Introducció a Clúster, Cloud i DevOps

Remo Suppi Boldrito

PID_00239612

Universitat Oberta de Catalunya

Índex

Introducción

Els avenços en la tecnologia han portat, d'una banda, al desenvolupament de processadors més ràpids, amb més d'un element de còmput (nuclis o *cores*) en cadascun, de baix cost i amb suport a la virtualització maquinari i, d'una altra, al desenvolupament de xarxes altament eficients. Això, juntament amb el desenvolupament de sistemes operatius com GNU/Linux, ha afavorit un canvi radical en la utilització de sistemes de processadors de nuclis múltiples i altament interconnectats, en lloc d'un únic sistema d'alta velocitat (com els sistemes vectorials o els sistemes de processament simètric SMP). A més, aquests sistemes han tingut una corba de desplegament molt ràpida, ja que la relació preu/prestacions ha estat, i ho és cada dia més, molt favorable tant per al responsable dels sistemes TIC d'una organització com per als usuaris finals en diferents aspectes: prestacions, utilitat, fiabilitat, facilitat i eficiència. D'altra banda, les creixents necessitats de còmput (i d'emmagatzematge) s'han convertit en un element tractor d'aquesta tecnologia i el seu desenvolupament, vinculats a la provisió dinàmica de serveis, contractació d'infraestructura i utilització per ús -fins i tot en minuts- (i no per compra o per lloguer), la qual cosa ha generat una evolució total i important en la forma de dissenyar, gestionar i administrar aquests centres de còmput. No menys importants han estat els desenvolupaments, a més dels sistemes operatius, per a suportar totes aquestes tecnologies, en llenguatges de desenvolupament, API i entorns (*frameworks*) perquè els desenvolupadors puguin utilitzar la potencialitat de l'arquitectura subjacent per al desenvolupament de les seves aplicacions, perquè els administradors i gestors puguin desplegar, oferir i gestionar serveis contractats i valorats per minuts d'ús o bytes d'E/S, per exemple, i perquè els usuaris finals puguin fer un ús ràpid i eficient dels sistemes de còmput sense preocupar-se gens d'allò que existeix per sota, on està situat i com s'executa.

Per aquest motiu, en aquest apartat, es desenvolupen tres aspectes bàsics que són part de la mateixa idea, en diferents aspectes, quan es desitja oferir la infraestructura de còmput d'altes prestacions. O una plataforma com a servei o la seva utilització per un usuari final en el desenvolupament d'una aplicació, que permeti utilitzar totes aquestes infraestructures de còmput distribuïdes i d'altes prestacions. En sistemes de còmput d'altes prestacions (*High Performance Computing*-HPC), podem distingir dues grans configuracions:

1) Sistemes fortament acoblats (*tightly coupled systems*): són sistemes on la memòria és compartida per tots els processadors (*shared memory systems*) i la memòria de tots aquests "es veu" (per part del programador) com una única memòria.

2) Sistemes feblement acoblats (*loosely coupled systems*): no comparteixen memòria (cada processador posseeix la seva) i es comuniquen mitjançant missatges passats a través d'una xarxa (*message passing systems*).

En el primer cas, són coneguts com a *sistemes paral·lels de còmput* (*parallel processing system*) i en el segon, com a *sistemes distribuïts de còmput* (*distributed computing systems*).

En l'actualitat, la gran majoria de sistemes (bàsicament per la seva relació preu-prestacions) són del segon tipus i es coneixen com a clústers on els sistemes de còmput estan interconnectats per una xarxa de gran amplada de banda i tots treballen en estreta col·laboració, i l'usuari final els veu com un únic equip. És important indicar que cadascun dels sistemes que integra el clúster al seu torn pot tenir més d'un processador amb més d'un *core* en cadascun, per la qual cosa el programador haurà de tenir en compte aquestes característiques quan desenvolupi les seves aplicacions (llenguatge, memòria compartida/distribuïda, nivells de caché, etc.) i així podrà aprofitar tota la potencialitat de l'arquitectura agregada. El tipus més comú de clúster és l'anomenat Beowulf, que és un clúster implementat amb múltiples sistemes de còmput (generalment similars, però poden ser heterogenis) i interconnectats per una xarxa d'àrea local (generalment Ethernet, però hi ha xarxes més eficients com Infiniband o Myrinet). Alguns autors el denominen *MPP* (*massively parallel processing*) quan és un clúster, però disposen de xarxes especialitzades d'interconnexió (mentre que els clústers utilitzen maquinari estàndard a les seves xarxes) i estan formats per un alt nombre de recursos de còmput (1.000 processadors, no més). Avui dia, la majoria de sistemes publicats en el TOP500 (https://www.top500.org/statistics/list/) són clústers, i en l'última llista publicada (2016) ocupava el primer lloc l'ordinador Sunway TaihuLight (Xina) amb 10,6 milions de cores, 1.300 Terabytes de memòria (1,3 Petabytes) i un consum de 15,3 MW, i que pot executar 93.014 TFlops (és a dir, aproximadament 93.000.000.000.000.000 instruccions de punt flotant per segon).

L'altre concepte vinculat als desenvolupaments tecnològics esmentats a les noves formes d'entendre el món de les TIC en l'actualitat és el *cloud computing* (o informàtica/serveis en el núvol), que sorgeix com a concepte d'oferir serveis d'informàtica a través d'Internet en què l'usuari final no té coneixement d'on s'estan executant les seves aplicacions i tampoc necessita ser un expert per a contactar, pujar i executar les seves aplicacions en minuts. Els proveïdors d'aquest tipus de servei tenen recursos (generalment distribuïts a tot el món) i permeten que els seus usuaris contractin i gestionin aquests recursos en línia i sense la major intervenció i en forma gairebé automàtica, la qual cosa permet reduir els costos tenint avantatges (a més no s'ha d'instal·lar-mantenir la infraestructura física –ni l'obra civil) com la fiabilitat, flexibilitat, rapidesa en l'aprovisionament, facilitat d'ús, pagament per ús, contractació d'allò que es necessita, etc. Òbviament, també té els seus desavantatges, com són la depen-

dència d'un proveïdor i la centralització de les aplicacions/emmagatzematge de dades, servei vinculat a la disponibilitat d'accés a Internet, qüestions de seguretat, que les dades sensibles del negoci no resideixen en les instal·lacions de les empreses i poden generar riscos de sostracció/robatori d'informació, confiabilitat dels serveis prestats pel proveïdor (sempre és necessari signar una SLA –contracte de qualitat de servei), tecnologia susceptible al monopoli, serveis estàndard (només els oferts pel proveïdor), escalabilitat o privadesa.

Un tercer concepte vinculat a aquestes tecnologies, però probablement més del costat dels desenvolupadors d'aplicacions i *frameworks*, és el de DevOps. DevOps sorgeix de la unió de les paraules *Development* (desenvolupament) i *Operations* (operacions), i es refereix a una metodologia de desenvolupament de programari que se centra en la comunicació, col·laboració i integració entre desenvolupadors de programari i els professionals d'operacions/administradors a les tecnologies de la informació (TI). DevOps es presenta com una metodologia que dóna resposta a la relació entre el desenvolupament del programari i les operacions TI, tenint com a objectiu que els productes i serveis de programari desenvolupats per una entitat es puguin fer més eficientment, tinguin una alta qualitat i a més siguin segurs i fàcils de mantenir. El terme DevOps és relativament nou i va ser popularitzat mitjançant *DevOps Open Days* (Bèlgica, 2009), i com a metodologia de desenvolupament ha anat guanyant adeptes des de grans a petites companyies, per a fer més i millors productes i serveis i, sobretot, a causa de diferents factors amb una forta presència avui dia com l'ús dels processos i metodologies de desenvolupament àgil, necessitat d'una major taxa de versions, àmplia disponibilitat d'entorns virtualitzats/*cloud*, major automatització de centres de dades i augment de les eines de gestió de configuració.

En aquest mòdul, es veuran diferents formes de crear i programar un sistema de còmput distribuït (clúster/*cloud*), les eines i biblioteques més importants per a complir aquest objectiu i els conceptes i eines vinculat a les metodologies DevOps.

Objectius

En els materials didàctics d'aquest mòdul, trobareu els continguts i les eines procedimentals per a aconseguir els objectius següents:

1. Analitzar les diferents infraestructures i eines per al còmput d'altes prestacions (inclosa la virtualització) (HPC).
2. Configurar i instal·lar un clúster d'HPC i les eines de monitoratge corresponents.
3. Instal·lar i desenvolupar programes d'exemples en les principals API de programació: Posix Threads, OpenMPI, i OpenMP.
4. Instal·lar un clúster específic basat en una distribució *ad hoc* (Rocks).
5. Instal·lar una infraestructura per a prestar serveis en *cloud* (IasS).
6. Eines DevOps per a automatitzar un centre de dades i gestions de la configuració.

1. Clusterització

La història dels sistemes informàtics és molt recent (es pot dir que comença en la dècada dels seixanta). Al principi, eren sistemes grans, pesats, cars, de pocs usuaris experts, no accessibles i lents. En la dècada dels setanta, l'evolució va permetre millores substancials dutes a terme per tasques interactives (*interactive jobs*), temps compartit (*time sharing*), terminals i amb una considerable reducció de la grandària. La dècada dels vuitanta es caracteritza per un augment notable de les prestacions (fins avui dia) i una reducció de la grandària en els anomenats *microordinadors*. La seva evolució ha estat a través de les estacions de treball (*workstations*) i els avenços en xarxes (LAN de 10 Mb/s i WAN de 56 kB/s el 1973 a LAN d'1/10 Gb/s i WAN amb ATM, *asynchronous transfer mode* d'1,2 Gb/s en l'actualitat o xarxes d'alt rendiment com Infiniband -96Gb/s- o Myrinet -10Gb/s-), que és un factor fonamental en les aplicacions multimèdia actuals i d'un futur proper. Els sistemes distribuïts, per la seva banda, van començar la seva història en la dècada dels setanta (sistemes de 4 o 8 ordinadors) i el seu salt a la popularitat el van fer en la dècada dels noranta. Si bé la seva administració, instal·lació i manteniment poden tenir una certa complexitat (cada vegada menys) perquè continuen creixent en grandària, les raons bàsiques de la seva popularitat són l'increment de prestacions que presenten en aplicacions intrínsecament distribuïdes (aplicacions que per la seva naturalesa són distribuïdes), la informació compartida per un conjunt d'usuaris, la compartició de recursos, l'alta tolerància a les fallades i la possibilitat d'expansió incremental (capacitat d'agregar més nodes per a augmentar les prestacions i de manera incremental). Un altre aspecte molt important en l'actualitat és la possibilitat, en aquesta evolució, de la virtualització. Les arquitectures cada vegada més eficients, amb sistemes *multicores*, han permès que la virtualització de sistemes es transformi en una realitat amb tots els avantatges (i possibles desavantatges) que això comporta.

1.1. Virtualització

La virtualització és una tècnica que està basada en l'abstracció dels recursos d'un ordinador, anomenada *Hypervisor* o VMM (*Virtual Machine Monitor*) que crea una capa de separació entre el maquinari de la màquina física (*host*) i el sistema operatiu de la màquina virtual (*virtual machine*, *guest*), i és un mitjà per a crear una "versió virtual" d'un dispositiu o recurs, com un servidor, un dispositiu d'emmagatzematge, una xarxa o fins i tot un sistema operatiu, on es divideix el recurs en un o més entorns d'execució. Aquesta capa de programari (VMM) maneja, gestiona i administra els quatre recursos principals d'un ordinador (CPU, memòria, xarxa i emmagatzematge) i els reparteix de mane-

ra dinàmica entre totes les màquines virtuals definides en l'ordinador central. D'aquesta manera, ens permet tenir diversos ordinadors virtuals executant-se sobre el mateix ordinador físic.

La màquina virtual, en general, és un sistema operatiu complet que s'executa com si estigués instal·lat en una plataforma de maquinari autònoma.

Enllaç d'interès

Per a saber més sobre virtualització, podeu visitar http://en.wikipedia.org/wiki/Virtualization.
Es pot consultar una llista completa de programes de virtualització a http://en.wikipedia.org/wiki/Comparison_of_platform_virtual_machines.

1.1.1. Plataformes de virtualització

Els exemples més comuns de plataforma de virtualització amb llicències GPL o similars són Xen, KVM, Qemu (emulació), OpenVz, VirtualBox i Oracle VM (només el servidor); i entre les plataforma propietàries (algunes com *free to use*) VMware ESX/i, Virtual PC/Hyper-V, Parallels i Virtuozzo. Hi ha diferents taxonomies per a classificar la virtualització (i algunes plataformes poden executar-se en més d'una categoria), essent la més coneguda la següent:

1) Virtualització per HW: considerant com a *Full Virtualization* quan la màquina virtual simula tot el HW per a permetre a un sistema *guest* executar-se sense modificacions (sempre que estigui dissenyat per al mateix conjunt d'instruccions). Va ser la primera generació de VM per a processadors x86 i el que fa és una captura d'instruccions que accedeixen de manera prioritària a la CPU i les transforma en una crida a la VM perquè siguin emulades per programari. En aquest tipus de virtualització, poden executar-se Parallels, VirtualBox, Virtual Iron, Oracle VM, Virtual PC, Hyper-V, VMware per exemple.

2) Virtualització assistida per maquinari, on el maquinari dóna suport que facilita la construcció i treball de la VM i millora notablement les prestacions (a partir del 2005/6, Intel i AMD donen aquest suport com VT-x i AMD-V, respectivament). Els principals avantatges, respecte a altres formes de virtualització, és que no s'ha de tocar el sistema operatiu *guest* (com en paravirtualització) i s'obtenen millors prestacions, però com a contrapartida es necessita suport explícit en la CPU, la qual cosa no està disponible en tots els processadors x86/86_64. En aquest tipus de virtualització poden executar-se KVM, VMware Fusion, Hyper-V, Virtual PC, Xen, Parallels, Oracle VM, VirtualBox.

3) Paravirtualització: és una tècnica de virtualització en què la VM no necessàriament simula l'HW, sinó que presenta una API que pot ser utilitzada pel sistema *guest* (per la qual cosa, s'ha de tenir accés al codi font per a reemplaçar les crides d'un tipus per un altre). Aquest tipus de crides a aquesta API es denominen *hypercall* i Xen pot executar-se en aquesta forma.

4) Virtualització parcial: és el que es denomina *Address Space Virtualization*. La màquina virtual simula múltiples instàncies de l'entorn subjacent del maquinari (però no de tot), particularment l'*address space*. Aquest tipus de virtualit-

zació accepta compartir recursos i allotjar processos, però no permet instàncies separades de sistemes operatius *guest* i es troba en desús actualment.

5) Virtualització en un nivell del sistema operatiu: en aquest cas, la virtualització permet tenir múltiples instàncies aïllades i segures del servidor, totes executant-se sobre el mateix servidor físic i on el sistema operatiu *guest* coincidirà amb el sistema operatiu base del servidor (s'utilitza el mateix kernel). Plataformes dins d'aquest tipus de virtualització són FreeBSD jails (el pioner), OpenVZ, Linux-VServer, LXC, Virtuozzo.

La diferència entre instal·lar dos sistemes operatius i virtualitzar dos sistemes operatius és que en el primer cas tots els sistemes operatius que tinguem instal·lats funcionaran de la mateixa manera que si estiguessin instal·lats en diferents ordinadors, i necessitarem un gestor d'arrencada que en encendre l'ordinador ens permeti triar quin sistema operatiu volem utilitzar, però només en podrem tenir funcionant simultàniament un. En canvi, la virtualització permet executar moltes màquines virtuals amb els seus sistemes operatius i canviar de sistema operatiu com si es tractés de qualsevol altre programa; no obstant això, s'han de valorar molt bé les qüestions relacionades amb les prestacions, ja que si l'HW subjacent no és l'adequat, podrem notar moltes diferències en les prestacions entre el sistema operatiu instal·lat en base o el virtualitzat.

Entre els principals avantatges de la virtualització, hi ha:

1) Consolidació de servidors i millora de l'eficiència de la inversió en HW amb reutilització de la infraestructura existent.

2) Ràpid desplegament de nous serveis amb balanç dinàmic de càrrega i reducció dels sobredimensionaments de la infraestructura.

3) Increment d'*uptime* (temps que el sistema està al 100% en la prestació de serveis), increment de la tolerància a fallades (sempre que existeixi redundància física) i eliminació del temps de parada per a manteniment del sistema físic (migració de les màquines virtuals).

4) Manteniment a cost acceptable d'entorns de programari obsolets però necessaris per al negoci.

5) Facilitat de disseny i test de noves infraestructures i entorns de desenvolupament amb un baix impacte en els sistemes de producció i ràpida posada en marxa.

6) Millora de TCO (*Total Cost of Ownership*) i ROI (*Return on Investment*).

7) Menor consum d'energia que en servidors físics equivalents.

Com a desavantatges, podem esmentar:

1) Augmenta la probabilitat de fallades si no es considera redundància/alta disponibilitat (si es consoliden 10 servidors físics en un de potent equivalent, amb servidors virtualitzats, i deixen de funcionar tots els servidors en aquest, deixaran de prestar servei).

2) Rendiment inferior (possible) en funció de la tècnica de virtualització utilitzada i recursos disponibles.

3) Proliferació de serveis i màquines que incrementen les despeses d'administració/gestió (bàsicament, per l'efecte derivat de la facilitat de desplegament es tendeix a tenir-ne més dels necessaris).

4) Infraestructura desaprofitada (possible), ja que és habitual comprar una infraestructura major que la necessària en aquest moment per al possible creixement futur immediat.

5) Poden existir problemes de portabilitat, maquinari específic no suportat i compromís a llarg termini amb la infraestructura adquirida.

6) La presa de decisions en la selecció del sistema amfitrió pot ser complicada o condicionant.

Com és possible observar, els desavantatges es poden resoldre amb una planificació i presa de decisions adequada, i aquesta tecnologia és habitual en la prestació de serveis i totalment imprescindible en entorns de *cloud computing* (que veurem en aquest apartat també).

1.2. Beowulf

Beowulf [Rad, Beo, Beo1, Kur] és una arquitectura multiordenador que pot ser utilitzada per a aplicacions paral·leles/distribuïdes. El sistema consisteix bàsicament en un servidor i un o més clients connectats (generalment) a través d'Ethernet i sense la utilització de cap maquinari específic. Per a explotar aquesta capacitat de còmput, és necessari que els programadors tinguin un model de programació distribuït que, si bé és possible mitjançant UNIX (Sockets, RPC), pot implicar un esforç considerable, ja que són models de programació a nivell de *systems calls* i llenguatge C, per exemple; però aquesta forma de treball es pot considerar de baix nivell. Un model més avançat en aquesta línia de treball són els **Posix Threads**, que permeten explotar sistemes de memòria compartida i *multicores* de manera simple i fàcil. La capa de programari (interfície de programació d'aplicacions, API) aportada per sistemes com ***Parallel Virtual Machine*** (PVM) i ***Message Passing Interface*** (MPI) facilita notablement l'abstracció del sistema i permet programar aplicacions paral·leles/distribuïdes de manera senzilla i simple. Una de les formes bàsiques de treball és la de mestre-treballadors (*master-workers*), en què existeix un

servidor (mestre) que distribueix la tasca que faran els treballadors. En grans sistemes (per exemple, de 1.024 nodes) hi ha més d'un mestre i nodes dedicats a tasques especials, com per exemple entrada/sortida o monitoratge. Una altra opció no menys interessant, sobretot amb l'auge de processadors *multicores*, és **OpenMP**, que és una API per a la programació multiprocés de memòria compartida. Aquesta capa de programari permet afegir concurrència als programes sobre la base del model d'execució *fork-join* i es compon d'un conjunt de directives de compilador, rutines de biblioteca i variables d'entorn, que influencien el comportament en temps d'execució i proporcionen als programadors una interfície simple i flexible per al desenvolupament d'aplicacions paral·leles i arquitectures de CPU que puguin executar més d'un fil d'execució simultani (*hyperthreading*) o que disposin de més d'un nucli per processador accedint a la mateixa memòria compartida.

Hi ha dos conceptes que poden donar lloc a dubtes i que són Cluster Beowulf i COW (*Cluster of Workstations*). Una de les principals diferències és que Beowulf "es veu" com una única màquina on s'accedeix als nodes remotament, ja que no disposen de terminal (ni de teclat), mentre que un COW és una agrupació d'ordinadors que poden ser utilitzats tant pels usuaris del COW com per altres usuaris en forma interactiva, a través de la pantalla i el teclat. Cal considerar que Beowulf no és un programari que transforma el codi de l'usuari en distribuït ni afecta el nucli del sistema operatiu (com per exemple, Mosix). Simplement, és una forma d'agrupació (un clúster) de màquines que executen GNU/Linux i actuen com un superordinador. Òbviament, hi ha una gran quantitat d'eines que permeten obtenir una configuració més fàcil, biblioteques o modificacions al nucli per a obtenir millors prestacions, però és possible construir un clúster Beowulf a partir d'un GNU/Linux estàndard i de programari convencional. La construcció d'un clúster Beowulf de dos nodes, per exemple, es pot dur a terme simplement amb les dues màquines connectades per Ethernet mitjançant un concentrador (*hub*), una distribució de GNU/Linux estàndard (Debian), el sistema d'arxius compartit (NFS) i tenint habilitats els serveis de xarxa, com per exemple `ssh`. En aquestes condicions, es pot argumentar que es disposa d'un clúster simple de dos nodes. En canvi, en un COW no necessitem tenir coneixements sobre l'arquitectura subjacent (CPU, xarxa) necessària per a fer una aplicació distribuïda en Beowulf però sí que és necessari tenir un sistema operatiu (per exemple, Mosix) que permeti aquest tipus de compartició de recursos i distribució de tasques (a més d'una API específica per a programar l'aplicació que és necessària en els dos sistemes). El primer Beowulf es va construir el 1994 i el 1998 comencen a aparèixer sistemes Beowulf/Linux en les llistes del Top500 (Avalon en la posició 314 amb 68 *cores* i 19,3 Gflops, juny del 98).

1.2.1. Com es configuren els nodes?

Considerem que el servidor té dues interfícies de xarxa: *eth0* connectada a Internet i *eth1* connectada a una xarxa interna a la qual estan connectats els nodes. Primer de tot s'ha de modificar l'arxiu */etc/hosts* perquè contingui la

línia de `localhost` (amb 127.0.0.1) i la resta de noms a les IP internes dels restants nodes (aquest arxiu haurà de ser igual en tots els nodes). Per exemple:

```
    127.0.0.1 localhost
    192.168.0.254   srv.nteum.org srv
```

I afegir les IP dels nodes (i per a tots els nodes), per exemple:

```
    192.168.0.1 lucix1.nteum.org lucix1
    192.168.0.2 lucix2.nteum.org lucix2
...
```

S'ha de crear un usuari (vteum) en tots els nodes, crear un grup i afegir aquest usuari al grup:

```
    groupadd beowulf
    adduser vteum beowulf
    echo umask 007 >> /home/vteum/.bash_profile
```

Així, qualsevol arxiu creat per l'usuari vteum o qualsevol dins del grup serà modificable pel grup beowulf. S'ha de crear un servidor de NFS (i els altres nodes seran clients d'aquest NFS) i exportar el directori */home*, i així tots els clients veuran el directori *$HOME* dels usuaris. Per a això, sobre el servidor s'edita el */etc/exports* i s'agrega una línia com

```
/home 192.168.0.0/24*(rw,sync,no_root_esquaix)
```

Sobre cada node client es munta agregant en el */etc/fstab* perquè en l'arrencada el node munti el directori com a `192.168.0.254:/home /home nfs defaults 0 0`, també és aconsellable tenir en el servidor un directori */soft* (on s'instal·larà tot el programari perquè el vegin tots els nodes, i així ens evitem instal·lar aquest en cada node), i agregar-los a l'*export* com

```
/soft 192.16.0.0/24(rw,async,no_root_squash)
```

i en cada node agreguem en el *fstab*

```
192.168.0.254:/soft /soft nfs defaults 0 0
```

També haurem de fer la configuració de l'arxiu */etc/resolv.conf* i el `iptables` (per a fer un NAT) sobre el servidor (per exemple, amb la instrucció `iptables -t nat -A POSTORUTING -o eth0 -j MASQUERADE`) si volem que els nodes també hi tinguin accés (és molt útil sobretot a l'hora d'actualitzar el sistema operatiu).

