Container a Pisa

disclaimer: non sono quelli in figura

Rudi Gaol (https://commons.wikimedia.org/wiki/File:Container-kontainer.jpg), https://creativecommons.org/licenses/by-sa/4.0/legalcode

Tutorial Days 2024 – HPC
25 - 27 Novembre 2024
INFN – Sezione di Pisa

INFN
Istituto Nazionale di Fisica Nucleare

Commissione Calcolo Reti

Francesco Laruina

Sommario

- Il principio/valore del disaccoppiamento
- Dal chroot al container
- Un Data Center dockerizzato
- Container e HPC
- Conclusioni

Cosa fa l’utente

- Sottomette un job su un sistema di batch
  - → no ssh sui nodi!
- Trovato uno o più nodi che soddisfano i requisiti, eseguo il comando richiesto
- A fine operazione, raccolgo i risultati

Cosa fa il sistemista

- Batch system
- UI, worker, ...
  - Storage
  - Networking
  - AAI
- Installazione software necessario per gli utenti
- ….

Separazione ambienti utente

- PRO
  - Semplifica gestione dipendenze
  - Il workload utente “funziona” indipendentemente da versione/distribuzione installata sull’host
    - Posso aggiornare, (re-)installare nuovi nodi e aggiungerli senza modificare la parte esposta all’utente
- CONTRO
  - Trovare un buon modo per ottenerla!

Cosa offre linux? (1)

- Namespaces

Namespace	Flag	Page	Isolates
Cgroup	CLONE_NEWCGROUP	cgroup_namespaces(7)	Cgroup root directory
IPC	CLONE_NEWIPC	ipc_namespaces(7)	System V IPC, POSIX message queues
Network	CLONE_NEWNET	network_namespaces(7)	Network devices, stacks, ports, etc.
Mount	**CLONE_NEWNS**	**mount_namespaces(7)**	**Mount points**
PID	CLONE_NEWPID	pid_namespaces(7)	Process IDs
Time	CLONE_NEWTIME	time_namespaces(7)	Boot and monotonic clocks
User	CLONE_NEWUSER	user_namespaces(7)	User and group IDs
UTS	CLONE_NEWUTS	uts_namespaces(7)	Hostname and NIS domain name

Cosa offre linux? (1)

Namespace	Flag	Page	Isolates
Cgroup	CLONE_NEWCGROUP	cgroup_namespaces(7)	Cgroup root directory
IPC	CLONE_NEWIPC	ipc_namespaces(7)	System V IPC, POSIX message queues
Network	CLONE_NEWNET	network_namespaces(7)	Network devices, stacks, ports, etc.
Mount	**CLONE_NEWNS**	**mount_namespaces(7)**	**Mount points**
PID	CLONE_NEWPID	pid_namespaces(7)	Process IDs
Time	CLONE_NEWTIME	time_namespaces(7)	Boot and monotonic clocks
User	CLONE_NEWUSER	user_namespaces(7)	User and group IDs
UTS	CLONE_NEWUTS	uts_namespaces(7)	Hostname and NIS domain name

Cosa offre linux? (2)

- Cgroups

Control groups, usually referred to as cgroups, **are a Linux kernel feature which allow processes to be organized into hierarchical groups whose usage of various types of resources can then be limited and monitored.** The kernel's cgroup interface is provided through a pseudo-filesystem called cgroupfs. Grouping is implemented in the core cgroup kernel code, while resource tracking and limits are implemented in a set of per-resource-type subsystems (memory, CPU, and so on).

CHROOT

- CHROOT fa leva sul MOUNT namespace
- Ambienti utente in cartelle
  - Locali ai nodi o su fs di rete
- I job sottomessi vanno eseguiti nell’opportuno ambiente chroot
  - Servono script di supporto

Criticità e alternative

- Soluzioni *artigianali*
  - *Fanno quello che voglio (e se non lo fanno colpa mia)*
  - *Devo costruirle e mantenerle nel tempo*
- Alternative *industriali*
  - Macchine virtuali
  - Container: es. Docker

Macchine virtuali

- Macchine fisiche → Hypervisor
- Nodi di calcolo → VM

Ogni VM ha il suo kernel e i suoi applicativi.

Rispetto a Bare metal cosa cambia per:

- cpu
- ram
- networking (compreso interconnect)
- Storage

Container

- Stesso kernel del nodo “ospitante”
- Isolamento e controllo risorse basato su namespace e cgroups
- Performance analoghe a bare metal (e chroot) salvo componenti paravirtualizzate
  - servono davvero in ambiente di calcolo?

