2 minutes       0.0.0.0:3306->3306/tcp              server-mariadb
63cae6807bc4d     registry           "/docker-registry"     3 days ago        Up 39 minutes      0.0.0.0:5000->5000/tcp              local-registry
-bash-4.2#

```

Imaginea a pornit, logurile arată că totul este ok, iar directorul `/var/lib/mysql` este populat cu fișierele bazei de date.

Serviciul MySQL este disponibil și din rețeaua externă clusterului, putem să ne conectăm direct la `192.168.122.10` portul `3306`.



```

mihai@laptop-asus-lan:~$ mysql -h 192.168.122.10 -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 5
Server version: 10.0.22-MariaDB-1-jessie mariadb.org binary distribution

Copyright (c) 2000, 2015, Oracle, MariaDB Corporation Ab and others.

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

MariaDB [(none)]> use mysql;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [mysql]> pager less -SFX
PAGER set to 'less -SFX'
MariaDB [mysql]> select * from users;
ERROR 1146 (42S02): Table 'mysql.users' doesn't exist
MariaDB [mysql]> select * from user;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Host | User | Password | Select_priv | Insert_priv | Update_priv | Delete_priv | Create_priv | Drop_priv | Reload_priv | Shutdown_priv | Process_priv |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| % | root | *A092CEEEDAD52ABC79A85A986F983726CA4CFB6F | Y | Y | Y | Y | Y | Y | Y | Y | Y |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

MariaDB [mysql]>

```

## VII. Kubernetes și Mysql

Baza de date este gestionată în afara clusterului Kubernetes, dar dorim o integrare, astfel ca orice pod kubernetes să poată folosi baza de date. Pentru aceasta kubernetes are o facilitare interesantă numită servicii externe:

<http://kubernetes.io/v1.0/docs/user-guide/services.html#services-without-selectors>

Pe hostul master, creăm un fișier, mysql-service.yml

```
vi mysql-service.yml
```

cu următorul conținut:

```

apiVersion: v1
Kind: Service
metadata:
  name: mysql
labels:
  name: mysql
spec:
  ports:
  - port: 3306

```

Apoi rulăm comanda:

```
kubectl create -f mysql-service.yml
```

```
services/mysql
```

```
-bash-4.2# kubectl get svc
```

NAME	LABELS	SELECTOR	IP(S)	PORT(S)
kubernetes	component=apiserver,provider=kubernetes	<none>	10.254.0.1	443/TCP

```
mysql name=mysql
```

```

mihai@laptop-asus-lan:~$ kubectl create -f mysql-service.yml
services/mysql
mihai@laptop-asus-lan:~$ kubectl get svc
NAME          LABELS                                SELECTOR    IP(S)          PORT(S)
kubernetes    component=apiserver,provider=kubernetes <none>      10.254.0.1     443/TCP
mysql         name=mysql                            <none>      10.254.155.85  3306/TCP

```

Astfel am definit un serviciu în kubernetes, dar el nu poate fi încă folosit până nu îl conectăm la baza de date reală, de aceea vom defini și un endpoint:

```
-bash-4.2# vi mysql-endp.json
```

```
{
```

```
  "kind": "Endpoints",
```

```
  "apiVersion": "v1",
```

```

"metadata": {
  "name": "mysql"
},
"subsets": [
  {
    "addresses": [
      { "IP": "192.168.122.10" }
    ],
    "ports": [
      { "port": 3306 }
    ]
  }
]
}

```

```
-bash-4.2# kubectl create -f mysql-endp.json
```

```
endpoints/mysql
```

Acum baza de date mysql rulează în afara controlului clusterului kubernetes, dar este accesibilă din interiorul clusterului, pe un IP al rețelei definite la services: 10.254.155.85