A continuació s'ha de verificar que els serveis estan actius (amb amb l'ordre `systemctl status sshd; systemctl status nfs` o també amb la utilitat `chkconfig`, encara que a Debian per exemple no està disponible i s'haurà d'instal·lar). En cas que aquests serveis no estiguin habilitats haurem d'habilitar-los (per exemple, `sysmctl enable sshd`). El següent és poder-nos connectar des d'un determinat usuari a totes la màquines mitjançant *passwd* pel que utilitzarem una característica d'`ssh` que usa un mètode de clau pública-privada per a l'autentificació. Per a això verifiquem que estan habilitades a l'arxiu */etc/ssh/sshd_config* les següents línies (sense # davant):

RSAAuthentication yes AuthorizedKeysFile .ssh/authorized_keys Reiniciem el servei (`service sshd restart` o `systemctl restart ssh`) i ens connectem amb l'usuari que es desitja generar les claus (per exemple, *vteum*). A continuació executem `ssh-keygen -t rsa`, que crearà en *$HOME/.ssh* les claus *id_rsa* i *id_rsa.pub*, en referència a la clau privada i pública respectivament. Finalment executem `ssh-copy-id lucix1.nteum.org` per a copiar a el primer node la clau pública de l'usuari *vteum*. Com tindrem el directori *$HOME* muntat per a tots els nodes per NFS solament haurem de fer la còpia per al primer node i ja ens podrem connectar a els restants nodes, ja que la clau pública és la mateixa per tots ells. A continuació hem de comprovar que la connexió a *lucix1* des d'i cap a l'usuari *vteum* no requereix *passwd*.

És convenient també instal·lar NIS sobre el clúster, i així evitem haver de definir cada usuari en els nodes i un servidor DHCP per a definir tots els paràmetres de xarxa que cada node sol·liciti durant l'etapa de *boot*. Per a això, consulteu l'apartat de xarxa de l'assignatura Administració GNU/Linux, on s'expliquen amb detall aquestes configuracions.

A partir d'aquí, ja tenim un clúster Beowulf per a executar aplicacions amb interfície a MPI (com es veurà en els següents subapartats) per a aplicacions distribuïdes (també poden ser aplicacions amb interfície a PVM, però és recomanable MPI per qüestions d'eficiència i prestacions). Hi ha sobre diferents distribucions (Debian, FC incloses) l'aplicació `system-config-cluster`, que permet configurar un clúster sobre la base d'una eina gràfica o `clusterssh`, `parallel`, `pssh` o `dish`, que permeten gestionar i executar ordres en un conjunt de màquines de manera simultània (ordres útils quan necessitem fer operacions sobre tots els nodes del clúster, per exemple, apagar-los, reiniciar-los o fer còpies d'un arxiu a tots aquests).

Enllaç d'interès

Informació addicional: https://wiki.debian.org/highperformancecomputing

1.3. Beneficis del còmput distribuït

Quins són els beneficis del còmput en paral·lel? Veurem això amb un exemple [Rad]. Considerem un programa per a sumar nombres (per exemple, 4 + 5 + 6 + ...) anomenat `sumdis.c`:

```
#include <stdio.h>
#include <stdio.h>

int main (int argc, char** argv){
long inicial, final, resultado, tmp;
if (argc < 2) {
printf (" Uso: %s N° inicial N° final\n",argv[0]);
return (4); }
else {
inicial = atoll(argv[1]);
final = atoll(argv[2]);
resultado = 0;}
for (tmp = inicial; tmp <= final; tmp++){resultado += tmp; };
printf("%lu\n", resultado);
return 0;
            }
```

Ho compilem amb `gcc -o sumdis sumdis.c` i si mirem l'execució d'aquest programa, per exemple, amb

```
 time ./sumdis 1 1000000
500000500000

    real 0m0.005s
    user 0m0.000s
    sys     0m0.004s
```

es podrà observar que el temps en una màquina Debian sobre una màquina virtual (Virtualbox) amb processador i7 és de 5 mil·lèsimes de segon. Si, en canvi, fem des d'1 a 16 milions, el temps real puja fins a 0,050 s, és a dir, 10 vegades més, per la qual cosa, si es consideren 1.600 milions, el temps serà de prop de 4 segons.

La idea bàsica del còmput distribuït és repartir el treball. Així, si disposem d'un clúster de quatre màquines (lucix1–lucix4) amb un servidor i on l'arxiu executable es comparteix per NFS, és interessant dividir l'execució mitjançant ssh de manera que el primer sumi d'1 a 400.000.000, el segon, de 400.000.001 a 800.000.000, el tercer, de 800.000.001 a 1.200.000.000, i el quart, d'1.200.000.001 a 1.600.000.000. Les següents ordres mostren una possibilitat. Considerem que el sistema té el directori */home* compartit per NFS i l'usuari **adminp**, que té adequadament configurades les claus per a executar el codi sense *password* sobre els nodes, executarà:

```
    mkfifo out1     Crea una cua fifo en /home/adminp
    ./distr.sh & time cat out1 | awk '{total + = $1 } END {printf "%lf", total}'
```

S'executa l'ordre `distr.sh`; es recol·lecten els resultats i se sumen mentre es mesura el temps d'execució. El *shell script* `distr.sh` pot ser una cosa com:

```
ssh lucix1 /home/adminp/sumdis 1 400000000 > /home/adminp/out1 < /dev/null &
ssh lucix2 /home/adminp/sumdis 400000001 800000000 > /home/adminp/out1 < /dev/null &
ssh lucix3 /home/adminp/sumdis 800000001 1200000000 > /home/adminp/out1 < /dev/null &
ssh lucix4 /home/adminp/sumdis 1200000001 1600000000 > /home/adminp/out1 < /dev/null &
```

Podrem observar que el temps es redueix notablement (aproximadament en un valor proper a 4) i no exactament de manera lineal, però molt propera. Òbviament, aquest exemple és molt simple i només vàlid per a finalitats demostratives. Els programadors utilitzen biblioteques que els permeten portar a terme el temps d'execució, la creació i comunicació de processos en un sistema distribuït (per exemple MPI o OpenMP).

Una forma més eficient d'executar en forma paral·lela múltiples ordres és utilitzant `pssh`, `parallel` o `taktuk` (tots en el repositori de Debian). Per exemple, després d'instal·lat `parallel` executem:

```
parallel -S lucix1 -S lucix2 -S lucix3 --verbose ./sumdis 1 :::
10000 100000 100000000
```

En aquest cas, si mesurem el temps amb l'ordre `time` (anteposant-ho a l'ordre `parallel`) veurem que el temps d'execució és millor que en el cas anterior (li podem agregar igual que en l'ordre anterior l'awk perquè sumi tots els resultats també). l'ordre `parallel` admet una gran quantitat de paràmetres i opcions (igual que `pssh` i `taktuk`), que es poden consultar al tutorial [Parell].

1.3.1. Com cal programar per a aprofitar la concurrència?

Hi ha diverses maneres d'expressar la concurrència en un programa. Les tres més comunes són les següents:

1) Utilitzant fils (o processos) en el mateix processador (multiprogramació amb superposició del còmput i l'E/S).

2) Utilitzant fils (o processos) en sistemes *multicore*.

3) Utilitzant processos en diferents processadors que es comuniquen mitjançant missatges (MPS, *Message Passing System*).

Aquests mètodes poden ser implementats sobre diferents configuracions de maquinari (memòria compartida o missatges) i, si bé tots dos mètodes tenen els seus avantatges i desavantatges, els principals problemes de la memòria compartida són les limitacions en l'escalabilitat (ja que tots els *cores*/processadors utilitzen la mateixa memòria i el nombre d'aquests en el sistema està limitat per l'amplada de banda de la memòria) i, en els sistemes de pas de missatges, la latència i velocitat dels missatges a la xarxa. El programador haurà d'avaluar quin tipus de prestacions necessita, les característiques de l'aplicació subjacent i el problema que es desitja solucionar. No obstant això, amb els avenços de les tecnologies de *multicores* i de xarxa, aquests sistemes han crescut en popularitat (i en quantitat). Les API més comunes avui dia són Posix Threads i OpenMP per a memòria compartida i MPI (en les seves versions OpenMPI o Mpich) per a pas de missatges. Com hem esmentat anteriorment, hi ha una altra biblioteca molt difosa per a passos de missatges, anomenada PVM, però la versatilitat i prestacions que s'obtenen amb MPI l'ha deixat relegada a aplicacions petites o per a aprendre a programar en sistemes distribuïts. Aquestes biblioteques, a més, no limiten la possibilitat d'utilitzar fils (encara que en un nivell local) i tenir concurrència entre processament i entrada/sortida.

Per a portar a terme una aplicació paral·lela/distribuïda, es pot partir de la versió sèrie o mirar l'estructura física del problema i determinar quines parts poden ser concurrents (independents). Les parts concurrents seran candidates

a reescriure's com a codi paral·lel. A més, s'ha de considerar si és possible reemplaçar les funcions algebraiques per les seves versions paral·lelitzades (per exemple, ScaLapack *Scalable Linear Algebra Package* (es poden provar en Debian els diferents programes de prova que es troben en el paquet *scalapack-mpi-test*, per exemple i directament, instal·lar les biblioteques *libscalapack-mpi-dev*). També és convenient esbrinar si hi ha alguna aplicació similar paral·lela que pugui orientar-nos sobre la mode de construcció de l'aplicació paral·lel*.

*http://www.mpich.org/

Paral·lelitzar un programa no és una tasca fàcil, ja que s'ha de tenir en compte la **llei d'Amdahl**, que afirma que l'increment de velocitat (*speedup*) està limitat per la fracció de codi (f), que pot ser paral·lelitzat de la següent manera:

$$\text{speedup} = \frac{1}{1-f}$$

Aquesta llei implica que amb una aplicació seqüencial $f = 0$ i l'speedup = 1, mentre que amb tot el codi paral·lel $f = 1$ i l'speedup es fa infinit. Si considerem valors possibles, un 90% ($f = 0{,}9$) del codi paral·lel significa un speedup igual a 10, però amb $f = 0{,}99$ l'speedup és igual a 100. Aquesta limitació es pot evitar amb algorismes escalables i diferents models de programació d'aplicació (paradigmes):

1) Mestre-treballador: el mestre inicia a tots els treballadors i coordina el seu treball i el d'entrada/sortida.

2) *Single Process Multiple Data* (SPMD): el mateix programa que s'executa amb diferents conjunts de dades.

3) Funcional: diversos programes que porten a terme una funció diferent en l'aplicació.

En resum, podem concloure:

1) Proliferació de màquines multitasca (multiusuari) connectades per xarxa amb serveis distribuïts (NFS i NIS).

2) Són sistemes heterogenis amb sistemes operatius de tipus NOS (*Networked Operating System*), que ofereixen una sèrie de serveis distribuïts i remots.

3) La programació d'aplicacions distribuïdes es pot efectuar en diferents nivells:

a) Utilitzant un model client-servidor i programant a baix nivell (*sockets*) o fent servir memòria compartida a baix nivell (Posix Threads).

b) El mateix model, però amb API d'"alt" nivell (OpenMP, MPI).

c) Utilitzant altres models de programació com, per exemple, programació orientada a objectes distribuïts (RMI, CORBA, Agents, etc.).

1.4. Memòria compartida. Models de fils (*threading*)

Normalment, en una arquitectura client-servidor, els clients sol·liciten als servidors determinats serveis i esperen que aquests els contestin amb la major eficàcia possible. Per a sistemes distribuïts amb servidors amb una càrrega molt alta (per exemple, sistemes d'arxius de xarxa, bases de dades centralitzades o distribuïdes), el disseny del servidor es converteix en una qüestió crítica per a determinar el rendiment general del sistema distribuït. Un aspecte crucial en aquest sentit és trobar la manera òptima de manejar l'E/S, tenint en compte el tipus de servei que ofereix, el temps de resposta esperat i la càrrega de clients. No hi ha un disseny predeterminat per a cada servei, i escollir el correcte dependrà dels objectius i restriccions del servei i de les necessitats dels clients.

Les preguntes que hem de contestar abans de triar un determinat disseny són les següents: quant temps es triga en un procés de sol·licitud del client?, quantes d'aquestes sol·licituds és probable que arribin durant aquest temps?, quant temps pot esperar el client?, quant afecta aquesta càrrega del servidor les prestacions del sistema distribuït? A més, amb l'avenç de la tecnologia de processadors ens trobem que disposem de sistemes *multicore* (múltiples nuclis d'execució) que poden executar seccions de codi independents. Si es dissenyen els programes en forma de múltiples seqüències d'execució i el sistema operatiu ho suporta (i GNU/Linux és un d'aquests), l'execució dels programes es reduirà notablement i s'incrementaran en forma (gairebé) lineal les prestacions en funció dels *cores* de l'arquitectura.

1.4.1. Multifils (*multithreading*)

Segons les últimes tecnologies en programació per a aquest tipus d'aplicacions (i així ho demostra l'experiència), els dissenys més adequats són aquells que utilitzen models de multifils (*multithreading models*), en els quals el servidor té una organització interna de processos paral·lels o fils cooperants i concurrents.

> Un fil (*thread*) és una seqüència d'execució (fil d'execució) d'un programa, és a dir, diferents parts o rutines d'un programa que s'executen concurrentment en un únic processador i accediran a les dades compartides al mateix temps.

Quins avantatges aporta això respecte a un programa seqüencial? Considerem que un programa té tres rutines A, B i C. En un programa seqüencial, la rutina

C no s'executarà fins que s'hagin executat A i B. Si, en canvi, A, B i C són fils, les tres rutines s'executaran concurrentment i, si en aquestes hi ha E/S, tindrem concurrència d'execució amb E/S del mateix programa (procés), cosa que millorarà notablement les prestacions d'aquest programa. Generalment, els fils estan continguts dins d'un procés, i diferents fils d'un mateix procés poden compartir alguns recursos, mentre que diferents processos no. L'execució de múltiples fils en paral·lel necessita el suport del sistema operatiu i en els processadors moderns hi ha optimitzacions del processador per a suportar models multifil (*multithreading*) a més de les arquitectures *multicores* on hi ha múltiples nuclis i on cadascun pot executar un *thread*.

Generalment, hi ha quatre models de disseny per fils (en ordre de complexitat creixent):

1) Un fil i un client: en aquest cas, el servidor entra en un bucle sense fi, escoltant per un port i davant la petició d'un client s'executen els serveis en el mateix fil. Altres clients hauran d'esperar que acabi el primer. És fàcil d'implementar però només atén un client alhora.

2) Un fil i diversos clients amb selecció: en aquest cas, el servidor utilitza un sol fil, però pot acceptar múltiples clients i multiplexar el temps de CPU entre aquests. Es necessita una gestió més complexa dels punts de comunicació (*sockets*), però permet crear serveis més eficients, encara que presenta problemes quan els serveis necessiten una alta càrrega de CPU.

3) Un fil per client: és, probablement, el model més popular. El servidor espera per peticions i crea un fil de servei per a atendre cada nova petició dels clients. Això genera simplicitat en el servei i una alta disponibilitat, però el sistema no escala amb el nombre de clients i pot saturar el sistema molt ràpidament, ja que el temps de CPU dedicat davant una gran càrrega de clients es redueix notablement i la gestió del sistema operatiu pot ser molt complexa.

4) Servidor amb fils en granja (*worker threads*): aquest mètode és més complex però millora l'escalabilitat dels anteriors. Hi ha un nombre fix de fils treballadors (*workers*) als quals el fil principal distribueix el treball dels clients. El problema d'aquest mètode és l'elecció del nombre de treballadors: amb un nombre elevat, cauran les prestacions del sistema per saturació; amb un nombre massa baix, el servei serà deficient (els clients hauran d'esperar). Normalment, serà necessari sintonitzar l'aplicació per a treballar amb un determinat entorn distribuït.

Hi ha diferents formes d'expressar en un nivell de programació amb fils: paral·lelisme en un nivell de tasques o paral·lelisme a través de les dades. Triar el model adequat minimitza el temps necessari per a modificar, depurar i sintonitzar el codi. La solució a aquesta disjuntiva és descriure l'aplicació en termes de dos models basats en un treball en concret:

- Tasques paral·leles amb fils independents que poden atendre tasques independents de l'aplicació. Aquestes tasques independents seran encapsu-

lades en fils que s'executaran asincrònicament i s'hauran d'utilitzar biblioteques com Posix Threads (Linux/Unix) o Win32 Thread API (Windows), que han estat dissenyades per a suportar concurrència en un nivell de tasca.

- Model de dades paral·leles per a calcular llaços intensius; és a dir, la mateixa operació ha de repetir-se un nombre elevat de vegades (per exemple, comparar una paraula amb les paraules d'un diccionari). Per a aquest cas, és possible encarregar la tasca al compilador de l'aplicació o, si no és possible, que el programador descrigui el paral·lelisme utilitzant l'entorn OpenMP, que és una API que permet escriure aplicacions eficients sota aquest tipus de models.

Una aplicació d'informació personal (*Personal Information Manager*) és un bon exemple d'una aplicació que conté concurrència en un àmbit de tasques (per exemple, accés a la base de dades, llibreta d'adreces, calendari, etc.). Això podria ser en pseudocodi:

```
Function addressBook;
Function inBox;
Function calendar;
Program PIM     {
    CreateThread (addressBook);
    CreateThread (inBox);
    CreateThread (calendar);   }
```

Podem observar que hi ha tres execucions concurrents sense relació. Un altre exemple d'operacions amb paral·lelisme de dades podria ser un corrector d'ortografia, que en pseudocodi seria:

```
Function SpellCheck {loop (word = 1, words_in_file) compareToDictionary (word);}
```

S'ha de tenir en compte que tots dos models (fils paral·lels i dades paral·leles) poden existir en una mateixa aplicació. A continuació, es mostrarà el codi d'un productor de dades i un consumidor de dades basat en Posix Threads. Per a compilar sobre Linux, per exemple, s'ha d'utilitzar

```
gcc –o pc pc.c –lptread
```

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#define QUEUESIZE 10
#define LOOP 20

void *producer (void *args);
void *consumer (void *args);
typedef struct {  /* Estructura del buffer compartit i descriptors de threads */
int buf[QUEUESIZE]; long head, tail; int full, empty;
pthread_mutex_t *mut; pthread_cond_t *notFull, *notEmpty;
} queue;
queue *queueInit (void);  /* Prototip de funció: inicialització del buffer */
void queueDelete (queue *q);  /* Prototip de funció: esborrament del buffer*/
void queueAdd (queue *q, int in);  /* Prototip de funció: inserir element en el buffer */
```

```
void queueDel (queue *q, int *out);  /* Prototip de funció: treure element del buffer */

int main () {
    queue *fifo; pthread_t pro, con; fifo = queueInit ();
    if (fifo == NULL) { fprintf (stderr, " Error en crear buffer.\n"); exit (1); }
    pthread_create (&pro, NULL, producer, fifo);  /* Creació del thread productor */
    pthread_create (&con, NULL, consumer, fifo);  /* Creació del thread consumidor*/
    pthread_join (pro, NULL);  /* main () espera fins que s'acabin els dos threads */
    pthread_join (con, NULL);
    queueDelete (fifo);  /* Eliminació del buffer compartit */
    return 0; }  /* Fin */

void *producer (void *q) {  /*Funció del productor */
   queue *fifo; int i;
   fifo = (queue *)q;
   for (i = 0; i < LOOP; i++) {  /* Inserit en el buffer elements=LOOP*/
     pthread_mutex_lock (fifo->mut);  /* Semàfor per a entrar a inserir */
     while (fifo->full) {
      printf ("Productor: queue FULL.\n");
      pthread_cond_wait (fifo->notFull, fifo->mut); }
/* Bloqueig del productor si el buffer està ple, alliberant el semàfor mut */
/*perquè hi pugui entrar el consumidor. Continuarà quan el consumidor executi*/
     pthread_cond_signal (fifo->notFull);*/
     queueAdd (fifo, i);  /* Inserir element en el buffer */
     pthread_mutex_unlock (fifo->mut);  /* Allibero el semàfor */
     pthread_cond_signal (fifo->notEmpty); /*Desbloqueig consumidor si està bloquejat*/
     usleep (100000);  /* Dormo 100 mseg per a permetre que el consumidor s'activi */
     }
   return (NULL); }

void *consumer (void *q) {  /*Funció del consumidor */
   queue *fifo; int i, d;
   fifo = (queue *)q;
   for (i = 0; i < LOOP; i++) { /* Trec del buffer elements=LOOP*/
     pthread_mutex_lock (fifo->mut); /* Semàfor per a entrar a treure */
     while (fifo->empty) {
       printf (" Consumidor: queue EMPTY.\n");
       pthread_cond_wait (fifo->notEmpty, fifo->mut); }
/* Bloqueig del consumidor si el buffer està buit, alliberant el semàfor mut */
/*perquè pugui entrar el productor. Continuarà quan el consumidor executi */
     pthread_cond_signal (fifo->notEmpty);*/
     queueDel (fifo, &d); /* Trec element del buffer */
     pthread_mutex_unlock (fifo->mut); /* Allibero el semàfor */
     pthread_cond_signal (fifo->notFull); /*Desbloqueig productor si està bloquejat*/
     printf (" Consumidor: Rebut %d.\n", d);
     usleep(200000);/* Dormo 200 mseg per a permetre que el productor s'activi */
     }
   return (NULL); }

queue *queueInit (void) {
   queue *q;
   q = (queue *)malloc (sizeof (queue));  /* Creació del buffer */
   if (q == NULL) return (NULL);
   q->empty = 1; q->full = 0; q->head = 0; q->tail = 0;
   q->mut = (pthread_mutex_t *) malloc (sizeof (pthread_mutex_t));
   pthread_mutex_init (q->mut, NULL);  /* Creació del semàfor */
   q->notFull = (pthread_cond_t *) malloc (sizeof (pthread_cond_t));
   pthread_cond_init (q->notFull, NULL);  /* Creació de la variable condicional notFull*/
   q->notEmpty = (pthread_cond_t *) malloc (sizeof (pthread_cond_t));
   pthread_cond_init (q->notEmpty, NULL);  /* Creació de la variable condicional notEmpty*/
   return (q); }

void queueDelete (queue *q) {
   pthread_mutex_destroy (q->mut); free (q->mut);
 pthread_cond_destroy (q->notFull); free (q->notFull);
   pthread_cond_destroy (q->notEmpty); free (q->notEmpty);
   free (q); }

void queueAdd (queue *q, int in) {
   q->buf[q->tail] = in; q->tail++;
   if (q->tail == QUEUESIZE) q->tail = 0;
   if (q->tail == q->head) q->full = 1;
```

```
   q->empty = 0;
   return; }

void queueDel (queue *q, int *out){
   *out = q->buf[q->head]; q->head++;
    if (q->head == QUEUESIZE) q->head = 0;
    if (q->head == q->tail) q->empty = 1;
    q->full = 0;
    return; }
```

1.5. OpenMP

L'**OpenMP** (*Open-Multi Processing*) és una interfície de programació d'aplicacions (API) amb suport multiplataforma per a la programació en C/C++ i Fortran de processos amb ús de memòria compartida sobre plataformes Linux/Unix (i també Windows). Aquesta infraestructura es compon d'un conjunt de directives del compilador, rutines de la biblioteca i variables d'entorn que permeten aprofitar recursos compartits en memòria i en temps d'execució. Definit conjuntament per un grup dels principals fabricants de maquinari i programari, OpenMP permet utilitzar un model escalable i portàtil de programació, que proporciona als usuaris una interfície simple i flexible per al desenvolupament, sobre plataformes paral·leles, d'aplicacions d'escriptori fins a aplicacions d'altes prestacions sobre superordinadors. Una aplicació construïda amb el model híbrid de la programació paral·lela pot executar-se en un ordinador utilitzant tant OpenMP com Message Passing Interface (MPI) [Bla].

OpenMP és una implementació multifil, mitjançant la qual un fil mestre divideix la tasques sobre un conjunt de fils treballadors. Aquests fils s'executen de manera simultània i l'entorn d'execució porta a terme l'assignació d'aquests als diferents processadors de l'arquitectura. La secció del codi que està dissenyada per a funcionar en paral·lel està marcada amb una directiva de preprocessament que crearà els fils abans que la secció s'executi. Cada fil tindrà un identificador (`id`) que s'obtindrà a partir d'una funció (en C/C++ `omp_get_thread_num()`) i, després de l'execució paral·lela, els fils s'uniran de nou en la seva execució sobre el fil mestre, que continuarà amb l'execució del programa. Per defecte, cada fil executarà una secció paral·lela de codi independent però es poden declarar seccions de "treball compartit" per a dividir una tasca entre els fils, de manera que cada fil executi part del codi assignat. D'aquesta manera, és possible tenir en un programa OpenMP paral·lelisme de dades i paral·lelisme de tasques convivint.

Els principals elements d'OpenMP són les sentències per a la creació de fils, la distribució de càrrega de treball, la gestió de dades d'entorn, la sincronització de fils i les rutines en un nivell d'usuari. OpenMP usa en C/C++ les directives de preprocessament conegudes com a *pragma* per a diferents construccions (`#pragma omp <resta del pragma>`). Així, per exemple, `omp parallel` s'utilitza per a crear fils addicionals per a executar el treball indicat en la sentència paral·lela on el procés original és el fil mestre (`id=0`). El conegut programa que imprimeix "Hello, world" utilitzant OpenMP i multifils és*:

***Compileu amb** `gcc -fopenmp -o hello hello.c` **(en algunes distribucions -Debian està instal·lat per defecte- s'ha de tenir instal·lada la biblioteca GCC OpenMP (GOMP)** `apt-get install libgomp1`.