Docker@Pisa

- Container vs chroot: perche’ ha vinto docker?
- Immagini sia *from scratch* sia gestibili con Dockerfile
- Distribuzione via registry locale
- In definitiva, eredito i tool di docker
- La conversione da chroot a immagine docker e’ banale

```
tar -C chrootDir -c . | docker import - imageFromChroot
```

Docker@Pisa (2)

- Sui nodi: installazione e configurazione docker per puntare a registry locale
- Sul sistema di batch: *wrapping*
- Come si sceglie l’ambiente?
  - Default per coda
  - Bsub […] -a docker-cs7 cmd

```
# pull dell'immagine richiesta per aggiornare la copia locale

/usr/bin/docker pull ${IMG}


# start del container docker come utente

/usr/bin/docker run --name=${CNTNAME} --net=host --rm=true --cpus=${ncpu} \
                    -v /etc/resolv.conf:/etc/resolv.conf \
                    -v /afs:/afs -v /cvmfs:/cvmfs \
                    ${MNT} \
                    ${OPT} \
                    -e "DISPLAY=$DISPLAY" \
                    --env-file ${dscript}/.user_env \
                    -u ${ui}:${gi} ${ga} -w $HOME \
                    -i ${USE_TTY} ${IMG} ${lscript}
```

E l’HPC?

Chiesti 56 core, assegnati

- 32 core su nodo1
- 24 core su nodo2

- Lancio mpirun su un nodo:
  - faccio ssh su nodo* per lanciare eseguibile
- Deve esserci un ambiente configurato e contattabile via ssh prima di fare mpirun

HPC: container statico

```
for i in `ls /dev/${NET}`;
do
 OPDEV="${OPDEV} --device=/dev/${NET}/${i}"
done


docker run --name=wn-hpc --net=host --rm=true --ulimit memlock=-1 --cap-add=CAP_SYS_PTRACE --shm-size="8g" \
        ${OPDEV} \
        -e container=docker --privileged=true --security-opt seccomp:unconfined --cap-add=SYS_ADMIN \
        -v /afs:/afs -v /cvmfs:/cvmfs -v /gpfs/ddn:/gpfs/ddn -v /chrootlfs/home:/home/grid \
        -v /sys/fs/cgroup:/sys/fs/cgroup:ro \
        -d -t ${IMG} /bin/bash -c /usr/sbin/init
```

E l’HPC con container dinamico?

- Deve esserci un ambiente configurato e contattabile via ssh prima di fare mpirun
- Quando lo creo?

Pre-exec

```
# start del container (via ssh, dopo tutta una serie di controlli e configurazione via ssh)

/usr/bin/docker run -d -i --name=${cntname} --hostname=${HOSTNAME} -p $
{ssh_port}:22 \

        --rm=true --cpus=${ncpu} --memory=${mem}k --ulimit memlock=-1 \

        -v /etc/resolv.conf:/etc/resolv.conf \

        -v /afs:/afs -v /gpfs/ddn:/gpfs/ddn -v /cvmfs:/cvmfs \

        -v /chrootlfs/home:/home/grid -v /sys/fs/cgroup:/sys/fs/cgroup:ro \

        ${mnt} \  # -v /tmp/.ssh/${jobid}.config:/etc/ssh/ssh_config \

        ${opt} \

        ${img} ${lscript} # ${lscript} -> /etc/sysconfig/docker-pi/start
```

```
# Lista device da passare al container

for d in `find /dev/infiniband`

do

 opt="$opt --device=$d"

done
```

```
function portselect

 p=`find ${portarea} -links 1 | sort | head -n 1`

 port=${p#"${portarea}/"}

 ln $p ${portarea}/job-${LSB_JOBID}
```

```
Host o2wn* 10.10.13.* 10.1.13.*

 Port <porta>

 Protocol 2

 StrictHostKeyChecking no

 ForwardX11 no

 HostbasedAuthentication yes
```

Cosa ottengo dopo il preexec

- Ogni nodo avra’
  - Ssh “del nodo” fuori container
  - un ssh per ogni container di calcolo
- Ogni job ha una sua porta di riferimento e ogni container una configurazione client ad hoc per parlare sulla porta selezionata con gli altri nel cluster

Ci siamo quasi...

- Fatto il preexec, parte lo script di avvio:
  - Docker stop&run con:
    - porta non privilegiata
    - Uid, gid, gruppi addizionali
    - Ambiente utente e comando utente

E finalmente mpirun

mpirun puo’ usare ssh e lanciare gli eseguibili.

Ma l’HPC e’ meno “variegato” dell’HTC, potrei scegliere di semplificare e lasciare N ambienti in esecuzione e in ascolto su porte diversificate

Oppure… cambiare approccio

Riflessioni e futuro

- Tutto questo e’ stato pensato un po’ di tempo fa
- Sul fronte container: rootless? Sistema di batch container-aware?
- SSH unica alternativa? Se usassi direttamente un batch system MPI-aware?
  - PMI-1, PMI-2 or PMIx APIs

Conclusioni

- Ripensando con gli occhi del 2024 quasi ‘25:
  - Molto scripting fatto in casa potrebbe essere rivisto/snellito
  - Si potrebbero aprire interessanti scenari per un metodo unificato non solo di gestione code, ma soprattutto dei nodi HTC/HPC

Grazie per l’attenzione!