```

mihai@laptop-asus-lan:~$ kubectl create -f mysql-service.yml
mihai@laptop-asus-lan:~$ kubectl get svc
NAME          LABELS                                SELECTOR          IP(S)              PORT(S)
kubernetes    component=apiserver,provider=kubernetes <none>            10.254.0.1         443/TCP
mysql         name=mysql                            <none>            10.254.155.85      3306/TCP
mihai@laptop-asus-lan:~$ kubectl create -f mysql-endp.json
mihai@laptop-asus-lan:~$ kubectl get endpoints/mysql
NAME          ENDPOINTS
kubernetes    192.168.122.10:6443
mysql         192.168.122.10:3306
mihai@laptop-asus-lan:~$ cat mysql-endp.json
{
  "kind": "Endpoints",
  "apiVersion": "v1",
  "metadata": {
    "name": "mysql"
  },
  "subsets": [
    {
      "addresses": [
        { "IP": "192.168.122.10" }
      ],
      "ports": [
        { "port": 3306 }
      ]
    }
  ]
}

```

Din curiozitate, putem rula comanda iptables-save pentru a vizualiza cum a realizat kubernetes accesul la aceste servicii setând reguli în firewall.

```

-bash-4.2# kubectl get endpoints
NAME      ENDPOINTS
kubernetes 192.168.122.18:6443
mysql      192.168.122.18:3306
-bash-4.2# iptables-save
# Generated by iptables-save v1.4.21 on Sat Nov 14 18:03:16 2015
*nat
:PREROUTING ACCEPT [1801:60060]
:INPUT ACCEPT [1801:60060]
:OUTPUT ACCEPT [5282:319412]
:POSTROUTING ACCEPT [5286:319652]
:DOCKER - [0:0]
-A PREROUTING -m addrtype --dst-type LOCAL -j DOCKER
-A OUTPUT ! -d 172.17.0.0/8 -m addrtype --dst-type LOCAL -j DOCKER
-A POSTROUTING -s 172.17.0.0/16 ! -o docker0 -j MASQUERADE
-A POSTROUTING -s 172.17.0.1/32 -d 172.17.0.1/32 -p tcp -m tcp --dport 5000 -j MASQUERADE
-A POSTROUTING -s 172.17.0.17/32 -d 172.17.0.17/32 -p tcp -m tcp --dport 3306 -j MASQUERADE
-A DOCKER ! -i docker0 -p tcp -m tcp --dport 5000 -j DNAT --to-destination 172.17.0.1:5000
-A DOCKER ! -i docker0 -p tcp -m tcp --dport 3306 -j DNAT --to-destination 172.17.0.17:3306
COMMIT
# Completed on Sat Nov 14 18:03:16 2015
# Generated by iptables-save v1.4.21 on Sat Nov 14 18:03:16 2015
*filter
:INPUT ACCEPT [573313:71152520]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [569633:73960697]
:DOCKER - [0:0]
-A FORWARD -o docker0 -j DOCKER
-A FORWARD -o docker0 -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -i docker0 ! -o docker0 -j ACCEPT
-A FORWARD -i docker0 -o docker0 -j ACCEPT
-A DOCKER -d 172.17.0.1/32 ! -i docker0 -o docker0 -p tcp -m tcp --dport 5000 -j ACCEPT
-A DOCKER -d 172.17.0.17/32 ! -i docker0 -o docker0 -p tcp -m tcp --dport 3306 -j ACCEPT
COMMIT
# Completed on Sat Nov 14 18:03:16 2015
-bash-4.2#

```

Din nefericire, în acest punct un reboot a dat sistemul peste cap, iar soluția (temporară), până aflu și cauza, a fost să dezactivez IPV6:

```
-bash-4.2# vi /etc/sysconfig/network
```

```
# Created by anaconda
```

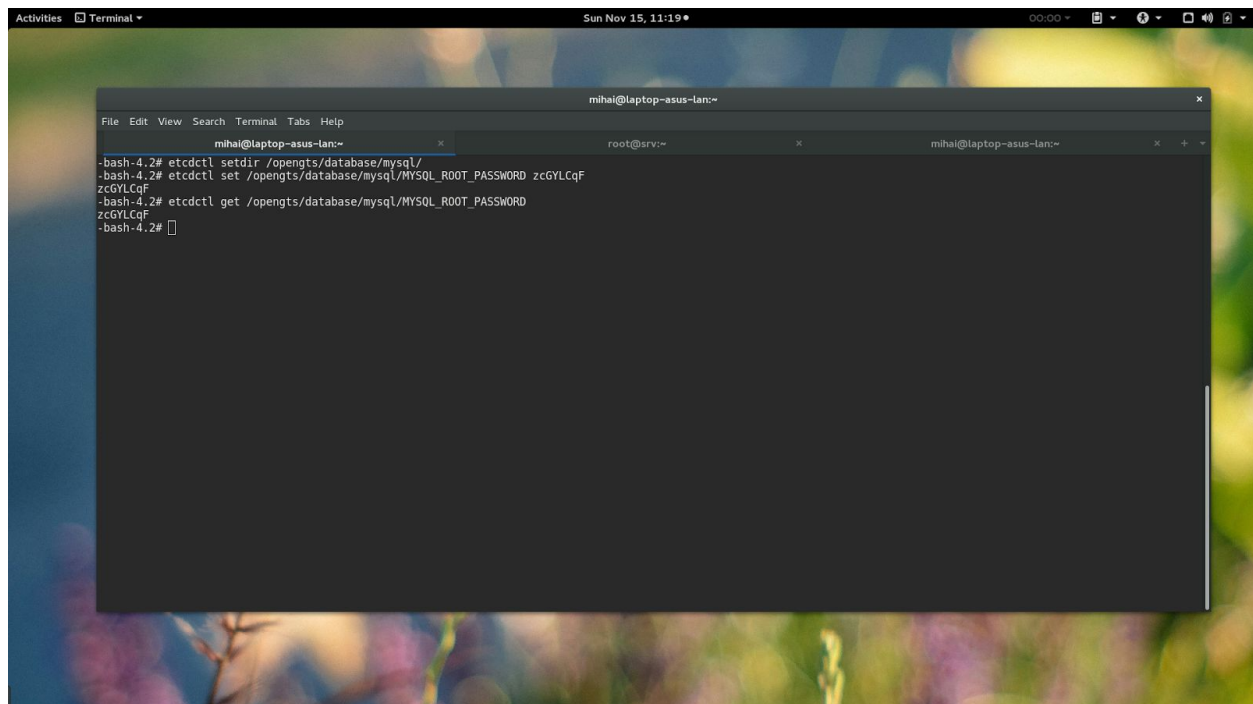
```
NETWORKING IPV6=no
```

```
IPV6INIT=no
```

## VIII. OpenGTS în cloud

Pentru a rula OpenGTS în cloud, am pregătit o imagine docker. Dar înainte de a crea resursele, ne folosim de etcd pentru face accesibilă parola bazei de date către imaginile docker. Parola MYSQL\_ROOT\_PASSWORD a fost aleasă aleator la crearea imaginii docker pentru serverul MySQL.

```
-bash-4.2# etcdctl setdir /opengts/database/mysql/
-bash-4.2# etcdctl set /opengts/database/mysql/MYSQL_ROOT_PASSWORD zcGYLCqF
zcGYLCqF
-bash-4.2# etcdctl get /opengts/database/mysql/MYSQL_ROOT_PASSWORD
zcGYLCqF
```



Imaginea docker pentru aplicația OpeGTS se află în registrul Docker, la adresa:

<https://hub.docker.com/r/mcsaky/opengts-cloud/>

și este generată automat din repo-ul github:

<https://github.com/mihaics/opengts-cloud>

de latfel, în github se află și fișierele de care avem nevoie, gts\_account.sh:

```
#!/bin/bash
etcdctl set /opengts/cartrack/MYSQL_DBNAME dbcartrack
etcdctl set /opengts/cartrack/MYSQL_DBUSER usrcartrack
```