```
#include <stdio.h>
#include <omp.h>

int main(int argc, char* argv[]){
   #pragma omp parallel
   printf("Hello, world.\n");
   return 0;}
```

On executarà un *thread* per a cada *core* disponible en l'arquitectura. Per a especificar *work-sharing constructs*, s'utilitza:

- **omp for** o **omp do**: reparteix les iteracions d'un llaç en múltiples fils.
- **sections**: assigna blocs de codi independents consecutius a diferents fils.
- **single**: especifica que el bloc de codi serà executat per un sol fil amb una sincronització (*barrier*) al final.
- **master**: similar a `single`, però el codi del bloc serà executat pel fil mestre i no hi ha *barrier* implicat al final.

Per exemple, per a inicialitzar els valors d'un *array* en paral·lel utilitzant fils per a fer una porció del treball (feu la compilació, per exemple, amb la instrucció `gcc -fopenmp -o init2 init2.c`):

```
#include <stdio.h>
#include <omp.h>
#define N 1000000
int main(int argc, char *argv[]) {
   float a[N]; long i;
    #pragma omp parallel for
   for (i=0;i<N;i++)  a[i]= 2*i;
return 0;
}
```

Si executem amb el *pragma* i després el comentem i calculem el temps d'execució (`time ./init2`), veiem que el temps d'execució passa de 0,003 s a 0,007 s, la qual cosa mostra la utilització dels múltiples *cores* (si es tenen disponibles) del processador (si s'està utilitzant un màquina virtual, cal verificar que aquesta té més d'un *core* assignat, per exemple, en VBox a l'apartat *System->Processor* de la MV).

Atès que OpenMP és un model de memòria compartida, moltes variables en el codi són visibles per a tots els fils per defecte. Tanmateix, de vegades és necessari tenir variables privades i passar valors entre blocs seqüencials del codi i blocs paral·lels, per la qual cosa és necessari definir atributs a les dades (*data clauses*) que permetin diferents situacions.

- **shared**: les dades a la regió paral·lela són compartides i accessibles per tots els fils de manera simultània.

- **private**: les dades a la regió paral·lela són privades per a cada fil, i cadascun en té una còpia sobre una variable temporal.

- **default**: permet al programador definir com seran les dades dins de la regió paral·lela (`shared`, `private` o `none`).

Un altre aspecte interessant d'OpenMP són les directives de sincronització:

- **critical section**: el codi emmarcat serà executat per fils, però només un per vegada (no hi haurà execució simultània) i es manté l'exclusió mútua.
- **atomic**: similar a `critical section`, però avisa el compilador perquè faci servir instruccions de maquinari especials de sincronització i així obtenir millors prestacions.
- **ordered**: el bloc és executat en l'ordre d'un programa seqüencial.
- **barrier**: cada fil espera que els restants hagin acabat la seva execució (implica sincronització de tots els fils al final del codi).
- **nowait**: especifica que els fils que acabin el treball assignat poden continuar.

A més, OpenMP proveeix de sentències per a la planificació (*scheduling*) del tipus `schedule(type, chunk)` (on el tipus pot ser `static`, `dynamic` o `guided`) o proporciona control sobre les sentències `if`, la qual cosa permetrà definir si es paral·lelitza o no en funció de si l'expressió és veritable o no. OpenMP també proporciona un conjunt de funcions de biblioteca, com per exemple:

- **omp_set_num_threads**: defineix el nombre de fils que cal fer servir a la següent regió paral·lela.
- **omp_get_num_threads**: obté el nombre de fils que s'estan usant en una regió paral·lela.
- **omp_get_max_threads**: obté la màxima quantitat possible de fils.
- **omp_get_thread_num**: retorna el número del fil.
- **omp_get_num_procs**: retorna el màxim nombre de processadors que es poden assignar al programa.
- **omp_in_parallel**: retorna un valor diferent de zero si s'executa dins d'una regió paral·lela.

Veurem a continuació alguns exemples simples (compileu amb la instrucció `gcc -fopenmp -o out_file input_file.c`):

```
    /* Programa simple multihreading amb OpenMP */
#include <omp.h>
#include <stdio.h>

int main() {
      int iam =0, np = 1;
      #pragma omp parallel private(iam, np)
       {
      #if defined (_OPENMP)
        np = omp_get_num_threads();
        iam = omp_get_thread_num();
      #endif
      printf("Hello from thread %d out of %d \n", iam, np);
      }
   }
```

```
     /* Programa simple amb threads imbricats amb OpenMP */
```

```
#include <omp.h>
#include <stdio.h>
    main(){
        int x=0,nt,tid,ris;

        omp_set_nested(2);
        ris=omp_get_nested();
        if (ris) printf("Paral·lelisme imbricat actiu %d\n", ris);
        omp_set_num_threads(25);
        #pragma omp parallel private (nt,tid)
            {
            tid = omp_get_thread_num();
            printf("Thread %d\n",tid);
            nt = omp_get_num_threads();
            if (omp_get_thread_num()==1)
            printf("Nombre de Threads: %d\n",nt);
            }
}
```

```
/* Programa simple d'integració amb OpenMP */
#include <omp.h>
#include <stdio.h>
#define N 100
    main() {
    double local, pi=0.0, w; long i;
    w = 1.0 / N;
    #pragma omp parallel private(i, local)
        {
        #pragma omp single
        pi = 0.0;
        #pragma omp for reduction(+: pi)
        for (i = 0; i < N; i++) {
          local = (i + 0.5)*w;
          pi = pi + 4.0/(1.0 + local*local);
          printf ("Pi: %f\n",pi);
            }
    }
```

```
/* Programa simple de reducció amb OpenMP */
#include <omp.h>
#include <stdio.h>
#define NUM_THREADS 2
    main () {
    int i; double ZZ, res=0.0;
        omp_set_num_threads(NUM_THREADS);
     #pragma omp parallel for reduction(+:res) private(ZZ)
        for (i=0; i< 1000; i++) {
       ZZ = i*i;
       res = res + ZZ;
       printf("ZZ: %f, res: %f\n", ZZ, res);}
             }
        }
```

Nota

Es recomana veure també els exemples que es poden trobar en la referència [opm].

1.6. MPI, *Message Passing Interface*

La definició de l'API de MPI [Proc] ha estat el treball resultant del MPI Forum (MPIF), que és un consorci de més de 40 organitzacions. MPI té influències de diferents arquitectures, llenguatges i treballs al món del paral·lelisme, com per exemple: WRC (Ibm), Intel NX/2, Express, nCUBE, Vertex, p4, Parmac i contribucions de ZipCode, Chimp, PVM, Chamaleon, PICL.

El principal objectiu de MPIF va ser dissenyar una API, sense relació particular amb cap compilador ni biblioteca, que permetés la comunicació eficient (*memory-to-memory copy*), còmput i comunicació concurrent i descàrrega de

comunicació, sempre que hi hagués un coprocessador de comunicacions. A més, es demanava que suportés el desenvolupament en ambients heterogenis, amb interfície C i F77 (incloent C++, F90), on la comunicació fos fiable i les fallades, resoltes pel sistema. L'API també havia de tenir interfície per a diferents entorns, disposar d'una implementació adaptable a diferents plataformes amb canvis insignificants i que no interferís amb el sistema operatiu (*thread-safety*). Aquesta API va ser dissenyada especialment per a programadors que utilitzessin el *Message Passing Paradigm* (MPP) en C i F77, per a aprofitar la seva característica més rellevant: la portabilitat. El MPP es pot executar sobre màquines multiprocessador, xarxes d'estacions de treball i fins i tot sobre màquines de memòria compartida. La primera versió de l'estàndard va ser MPI-1 (que si bé hi ha molts desenvolupaments sobre aquesta versió, es considera en EOL), la versió MPI-2 va incorporar un conjunt de millores, com ara creació de processos dinàmics, *one-sided communication,* entrada/sortida paral·lela entre d'altres, i finalment l'última versió, MPI-3 (considerada com una revisió major), inclou *nonblocking collective operations, one-sided operations* i suport per a Fortran 2008.[mpi3]

Molts aspectes han estat dissenyats per a aprofitar els avantatges del maquinari de comunicacions sobre SPC (*scalable parallel computers*), i l'estàndard ha estat acceptat de manera majoritària pels fabricants de maquinari en paral·lel i distribuït (SGI, SUN, Cray, HPConvex, IBM, etc.). Hi ha versions lliures (per exemple, Mpich, LAM/MPI i OpenMPI) que són totalment compatibles amb les implementacions comercials efectuades pels fabricants de maquinari i inclouen comunicacions punt a punt, operacions col·lectives i grups de processos, context de comunicacions i topologia, suport per a F77 i C, i un entorn de control, administració i *profiling*. [lam, ompi, Proc]

Tanmateix, hi ha també alguns punts que poden presentar alguns problemes en determinades arquitectures, com són la memòria compartida, l'execució remota, les eines de construcció de programes, la depuració, el control de fils, l'administració de tasques i les funcions d'entrada/sortida concurrents (la major part d'aquests problemes de falta d'eines estan resolts a partir de la versió 2 de l'API –MPI2). Un dels problemes de MPI1, ja que no té creació dinàmica de processos, és que només suporta models de programació MIMD (*Multiple Instruction Multiple Data*) i comunicant-se via les anomenades MPI. A partir de MPI-2 i amb els avantatges de la creació dinàmica de processos, ja es poden implementar diferents paradigmes de programació com ara *master-worker/farmer-tasks*, *divide & conquer*, paral·lelisme especulatiu, etc. (o almenys sense tanta complexitat i major eficiència en la utilització dels recursos).

Per a la instal·lació de MPI es recomana utilitzar la distribució (en alguns casos la compilació pot ser complexa a causa de les dependències d'altres paquets que pot necessitar). Debian inclou la versió OpenMPI (sobre Debian 8.5 és versió 2, però es poden baixar les fonts i compilar-les) i Mpich2 (Mpich3 disponible en http://www.mpich.org/downloads/). La millor elecció serà OpenMPI, ja que combina les tecnologies i els recursos d'altres projectes diversos (FT-MPI, LA-MPI, LAM/MPI i PACX-MPI) i suporta totalment l'estàndard MPI-2 (i des de

la versió 1.75, suporta la versió MPI3). Entre altres característiques d'OpenMPI tenim: és conforme a MPI-2/3, *thread safety & concurrency*, creació dinàmica de processos, alt rendiment i gestió de treballs tolerants a fallades, instrumentació en temps d'execució, *job schedulers*, etc. Per a això s'han d'instal·lar els paquets `openmpi-dev`, `openmpi-bin`, `openmpi-common` i `openmpi-doc`. A més, Debian 8.5 inclou una altra implementació de MPI anomenada LAM (paquets `lam*`). S'ha de considerar que si bé les implementacions són equivalents des del punt de vista de MPI, tenen diferències quant a la gestió i procediments de compilació/execució/gestió.

1.6.1. Configuració d'un conjunt de màquines per a fer un clúster adaptat a OpenMPI

Per a la configuració d'un conjunt de màquines per a fer un clúster adaptat a OpenMPI [tec], s'han de seguir els següents passos:

1) Cal tenir les màquines "visibles" (per exemple, mitjançant un `ping`) a través de TCP/IP (IP pública/privada).

2) És recomanable que totes les màquines tinguin la mateixa arquitectura de processador, així és més fàcil distribuir el codi, amb versions similars de Linux (si pot ser, amb el mateix tipus de distribució).

3) Es recomana tenir NIS o, si no, s'ha de generar un mateix usuari (per exemple, mpiuser) a totes les màquines i el mateix directori `$HOME` muntat per NFS.

4) Nosaltres anomenarem les màquines `slave1`, `slave2`, etc. (ja que després resultarà més fàcil fer les configuracions), però es poden anomenar les màquines com cadascú prefereixi.

5) Una de les màquines serà el mestre i les restants, *slaveX*.

6) S'ha d'instal·lar en tots els nodes (suposem que tenim la distribució Debian): `openmpi-bin`, `openmpi-common`, `libopenmpi-dev`. Cal verificar que en totes les distribucions es treballa amb la mateixa versió d'OpenMPI.

7) En Debian, els executables estan en */usr/bin* però si estan en un *path* diferent, haurà d'agregar-se a mpiuser i també verificar que `LD_LIBRARY_PATH` apunta a */usr/lib*.

8) En cada node esclau ha d'instal·lar-se el *SSH server* (instal·leu el paquet `openssh-server`), i sobre el mestre, el client (paquet `openssh-client`).

9) S'han de crear les claus públiques i privades fent `ssh-keygen -t dsa` (tampoc hi ha problema si s'utilitza rsa) i copiar a cada node amb `ssh-copy-id` per a aquest usuari (només s'ha de fer en un node, ja que, com tindrem el directori *$HOME* compartit per NFS per a tots els nodes, amb una còpia n'hi ha prou).

10) Si no es comparteix el directori, cal assegurar que cada esclau coneix que l'usuari *mpiuser* es pot connectar sense *passwd*, per exemple fent: `ssh slave1`.

11) S'ha de configurar la llista de les màquines sobre les quals s'executarà el programa, per exemple */home/mpiuser/.mpi_hostfile* i amb el següent contingut:

```
# The Hostfile for Open MPI
# The master node, slots=2 is used because it is a dual-processor machine.
    localhost slots=2
# The following slave nodes are single processor machines:
    slave1
    slave2
    slave3
```

12) OpenMPI permet utilitzar diferents llenguatges, però aquí utilitzarem C. Per a això, cal executar sobre el mestre, per exemple,

```
mpicc testprogram.c -o testprogram
```

Si es desitja veure què incorpora `mpicc`, es pot fer `mpicc -showme`.

13) Per a executar en local, podríem fer `mpirun -np 2 ./testprogram` i per a executar sobre els nodes remots (per exemple 5 processos),

```
mpirun -np 2 -hostfile ./mpi_hostfile ./testprogram
```

És important notar que `np` és el nombre de processos o processadors en què s'executarà el programa i es pot posar el nombre que es desitgi, ja que OpenMPI intentarà distribuir els processos de manera equilibrada entre totes les màquines. Si hi ha més processos que processadors, OpenMPI/Mpich utilitzarà les característiques d'intercanvi de tasques de GNU/Linux per a simular l'execució paral·lela. A continuació, es veuran dos exemples: Srtest és un programa simple per a establir comunicacions entre processos punt a punt, i cpi calcula el valor del nombre π de manera distribuïda (per integració).

```
/* Srtest Progam */
#include "mpi.h"
#include <stdio.h>
#include <string.h>
#define BUFLEN 512

int main(int argc, char *argv[]){
    int myid, numprocs, next, namelen;
    char buffer[BUFLEN], processor_name[MPI_MAX_PROCESSOR_NAME];
    MPI_Status status;
    MPI_Init(&argc,&argv);  /* Ha de posar-se abans d'altres crides MPI, sempre */
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);    /*Integra el procés en un grup de comunicacions*/
    MPI_Get_processor_name(processor_name,&namelen); /*Obté el nom del processador*/

    fprintf(stderr,"Procés %d sobre %s\n", myid, processor_name);
    fprintf(stderr,"Procés %d de %d\n", myid, numprocs);
    strcpy(buffer,"hello there");
    if (myid == numprocs-1) next = 0;
    else next = myid+1;

    if (myid == 0) { /*Si és l'inicial, envia string de buffer*/
        printf("%d sending '%s' \n",myid,buffer);fflush(stdout);
    MPI_Send(buffer, strlen(buffer)+1, MPI_CHAR, next, 99, MPI_COMM_WORLD);
        /*Blocking Send, 1:buffer, 2:size, 3:tipus, 4:destinació, 5:tag, 6:context*/
    printf("%d receiving \n",myid);fflush(stdout);
```

```
     MPI_Recv(buffer, BUFLEN, MPI_CHAR, MPI_ANY_SOURCE, 99, MPI_COMM_WORLD,&status);
     printf("%d received '%s' \n",myid,buffer);fflush(stdout);
        /* mpdprintf(001,"%d receiving \n",myid); */
         }
     else {
        printf("%d receiving  \n",myid);fflush(stdout);
    MPI_Recv(buffer, BUFLEN, MPI_CHAR, MPI_ANY_SOURCE, 99, MPI_COMM_WORLD,&status);
        /* Blocking Recv, 1:buffer, 2:size, 3:tipus, 4:font, 5:tag, 6:context, 7:status*/
    printf("%d received '%s' \n",myid,buffer);fflush(stdout);
    /* mpdprintf(001,"%d receiving \n",myid); */
    MPI_Send(buffer, strlen(buffer)+1, MPI_CHAR, next, 99, MPI_COMM_WORLD);
    printf("%d sent '%s' \n",myid,buffer);fflush(stdout);
        }
     MPI_Barrier(MPI_COMM_WORLD); /*Sincronitza tots els processos*/
     MPI_Finalize(); /*Allibera els recursos i acaba*/
     return (0);
     }


/* CPI Program */
#include "mpi.h"
#include <stdio.h>
#include <math.h>
double f( double );
double f( double a) { return (4.0 / (1.0 + a*a)); }
int main( int argc, char *argv[] ) {
    int done = 0, n, myid, numprocs, i;
    double PI25DT = 3.14159265358979323846264338327950288419716939937510;
    double mypi, pi, h, sum, x; double startwtime = 0.0, endwtime;
    int namelen; char processor_name[MPI_MAX_PROCESSOR_NAME];
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs); /*Indica el nombre de processos en el grup*/
    MPI_Comm_rank(MPI_COMM_WORLD,&myid); /*Id del procés*/
    MPI_Get_processor_name(processor_name,&namelen); /*Nom del procés*/
    fprintf(stderr, "Procés %d sobre %s\n", myid, processor_name);
    n = 0;
    while (!done) {
        if (myid ==0) { /*Si és el primer...*/
            if (n ==0) n = 100; else n = 0;
            startwtime = MPI_Wtime();} /* Time Clock */
        MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD); /*Broadcast a la resta*/
            /*Envia des del 4 arg. a tots els processos del grup. Els restants que no són 0
            copiaran el buffer des de 4 o arg -procés 0-*/
            /*1:buffer, 2:size, 3:tipus, 5:grup */
        if (n == 0) done = 1;
        else {h = 1.0 / (double) n;
            sum = 0.0;
            for (i = myid + 1; i <= n; i += numprocs) {
                x = h * ((double)i - 0.5); sum += f(x); }
            mypi = h * sum;
            MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0,MPI_COMM_WORLD);
                /* Combina els elements del Send Buffer de cada procés del grup usant
                l'operació MPI_SUM i retorna el resultat en el Recv Buffer. Ha de ser cridada
                per tots els processos del grup fent servir els mateixos arguments*/

                /*1:sendbuffer, 2:recvbuffer, 3:size, 4:tipus, 5:oper, 6:root, 7:context*/
            if (myid == 0){ /*només el P0 imprimeix el resultat*/
                printf("Pi és aproximadament %.16f, l'error és %.16f\n", pi,fabs(pi - PI25DT));
                endwtime = MPI_Wtime();
                printf("Temps d'execució = %f\n", endwtime-startwtime); }
            }
    }
    MPI_Finalize(); /*Allibera recursos i acaba*/
    return 0;
}
```

Per a visualitzar l'execució d'un codi paral·lel/distribuït en MPI, hi ha una aplicació anomenada XMPI (en Debian xmpi) que permet "veure", durant l'execució, l'estat de l'execució de les aplicacions sobre MPI, però està vinculada

al paquet LAM/MPI. Per a visualitzar i analitzar codi sobre OpenMPI, s'hauria de (i és recomanable) baixar i compilar el codi de MPE* o TAU**, que són eines de *profiling* molt potents i que no requereixen gaire treball ni dedicació per a engegar-les.

***http://www.mcs.anl.gov /research/projects/perfvis /software/mpe/**
****http://www.cs.uoregon.edu /research/tau/home.php**

1.7. Rocks Cluster

Rocks Cluster és una distribució de Linux per a clústers d'ordinadors d'alt rendiment. Les versions actuals de Rocks Cluster estan basades en CentOS (CentOS 6.6 a maig 2015) i, com a instal·lador, Anaconda amb certes modificacions, que simplifica la instal·lació "en massa" en molts ordinadors. Rocks Cluster inclou moltes eines (com ara MPI) que no formen part de CentOS però són els components que transformen un grup d'ordinadors en un clúster. Les instal·lacions poden personalitzar-se amb paquets de programari addicionals anomenats *rolls*. Els *rolls* estenen el sistema integrant automàticament els mecanismes de gestió i empaquetament usats pel programari bàsic, i simplifiquen àmpliament la instal·lació i configuració d'un gran nombre d'ordinadors. S'ha creat una gran quantitat de *rolls*, com per exemple SGE *roll*, Cóndor *roll*, Xen *roll*, el Java *roll*, Ganglia *roll*, etc.*** Rocks Cluster és una distribució molt emprada en l'àmbit de clústers, per la seva facilitat d'instal·lació i incorporació de nous nodes i per la gran quantitat de programes per al manteniment i monitoratge del clúster.

*****http://www.rocksclusters.org /wordpress/?page_id=4**

Enllaços d'interès

Per a una llista detallada de les eines incloses en Rocks Cluster, consulteu http://www.rocksclusters.org /roll-documentation/base /5.5/.

Les principals característiques de Rocks Cluster són:

1) Facilitat d'instal·lació, ja que només és necessari completar la instal·lació d'un node anomenat *node mestre* (*frontend*), la resta s'instal·la amb Avalache, que és un programa P2P que ho fa de manera automàtica i evita haver d'instal·lar els nodes un a un.

2) Disponibilitat de programes (conjunt molt ampli de programes que no és necessari compilar i transportar) i facilitat de manteniment (només s'ha de mantenir el node mestre).

3) Disseny modular i eficient, pensat per a minimitzar el trànsit de xarxa i utilitzar el disc dur propi de cada node per a només compartir la informació mínima i imprescindible.

La instal·lació es pot seguir pas a pas des del lloc web de la distribució [Rock] i els autors garanteixen que no es triga més d'una hora per a una instal·lació bàsica. Rocks Cluster permet, en l'etapa d'instal·lació, diferents mòduls de programari, els *rolls*, que contenen tot el necessari per a fer la instal·lació i la configuració de sistema amb aquestes noves addicions de manera automàtica i, a més, decidir en el *frontend* com serà la instal·lació en els nodes esclaus, quins *rolls* estaran actius i quina arquitectura s'usarà. Per al manteniment, inclou un sistema de còpia de seguretat de l'estat, anomenat *Roll Restore*. Aquest

roll guarda els arxius de configuració i *scripts* (i fins i tot, es poden afegir els arxius que es vulguin).

1.7.1. Guia ràpida d'instal·lació

Aquest és un resum breu de la instal·lació proposada en [Rock] i es parteix de la base que el *frontend* té (mínim) 30 GB de disc dur, 1 GB de RAM, arquitectura x86-64 i 2 interfícies de xarxa (eth0 per a la comunicació amb Internet i eth1 per a la xarxa interna); per als nodes 30 GB de disc dur, 512 MB de RAM i 1 interfície de xarxa (xarxa interna). Després d'obtenir els discos *Kernel/Boot Roll, Base Roll, US Roll CD1/2* (o en defecte d'això, DVD equivalent), inserim *kernel boot* i seguim els següents passos:

Enllaços d'interès

És possible descarregar els discos des de: http://www.rocksclusters.org/wordpress/?page_id=80

1) Arrenquem el *frontend* i veurem una pantalla en la qual introduïm `build` i ens preguntarà la configuració de la xarxa (IPV4 o IPV6).

2) El següent pas que s'ha de fer és seleccionar els `rolls` (per exemple, seleccionant els "CD/DVD-based Roll" i anem introduint els següents *rolls*) i marcar en les successives pantalles quins són els que volem.

3) La següent pantalla ens demanarà informació sobre el clúster (és important definir bé el nom de la màquina, *Fully-Qualified Host Name*, ja que en cas contrari fallarà la connexió amb diferents serveis) i també informació per a la xarxa privada que connectarà el *frontend* amb els nodes i la xarxa pública (per exemple, la que connectarà el *frontend* amb Internet) així com DNS i passarel·les.

4) A continuació, se sol·licitarà la contrasenya per al *root*, la configuració del servei de temps i la partició del disc (es recomana escollir "auto").

5) Després de formatar els discos, sol·licitarà els CD de *rolls* indicats i instal·larà els paquets i farà un *reboot* del *frontend*.

6) Per a instal·lar els nodes, s'ha d'entrar com a *root* en el *frontend* i executar `insert-ethers`, que capturarà les peticions de DHCP dels nodes i els agregarà a la base de dades del *frontend*, i seleccionar l'opció "*Compute*" (per defecte, consulteu la documentació per a les altres opcions).

7) Posem en marxa el primer node i en el *boot order* de la BIOS generalment es tindrà CD, PXE (Network Boot), Hard Disk (si l'ordinador no suporta PXE, llavors feu el *boot* del node amb el *Kernel Roll CD*). En el *frontend* es veurà la petició i el sistema l'agregarà com a `compute-0-0` i començarà la descàrrega i instal·lació dels arxius. Si la instal·lació falla, s'hauran de reiniciar els serveis httpd, mysqld i autofs en el *frontend*.

8) A partir d'aquest punt, es pot monitorar la instal·lació executant la instrucció `rocks-console`, per exemple amb `rocks-console compute-0-0`. Després que s'hagi instal·lat tot, es pot sortir de `insert-ethers` prement la

tecla F8. Si es disposa de dos *racks*, després d'instal·lat el primer es pot començar el segon fent `insert-ethers -cabinet=1`, els quals rebran els noms `compute-1-0`, `compute-1-1`, etc.

9) A partir de la informació generada en consola per `rocks list host`, generarem la informació per a l'arxiu `machines.conf`, que tindrà un aspecte com:

```
nteum slot=2
compute-0-0 slots=2
compute-0-1 slots=2
compute-0-2 slots=2
compute-0-3 slots=2
```

i que després haurem d'executar amb

```
mpirun -np 10 -hostfile ./machines.conf ./mpi_program_to_execute.
```

Un exercici interessant per fer és instal·lar un clúster funcional virtualitzat que òbviament no tindrà grans prestacions, però que és útil per aprendre tots els conceptes que intervenen i com a prova de concepte. Per a això utilitzarem VirtualBox sobre un processador de 64 bits (les últimes versions de Rocks solament estan per a 64 bits) i que tinguin les VTx/AMD-V del processador. El procediment seria:

1) Descarregar la iso del DVD des d'http://www.rocksclusters.org.

2) Crear una màquina virtual (per exemple, RocksFE) amb 28 GB de disc i dos adaptadors de xarxa (un connectat a la xarxa externa -*eth0*- i un altre a una xarxa interna -*eth1*- que l'anomenarem *rocksnet*). Aquesta màquina serà el *Frontend* del nostre clúster. Després inserim el DVD sobre el DVD virtual i arrenquem la màquina virtual.

3) Introduir a la primera pàgina, `build ksdevice=eth0 ip=IP-xarxa-externa gateway=gateway-extern netmask=255.255.255.0`.

4) Seguir els passos abans indicats, seleccionant la mitjana des del DVD i els *rolls* (mínim: os, kernel, base) que es desitgin i contestant a les preguntes de la xarxa i DNS.

5) Després de la instal·lació podrem entrar i veure l'estat de la màquina.

6) Crear una segona màquina (per exemple, RocksCN) similar a la primera però amb només una targeta de xarxa (que configurarem com a xarxa interna i la connectarem a *rocksnet*) i configurar l'arrencada (a *System*) com *Network*.

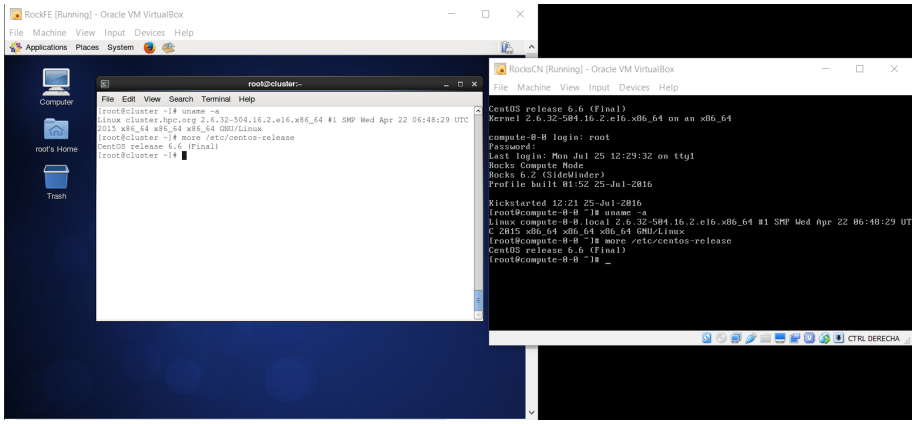
7) Sobre el *Frontend* i en una terminal executar *insert-ethers* seleccionant *Compute* i engegar la màquina virtual. Veurem després d'un moment que la reconeix dins de la finestra com *compute-0-0* i arrenca i s'instal·la el SO sobre RocksCN.

8) Quan finalitzi la instal·lació sobre RocksCN hem d'apagar la màquina i activar el *boot* des del disc dur i tornar a arrencar la màquina.

9) Per a finalitzar veurem que està donada d'alta a */etc/hosts* sobre el *Frontend* i que podem accedir a ella a través d'un `ssh compute-0-0`.

La figura 1 mostra l'execució sobre VirtualBox de la prova de concepte del *Frontend* (amb GUI) i la d'un node (en mode text).

Figura 1



1.8. Desplegament automàtic

El desplegament automàtic (*Automatic Provisioning*) és una de les grans preocupacions i suposa un consum de temps per als administradors, sobretot quan s'ha de desplegar un conjunt important de màquines. Això explica l'esforç d'alguns projectes per simplificar aquesta tasca, per exemple:

- **Kickstart [KS]**, per a màquines RH, Fedora i CentOS, bàsicament.
- **FAI [FG]**, per a sistemes Debian però que poden instal·lar un altre tipus de distribucions.
- **Cobbler [Co]**, que si bé és per a sistemes de la branca d'RH hi ha experiències sobre altres sistemes (internament utilitza KickStart).
- **Spacewalk [SW]**, per a servidors RH però que poden instal·lar un altre tipus de distribucions.
- **OpenQRM [OQRM]**, per a diferents servidors (versió *community*).

En el nostre cas, per seguir amb la mateixa distribució, veurem el cas de FAI sobre un sistema Debian.

1.8.1. FAI (Fully Automatic Installation)

FAI és una eina d'instal·lació automatitzada per a desplegar Linux en un clúster o en un conjunt de màquines en forma desatesa. És equivalent a *Kickstart*

de RH, o Alice de SuSE. FAI pot instal·lar Debian, Ubuntu i RPM de distribucions Linux. Els seus avantatges són que mitjançant aquesta eina es pot desplegar Linux sobre un node (físic o virtual) simplement fent que arrenqui per PXE i quedi preparat per a treballar i totalment configurat (quan es diu un, es podria dir 100 amb el mateix esforç per part de l'administrador) i sense cap interacció pel mig. Per tant, és un mètode escalable per a la instal·lació i actualització d'un clúster Beowulf o una xarxa d'estacions de treball sense supervisió i amb poc esforç. FAI utilitza la distribució Debian, una col·lecció d'*scripts* (la majoria en Perl) i *cfengine* i *Preseeding d-i* per al procés d'instal·lació i canvis en els arxius de configuració.

És una eina pensada per a administradors de sistemes que han d'instal·lar Debian en desenes o centenars d'ordinadors i pot utilitzar-se a més com a eina d'instal·lació de propòsit general per a instal·lar (o actualitzar) un clúster de còmput HPC, de servidors web, o un *pool* de bases de dades i configurar la gestió de la xarxa i els recursos en forma desatesa. Aquesta eina permet tractar (mitjançant un concepte que incorpora anomenat *classes*) amb un maquinari diferent i diferents requisits d'instal·lació en funció de la màquina i característiques de què es disposi en la seva preconfiguració, la qual cosa permet fer desplegaments massius eficients i sense intervenció d'administrador (una vegada que l'eina hagi estat configurada de manera adequada). [fai1, fai2, fai3]

1.9. Guia d'instal·lació de FAI

En aquest apartat veurem una guia d'instal·lació bàsica en màquines virtuals (en Virtualbox) per desplegar altres màquines virtuals, però que, amb unes intervencions mínimes, es pot adaptar/ampliar a les necessitats de l'entorn. FAI és una eina que no compta amb *daemons*/DB; només consisteix en *scripts* (consulteu les referències indicades).

La manera més fàcil de fer una prova de concepte és a través de les instal·lacions modulars que proveeix el projecte (on es pot utilitzar una màquina virtual, tal com farem nosaltres). Per a això, s'utilitza una ISO en un USB o en un CD descarregable des de http://fai-project.org/fai-cd/. Les imatges autoinstal·lables disponibles són:

1) FAI ISO large (953 MB): imatge per defecte que inclou XFCE i GNOME (però només la base per CentOS 7 i Ubuntu 16.04).

2) FAI ISO small (585 MB): imatge que inclou només XFCE. Els altres paquets es descarreguen des d'Internet.

3) FAI ISO Ubuntu (1.1 GB): tot Ubuntu 16.04 LTS.

4) Autodiscover CD (19 MB): imatge que només permet l'arrencada i que busca un servidor FAI.

Totes les imatges anteriors utilitzen Debian Jessie, FAI 5.1.2, nucli 3.16, arquitectura amd64. A més, està disponible la imatge **FAI ISO small** (stretch) (585 MB), que és la versió FAI ISO small, però que utilitza Debian stretch.

Per inserir les imatges en un USB es pot utilitzar la comanda

```
dd if=faicd64-large_5.1.2.iso of=/dev/sdX
```

on s'ha de reemplaçar */dev/sdX* amb el dispositiu de l'USB (p. ex., */dev/sdg*).

Amb aquestes imatges es pot instal·lar una Debian 8 (XFCE/GNOME), Ubuntu 16.04 o CentOS 7 sense que calgui tenir un servidor (tant l'usuari *fai*, com *demo*, com *root* té per *passwd* **fai**). S'ha d'anar amb compte, ja que el sistema quedarà instal·lat sobre el primer disc, i totes les dades seran esborrades. Per instal·lar-lo en una màquina Virtualbox només cal crear una màquina amb un disc de, per exemple, 12 GB, carregar la imatge en el CD-Rom virtual i arrencar la màquina seleccionant a l'inici tan sols allò que es desitja. Després, la instal·lació és desatesa i tindrem la màquina instal·lada sense cap intervenció.

1.9.1. Instal·lació del servidor

Si es vol instal·lar un servidor per aprovisionar altres màquines i adaptar les configuracions a les necessitats de la instal·lació, es pot seguir la guia del projecte*. No obstant això, sobre Jessie, presenta alguns problemes amb certs paquets i s'ha d'invertir una gran quantitat de temps per compatibilitzar tots els requeriments. Per aquest motiu és més fàcil utilitzar la imatge FAI ISO Large, ja que permet instal·lar el servidor en un temps breu i, després, adaptar-lo a les necessitats de la instal·lació i dels clients. Per això en aquesta prova de concepte farem:

***http://fai-project.org/fai-guide/**

1) Descarregar la imatge: http://fai-project.org/fai-cd/faicd64-large_5.1.2.iso.

2) Crear a VirtualBox una màquina virtual amb 16 Gb de disc i una targeta de xarxa amb NAT (després ho canviarem), i associar aquesta imatge al DVD virtual.

3) Arrencar la màquina virtual i seleccionar *FAI server Installation using external DHCP*. Introduir com a usuari *fai* i *install* com a *passwd*, i començarà la instal·lació.

4) Quan finalitzi i es torni a arrencar la màquina, seleccionarem *Boot US of First Partition (1st disc)* (o, si no, traurem la imatge del DVD) i començarà la instal·lació del servidor durant uns minuts.

5) Quan finalitzi, iniciarem la sessió com a **root** i *passwd* **fai** i farem els següents ajustos:

a) Canviar el teclat a */etc/default/keyboard* a *XKBLAYOUT = "es"* i executar `service keyboard-setup restart`.

b) Editar */etc/hosts* i canviar la IP del faiserver a 192.168.33.1, ja que volem que l'aprovisionament ho faci per l'altra xarxa que afegirem.

c) Editar l'arxiu */etc/network/interfaces* i afegir la configuració per a *eth1* com:

```
auto lo eth0 eth1
...
iface eth1 inet static
        address 192.168.33.1
        netmask 255.255.255.0
```

d) Modificar */etc/exports* i, on diu 10.0.2.0/24, canviar-ho per 192.168.33.0/24, perquè els clients puguin muntar tots els recursos del servidor.

e) Canviar a */etc/dhcp/dhcpd.conf* la IP de l'opció de *domain-name-servers* (per exemple, a 192.168.33.1).

f) Reiniciar el sistema i, abans que arrenqui, aturar-lo i afegir un segon adaptador (*eth1*) connectat a la xarxa interna (serà la mateixa xarxa a la qual després connectarem els clients). A continuació, treure el DVD (si no ho hem fet abans) perquè arrenqui directament des del disc dur. Iniciar la sessió com a *root*, i verificar que tot és correcte i que tenim els serveis tftp i dhcpd funcionant (per exemple, amb `systemctl status`).

g) **Consideracions que cal tenir en compte**. Quan es crea la MV per als clients, cal escollir el tipus Linux64, ja que, si no ho fem, durant l'arrencada del client donarà un error perquè intentarà instal·lar un Linux32. Quan s'ha instal·lat satisfactòriament una màquina, en queda un registre al servidor i, si volem fer una altra prova sobre la mateixa MV, haurem canviar-li la MAC per reutilitzar la mateixa màquina, ja que, si no, donarà un error perquè consta com instal·lada.

1.9.2. Configuració del node

Farem la nostra prova de concepte per a un node en Virtualbox. Primer, cal instal·lar les expansions de la versió que tenim instal·lada. Per fer això, haurem d'anar a https://www.virtualbox.org i, per a la distribució instal·lada, descarregar i instal·lar el paquet *VirtualBox Extension Pack All supportedplatforms* (això ens permetrà tenir un dispositiu que faci *boot* per PXE). Podrem verificar que estan instal·lades al menú de VirtualBox -> File -> Preferences -> Extensions. Després, haurem de crear una nova màquina virtual amb un disc buit (per exemple, 8 GB si és un client text, o 14-16 GB si és un client XFE/Gnome) i una targeta de xarxa connectada tant a una xarxa interna com a la mateixa xarxa en què es troba al servidor en la targeta *eth1*. A continuació, en primer lloc, s'haurà de seleccionar al *System* d'aquesta màquina l'opció de *boot order network*. Finalment, arrencarem la màquina, i veurem que rep IP (verificar els possibles errors a */var/log/messages |syslog*) i que es comença a baixar el *initrd-img* i, després, el nucli. Ens demanarà quin tipus de client volem instal·lar i, al cap d'uns minuts, el sistema estarà instal·lat. Quan finalitzi la instal·lació,

haurem d'apagar el sistema i canviar l'ordre de *boot* a *System* perquè torni a arrencar, en primer lloc, des del disc dur.

La figura 2 mostra les proves realitzades amb la selecció d'un client simple i la figura 3, amb un client Gnome (a l'esquerra es visualitza el servidor d'aprovisionament).

Figura 2

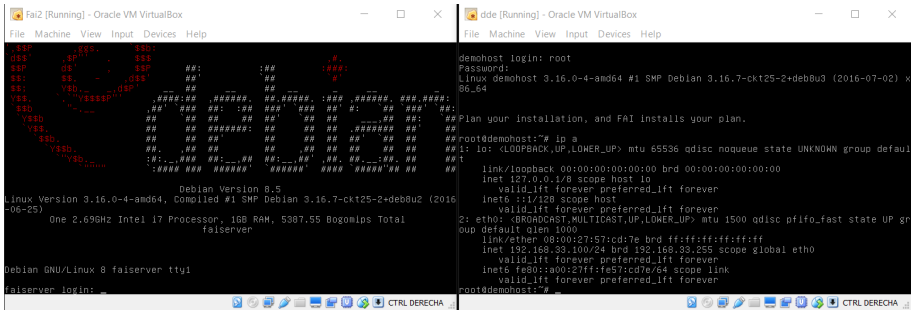
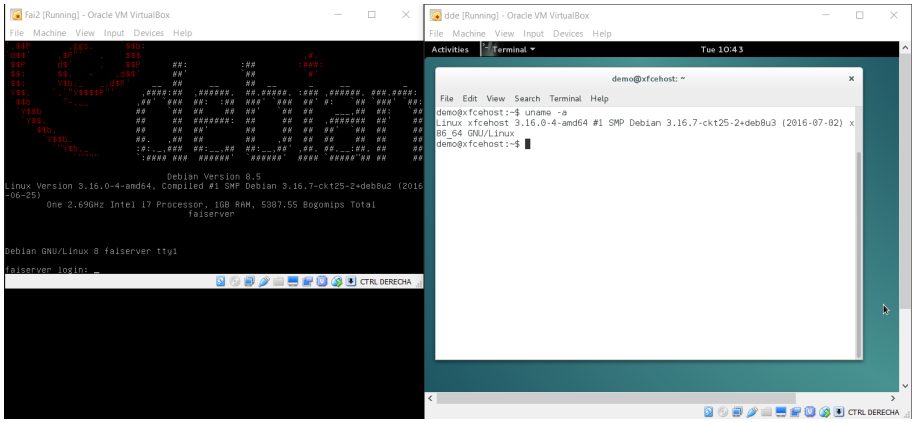


Figura 3



1.10. *Logs*

Linux manté un conjunt de registres anomenats *system logs* o *logs* simplement, que permeten analitzar què ha passat i quan en el sistema, a partir dels esdeveniments que recull del propi *kernel* o de les diferents aplicacions i gairebé totes les distribucions es troben en */var/log*. Probablement, els dos arxius més importants (i que amb més o menys presència estan en totes les distribucions) són */var/log/messages* i */var/log/syslog*, els que tenen registres (ASCII) d'esdeveniments com ara errors del sistema, (re)inicis/apagats, errors d'aplicacions, advertiments, etc. Existeix una ordre, `dmesg`, que permet a més veure els missatges d'inici (en algunes distribucions són els que apareixen en la consola o amb la tecla Esc en la manera gràfica d'arrencada, o Crtl+F8 en altres distribucions) per a visualitzar els passos seguits durant l'arrencada (tenint en compte que pot ser molt extens, es recomana executar `dmesg | more`).

Per la qual cosa, el sistema ens permetrà analitzar què ha passat i esbrinar les causes que han produït aquest registre i on/quan. El sistema inclòs en Debian és `rsyslog` (http://www.rsyslog.com/) amb un servei (a través del *daemon* **rsyslogd**) que reemplaça a l'original *syslog*. És un sistema molt potent i versàtil i la seva configuració és través de l'arxiu */etc/rsyslog.conf* o arxius en el directori */etc/rsyslog.d*. Podeu mirar la documentació sobre la seva configuració (`man rsyslog.conf` o a la pàgina indicada), però en resum inclou una sèrie de directives, filtres, *templates* i regles que permeten canviar el comportament dels *logs* del sistema. Per exemple, les regles són de tipus **recurs.nivell acció** però es pot combinar més d'un recurs separat per ',', o diferents nivells, o tots '*' o negar-ho '!', i si posem '=' davant del nivell, indica només aquest nivell i no tots els superiors, que és el que ocorre quan només s'indica un nivell (els nivells poden ser de menor a major importància *debug, info, notice, warning, err, crit, alert, emerg*). Per exemple, `*.warning /var/log/messages` indicarà que tots els missatges de *warning,err,crit,alert,emerg* de qualsevol recurs vagin a parar a aquest arxiu. Es pot incloure també un '-' davant del nom del fitxer, que indica que no se sincronitzi el fitxer després de cada escriptura per a millorar les prestacions del sistema i reduir la càrrega.

Un aspecte interessant dels *logs* és que els podem centralitzar des de les diferents màquines de la nostra infraestructura en un determinat servidor, per a no haver de connectar-nos si volem "veure"què està ocorrent en els *logs* en aquestes màquines (sobretot si el nombre de màquines és elevat). Per a això, en cada client hem de modificar l'arxiu */etc/rsyslog.conf* (on la IP serà la del servidor que recollirà els *logs*):

```
# Provides TCP forwarding.
*.*                             @@192.168.168.254:514
```

En el servidor, haurem de treure el comentari (#) dels mòduls de recepció per TCP en */etc/rsyslog.conf*:

```
# Provides TCP syslog reception
$ModLoad imtcp
$InputTCPServerRun 514
```

Després de fer les modificacions de cada arxiu, és important no oblidar fer un `/etc/init.d/rsyslog restart`. A partir d'aquest moment, podem mirar el */var/log/syslog* del servidor, per exemple, i veurem els registres marcats amb el nom (o IP) de la màquina on s'ha generat el *log*.

Existeix un paquet anomenat `syslog-ng` que presenta característiques avançades (per ex., xifratge o filtres basats en continguts) o altres d'equivalents amb `rsyslog`. Trobareu més informació en http://www.balabit.com/network-security/syslog-ng.

1.11. Visualització de logs

Un aspecte important, quan es tenen els *logs* –registres –(inclosos d'altres màquines), és com visualitzar-los i com fer una recerca selectiva per prevenir situacions o saber l'estat del sistema, també per generar informes o visualitzar de manera integrada diferents fonts d'informació. Hi ha eines que ens permeten visualitzar (*log browsing*) i buscar de manera simple, com per exemple:

1) glogg (http://glogg.bonnefon.org/)

2) ksystemlog (https://www.kde.org/applications/system/ksystemlog/)

3) lnav (http://lnav.org/)

4) multitail (https://www.vanheusden.com/multitail/), que es troben a l'arxiu de Debian

5) logio (http://logio.org/), quan la visualització s'ha de fer remotament.

Quan es vol fer una supervisió i una gestió de manera eficient i escalable, amb capacitats de pre-processat, filtrat, indexació i emmagatzematge de gran quantitat d'esdeveniments (per a la seva anàlisi *on-line* o posterior), cal comptar amb eines adequades a aquestes funcions. En aquesta categoria es poden esmentar eines com:

1) LogAnalyzer (http://loganalyzer.adiscon.com/). És una eina fàcil de configurar/usar, amb una interfície versàtil per buscar i analitzar dades d'esdeveniments d'un sistema de diferents fonts de xarxa (syslog/rsyslog). Per exemple, aquesta aplicació recull les dades produïdes per *rsyslog* (estàndard a Debian) i les visualitza a través d'un servidor web i d'un navegador. L'aplicació està escrita en PHP i permet la connexió a MySQL (o altres bases de dades SQL-NoSQL), quan són necessàries opcions avançades o un processament eficient.

2) Logstash (https://www.elastic.co/products/logstash). Aquesta eina *open source* permet recol·lectar, analitzar i emmagatzemar els arxius de *logs* per a la seva anàlisi posterior. Incorpora un conjunt de connectors per facilitar diferents formats, descodificació i regles de filtre (per exemple, *logs* de S3, RabbitMQ, Syslog, collectd, sockets TCP/UDP).

3) Collectd (https://collectd.org/). Si bé és una eina més aviat de monitorització, permet, a través de diversos connectors d'entrada/sortida, processar diferents arxius de *logs* (locals o remots). Està inclosa en el repositori de Debian.

4) Fluentd (http://www.fluentd.org/ https://github.com/fluent/fluentd).
És una eina que presenta una capa unificada d'agregació de *logs* i que permet el processament *in-stream* per a una gran varietat de flux de dades i d'arxius de *logs*. La seva arquitectura és extensible i compta amb més de 300 connectors per a diverses fonts d'entrada/sortida d'interfícies diferents (per exemple, Apache, syslog | rsyslog, bases de dades SQL-NoSQL, S3...).

5) Flume (https://flume.apache.org). És el projecte desenvolupat per Apache com a servei distribuït i eficient per recollir, afegir i moure grans quantitats de dades de *logs*. Aquesta eina presenta una arquitectura flexible que funciona sobre fluxos de dades (*streaming data flows*). És robusta i tolerant a fallades, i utilitza un model simple de dades extensible, la qual cosa permet que pugui ser utilitzada com a eina de gestió analítica i de presa de decisions.

Com a exemple, veurem la instal·lació i configuració de LogAnalyzer, que destaca per la seva simplicitat i les seves prestacions:

1) Verifiquem que disposem d'Apache2 ja instal·lat, i descarreguem el paquet des del web del desenvolupador: http://loganalyzer.adiscon.com/downloads/.

2) El descomprimim `tar xzvf loganalyzer-xyztar.gz`, on la versió corresponent és x.y.z, i movem la carpeta interna *src* al directori *DocumentRoot* d'Apache, per exemple,

```
mv loganalyzer-x.y.z/src /var/www/html/log2
```

i també l'arxiu *contrib /configure.sh*, Per exemple,

```
cp loganalyzer-x.y.z/contrib/configure.sh /var/www/html/log2/
```

Executem

```
chmod +x /var/www/html/log2/configure.sh
```

i, després, `/var/www/html/log2/configure.sh`.

3) Amb això i connectant-nos al servidor /logs2, podrem realitzar la configuració restant i, quan hàgim acabat, tindrem la visualització dels *logs*, però no podrem accedir a alguna de les funcionalitats i configuracions reservades a l'administrador. Per a això és necessari disposar d'una base de dades (MySQL per defecte). Per instal·lar la base de dades, farem `apt-get install mysqlserver php5-mysql` (hem de recordar el *passwd* per a l'usuari *root* de la BD, ja que després serà necessari). Creem una base de dades anomenada *loganalyzer* (per exemple, executant `mysql -u root -p` i, després, `create database loganalyzer;`).

4) Des de qualsevol navegador connectem el servidor/aplicació. En el nostre cas, http://srv.nteum.org/log2/, i seguim les finestres de configuració. No s'ha de tocar res fins al pas 3 (*frontend options*), a la finestra inferior del qual, *user database options*, s'ha de seleccionar *Enable User Database* = *yes* i configurar com a usuari *root* i el *passwd* introduït quan es va crear la base de dades. A la finestra següent es crearan les taules i es demanarà un usuari i un *passwd* per accedir a la gestió. Amb això ja podrem veure els *log* des de *syslog*, afegir noves fonts de *log*, seleccionar-los, cercar o entrar al *admin Center* per canviar les fonts de *logs*, l'aspecte, els informes, els usuaris, els camps, les vistes, les estadístiques, els gràfics i una gran quantitat d'opcions addicionals. Podem trobar informació addicional sobre LogAnalyzer a [logA] (o mitjançant el botó *Help* de l'aplicació). També podem trobar informació sobre com generar informes setmanalment o diàriament (l'exemple està per a W, però és equivalent en Linux) a [LogAR].

5) Una vegada finalitzada la configuració, és convenient canviar les protecci-ons del fitxer *config.php*, per exemple,

```
chmod 644 /var/www/html/log2/config.php
```

Cal tenir en compte les recomanacions de *Securing LogAnalyzer* a propòsit de la documentació indicada.

Les figures 4 i 5 mostren dues imatges de LogAnalyzer (la primera del *syslog* i la segona de auth.log, com es veu en el gestor, a la part superior dreta).

Figura 4

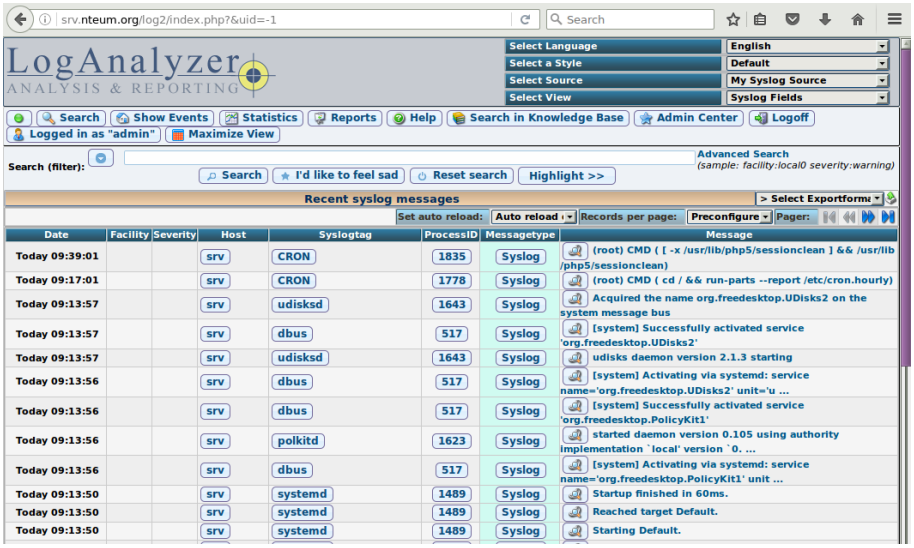
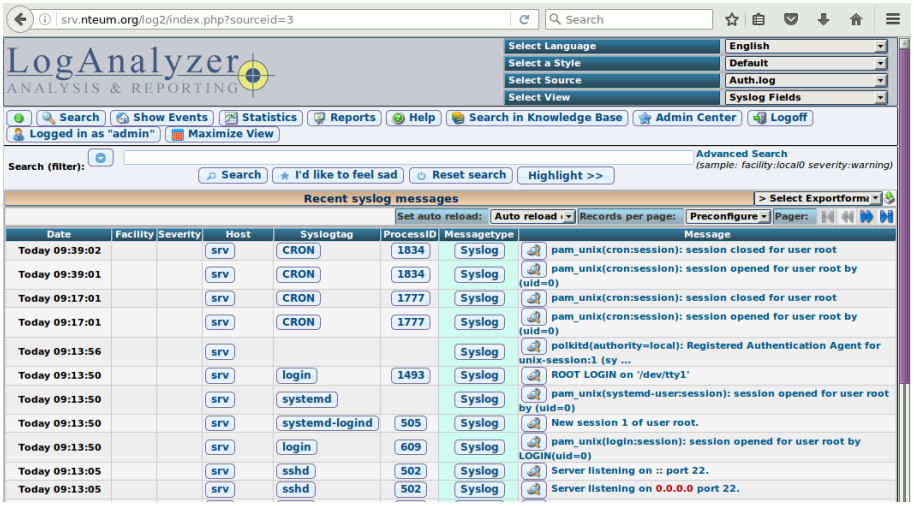


Figura 5



2. *Cloud*

Les infraestructures *cloud* es poden classificar en 3 + 1 grans grups en funció dels serveis que presten i a qui els presten:

1) Públiques: els serveis es troben en servidors externs i les aplicacions dels clients es barregen en els servidors i amb altres infraestructures. L'avantatge més clar és la capacitat de processament i emmagatzematge sense instal·lar màquines localment (no hi ha inversió inicial ni manteniment) i es paga per ús. Té un retorn de la inversió ràpid i pot resultar difícil integrar aquests serveis amb altres de propis.

2) Privades: les plataformes es troben dins de les instal·lacions de l'empresa/institució i són una bona opció per a aquells que necessiten una alta protecció de dades (aquests continuen dins de la pròpia empresa). És més fàcil integrar aquests serveis amb altres de propis, però hi ha inversió inicial en infraestructura física, sistemes de virtualització, amplada de banda, seguretat i despesa de manteniment, la qual cosa implica un retorn més lent de la inversió. No obstant això, entre tenir infraestructura i tenir-ne *cloud* i virtualizada, aquesta última és millor pels avantatges de major eficiència, millor gestió i control i major aïllament entre projectes i rapidesa en la provisió.

3) Híbrides: combinen els models de núvols públics i privats i permeten mantenir el control de les aplicacions principals a la vegada que s'aprofita el *cloud computing* en els llocs on tingui sentit amb una inversió inicial moderada i, alhora, permet comptar amb els serveis que es necessitin sota demanda.

4) Comunitat: canalitzen necessitats agrupades i les sinergies d'un conjunt o sector d'empreses per a oferir serveis de *cloud* a aquests grups d'usuaris.

A més, hi ha diferents capes sota les quals es poden contractar aquests serveis:

1) *Infraestructure as a Service* (IaaS): es contracta capacitat de procés i d'emmagatzematge que permeten desplegar aplicacions pròpies que per motius d'inversió, infraestructura, cost o falta de coneixements no es volen instal·lar en la pròpia empresa (exemples d'aquest tipus són EC2/S3 d'Amazon i Azure de Microsoft).

2) *Platform as a Service* (PaaS): es proporciona a més un servidor d'aplicacions (on s'executaran les nostres aplicacions) i una base de dades, on es podran instal·lar les aplicacions i executar-les, les quals s'hauran de desenvolupar d'acord amb unes indicacions del proveïdor (per exemple, Google App Engine).

3) *Programari as a Service* (SaaS): comunament s'identifica amb *cloud*, on l'usuari final paga un lloguer per a l'ús de programari sense adquirir-lo en propietat, instal·lar-lo, configurar-lo i mantenir-lo (en són exemples Adobe Creative Cloud, Google Docs o Office365).

4) *Business Process as a Service* (BPaaS): és la capa més nova (i se sustenta damunt de les altres 3), on el model vertical (o horitzontal) d'un procés de negoci es pot oferir sobre una infraestructura *cloud*.

Si bé els avantatges són evidents i molts actors d'aquesta tecnologia la basen principalment en l'abaratiment dels costos de servei, comencen a haver-hi opinions en contra (http://deepvalue.net/ec2-is-380-more-expensive-than-internal-cluster/) que consideren que un *cloud* públic potser no és l'opció més adequada per a determinat tipus de serveis/infraestructura. Entre altres aspectes, els negatius poden ser la fiabilitat en la prestació de servei, seguretat i privadesa de les dades/informació en els servidors externs, *lock-in* de dades en la infraestructura sense possibilitat d'extreure-les, estabilitat del proveïdor (com a empresa/negoci) i posició de força al mercat no subjecte a la variabilitat del mercat, colls d'ampolla en les transferències (empresa/proveïdor), rendiment no predictible, temps de resposta a incidents/accidents, problemes derivats de la falta de maduresa de la tecnologia/infraestructura/gestió, acords de serveis (SLA) pensats més per al proveïdor que per a l'usuari, etc. No obstant això, és una tecnologia que té els seus avantatges i que amb l'adequada planificació i presa de decisions, valorant tots els factors que influeixen en el negoci i escapant de conceptes superficials (tots el tenen, tots l'utilitzen, baix cost, expansió il·limitada, etc.), pot ser una elecció adequada per als objectius de l'empresa, el seu negoci i la prestació de serveis en IT que necessita o que forma part de les seves finalitats empresarials.

És molt àmplia la llista de proveïdors de serveis *cloud* en les diferents capes, però d'acord amb la informació de Synergy Research Group el 2015*, els líders al mercat d'infraestructures *cloud* són:

***https://www.srgresearch.com/articles/2015-review-shows-110-billion-cloud-market-growing-28-annually**

1) Públic IaaS/PaaS: mercat dominat per Amazon i Microsoft (aproximadament el 50%)

2) Privat i Híbrid: IBM i Amazon (aproximadament el 45%)

3) SaaS: Salesforce i Microsoft (aproximadament el 30%)

4) UCaaS (Unified Communications as a Service): Cisco i Citrix (aproximadament el 18%)

5) Infraestructura (hw & sw): Públic: Cisco-HPE (aproximadament el 30%) i Privat: HPE-Cisco (aproximadament el 18%)

El volum de negoci (segons aquest informe) creix un 28% anualment i va aconseguir els 110.000 milions de dòlars. Això significa un canvi apreciable

en relació amb les dades del 2014, en què es pot observar una variabilitat de l'oferta molt alta*.

*https://www.srgresearch.com/articles/amazon-salesforce-and-ibm-lead-cloud-infrastructure-service-segments

Quant a plataformes per a desplegar *clouds* amb llicències GPL, Apache i BSD (o similars), podem comptar entre les més referenciades (i per ordre alfabètic):

1) AppScale: és una plataforma que permet als usuaris desenvolupar i executar/emmagatzemar les pròpies aplicacions basades en Google AppEngine i pot funcionar com a servei o en local. Es pot executar sobre AWS EC2, Rackspace, Google Compute Engine, Eucalyptus, Openstack, CloudStack, així com sobre KVM i VirtualBox i suporta Python, Java, Go i plataformes PHP Google AppEngine. http://www.appscale.com/

2) Cloud Foundry: és una plataforma *open source* PaaS desenvolupada per VMware escrita bàsicament en Ruby and Go. Pot funcionar com a servei i també en local. http://cloudfoundry.org/

3) Apache CloudStack: dissenyada per a gestionar grans xarxes de màquines virtuals com IaaS. Aquesta plataforma inclou totes les característiques necessàries per al desplegament d'un IaaS: CO (*compute orchestration*), *Network-as-a-Service*, gestió d'usuaris/comptes, API nativa, *resource accounting* i UI millorada (*User Interface*). Suporta els *hypervisors* més comuns, gestió del *cloud* via web o CLI i una API compatible amb AWS EC2/S3 que permeten desenvolupar *clouds* híbrids. http://cloudstack.apache.org/

4) Eucalyptus: plataforma que permet construir *clouds* privats compatibles amb AWS. Aquest programari permet aprofitar els recursos de còmput, xarxa i emmagatzematge per a oferir autoservei i desplegament de recursos de *cloud* privat. Es caracteritza per la simplicitat en la instal·lació i estabilitat en l'entorn amb una alta eficiència en l'ús dels recursos. https://www.eucalyptus.com/

5) Nimbus: és una plataforma que una vegada instal·lada sobre un clúster, proporciona IasS per a la construcció de *clouds* privats o de comunitat i pot ser configurada per a suportar diferents virtualitzacions, sistemes de cues o Amazon EC2. http://www.nimbusproject.org/

6) OpenNebula: és una plataforma per a gestionar tots els recursos d'un centre de dades i permetre la construcció de IasS privats, públics i híbrids. Proveeix d'una gran quantitat de serveis, prestacions i adaptacions que han permès que sigui una de les plataformes més difoses en l'actualitat. http://opennebula.org/

7) OpenQRM: és una plataforma per al desplegament de *clouds* sobre un centre de dades heterogènies. Permet la construcció de *clouds* privats, públics i híbrids amb IaaS. Combina la gestió del la CPU/emmagatzematge/xarxa per a oferir serveis sobre màquines virtualitzades i permet la integració amb recursos remots o altres *clouds*. http://www.openqrm-enterprise.com/index-2.html

8) OpenShift: aposta important de la companyia RH i és una plataforma (versió Origin) que permet prestar servei *cloud* en modalitat PasS. OpenShift suporta l'execució de binaris que són aplicacions web tal com s'executen en

RHEL, per la qual cosa permet un gran nombre de llenguatges i *frameworks*. https://www.openshift.com/products/origin

9) OpenStack és una arquitectura programari que permet el desplegament de *cloud* en la modalitat de IaaS. Es gestiona mitjançant una consola de control via web que permet la provisió i control de tots els subsistemes i l'aprovisionament dels recursos. El projecte iniciat (2010) per Rackspace i NASA és actualment gestionat per OpenStack Foundation i hi ha més de 200 companyies adherides al projecte, entre les quals es troben les grans proveïdores de serveis *cloud* públics i desenvolupadores de SW/HW (ATT, AMD, Canonical, Cisco, Dell, EMC, Ericsson, HP, IBM, Intel, NEC, Oracle, RH, SUSE Linux, VMware, Yahoo, entre d'altres). És una altra de les plataformes més referenciades. http://www.openstack.org/

10) PetiteCloud: és una plataforma programari que permet el desplegament de *clouds* privats (petits) i no orientats a dades. Aquesta plataforma pot ser utilitzada sola o en unió amb altres plataformes *cloud* i es caracteritza per la seva estabilitat/fiabilitat i facilitat d'instal·lació. http://www.petitecloud.org

11) oVirt: si bé no es pot considerar una plataforma *cloud*, oVirt és una aplicació de gestió d'entorns virtualitzats. Això significa que es pot utilitzar per a gestionar els nodes HW, l'emmagatzematge o la xarxa i desplegar i monitorar les màquines virtuals que s'estan executant al centre de dades. Forma part de RH Enterprise Virtualization i és desenvolupada per aquesta companyia amb llicència Apache. http://www.ovirt.org/

2.1. Opennebula

Tenint en compte les opinions a [19] [20], realitzarem la nostra prova de concepte sobre infraestructura *cloud* a OpenNebula. Probablement, una de les maneres més fàcils de provar les funcionalitats d'OpenNebula i fer un desplegament immediat és utilitzar el Sandbox virtualitzat i preconfigurat que proveeix el desenvolupador. Per això, en el nostre cas, utilitzarem la versió de VirtualBox que està sobre una màquina virtual CentOS 7, amb OpenNebula 5.0.0, fent servir QEMU per executar les màquines virtuals. A més, poden afegir qualsevol altre node físic, amb alguns dels *hipervisors* suportats per OpenNebula, per construir un *cloud* a petita escala. En aquesta prova l'usuari podrà connectar-se al seu OpenNebula *cloud* privat, mirar els recursos gestionats i llançar noves instàncies de màquines virtuals sense les dificultats de configurar la infraestructura física. Els requeriments seran 1 GB RAM lliure, 10 GB d'espai lliure de disc, un processador / SO de 64 bits i les *Virtualization Extensions* de Virtualbox instal·lades.

Documentació OpenNebula
http://docs.opennebula.org/5.0/

1) Descarreguem el *virtual appliance* en format OVA des de la següent pàgina web: http://opennebula.org/tryout/sandboxvirtualbox/.

2) Importem l'*appliance* des de VirtualBox: *File-> Import Appliance*. Posem en marxa la màquina virtual i sortirà *one-sandbox login:*. Ens connectem com a *root* i *passwd* "Opennebula".

3) Canviem el teclat amb `localectl set-keymap es` i recreem les cau dels repositoris `yum makecache fast`.

4) Com farem servir la mateixa màquina virtual per a la prova, instal·larem un entorn gràfic mínim fent:

```
yum groupinstall "X Window System"
yum groupinstall "Fonts"
yum install kde-workspace
yum install gdm firefox
unlink /etc/systemd/system/default.target
ln -sf /lib/systemd/system/graphical.target \
/etc/systemd/system/default.target
reboot
```

(o també `systemctl isolate graphical.target`)

5) Accedim a l'escriptori (es pot configurar el teclat des de les opcions de l'entorn KDE); a continuació, obrim Firefox i ens connectem a *127.0.0.1:9869*; i sortirà la pantalla d'OpenNebula. Introduïm com a usuari *oneadmin* i, com a *passwd*, "Opennebula". Així, accedirem a l'escriptori d'OpenNebula.

6) A partir d'aquest punt veurem el *dashboard* d'OpenNebula i, després de revisar els paràmetres i ajustar-los, podrem crear una màquina virtual (signe + en VM) amb el *template* que hi ha disponible. Un cop creada, podrem accedir-hi i realitzar totes les accions necessàries, i accedir o tornar a *Dashboard* i veure els recursos consumits.

Les figures 6, 7 i 8 mostren un exemple del *Dashboard* i l'execució de la màquina virtual en el mateix servidor. A la figura 8 es poden veure en CLI les ordres útils (en el requadre groc) per obtenir informació dels recursos i d'aquells que estan funcionant (cal tenir en compte que s'han de fer com a usuari *oneadmin*).

Figura 6

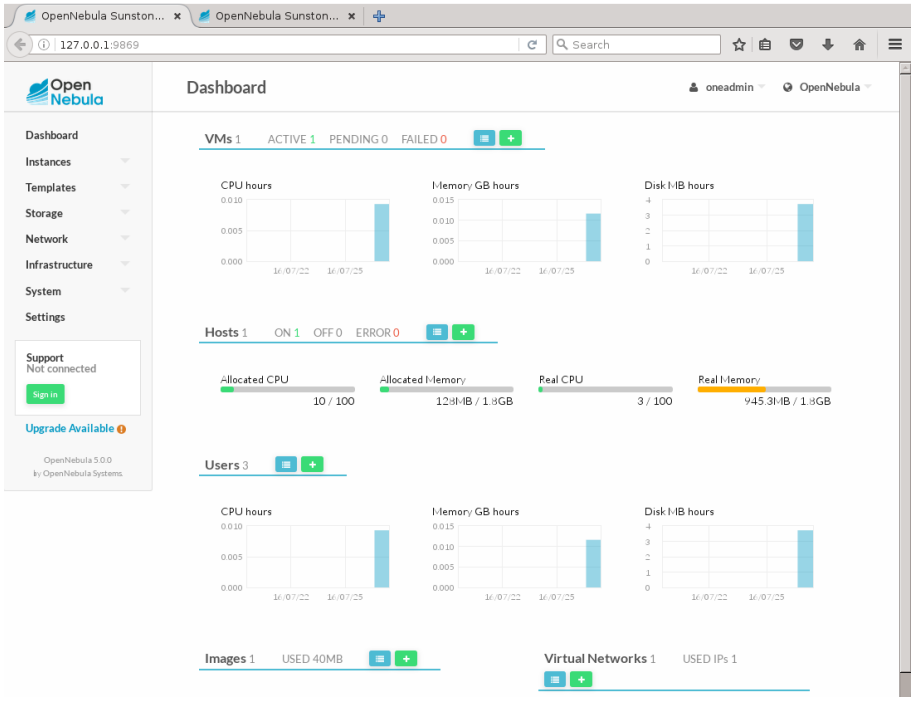


Figura 7

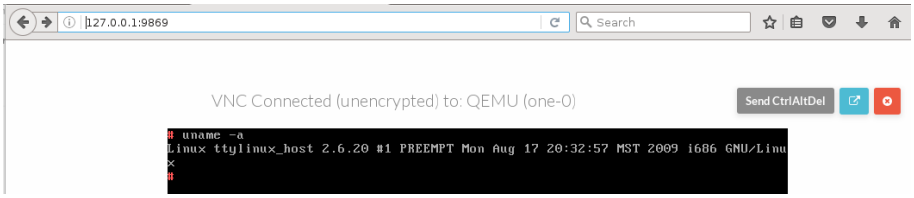
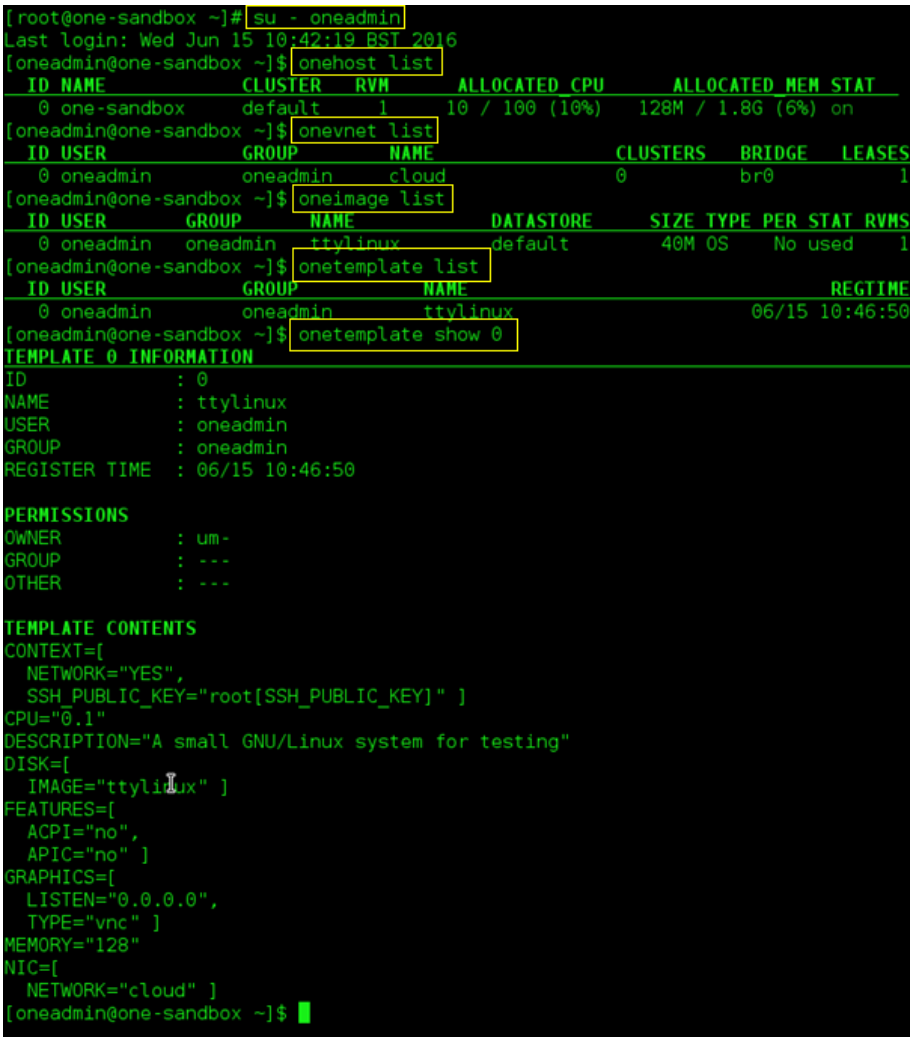


Figura 8



La instal·lació de Debian, per exemple del *Frontend*, es pot fer fàcilment amb (veure [ONDe]):

1) `wget -q -O- http://downloads.opennebula.org/repo/Debian/repo.key | apt-key add -`

2) `echo "deb http://downloads.opennebula.org/repo/5.0/Debian/8 stable opennebula"> /etc/apt/sources.list.d/opennebula.list`

3) `apt-get update`

4) `apt-get install opennebula opennebula-sunstone opennebula-gate opennebula-flow`

Els paquestes que s'instal·laran són: *opennebula-common* (arxius comuns), *ruby-opennebula* (API Ruby), *libopennebula-java* (API Java), *libopennebula-java-doc* (documentació), *opennebula-node* (prepara un node com a opennebula-node), *opennebula-Sunstone* (GUI), *opennebula-tools* (CLI), *opennebula-gate* (comunicació entre VMs i OpenNebula), *opennebula-flow* (gestiona els serveis i l'elasticitat) i *opennebula* (Daemon).

5) Executem `/usr/share/one/install_gems` (instal·la paquets addicionals).

6) Si l'entorn és de producció, és recomanable substituir SQLite per MySQL (veure [ONDe]).

7) Fem `su - oneadmin` i després