```
etcdctl set /opengts/cartrack/MYSQL_DBPASSWORD $( < /dev/urandom tr -dc A-Z-a-z-0-9
| head -c${2:-10};echo;)
etcdctl set /opengts/cartrack/SYSADMIN_PASSWORD $( < /dev/urandom tr -dc
A-Z-a-z-0-9 | head -c${2:-10};echo;)
etcdctl set /opengts/cartrack/CREATE_DATABASE true

kubectl create -f opengts-rc-cartrack.yml
```

Acest script generează cheile care conțin parole (random), în etcd. Parolele vor fi citite de scripturile care inițializează aplicația, de aceea accesul la etcd trebuie securizat.

Ultima linie din script `kubectl create -f opengts-rc-cartrack.yml` generează pod-urile care rulează efectiv aplicația. Din acest punct, kubernetes se va ocupa de tot, va descărca imaginile necesare din docker hub, va distribui aplicația pe noduri și va crea serviciile necesare accesării aplicației. În funcție de resurse, va trebui să așteptăm poate și peste 10 minute, dar în final vom avea două instanțe ale aplicației configurate și rulând pe noduri. Conținutul fișierului `opengts-rc-cartrack.yml` este:

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: app-cartrack
  labels:
    name: app-cartrack
spec:
  replicas: 2
  selector:
    app: app-cartrack
  template:
    metadata:
      labels:
        app: app-cartrack
    spec:
      containers:
        - image: mcsaky/opengts-cloud
          name: app-cartrack
          env:
            - name: ETCD_SRV_ADDR
              value: 192.168.122.10
            - name: OPENGTS_CLIENT_ID
```

```
    value: cartrack
  ports:
    - containerPort: 8080
      name: http-cartrack
    - containerPort: 8009
      name: ajp-cartrack
  -----
apiVersion: v1
kind: Service
metadata:
  name: svc-cartrack
labels:
  name: svc-cartrack
spec:
  ports:
    - port: 8080
      targetPort: 8080
      protocol: TCP
      name: http-cartrack
    - port: 8009
      targetPort: 8009
      protocol: TCP
      name: ajp-cartrack
  selector:
    app: app-cartrack
  type: NodePort
```

și definește resursele necesare rulării aplicației.

rezultatele se pot consulta pe master:

```

-bash-4.2# kubectl get no
NAME                                LABELS                                STATUS
192.168.122.101                     kubernetes.io/hostname=192.168.122.101 Ready
192.168.122.102                     kubernetes.io/hostname=192.168.122.102 Ready
-bash-4.2# kubectl get po
NAME                                READY    STATUS    RESTARTS   AGE
app-cartrack-az668                 1/1      Running   0           3h
app-cartrack-t84rw                 1/1      Running   0           3h
-bash-4.2# kubectl get svc
NAME                                LABELS                                SELECTOR                                IP(S)                                PORT(S)
kubernetes                         component=apiserver,provider=kubernetes <none>                                10.254.0.1                           443/TCP
mysql                             name=mysql                            <none>                                10.254.155.85                       3306/TCP
svc.cartrack                       name=svc.cartrack                     app=app-cartrack                      10.254.176.87                       8080/TCP
-bash-4.2# kubectl describe po app-cartrack-az668
Name:                               app-cartrack-az668
Namespace:                           default
Image(s):                           mcsaky/opengts-cloud
Node:                               192.168.122.102/192.168.122.102
Labels:                             app=app-cartrack
Status:                             Running
Reason:
Message:
IP:                                 172.16.70.3
Replication Controllers:             app-cartrack (2/2 replicas created)
Containers:
  app-cartrack:
    Image:                           mcsaky/opengts-cloud
    State:                           Running
      Started:                       Sun, 15 Nov 2015 11:50:15 +0200
    Ready:                           True
    Restart Count:                   0
Conditions:
  Type                               Status
  Ready                              True
No events.
-bash-4.2#

```

## IX. Frontend

Aplicația noastră rulează în cluster, este distribuită automat și este accesibilă celorlalte aplicații în cluster, dar nu este accesibilă în exterior.

Pentru un cluster mic, cu puține aplicații, cum este acesta, am putea rezolva simplu accesul din exterior cu ajutorul unor reguli de port-forward în firewall. Aceasta nu este însă o soluție elegantă, am ales să folosesc un load-balancer cu nginx și generarea automată a configurației cu confd.

De ce generare automată: kubernetes ne garantează că aplicația noastră va rula, că va reporni automat dacă se întâmplă ceva, că vor rula întotdeauna câte instanțe avem nevoie, dar nu ne garantează un IP fix al pod-ului și nici pe ce nod va rula. confd va trebui să determine orice schimbare în cluster și să anunțe serverul proxy nginx de aceste schimbări.