```
echo "oneadmin:mypassword" > .one/one_auth
```

per canviar el *passwd* generat automàticament.

8) Executem `service opennebula start`

9) Executem `service opennebula-sunstone start`

10) Provem el servei executant `oneuser show` i ens donarà alguna cosa com ara:

```
USER 0 INFORMATION
ID                              : 0
NAME                            : oneadmin
GROUP                           : oneadmin
PASSWORD                        : e6b47f32bdbcdb96aca2df6c8ccc190df3020828
AUTH_DRIVER                     : core
ENABLED                         : Yes
USER TEMPLATE
TOKEN_PASSWORD="f605ba14857d6e4ee28bce2735c832225727bb7a"
RESOURCE USAGE & QUOTAS
```

11) Després, ens connectem a la màquina i al port 9869, en el nostre cas http://srv.nteum.org:9869, i introduïm com a usuari *oneadmin* i, com a *passwd*, el canviat anteriorment. D'aquesta manera, tornarem a veure una imatge similar a la del *Dashboard* que s'ha mostrat abans.

12) El següent pas és configurar un node (on s'aprovisionaran les màquines virtuals). No té més dificultats i es pot fer seguint la guia des d'[ONNo].

13) Per fer una prova de concepte sobre la mateixa màquina i atès que és una màquina virtual sobre VirtualBox i no podem utilitzar KVM, utilitzarem QEMU. Per a això instal·lem

```
install libvirt-bin libvirt0 libvirt-daemon libvirt-clients
apt-get install qemu qemu-kvm qemu-system-x86 qemu-utils
```

14) A continuació, haurem de modificar els fitxers de configuració per afegir:

```
/etc/libvirt/qemu.conf
user = "oneadmin"
group = "oneadmin"
```

```
dynamic_ownership = 0
/etc/libvirt/libvirtd.conf
listen_tls = 0
listen_tcp = 1
mdns_adv = 0
unix_sock_group = "oneadmin"
unix_sock_rw_perms = "0777"
auth_unix_ro = "none"
auth_unix_rw = "none"
/etc/one/oned.conf
        VM_MAD = [
                  NAME              = "kvm",
                  SUNSTONE_NAME     = "KVMQemu",
...
                  TYPE = "qemu",
                  ...
```

15) Reiniciem la màquina i arrenquem els *daemons* d'*opennebula* i *opennebula-sunstone*. Entrem a la pàgina web (http://srv.nteum.org:9869/) i des de *Storage* -> *App* descarreguem una imatge de prova (per exemple, la ID = 0 ttylinux -- kvm) indicant-li que volem que passi a OpenNebula des del MarketPlace. Quan hagi finalitzat la descàrrega, la veurem a *Storage* -> *Images* com a *Ready* i també com a *template*.

16) Després, a Infraestructura, crearem un *host* seleccionant com a controlador el nom que li hem posat (per exemple, nosaltres li hem donat com a nom KVMQemu) i com a màquina, *localhost*. L'haurem de veure en l'estatus *ON* i amb recursos assignats.

17) Finalment, podrem crear una màquina virtual (des d'*Instances* o des del *DashBoard*) amb el *template* carregat i sobre la màquina creada. La veurem en estat *running* i podrem accedir a una consola per verificar el seu funcionament. S'ha de tenir en compte que no li hem assignat una xarxa, amb la qual cosa la màquina no tindrà dispositiu de xarxa, però, com a prova, sí que podrem veure'n el desplegament i accedir a la consola.

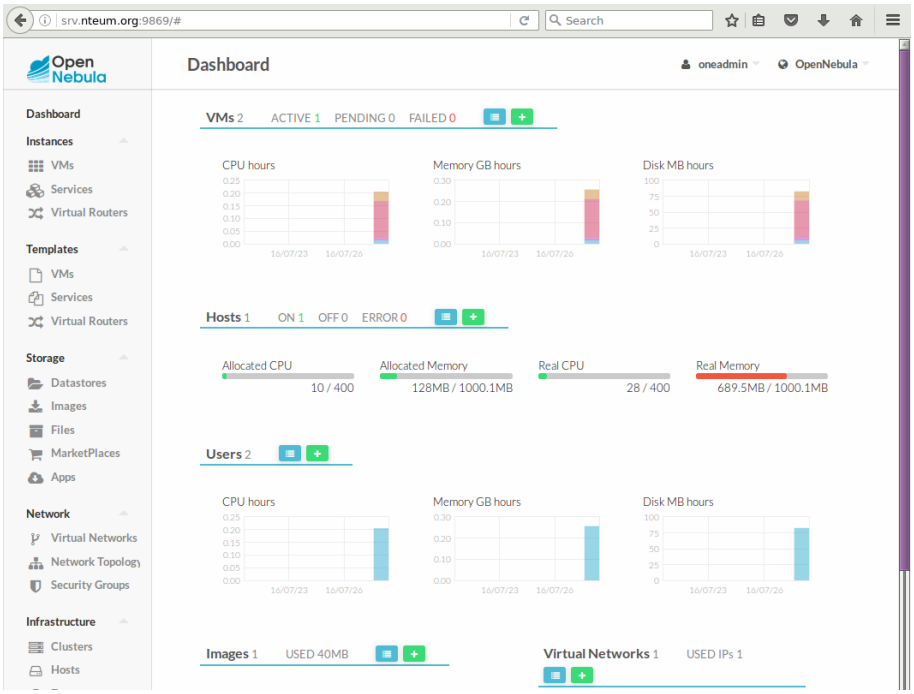
18) Per afegir una xarxa, haurem de configurar el sistema com a *bridge* modificant l'arxiu */etc/network/interfaces*:

```
auto lo br0
iface lo inet loopback
allow-hotplug eth0 eth1
iface eth0 inet dhcp
iface br0 inet static
        address 172.16.1.1
        network 172.16.1.0
        netmask 255.255.255.0
        broadcast 172.16.1.255
        #gateway 192.168.0.1
        bridge_ports eth1
        bridge_fd 9
        bridge_hello 2
        bridge_maxage 12
        bridge_stp off
```

Després, seleccionem Network -> VirtualNetwork al *DashBoard* i generem la xarxa associada a *br0* amb els paràmetres corresponents. A continuació, es pot crear una nova màquina virtual i assignar la xarxa verificant que, en arrencar, tindrà la IP i la resta de paràmetres de xarxa assignats.

La figura 9 mostra l'execució d'aquesta última màquina virtual sobre el servidor configurat (http://srv.nteum.org:9869).

Figura 9



Com s'ha pogut comprovar, la instal·lació està prou automatitzada però s'ha de tenir en compte que és una infraestructura complexa i que s'ha de dedicar temps i anàlisi a determinar les causes per les quals es produeixen els errors i solucionar-los. Existeix una gran quantitat de documentació i llocs a Internet, però recomanem començar per les fonts [on1] i [on2].

En aquest apartat, s'ha vist una prova de concepte funcional però OpenNebula és molt extens i flexible, i permet gran quantitat d'opcions/extensions. Amb OpenNebula MarketPlace (http://marketplace.opennebula.systems/appliance), es podran trobar i descarregar imatges de màquines virtuals creades per OpenNebula i llistes per a posar-les en funcionament (són les que es veuen des del MarketPlace de la interfície web), però en descarregar-les sobre el nostre servidor les podrem utilitzar sempre que les necessitem i no s'hauran de descarregar cada vegada (no s'han de descomprimir ni s'ha de fer res, s'han d'utilitzar tal com estan).

3. DevOps

Tal com hem comentat a l'inici d'aquest apartat, Devops es pot considerar, en paraules dels experts, un moviment tant en l'aspecte professional com en l'aspecte cultural del món TI. Si bé no hi ha totes les respostes encara, un administrador trobarà diferents "comunitats" (i ell mateix formarà part d'alguna), que tindran noves necessitats de desenvolupament i producció de serveis/productes dins del món TI i amb les premisses de "més ràpid", "més eficient", "de major qualitat" i totalment adaptable als "diferents entorns". És a dir, el grup de treball haurà de trencar les barreres entre els departaments d'una empresa i permetre que un producte passi ràpidament des del departament de recerca al de disseny, després al de desenvolupament + producció, després al de test + qualitat i finalment, a vendes, i amb les necessitats d'eines que permetin desplegar totes aquestes activitats en cadascuna de les fases i portar el control de tots pels responsables de cadascun dels àmbits, inclosos els funcionals i els de l'organització/directius. És per això que un administrador necessitarà eines de gestió, control, desplegament, automatització i configuració. Una llista (curta) de les eines que podríem considerar d'acord amb els nostres objectius de llicència GPL-BSD-Apache o similars (és interessant el lloc http://devs.info/, ja que permet accedir a la major part d'eines/entorns/documentació per a programadors i una llista més detallada -que inclou SW propietari- es pot trobar a [NR]):

1) **Linux**: Ubuntu|Debian^{v}, Fedora^{v}|CentOS|SL

2) **IasS**: Cloud Foundry, OpenNebula^{v}, OpenStack

3) **Virtualització**: KVM^{v}, Xen, VirtualBox^{v}, Vagrant^{v}

4) Contenidors: LXC^{v}, Docker^{v}

5) Instal·lació SO: Kickstart i Cobbler (rh), Preseed|Fai^{v} (deb), Rocks^{v}

6) Gestió de la configuració: Puppet^{v}, Chef^{v}, CFEngine, SaltStack, Juju, bcfg2, mcollective, fpm (effing), Ansible^{v}

7) Servidors web i acceleradors: Apache^{v}, nginx^{v}, varnish, squid^{v}

8) BD: MySQL^{v}|MariaDB^{v}, PostgreSQL^{v}, OpenLDAP^{v}, MongoDB^{v}, Redis

9) Entorns: Lamp, Lamr, AppServ, Xampp, Mamp

10) Gestió de versions: Git^{v}, Subversion^{v}, Mercurial^{v}

11) Monitoratge/supervisió: Nagios^{v}, Icinga, Ganglia^{v}, Cacti^{v}, Monin^{v}, MRTG^{v}, XYmon^{v}

12) Misc: pdsh, ppsh, pssh, GNUparallel^{v}, nfsroot, Multihost SSH Wrapper, lldpd, Benchmarks^{v}, Biblioteques^{v}

13) Supervisió: Monit^{v}, runit, Supervisor^{v}, Godrb, BluePill-rb, Upstart, Systemd^{v}

14) Security: OpenVas^{v}, Tripwire^{v}, Snort^{v}

15) Desenvolupament i test: Jenkins, Maven, Ant, Gradle, CruiseControl, Hudson

16) Desplegament i *workflow*: Capistrano

17) Servidors d'aplicacions: JBoss, Tomcat, Jetty, Glassfish,

18) Gestió de *logs*: Rsyslog^{v}, Octopussy^{v}, Logstash

Excepte les que són molt orientades a desenvolupament d'aplicacions i serveis, en les dues assignatures se n'ha vist (o es veuran dins d'aquest apartat) gran part (marcades amb ^{v}) i sens dubte, amb els coneixements obtinguts, l'alumne podrà ràpidament desplegar totes aquelles que necessiti i que no s'han tractat en aquests cursos.

A continuació, veurem algunes eines (molt útils en entorns DevOps) més orientades a generar automatitzacions en les instal·lacions o generar entorns aïllats de desenvolupaments/test/execució que permeten de manera simple i fàcil (i sense pèrdues de prestacions/rendiment) disposar d'eines i entorns adequats a les nostres necessitats.

3.1. Linux Containers, LXC

LXC (Linux Containers) és un mètode de virtualització en un nivell del sistema operatiu per a executar múltiples sistemes Linux aïllats (anomenats *contenidors*) sobre un únic *host*. El *kernel* de Linux utilitza `cgroups` per a poder aïllar els recursos (CPU, memòria, E/S, *network*, etc.), la qual cosa no requereix iniciar cap màquina virtual. *Cgroups* també proveeix aïllament dels espais de noms per a aïllar per complet l'aplicació del sistema operatiu, inclosos arbre de processos, xarxa, ID d'usuaris i sistemes d'arxius muntats. Mitjançant una API molt potent i eines simples, permet crear i gestionar contenidors de sistema o aplicacions. LXC utilitza diferents mòduls del *kernel* (ipc, uts, mount, pid, network, user) i d'aplicacions (Apparmor, SELinux profiles, Seccomp policies, Chroots -pivot_root- i Control groups -cgroups-) per a crear i gestionar els contenidors. Es pot considerar que LXC és a mig camí d'un "potent" `chroot` i una màquina virtual, i ofereix un entorn molt proper a un Linux estàndard però sense necessitat de tenir un kernel separat. Això és més eficient que utilitzar virtualització amb un `hypervisor` (KVM, VirtualBox) i més ràpid de (re)iniciar, sobretot si s'estan fent desenvolupaments i és necessari fer-ho freqüentment, i el seu impacte en el rendiment és molt baix (el contenidor no ocupa recursos) i tots es dedicaran als processos que s'estiguin executant.

La instal·lació es fa mitjançant l'ordre `apt-get install lxc lxctl` i es poden instal·lar altres paquets addicionals que són opcionals (per exemple, `bridge-utils, libvirt-bin, debootstrap`), no obstant això, si volem que els contenidors tinguin accés a la xarxa, amb IP pròpia és convenient instal·lar `apt-get install bridge-utils`. A Debian Jessie no s'ha de fer res més, però en versions anteriors s'han de fer ajustos amb el directori *cgroups* (consultar la documentació a [LXC1, LXC2, LXC3]) A partir d'aquest moment, podem verificar la instal·lació amb `lxc-checkconfig`, que donarà una sortida similar a:

```
Kernel configuration not found at /proc/config.gz; searching...
Kernel configuration found at /boot/config-3.16.0-4-amd64
--- Namespaces ---
Namespaces: enabled
Utsname namespace: enabled
Ipc namespace: enabled
Pid namespace: enabled
User namespace: enabled
Network namespace: enabled
Multiple /dev/pts instances: enabled

--- Control groups ---
Cgroup: enabled
Cgroup clone_children flag: enabled
Cgroup device: enabled
Cgroup sched: enabled
Cgroup cpu account: enabled
Cgroup memory controller: enabled
Cgroup cpuset: enabled

--- Misc ---
Veth pair device: enabled
Macvlan: enabled
Vlan: enabled
File capabilities: enabled

Note : Before booting a new kernel, you can check its configuration
usage : CONFIG=/path/to/config /usr/bin/lxc-checkconfig
```

Si sorgeixen opcions deshabilitades, s'ha de mirar la causa i solucionar-ho (si bé en algunes configuracions algunes opcions poden estar-ho).

En el directori */usr/share/lxc/templates* hi ha un conjunt de plantilles per crear diferents contenidors, si bé alguna pot tenir necessitats específiques. Per crear un contenidor Debian, per exemple, executem:

```
lxc-create -n debian8 -t debian -- -r jessie
```

Veurem, després d'uns instants d'instal·lació, que a */var/lib/xlc*, tenim un directori que es diu *debian8* (el nom del contenidor) i fa 287 MB de mida. És important estar atent a la informació que dóna la creació del contenidor, ja que allà ens informarà de l'usuari i del *passwd* (especialment del *root*). Les comandes més útils per treballar amb el contenidor seran:

1) `lxc-ls` per llistar els contenidors i `lxc-info` per obtenir informació sobre els contenidors disponibles i aquells que s'estan executant (amb el paràmetre `--active`).

2) `lxc-start -n debian8 -d` per posar en marxa un contenidor a *background* (`-d = daemon`).

3) `lxc-console -n debian8` per connectar-se a la consola del contenidor (amb el *passwd* generat pel *root* durant la seva creació). És recomanable verificar */etc/issue.net* (o qualsevol altre arxiu que indiqui la versió). Farem servir `uname -a` per verificar el nucli i canviar el *passwd* del *root*. Per tornar al sistema operatiu original haurem de pressionar les tecles Ctrl+a i, després, 'q'.

4) `lxc-stop -n debian8` per a l'execució del contenidor.

5) `lxc-destroy -n debian8` per eliminar un contenidor.

6) `lxc-clone debian8 debian8V2` per clonar un contenidor.

7) Per muntar un directori del *host* principal al contenidor, es pot afegir a */var/lib/lxc/containername/config* la sentència

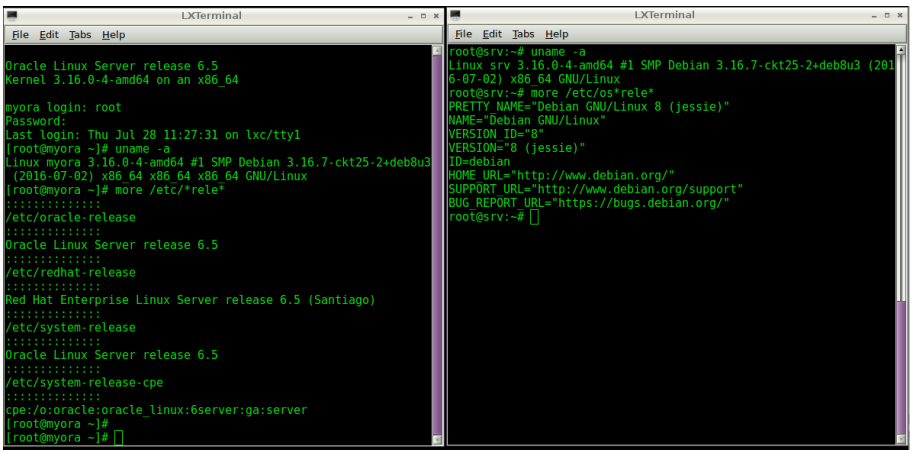
lxc.mount.entry = /usr/bin mnt none ro, bind 0.0,

que muntarà el */usr/bin* de *host* en el directori */mnt* del contenidor.

8) Si volem instal·lar un contenidor d'una branca diferent de la del *host* (p. ex., Oracle basat en Red Hat), s'han d'instal·lar una sèrie de dependències, com ara els paquets *rpm* i *yum* per a Debian (`apt-get install rpm yum`). Després, podrem crear el contenidor amb `lxc-create -n myoracle -t oracle`, i veurem que es descarrega una gran quantitat de paquets i que el directori (una vegada s'ha completat la instal·lació) ocupa una mica més que la distribució Debian prèviament instal·lada (448 MB).

La figura 10 mostra el *host* a la dreta (Debian 8.5) i el contenidor a l'esquerra (Oracle 6.5), funcionant sobre el mateix nucli, com es pot veure en l'execució de la comanda `uname -a` des de cadascun d'ells.

Figura 10



És interessant instal·lar el paquet *lxctl* (`apt-get install lxctl`) per gestionar de manera més simple els contenidors LXC.

Si es desitja crear un Ubuntu des d'un *host* Debian amb el *template* proporcionat abans, s'hauran d'instal·lar les *keys* d'Ubuntu amb

```
apt-get install ubuntu-archive-keyring
```

La configuració de la xarxa és simple gràcies al paquet *bridge-utils* (es poden seguir els passos indicats a [23-25]), però, per simplificar, creem sobre el *host* un dispositiu de *Bridge*, a */etc/network/interfaces* (en aquest cas, el contenidor està sobre una màquina virtualitzada en VBox on el primer adaptador està com a *Bridged*):

```
auto lo br0
iface lo inet loopback

iface br0 inet static
        address ip_dentro de la red
        netmask 255.255.255.0
        gateway gw_dentro_de_red
        bridge_ports eth0
        bridge_maxwait 0
        bridge_fd 0
```

Després, a /var/lib/lxc/container_name/config modifiquem (en el nostre cas és un contenidor Ubuntu creat anteriorment):

```
lxc.network.type = veth
lxc.network.flags = up
lxc.network.link = br0
#lxc.network.ipv4 = 158.109.74.24/24
lxc.network.hwaddr = 00:1E:1E:1a:00:00
#lxc.network.ipv4.gateway = 158.109.64.1
```

En aquest cas (i per evitar el temps d'espera del dhcp), no li indiquem la IP ni el GW, i les assignarem de manera estàtica dins del contenidor. Arrenquem el contenidor i modifiquem /etc/network/interfaces amb:

```
auto lo
iface lo inet loopback
auto eth0
iface eth0 inet static
address ip_dentro_red_del_host
netmask 255.255.255.0
gateway ip_gw_igual_host
```

Reiniciem el servei de xarxa i verifiquem la connexió entre el contenidor i el *host* (provem un *ping* o una connexió *ssh*).

Tanmateix, si es desitja una xarxa **ALL-Nat**, és a dir, VirtualBox configurat en NAT, en el primer adaptador (i Debian amb dhcp sobre *eth0*), i volem que el contenidor tingui sortida cap a fora i cap a NAT, el millor és crear una xarxa virtual. Per fer això, instal·lem `apt-get install libvirt-bin ebtables dnsmasq`. Reiniciem i podrem veure el dispositiu per defecte amb

`virsh net-info default` (en el nostre cas, *vrbr0*); els paràmetres es poden modificar a */etc/libvirt/qemu/networks/default.xml*). Tot seguit, iniciem la xarxa amb `virsh net-start default` i veurem el dispositiu en marxa amb `ip address`. A continuació, editem */var/lib/lxc/container_name/config* i configurem:

```
[...]
lxc.network.type = veth
lxc.network.flags = up
lxc.network.link = virbr0
lxc.network.hwaddr = 00:FF:AA:00:00:01
lxc.network.ipv4 = 0.0.0.0/24
[...]
```

Posem en marxa el contenidor i verifiquem la connexió externa amb l'ordre `ping 8.8.8.8`

Si es vol donar als contenidors la mateixa IP (mapejats per la MAC), es pot fer en el fitxer anterior:

```
<dhcp>
  <range start="192.168.122.100" end="192.168.122.254" />
  <host mac="00:FF:AA:00:00:01" name="foo.example.com" ip="192.168.122.101" />
  <host mac="00:FF:AA:00:00:02" name="bar.example.com" ip="192.168.122.102" />
</dhcp>
```

Si, a més, es vol crear el dispositiu virtual quan el *host* arrenca, es pot fer

```
virsh net-autostart default; virsh net-info default
```

S'ha d'anar amb compte quan s'inicia el contenidor sense `-d`, ja que no hi ha manera de sortir (en la versió actual, el 'Ctrl+a q' no funciona). Cal iniciar sempre els contenidors en *background* (amb `-d`) tret que necessiti depurar perquè el contenidor no arrenca. Una altra consideració que cal tenir és que l'ordre `lxc-halt` executarà el `telinit` sobre l'arxiu */run/initctl* si existeix i apagarà el *host*, per la qual cosa cal apagar-lo amb `lxc-stop` i no fer un `shutdown -h now` dins del contenidor, ja que també apagarà el *host*. També existeix un panell gràfic (si bé el desenvolupament no ha estat actualitzat últimament) per a gestionar els contenidors via web, disponible a l'adreça http://lxc-webpanel.github.io/install.html (en Debian hi ha problemes amb l'apartat de xarxa; alguns els soluciona https://github.com/vaytess/lxc-web-panel/tree/lwp-backup. Per a això, hem de clonar el lloc

```
git clone https://github.com/vaytess/lxc-web-panel.git
```

i després reemplaçar el directori obtingut per */srv/lwp* -renombrar aquest abans- i fer un `/etc/init.d/lwp restart`).

https://linuxcontainers.org/downloads/

3.2. Docker

Docker és una plataforma oberta per al desenvolupament, empaquetatge i execució d'aplicacions de manera que es puguin posar en producció o com-

partir més ràpidament separant aquestes de la infraestructura, de manera que sigui menys costós en recursos (bàsicament espai de disc, CPU i engegada) i que estigui tot preparat per al següent desenvolupador amb tot el que es va posar però res de la part d'infraestructura. Això significarà menys temps per a provar i accelerarà el desplegament escurçant de manera significativa el cicle entre que s'escriu el codi i passa a producció. Docker empra una plataforma (contenidor) de virtualització lleugera amb fluxos de treball i eines que l'ajuden a administrar i implementar les aplicacions i proporcionar una manera d'executar gairebé qualsevol aplicació en forma segura en un contenidor aïllat. Aquest aïllament i seguretat permeten executar molts contenidors de manera simultània en el *host* i, atesa la naturalesa (lleugera) dels contenidors, tot això s'executa sense la càrrega addicional d'un *hypervisor* (que seria l'altra manera de gestionar aquestes necessitats compartint la VM), la qual cosa significa que podem obtenir millors prestacions i utilització dels recursos. És a dir, amb una VM cada aplicació virtualizada inclou no només l'aplicació, que poden ser desenes de Mbytes -binaris + biblioteques-, sinó també el sistema operatiu *guest*, que poden ser diversos Gbytes; en canvi, en Docker el que es denomina DE (Docker Engine) només comprèn l'aplicació i les seves dependències, que s'executen com un procés aïllat a l'espai d'usuari del SO *host*, compartint el *kernel* amb altres contenidors. Per tant, té el benefici de l'aïllament i l'assignació de recursos de les màquines virtuals, però és molt més portàtil i eficient transformant-se en l'entorn perfecte per a donar suport al cicle de vida del desenvolupament de programari, test de plataformes, entorns, etc.[d1]

L'entorn està format per dos grans components: **Docker** [d2] (és la plataforma de virtualització –contenidor) i **Docker Hub** [d4] (una plataforma SasS que permet obtenir i publicar/gestionar contenidors ja configurats). En la seva arquitectura, Docker utilitza una estructura client-servidor en què el client (CLI **Docker**) interactua amb el *daemon* Docker, que fa el treball de la construcció, execució i distribució dels contenidors de Docker. Tant el client com el *daemon* es poden executar en el mateix sistema, o es pot connectar un client a un *daemon* de Docker remot. El client Docker i el servei es comuniquen mitjançant *sockets* o una API RESTful.

No s'ha de confondre una vella aplicació anomenada *docker* a Debian amb **la Plataforma Docker** (per a evitar confusions a Debian es diu docker.io). Per afegir el repositori a Jessie fem:

Més detalls sobre l'arquitectura a https://docs.docker.com /engine/understanding-docker/.

1) `apt-get install apt-transport-https ca-certificates`

2) `apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80\`
`--recv-keys 58118I89F3A912897C070ADBF76221572C52609D`

3) `echo "deb https://apt.dockerproject.org/repo debian-jessie\`
`main» /etc/apt/sources.list.d/docker.list`

4) `apt-get update`

5) Es pot verificar que veiem el repositori amb
`apt-cache policy docker-engine`.

6) Per instal·lar-ho `apt-get install docker-engine`.

7) Iniciar el *daemon* `service docker start`

8) Verificar que està instal·lat correctament `docker run hello-world`

9) Instal·lar un contenidor Ubuntu `docker run -it ubuntu bash` (com el contenidor no existeix ho descarregarà i ho executarà per la qual cosa veurem el *prompt* del *bash* d'Ubuntu). Si fem `cat /etc/os-release`, veurem que estem a Ubuntu. 16.04 LTS (Xenial Xerus). Amb `exit` (o `Crtl-D`) sortim a l'*host* novament.

Les ordres per a iniciar-se (a més de les que hem vist) són:

1) `docker`: mostra totes les opcions.

2) `docker images`: fa una llista de les imatges localment.

3) `docker search` *patró*: busca contenidors/imatges que tinguin aquest patró.

4) `docker pull` *nom*: obté imatges del *hub*. El nom pot ser <username>/<repository> per als particulars.

5) `docker run name cmd`: executarà el contenidor i a dins, l'ordre indicada per `cmd`.

6) `docker ps -l`: permet obtenir l'ID i informació d'una imatge.

7) `docker commit ID name`: actualitza la imatge amb què s'ha instal·lat fins a aquest moment (amb 3-4 nombres de l'ID és suficient).

8) `docker inspect`: mostra les imatges corrent (similar a docker ps).

9) `docker push`: salva la imatge en el *hub* (hem de registrar-nos primer) i està disponible per a altres usuaris/*hosts* que la vulguin instal·lar amb tot el que hem configurat dins.

10) `docker cp`: copia arxius/directoris des del contenidor al *host*.

11) `docker export/import`: exporta/importa el contenidor en un arxiu tar.

12) `docker history`: mostra la història d'una imatge.

13) `docker info`: mostra informació general (imatges, directoris, etc.).

14) `docker kill`: per a l'execució d'un contenidor.

15) `docker restart`: reinicia un contenidor.

16) `docker rm`: elimina un contenidor.

17) `docker rmi`: elimina imatges.

18) `docker start/stop`: inicia/atura un contenidor.

3.3. Instal·lació d'un servidor Apache sobre un contenidor Docker

Per a aquest objectiu, començarem amb un contenidor base, instal·larem Apache i, després, hi accedirem des del *host* fent un mapatge dels ports del contenidor. Per a això:

1) Executem `docker run -it ubuntu /bin/bash` per accedir al contenidor. Si no està instal·lat, es baixarà i s'instal·larà.

2) Dins el contenidor executem `apt-get update` i, a continuació fem servir `apt-get install apache2 vim`. Per altra banda podem provar Apache2 fent `service apache2 start`, i veure que arrenca sense inconvenients (només mostrarà un advertiment, que no té el *ServerName* configurat). Es pot veure amb un `ps -edaf`. Modifiquem la pàgina inicial per posar alguna cosa relativa a Docker i, d'aquesta manera, poder identificar la `vi /var/www/html/index.html`.

3) És important no sortir del contenidor, sinó fer Crtl-P i, després, Crtl-Q per sortir al *host*, ja que, si fem Crlt-D, apagarem el contenidor i perdrem els canvis. També es pot fer des d'un altre terminal.

4) Amb el contenidor en marxa, verifiquem que està a *UP* amb `docker ps`; fem `docker commit ID_contenidor apache`, on `ID_contenidor` és el número que ens dóna `docker ps`, i només hem de posar els 3-4 primers nombres. Això salvarà una nova imatge anomenada *apache*, que podem verificar amb `docker images`.

5) Ara, sortim de la sessió del contenidor (Crtl-D o *exit*) o l'apaguem amb `docker stop ID_contenidor;docker ps` ens haurà de mostrar que no hi ha cap imatge en execució.

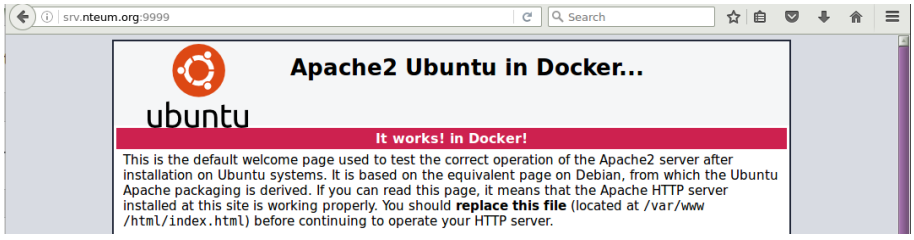
6) Posem en marxa el contenidor amb

```
docker run -d -p 9999: 80 apache /usr/sbin/ apache2ctl -D FOREGROUND;
```

allà on li indiquem amb `-d` que s'executi en mode *daemon*, amb `-p` li indiquem que faci un mapatge del port 80 del contenidor en el 9999 del *host*; i executem la comanda `apache2ctl` indicant-li que posi en marxa Apache2.

7) A continuació, des del *host* podem obrir un navegador amb *localhost: 9999* o amb el nom del sistema principal del qual s'ha fet un mapatge en una IP del *host* com a *srv.nteum.org:9999*, i veurem la pàgina modificada del contenidor com es mostra a la figura 11.

Figura 11



Lectura complementària

Per a més informació sobre com utilitzar els contenidors, aneu a [DoUs]

3.4. Puppet

Puppet és una eina de gestió de configuració i automatització en sistemes TI (llicència Apache –abans GPL). El seu funcionament és gestionat mitjançant arxius (anomenats *manifiests*) de descripcions dels recursos del sistema i els seus estats, utilitzant un llenguatge declaratiu propi de l'eina. El llenguatge Puppet pot ser aplicat directament al sistema, o compilat en un catàleg i distribuït al sistema de destinació utilitzant un model client-servidor (mitjançant una API REST), en què un agent interpel·la els proveïdors específics del sistema per a aplicar el recurs especificat en els *manifiests*. Aquest llenguatge permet una gran abstracció que habilita els administradors per a descriure la configuració en termes d'alt nivell, com ara usuaris, serveis i paquets sense necessitat d'especificar les ordres del sistema operatiu (`apt`, `dpkg`, etc.).[p1, p2, p12] L'entorn Puppet té dues versions: Puppet (*Open Source*) i Puppet Enterprise (producte comercial). Les diferències entre aquestes es poden veure a http://puppetlabs.com/puppet/enterprise-vs-open-source. A més s'integren amb aquests entorns altres eines com MCollective (*orchestration framework*), Puppet Dashboard (consola web però abandonada en el seu desenvolupament), PuppetDB *(datawarehouse* per a Puppet), Hiera (eina de cerca de dades de configuració), Facter (eina per a la creació de catàlegs) i Geppetto (IDE -*integrated development environment*- per a Puppet).

3.4.1. Instal·lació

És important començar amb dues màquines que tinguin els seus *hostname* i definició a */etc/hosts* correctament i que siguin accessibles mitjançant la xarxa amb les dades obtingudes del */etc/hosts* (és important que el nom de la màquina -sense domini- d'aquest arxiu coincideixi amb el nom especificat en el *hostname* ja que, si no, hi haurà problemes quan es generin els certificats).

Sobre el servidor instal·lem *puppetmaster* (considerem que tenim una Debian Jessie ja que els paquets estan en el repositori; per a versions anteriors podeu consultar la documentació):

```
apt-get install puppetmaster
```

Si està sobre una màquina virtual, reduïm els requeriments de memòria editant */etc/default/puppetserver* i posant: *JAVA_ARGS=-Xms1g -Xmx1g"* si es desitja donar 1 GB o *JAVA_ARGS=-Xms512m -Xmx512m"* per 512Mb. A continuació executem: `service puppetmaster restart` Sobre el client, instal·lem *puppet*: `apt-get install puppet` Sobre el client modifiquem */etc/puppet/puppet.conf* per afegir a la secció *[main]* el servidor (*srv.nteum.org*, en el nostre cas) i reiniciem:

[main] server=srv.nteum.org

```
puppet resource service puppet ensure=running enable=true service
puppet restart
```

Després executem sobre el *master*: `puppet cert list`. Veurem alguna cosa com *"node.nteum.org"(SHA256): 7A:B7:CE:...:01:6D*

Signem el certificat:

```
puppet cert sign node.nteum.org.
```

La sortida serà alguna cosa com:

Notice: Signed certificate request for node.nteum.org Notice: Removing file Puppet::SSL::CertificateRequest node,teum.orgat '/var/lib/puppet/ssl/ca/requests/node.nteum.org'

Això significa que *node.nteum.org* és acceptada pel *master (srv.nteum.org)* i es pot verificar amb `puppet cert list –all`. La sortida serà alguna cosa com:

+ "node.nteum.org"(SHA256) 7A:B7:CE:...:01:6D + "srv.nteum.org"(SHA256) FF:I2:49:....:62:5F (alt names: "DNS:puppet", "DNS:puppet.nteum.org", "DNS:srv.nteum.org")

En aquesta sortida el signe "+ïndica que els certificats estan correctes.

Sobre el client podem fer `puppet agent --fingerprint`. I veurem la clau SHA256 corresponent.

```
puppet agent --test
```

Veurem que obté tots els *pluginfacts*, *puglins*, *catalogs* i mostra la versió de la configuració (aquesta informació sortirà sobre el terminal en color verd; si hi ha alguna cosa que no funciona o un *warning*, sortirà en vermell i s'haurà de solucionar).

3.5. *Main Manifest File*

Puppet utilitza un llenguatge de domini específic per descriure les configuracions del sistema. Aquestes descripcions es desen en arxius anomenats *manifestos*, que tenen una extensió *name.pp*. El fitxer de manifest principal per defecte es troba a */etc/puppet/manifests/site.pp* (per crear-lo podem fer servir l'ordre `touch /etc/puppet/manifests/site.pp`).

La comanda `puppet apply` permet executar sota demanda manifestos que no són relatius al principal. En aquest cas, s'aplicarà el manifest sobre el node que s'hagi indicat, per exemple:

```
puppet apply /etc/puppet/modules/test/init.pp.
```

Executar el manifest d'aquesta manera només és útil si es desitja provar un nou manifest sobre un agent, o si només es busca executar-lo una sola vegada per inicialitzar l'agent a un estat determinat.

Crearem un manifest simple editant */etc/puppet/manifests/site.pp* i afegim:

```
file {'/tmp/example-ip':                                      # resource type file and filename
  ensure  => present,                                           # make sure it exists
  mode    => 0644,                                              # file permissions
  content => "Here is my Public IP Address: ${ipaddress_eth0}.\n",  # note the ipaddress_eth0 fact
}
```

Aquí s'assegura que l'arxiu existeix i que té permisos *rw-r - -r - -*, i insereix un text amb la IP pública. Es pot esperar fins que l'agent verifiqui automàticament; també es pot executar sobre el client `puppet agent --test` i després verificar `cat /tmp/example-ip`. Això donarà com a resultat *Here is my Public IP Address: 172.16.1.2.* A *site.pp*es pot afegir una especificació concreta per a un node, per exemple. Si només volem que s'instal·li sobre el node anomenat *nodo.nteum.org*, es posa el contingut *file* dins d'una sentència *node 'nodo.nteum.org '{ File { ... } }*, i veurem que només s'aplica a aquest node. Cal tenir en compte que, si no definim un recurs, Puppet farà el necessari per no tocar-lo, amb la qual cosa, encara que s'esborrin aquests recursos del manifest Puppet, no s'esborraran els recursos creats. Si, per exemple, volem esborrar aquests fitxers, s'ha de canviar a *ensure* l'etiqueta *present* per *absent*.

Una altra possibilitat que inclou Puppet, i que ara utilitzarem, són els *modules*. Els mòduls són útils per agrupar tasques i estan disponibles per a la comunitat Puppet, si bé cadascú pot crear els seus propis mòduls. Per instal·lar Apache utilitzarem el *puppetlabs-apache module* des de *forgeapi* executant `sudo puppet module install puppetlabs-apache`. La instal·lació d'aquest mòdul eliminarà totes les configuracions anteriors. A continuació, editem `vi /etc/puppet/manifest/site.pp` i inserim:

```
node 'node' {
class { 'apache': }                             # use apache module
        apache::vhost { 'node.nteum.org':       # define vhost resource
                port    => '80',
                docroot => '/var/www/html'
                }
}
```

Si executem sobre node `puppet agent -test`, veurem que, després d'uns quants missatges, hi haurà instal·lat i configurat Apache i podrem connectar-nos, per exemple, des de *srv.nteum.org* a http://nodo.nteum.org. Instal·lar un paquet, per exemple *nmap*, sobre el node és tan fàcil com inserir el següent codi dins el *node 'node'{ ... }*:

```
package {'nmap':
      ensure => installed,
      }
```

Finalment, per crear un mòdul que pugui generar un usuari es pot fer:

1) `mkdir -p /etc/puppet/modules/accounts/manifests`

2) `vi /etc/puppet/modules/accounts/manifests/groups.pp`

```
class accounts::groups {
        group { 'username':
        ensure  => present,}
                }
```

3) `vi /etc/puppet/modules/accounts/manifests/init.pp`

```
class accounts {
        include groups
        user { 'usernteum':
                ensure      => present,
                home        => '/home/usernteum',
                shell       => '/bin/bash',
                managehome  => true,
                gid         => 'username',
                password        => '$1$v9ESs.OZ$p.Y39eCnQ1T8A.BsHGGvO1'
                }
}
```

Per generar el *passwd* utilitzem `openssl passwd -1`

4) Incloem com a primera línia a */etc/puppet/manifets/sites.pp*

```
node 'node' {
include accounts
...
}
```

5) Executem sobre el client `puppet agent - -test` i, després, verifiquem que ens podem connectar a `ssh usernteum@localhost`.

Lectura recomanada

Vegeu la configuració i els detalls de la potencialitat del llenguatge Puppet i de la seva arquitectura a [PuDoc]. Cal tenir en compte la versió instal·lada (`puppet -V`).

Una eina interessant que es complementa amb Puppet és Foreman **[39, 40]**. Aquesta eina *open source* permet gestionar els servidors en tot el seu cicle de vida, des de l'aprovisionament i la configuració de l'orquestració i el seguiment. L'ús de Pupper, Xef i Ansible i l'arquitectura de proxy intel·ligent de Foreman permet automatitzar fàcilment tasques repetitives, desplegar aplicacions i gestionar de forma proactiva el canvi, tant en les instal·lacions amb màquines virtuals i *baremetal* o en el núvol (*cloud*). Presenta una interfície web, una CLI i una API REST per als diferents nivells d'interacció i permet gestionar des de desenes a desenes de milers de servidors. La seva instal·lació no és complexa encara que sí trigarà uns minuts i són importants els requeriments de memòria (per exemple per a aquesta prova de concepte s'ha utilitzat una MV a VirtualBox amb 2,5 GB de memòria RAM).

La instal·lació s'ha realitzat per a Debian Jessie (recomanada) encara que es pot fer per a altres versions (**vegeu [FOR]**). Els passos que s'han de seguir són els següents

1) Activem els repositoris i certificats:

```
apt-get -i install ca-certificates
```

```
wget https://apt.puppetlabs.com/puppetlabs-release-pc1-jessie.deb
dpkg -i puppetlabs-release-pc1-jessie.deb
echo "deb http://deb.theforeman.org/ jessie 1.12" > /etc/apt/sources.list.d/foreman.list
tiro "deb http://deb.theforeman.org/ plugins 1.12" >> /etc/apt/sources.list.d/foreman.list
apt-get -i install ca-certificates
wget -q https://deb.theforeman.org/pubkey.gpg -O- | apt-key add -
```

2) Descarreguem l'instal·lador:.

```
apt-get update && apt-get -i install foreman-installer
```

3) Executem l'instal·lador. La instal·lació és no-interactiva però si es desitgen ajustar alguns paràmetres es pot passar el paràmetre `i` a la comanda `foreman-installer`.

Després d'uns minuts tindrem una informació que cal verificar que ha acabat sense errors i en el cas que els hi hagi s'hauran de solucionar i repetir la instal·lació.

Foreman is running at https://srv.nteum.org Initial credentials llauri admin / 4eftk... Foreman Proxy is running at https://srv.nteum.org:8443 Puppetmaster is running at port 8140

El *log* queda registrat en */var/log/foreman-installer/foreman-installer.log*.

4) Reiniciem la màquina i verifiquem amb `service puppetmaster status` que ens dóna una informació similar a:

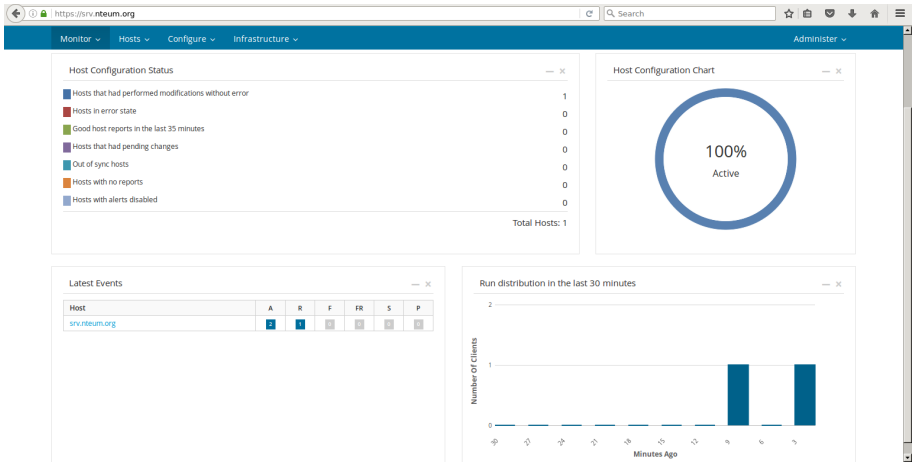
puppetserver.service - LSB: puppetserver Loaded: loaded (/etc/init.d/puppetserver) Activi: activi (running) since Sun 2016-07-31 10:53:44 BST; 22min ago CGroup: /system.slice/puppetserver.service +-602 /usr/bin/java -XX:OnOutOfMemoryError=kill -9 %p -Djava.secur.

En el cas que ens doni errors, haurem de verificar que són de memòria a *var/log/puppetlabs/puppetserver/* i modificarem */etc/default/puppetserver* amb, per exemple, *JAVA_ARGS=-Xms1g -Xmx1g -XX:MaxPermSize=256m"*.

El següent és accedir a l'adreça *https://srv.nteum.org* i acceptar el certificat i connectar-nos a *admin* amb el *passwd* indicat anteriorment. Per fer una prova es pot executar des de la mateixa màquina `puppet agent --test` (donarà uns missatges d'advertiment) i veurem que en *Tab->Host->All Hosts* de Foreman queda registrada l'activitat i amb l'estatus d'OK.

A continuació instal·larem un mòdul de Puppet per a conduir *module* amb la intenció de conduir el servei NTP executant `puppet moduli install puppetlabs/ntp`. A Foreman, anem a *Configure->Classes->Import from hostname* (a dalt a la dreta) i la classe "ntp" apareixerà instal·lada correctament. Fent un clic sobre "ntp" i en el tab *Smart Class Parameters seleccionar* marquem *Override*; així, Foreman pot conduir aquesta classe. Sobre el tab *Hosts*, accedim al servidor->tab *Puppet Classes*->ntp moduli i agreguem '+' per habilitar aquesta classe sobre *host*, i finalment salvem. Si des de un terminal executem `puppet agent --test`, veurem com s'instal·la el servei automàticament i s'engega (es pot comprovar amb `ntpq -p`) `[FOR]`. La figura 12 mostra el *Dashboard* de Foreman després de l'acció.

Figura 12



3.6. Chef

Chef és una altra de la grans eines de configuració amb funcionalitats similars a Puppet (hi ha un bon article en què s'efectua una comparació sobre el dilema de Puppet o Chef [ch1]). Aquesta eina utilitza un llenguatge (basat en Ruby) DSL (*domain-specific language*) per a escriure les "receptes" de configuració que seran utilitzades per a configurar i administrar els servidors de la companyia/institució i que a més pot integrar-se amb diferents plataformes (com Rackspace, Amazon EC2, Google i Microsoft Azure) per a, automàticament, gestionar la provisió de recursos de noves màquines. L'administrador comença escrivint les receptes que descriuen com maneja Chef les aplicacions del servidor (com ara Apache, MySQL, o Hadoop) i com seran configurades, i indica quins paquets hauran de ser instal·lats, els serveis que hauran d'executar-se i els arxius que s'hauran de modificar, i a més informarà de tot això en el servidor perquè l'administrador tingui el control del desplegament. Chef, juntament amb Puppet, CFEngine, Ansible i Bcfg2, és una de les eines més utilitzades per a aquest tipus de funcionalitat i que ja forma part de les mitjanes-grans instal·lacions actuals de GNU/Linux.

3.7. Estructura de Chef

L'estructura de Chef es basa en una **estació de treball** (*Workstation*) des d'on es gestiona tota la infraestructura, incloent-hi Chef DK, les receptes i l'ús d'eines com *kichen*, *chef-cero*; les eines de línia d'ordres com Knife (per interactuar amb el servidor de Chef); el Chef mateix (per interactuar amb el seu chef-repo local); i recursos com l'entorn bàsic de Chef (per a la creació de receptes) i d'InSpec (per a la construcció dels controls de seguretat i fluxos de treball). Aquesta estació de treball està configurada per permetre als usuaris crear, provar i mantenir els *cookbooks*, que es carreguen al servidor i que poden ser propis per a l'organització o d'altres disponibles al *Chef Supermarket*.

Chef-cero

Chef-cero és una eina de línia d'ordres que s'executa localment com si estigués connectat a un servidor autèntic Chef.

Els **nodes** són les màquines-físics, virtuals, núvols, etc. que estan sota la gestió de Chef. El chef-client s'instal·la a cada node i és qui du a terme l'automatització en aquesta màquina.

El **servidor Chef** actua com un centre d'informació. Els *cookbooks* i les polítiques de configuració es carreguen al servidor Chef des de les estacions de treball, encara que també es poden mantenir des del propi servidor Chef a través de la interfície web de la consola de gestió de Chef.

El **chef-client** s'executa en el node i té accés al servidor de Chef per obtenir les dades de configuració, fer cerques de dades històriques. Un cop finalitzada l'execució del Chef i del client, actualitza les dades d'execució al servidor com a node actualitzat.

Chef Supermarket és el lloc on s'emmagatzemen els *cookbooks* de la comunitat, que poden ser utilitzats per qualsevol usuari Chef.

Una manera de provar les receptes és utilitzar el programa **Chef-solo** per, després, passar-les al servidor. Tanmateix, això ha quedat obsolet, ja que el client actual pot funcionar en mode local (i emula tota la funcionalitat de Chef-Solo). Un programa complementari és **Chef-Cero**, que va sorgir com una experiència de laboratori. És un servidor Chef en memòria dissenyat per a finalitats de desenvolupament, sense dades al disc ni tampoc autenticació o autorització; està inclòs en Chef client (a partir de la versió 11.8.0) i permet passar el paràmetre `--local-mode` al chef-client. El mode local posa en marxa el servidor local Chef limitat només a *localhost*. Es carreguen totes les receptes locals, s'executa el Chef client, es realitzen les accions i, a continuació, acaba el servidor Chef-Zero. L'experiència de l'usuari final és la mateixa que si comptés amb la infraestructura completa, però només en una màquina local. A Debian es pot instal·lar Chef (`apt-get install xef`), que instal·larà Chef-Zero, i ens permetrà crear les receptes i aplicar-les en el mateix node.

Enllaç d'interès

La millor manera de veure la potencialitat de Chef és utilitzar la màquina virtual Ubuntu, que proveeix el desenvolupador (https://learn.chef.io/learn-the-basics/ubuntu/get-set-up/) i que s'executa per a l'usuari durant 12 hores. En un pas-a-pas ensenya les potencialitats de Chef sense grans inversions en instal·lació i configuració.

Com a prova de concepte provem algunes receptes:

1) Creem un directori de receptes: `mkdir`
HOME/*repo*; *cd*
`HOME / repo`

2) Creem la primera recepta `vi hello.rb`, que genera el fitxer */tmp/motd* amb el contingut '*hello world* ':

```
file '/tmp/motd' do
  content 'hello world'
end
```

3) Executem `chef-client -local-mode hello.rb`, que ens donarà alguna cosa com ara:

...

```
Starting Chef Client, version 11.12.8
resolving cookbooks for run list: []
Synchronizing Cookbooks:
Compiling Cookbooks...
[2016-07-31T23:43:58+01:00] WARN: Node srv.nteum.org has an empty run list.
Converging 1 resources
Recipe: @recipe_files::/root/recetas/hello.rb
  * file[/tmp/motd] action create
    - create new file /tmp/motd
    - update content in file /tmp/motd from none to c38c60
        --- /tmp/motd    2016-07-31 23:43:58.287615557 +0100
        +++ /tmp/.motd20160731-19247-1wil5o5     2016-07-31 23:43:58.287615557 +0100
        @@ -1 +1,2 @@
        +hello chef
Running handlers complete
Chef Client finished, 1/1 resources updated in 1.531396599 second
```

4) Podem verificar que existeix l'arxiu amb el contingut correcte i es pot provar d'executar la comanda per segona vegada. Veurem que no hi ha canvis (potser, si canviem el text del missatge i tornem a executar, veurem que Chef actualitza el contingut; o, si canviem el contingut de l'arxiu Chef, restituirà el contingut original). Si es canvia *content 'hello world '* per *action: delete*, esborrarem l'arxiu.

5) Per instal·lar un paquet, primer, haurem de fer un *update* dels paquets i, després, haurem d'instal·lar, per exemple, *apache2*: `vi webserver.rb`

```
execute 'apt-update' do
  command 'apt-get update'
end

package 'apache2' package 'apache2'
```

Es pot comprovar que després tenim Apache instal·lat. Si volguéssim canviar l'arxiu inicial, només hauríem d'agregar després de *package*:

```
file '/var/www/html/index.html' do
  content '<html>
  <body>
    <h1>hello world</h1>
  </body>
</html>'
end
```

Com es pot veure, la potencialitat i facilitat de les receptes és gran i és fàcil fer una descripció del que es desitja per configurar un node. Una manera d'organitzar les receptes i d'adaptar-les a diferents escenaris és construir un *cookbook* que sigui una estructura que es pugui lligar per agafar diferents parts d'ella i que ho mantingui tot organitzat [ChLB].

Els programaris (*ChefDK, Server, Client, Automate ...*) es poden descarregar des de https://downloads.chef.io/ (cal tenir en compte que alguns paquets tenen limitacions de nodes –fins a 25–). Tots estan per a una arquitectura Ubuntu. Hi ha diversos tutorials, per exemple [DO], que expliquen pas a pas com instal·lar el servidor i la infraestructura, si l'usuari està interessat a provar tota la infraestructura, però requereixen infraestructura específica o màquines virtuals amb altes prestacions que queden fora de l'objectiu d'aquest subapartat.

3.8. Ansible

Ansible és una plataforma codi obert per configurar i administrar molt fàcilment sistemes informàtics, i implementar aplicacions de programari en els nodes. I tot això utilitzant només SSH. Com hem vist, les eines de desplegament esmentades anteriorment necessiten un agent al *host* remot. En canvi, Ansible només necessita una connexió SSH i Python (2.4 o posterior) per dur a terme una acció en els nodes que s'han de configurar. A més, integra mòduls que treballen sobre format JSON i utilitza YAML per descriure configuracions reutilitzables. Els desenvolupadors són els mateixos que els del programari d'aprovisionament Cobbler i té una arquitectura que distingeix entre el **controlador** i els **nodes**. En el controlador s'inicia l'orquestració i és ell qui gestiona per `ssh` els nodes que coneix a través d'un inventari.

Ansible insereix mòduls als nodes (mitjançant `ssh`) que es desen temporalment i es comuniquen amb el controlador mitjançant el protocol JSON sobre una sortida estàndard.

El seu disseny està basat en una arquitectura minimalista (sense imposar dependències addicionals), consistent, segura i d'alta fiabilitat. Com que cada mòdul pot ser escrit en qualsevol llenguatge estàndard (Python, Perl, Ruby, Bash, etc.), la corba d'aprenentatge és suau i permet estar operatiu ràpidament, fins i tot per a infraestructures complexes.

La seva instal·lació i comprovació és molt fàcil:

1) Instal·lem el paquet sobre el controlador: `apt-get install ansible`

2) En verifiquem la versió: `ansible --version`

3) Generem les claus SSH:

```
ssh-keygen -t rsa -b 4096 -C "admin@srv.nteum.org"
```

4) Les copiem als clients: `ssh-copy-id` root@172.16.1.2. Fem el mateix per a tots els clients. Primer, haurem de verificar que tinguem en el client el paràmetre *PermitRootLogin yes* a */etc/ssh/sshd_config*. Si no és així, caldrà canviar-lo i reiniciar el servei.

5) Provem que ens podem connectar a tots els nodes sense *passwd*:
`ssh root@172.16.1.2`

6) Editem `vi /etc/ansible/hosts` i afegim tots els *hosts* clients (sota una mateixa etiqueta, per exemple), en el nostre cas *webservers*.

7) Comprovem `ansible -m ping webservers`, que ens donarà una resposta com ara

```
172.16.1.2 | success >> {
    "changed": false,
```

```
    "ping": "pong"
}
```

8) Executem la comanda `ansible -m command -a "df -h"webservers` que ens donarà com a resposta:

```
172.16.1.2 | success | rc=0 >>
Filesystem              Size  Used  Avail  Use%  Mounted on
/dev/sda1       3.7G    1.6G  1.9G   46%   /
udev                    10M           0      10M   0%    /dev
tmpfs                   201M  4.5M    196M   3%    /run
tmpfs                   501M          0      501M  0%    /dev/shm
tmpfs                   5.0M          0      5.0M  0%    /run/lock
tmpfs                   501M          0      501M  0%    /sys/fs/cgroup
tmpfs                   101M          0      101M  0%    /run/user/0
```

9) Ara podem executar altres ordres com

```
ansible -m command -a "free -mt"web-servers
ansible -m command -a "free -mt"webservers
ansible -m command -a üptime"webservers
ansible -m command -a "arch"webservers
ansible -m shell -a "hostname"webservers
ansible webservers -m copy -a "src=/etc/hosts dest=/tmp/hosts"
ansible all -m user -a 'password=$6$n... n2Mn1 name=testing'
```

per crear un usuari *testing*, el *passwd* es genera amb `mkpasswd -method=SHA-512`

Com es pot observar, és molt simple de posar en marxa i molt potent; no presenta particularitats/complexitats d'instal·lació/configuració de Puppet, Capistrano, Salt o Chef. Atès que utilitza com a agent SSH (generalment disponible per defecte en tots els clients Linux), el seu desplegament és molt fàcil. El nostre pas següent és configurar *playbooks* i treballar amb ells. [Ans]

> Els *playbooks* són descripcions en text pla (en format YAML) que permeten organitzar tasques complexes mitjançant ítems i parells *key: values*.

Dins d'un *playbook* podem trobar un o més grups de *hosts* (cadascun d'aquests és anomenat *play*) on es duran a terme les tasques. Per exemple:

```
---
- hosts: webservers
remote_user: root
vars:
variable1: value1
variable2: value2
remote_user: root
tasks:
- name: description for task1
task1: parameter1=value_for_parameter1 parameter2=value_for_parameter2
- name: description for task2
task2: parameter1=value_for_parameter1 parameter2=value_for_parameter2
handlers:
- name: description for handler 1
service: name=name_of_service state=service_status
```

```
- hosts: dbservers
remote_user: root
vars:
variable1: value1
variable2: value2
...
```

Els *handlers* són accions que s'executaran al final d'una tasca, com per exemple, en reiniciar un servei, o la màquina on s'han instal·lat uns paquets. Un exemple per desplegar Apache seria:

1) `mkdir /etc/ansible/playbooks; vi /etc/ansible/playbooks/apache.yml` amb el contingut següent:

```
---
- hosts: webservers
  tasks:
    - name: install apache2
      apt: name=apache2 update_cache=yes state=latest
    - name: copy index.html
      copy: src=/tmp/index.html dest=/var/www/html/ mode=0644
  handlers:
    - name: restart apache2
      service: name=apache2 state=restarted
```

2) Generem l'arxiu `vi /tmp/index.html`

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8"/>
  </head>
  <body>
    <h1>Apache was started in this host via Ansible</h1><br>
    <h2>Ansible is Easy!!</h2>
  </body>
</html>
```

3) Executem el *playbook*

```
ansible-playbook /etc/ansible/playbooks/apache.yml
```

i accedim al servidor posant a l'URL 172.16.1.2 (la IP o el nom del node desplegat).

Figura 13

Apache was started in this host via Ansible

Ansible is Easy!!!

Hi ha una gran quantitat de tutorials (per exemple, [An4H]) que mostren com fer *playbooks* més complexos i com gestionar diferents recursos. D'aquesta manera, l'administrador estarà en condicions d'implementar els seus propis *playbooks* d'una manera molt senzilla i amb un desplegament ràpid.

Ansible disposa d'una GUI anomenada Ansible Tower, basada en web, que permet totes les opcions de la CLI, a banda de característiques especials com ara monitorització dels nodes en temps real, interfície REST API per a Ansible, *job scheduling*, i d'una interfície gràfica per a la gestió de l'inventari i d'integració amb el *cloud* (per exemple, EC2, Rackspace, Azure); tanmateix, és un producte comercial (fins a 100 nodes U$S 5.000/any).

Com a alternatives, podem trobar RunDeck*, que permet executar *playbooks* d'Ansible (mitjançant un *plugin* que s'ha de descarregar**) o Semaphore*** que és una UI per a Ansible *Open Source*.

***http://rundeck.org/**
****https://github.com/Batix/rundeck-ansible-plugin**
*****https://github.com/ansible-semaphore/semaphore**

3.9. Vagrant

Aquesta eina és útil en entorns DevOps i té un paper diferent del de Docker però orientat cap als mateixos objectius: proporcionar entorns fàcils de configurar, reproduïbles i portàtils amb un únic flux de treball que ajudarà a maximitzar la productivitat i la flexibilitat en el desenvolupament d'aplicacions/serveis. Pot proveir de màquines de diferents proveïdors (VirtualBox, VMware, AWS, o altres) utilitzant *scripts*, Chef o Puppet per a instal·lar i configurar automàticament el programari de la VM. Per als desenvolupadors, Vagrant aïllarà les dependències i configuracions dins d'un únic entorn disponible, consistent, sense sacrificar cap de les eines que el desenvolupador utilitza habitualment i tenint en compte un arxiu anomenat *Vagrantfile*. La resta de desenvolupadors tindrà el mateix entorn encara que treballin des d'uns altres SO o entorns, de manera que s'aconseguirà que tots els membres d'un equip estiguin executant codi en el mateix entorn i amb les mateixes dependències. A més, en l'àmbit TI permet tenir entorns d'un sol ús amb un flux de treball coherent per a desenvolupar i provar *scripts* d'administració de la infraestructura, ja que ràpidament es poden fer proves de *scripts*, *cookbooks* de Chef, mòduls de Puppets i altres que utilitzen la virtualització local, com VirtualBox o VMware. Després, amb la mateixa configuració, pot provar aquests *scripts* en el *cloud* (per exemple, AWS o Rackspace) amb el mateix flux de treball.

En primer lloc, s'ha d'indicar que haurem de treballar sobre un GNU/Linux base (el que es defineix com a *bare metal*, és a dir, sense virtualizar, ja que necessitarem les extensions de HW visibles i si hem virtualitzat amb VirtualBox, per exemple, això no és possible). És recomanable instal·lar la versió de VirtualBox (pot ser la del repositori Debian), verificar que tot funciona i després descarregar l'última versió de Vagrant des de *http://www.vagrantup.com/downloads.html* i instal·lar-la amb `dpkg -i vagrant_x.y.z_x86_64.deb` (on x.y.z serà la versió que hem descarregat).

A partir d'aquest punt, és molt simple executant l'ordre [v1] `vagrant init hashicorp/precisi32`, que inicialitzarà/generarà un arxiu anomenat **Vagrantfile** amb les definicions de la VM, i quan executem **vagrant up** descarregarà del repositori *cloud* de Vagrant una imatge d'Ubuntu 12.04-32b i

l'engegarà. Per a accedir-hi, simplement hem de fer `vagrant ssh`, i si volem rebutjar-la, `vagrant destroy`. En el *cloud* de Vagrant [v2], podem accedir a diferents imatges preconfigurades* que podrem carregar simplement fent `vagrant box add nom`, per exemple `vagrant box add chef/centos-6.5`. Quan s'executi l'ordre `up` veurem que ens dóna una sèrie de missatges, entre els quals ens indicarà el port per a connectar-se a aquesta màquina virtual directament (per exemple, `ssh vagrant@localhost -p 2222` i amb *passwd* **vagrant**). Per a utilitzar una VM com a base, podem modificar l'arxiu *Vagrantfile* i canviar-ne el contingut:

*https://vagrantcloud.com/discover/featured

```
Vagrant.configure("2") do |config|
  config.vm.box = "hashicorp/precise32"
end
```

Si desitgem treballar amb dues màquines de manera simultània, haurem de modificar l'arxiu Vagrantfile amb el següent:

```
Vagrant.configure("2") do |config|
  config.vm.define "centos" do |centos|
    centos.vm.box = "chef/centos-6.5"
  end
  config.vm.define "ubu" do |ubu|
    ubu.vm.box = "hashicorp/precise32"
  end
end
```

Podrem veure que assigna un port SSH a cadascuna per a evitar col·lisions quan fem el `vagrant up`. Des de la VM podríem instal·lar el programari com es fa habitualment, però per a evitar que cada persona faci el mateix hi ha una forma d'aprovisionament que s'executarà quan es faci el `up` de la VM. Per a això, escrivim un *script init.sh* amb el següent contingut:

```
#!/usr/bin/env bash
apt-get update
apt-get install -y apache2
rm -rf /var/www
ln -fs /tmp/var/www
```

En aquest *script* (per a no tenir problemes de permisos per a aquesta prova) hem apuntant el directori */var/www* a */tmp*. A continuació, modifiquem el Vagrantfile (només hem deixat una VM per accelerar els processos de càrrega):

```
Vagrant.configure("2") do |config|
  config.vm.define " ubu" do |ubu|
    ubu.vm.box = "hashicorp/precise32"
    ubu.vm.provision :shell, path: " init.sh"
  end
end
```

Després, haurem de fer `vagrant reload -provision`. Veurem com s'aprovisiona i carrega Apache i després, si entrem a la màquina i creem un fitxer */tmp/index.html* i fem `wget 127.0.0.1` (o la IP interna de la màquina), veurem que accedim a l'arxiu i el baixem (igualment, si fem `ps -edaf | grep apache2` veurem que està funcionant). Finalment, i per a veure-la des del *host*,

hem de redireccionar el port 80 per exemple al 4444. Per a això, hem d'agregar en el mateix arxiu (sota ubu.vm.provision) la línia:

```
config.vm.network :forwarded_port, host: 4444, guest: 80.
```

Tornem a fer `vagrant reload –provision` i des del *host* ens connectem a l'URL: *127.0.0.1:4444* i veurem el contingut de l'*index.html*.

En aquestes proves de concepte, hem mostrat alguns aspectes interessants però és una eina molt potent que s'ha d'analitzar amb cura per a configurar les opcions necessàries per al nostre entorn, com per exemple aspectes del *Vagrant Share* o qüestions avançades del *Provisioning* que aquí només hem tractat superficialment.[v1]

Activitats

1. Instal·leu i configureu OpenMPI sobre un node; compileu i executeu el programa `cpi.c` i observeu el seu comportament.

2. Instal·leu i configureu OpenMP; compileu i executeu el programa de multiplicació de matrius (https://computing.llnl.gov/tutorials/openMP/exercise.html) en 2 *cores* i obteniu proves de la millora en l'execució.

3. Utilitzant Rocks i VirtualBox, instal·leu dues màquines per a simular un clúster.

4. Instal·leu Docker i creeu 4 entorns diferents.

5. Instal·leu Puppet i configureu un màquina client.

6. Ídem que en el punt anterior amb Chef.

7. Ídem amb Ansible.

8. Amb Vagrant creeu dues VM, una amb Centos i una altra amb Ubuntu i aprovisioneu la primera amb Apache i la segona amb MySQL. S'ha de poder accedir a les màquines des del *host* i s'han de poder comunicar entre elles.

Bibliografia

Tots els enllaços visitats al juliol de 2016.

[An4H] Ansible Tutorial. <https://serversforhackers.com/an-ansible-tutorial>

[Ans] Ansible: Getting Started <http://docs.ansible.com/ansible/intro_getting_started.html>

[Beo] *Beowulf cluster*.
<http://en.wikipedia.org/wiki/Beowulf_cluster>

[Beo1] **S. Pereira** *Building a simple Beowulf cluster with Ubuntu*.
<https://www-users.cs.york.ac.uk/ mjf/pi_cluster/src/Building_a_simple_Beowulf_cluster.html>

[Bla] **Barney, B.** *OpenMP*. Lawrence Livermore National Laboratory.
<https://computing.llnl.gov/tutorials/openMP/>

[ch1] *Puppet or Chef: The configuration management dilemma*.
<http://www.infoworld.com/d/data-center/puppet-or-chef-the-configuration-management-dilemma-215279?page=0,0>

[ch2] **A. Gale**. *Getting started with Chef*.
<http://gettingstartedwithchef.com/>

[ch3] *Download Chef: Server and Client*.
<http://www.getchef.com/chef/install/>

[ch4] *Chef Cookbooks*.
<https://supermarket.getchef.com/cookbooks-directory>

[Co] Cobbler. <http://cobbler.github.io/>

[ChLB] Learn the Chef basics on Ubuntu <https://learn.chef.io/learn-the-basics/ubuntu/>

[ChSer12] How To Set Up a Chef 12 Configuration Management System on Ubuntu 14.04 Servers. <https://www.digitalocean.com/community/tutorial_series/getting-started-managing-your-infrastructure-using-chef>

[d1] *Understanding Docker*.
<https://docs.docker.com/engine/understanding-docker/>

[d2] *Docker Docs*.
<https://docs.docker.com/>

[d4] *Docker HUB*.
<https://registry.hub.docker.com/>

[DO] Getting Started Managing Your Infrastructure Using Chef.
<https://www.digitalocean.com/community/tutorial_series/getting-started-managing-your-infrastructure-using-chef>

[DoUs] Working with containers. <https://docs.docker.com/v1.8/userguide/usingdocker/>

[empi] *MPI Examples*.
<http://www.mcs.anl.gov/research/projects/mpi/usingmpi2/examples/starting/main.htm>

[fai1] *FAI (Fully Automatic Installation) for Debian GNU/Linux*.
<https://wiki.debian.org/FAI>

[fai2] *FAI (Fully Automatic Installation) project and documentation*.
<http://fai-project.org/>

[fai3] *El libro del administrador de Debian. Instalación automatizada*.
<http://debian-handbook.info/browse/es-ES/stable/sect.automated-installation.html>

[faqmpi] *FAQ OpenMPI*.
<http://www.open-mpi.org/faq/>

[FG] FAI. <http://fai-project.org/fai-guide>

[FOR] Foreman QuickStart. <https://theforeman.org/manuals/1.12/index.html#2.Quickstart>

[KG] Kickstart
https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Installation_Guide/ch-kickstart2.html

[Kur] **Swendson, K**. *Beowulf HOWTO* (tlpd).
<http://www.tldp.org/HOWTO/Beowulf-HOWTO/>

[kvm] *KVM*.
<https://wiki.debian.org/es/KVM>

[il1] *Eucalyptus, CloudStack, OpenStack and OpenNebula: A Tale of Two Cloud Models*.
<http://opennebula.org/eucalyptus-cloudstack-openstack-and-opennebula-a-tale-of-two-cloud-models/>

[il2] *OpenNebula vs. OpenStack: User Needs vs. Vendor Driven*.
<http://opennebula.org/opennebula-vs-openstack-user-needs-vs-vendor-driven/>

[lam] *LAM/MPI*.
<http://lam.fries.net/>

[LogA] LogAnalyzer. <http://loganalyzer.adiscon.com/doc/>

[LogAR] How to create automated daily-weekly reports.
<http://loganalyzer.adiscon.com/articles/how-to-create-automated-dailyweekly-reports/>

[LXC1] *LXC*.
<https://wiki.debian.org/LXC>

[LXC2] *LXC en Ubuntu*.
<https://help.ubuntu.com/lts/serverguide/lxc.html>

[LXC3] **S. Graber***LXC 1.0*.
<https://wiki.debian.org/BridgeNetworkConnections>

[LXC4] *LXC: Step-by-Step Guide*.
<https://www.stgraber.org/2013/12/20/lxc-1-0-blog-post-series/>

[mpi3] *MPI 3*.
<http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf>

[NR] *A Short List of DevOps Tools*.
<http://newrelic.com/devops/toolset>

[on1] *OpenNebula Documentation*.
<http://opennebula.org/documentation/>

[on2] *Quickstart: OpenNebula on Ubuntu 14.04 and KVM*.
<http://docs.opennebula.org/4.6/design_and_installation/quick_starts/qs_ubuntu_kvm.html>

[ONDe] OpenNebula Deployment - Front-end Installation.
<http://docs.opennebula.org/5.0/deployment/opennebula_installation/frontend_installation.html>

[ONNo] OpenNebula. KVM Node Installation.
<http://docs.opennebula.org/5.0/deployment/node_installation/kvm_node_installation.html#kvm-node>

[ompi] *OpenMPI*.
<http://www.open-mpi.org/>

[opm] *OpenMP Exercise*.
<https://computing.llnl.gov/tutorials/openMP/exercise.html>

[OQRM] OpenQRM. <https://sourceforge.net/projects/openqrm/>

[p1] *Puppet Labs Documentation*.
<http://docs.puppetlabs.com/>

[p2] *Puppet - Configuration Management Tool*.
<http://puppet-cmt.blogspot.com.es/>

[p3] *Foreman - A complete lifecycle management tool*.
<http://theforeman.org/>

[p4] *Foreman - Quick Start Guide*.
<https://theforeman.org/manuals/1.12/quickstart_guide.html>

[pr] *Preseeding d-i en Debian*.
<https://wiki.debian.org/DebianInstaller/Preseed>

[p12] *Learning Puppet*.
<http://docs.puppetlabs.com/learning/ral.html>

[Par] GNU Parallel Tutorial <https://www.gnu.org/software/parallel/parallel_tutorial.html>

[Proc] *MPICH, High-Performance Portable MPI*.
<http://www.mpich.org/>

[PuDoc] The Puppet Language.
<https://docs.puppet.com/puppet/3.7/reference/lang_visual_index.html>

[Qe] *Qemu*.
<http://wiki.qemu.org/Main_Page>

[que] *QEMU. Guía rápida de instalación e integración con KVM y KQemu*.
<http://wiki.debian.org/QEMU>

[Rad] **Radajewski, J.; Eadline, D.** *Beowulf: Installation and Administration*. TLDP.
<http://www2.ic.uff.br/ vefr/research/clcomp/Beowulf-Installation-and-Administration-HOWTO.html>

[Rock] *Rocks Cluster. Base Users Guide*.
<http://central6.rocksclusters.org/roll-documentation/base/6.1.1/>

[tec] **Woodman, L.** *Setting up a Beowulf cluster Using Open MPI on Linux*.
<http://techtinkering.com/2009/12/02/setting-up-a-beowulf-cluster-using-open-mpi-on-linux/>

[v1] *Vagrant: Getting Started*.
<https://www.vagrantup.com/docs/getting-started/>

[v2] *Vagrant Cloud*.
<https://vagrantcloud.com/>

[SW] Spacewalk. <http://spacewalk.redhat.com/>

[xen] *The Xen hypervisor*.
<http://www.xen.org/>

[xeni] *Guía rápida de instalación de Xen*.
<http://wiki.debian.org/Xen>