Containers in HTCondor

Mauro Patano

TUTORIAL DAYS 2024 - HPC

PISA, 25 - 27 NOVEMBRE

Overview

HTCondor ed i container

HTCondor è un sistema software che crea un ambiente di Calcolo ad Alta Capacità (HTC), sfruttando efficacemente la potenza di calcolo di macchine connesse in network. La potenza deriva dalla capacità di sfruttare risorse condivise con proprietà distribuite.

- L'utente invia i job a HTCondor, che li esegue sulle macchine disponibili.
- HTCondor è utile quando un job deve essere eseguito molte volte, con centinaia di set di dati differenti.

HTCondor gestisce in modo efficente le risorse, abbinando fornitori e utilizzatori attraverso il sistema di *ClassAds*. Le macchine pubblicano le proprie caratteristiche in un annuncio di offerta risorsa, mentre l'utente invia una richiesta per definire le proprietà desiderate per eseguire il lavoro. HTCondor agisce da intermediario, abbinando e classificando gli annunci di offerta e richiesta risorse, tenendo conto di priorità multiple.
HTCondor supporta l'esecuzione di container per migliorare l'isolamento e la portabilità delle applicazioni.

Cosa sono i container?

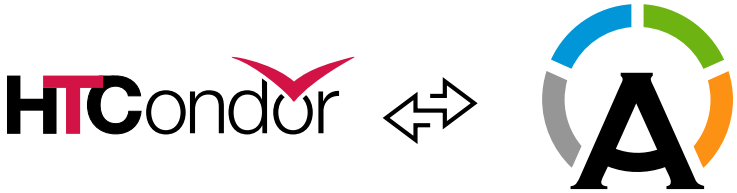
I container sono una soluzione ad alcuni dei problemi delle architetture monolitiche. Sebbene queste ultime abbiano dei punti di forza, impediscono alle organizzazioni di evolvere in modo agile.
I container permettono di suddividere i sistemi in microservizi.
Fondamentalmente, un container è un pacchetto di applicazioni composto da componenti leggeri, come dipendenze, librerie e file di configurazione, che vengono eseguiti in un ambiente isolato sopra i sistemi operativi tradizionali o in ambienti virtualizzati per garantire portabilità e flessibilità.
In sintesi, i container forniscono isolamento sfruttando tecnologie del kernel come cgroups, namespace del kernel e SELinux. I container condividono il kernel con l'host, il che consente loro di utilizzare meno risorse rispetto a una macchina virtuale (VM).

Vantaggi dei container

I container supportano un modello flessibile in termini di risorse di calcolo e memoria, e consentono alle applicazioni di consumare le risorse necessarie all'interno di confini definiti.
In altre parole, i container offrono scalabilità e flessibilità che non si ottengono eseguendo un'applicazione su una macchina virtuale o su bare metal.
I container semplificano la condivisione e il deploy delle applicazioni su cloud pubblici o privati. Più importante ancora, offrono coerenza che aiuta i team operativi e di sviluppo a ridurre la complessità dei deployment multi-piattaforma.
I container abilitano un set comune di componenti riutilizzabili in ogni fase dello sviluppo, per ricreare ambienti identici per sviluppo, testing, staging e produzione, estendendo il concetto di "scrivi una volta, distribuisci ovunque".
Rispetto alla virtualizzazione, i container semplificano la realizzazione di flessibilità, coerenza e la capacità di distribuire le applicazioni più velocemente: i principi fondamentali del DevOps.

Containerization Tech.	Apptainer	Docker
Architettura	Daemonless, single file format	Client-Server with daemon
Licenza	Licenza open-source (licenza LGPL)	Licenza open-source (licenza Apache 2.0)
Sicurezza	Integrazione con il sistema host, senza necessità di privilegi root	Richiede privilegi root per alcune features
Tipo di immagine	Immagini in formato Singularity. Compatibile con Docker	Immagini in formato Docker. Supporto per registry esterni
Tipo di utilizzo	Containerizzazione scientifica	Containerizzazione generale
Integrazione con HPC	Ottimizzato per ambienti HPC (High-Performance Computing)	Supporta ambienti cloud e containerizzazione su larga scala
Comunità e Supporto	Comunità focalizzata sulla ricerca scientifica e HPC	Comunità vasta, supporto per DevOps e applicazioni aziendali
Facilità d'uso	Comando simile a Docker, ma con alcune differenze in termini di gestione e configurazione	Interfaccia utente ben documentata e molto usata, facile da imparare
Portabilità	Eccellente portabilità grazie alla compatibilità con altre piattaforme di containerizzazione	Eccellente portabilità, con ampio supporto in ambienti cloud e locali

I container Apptainer in HTCondor



HTCondor supporta l'uso degli ambienti Apptainer (precedentemente noti come Singularity) per i job nel sistema High Throughput Computing.
Gli ambienti Apptainer consentono agli utenti di preparare software e ambienti di calcolo portabili, che possono essere inviati a molti job. Ciò significa che i job verranno eseguiti in uno spazio più coerente e facilmente riproducibile da altri.
I job container possono sfruttare meglio le risorse ad alta capacità del cluster HTCondor, poiché il sistema operativo su cui il job viene eseguito non deve necessariamente corrispondere al sistema operativo su cui è stato costruito il container.

Un esempio per gli impazienti

Submission file (.sub) per container Apptainer:

```
universe = vanilla
executable = /app/script_mio.py
use_container = true
container_image = nome_del_container.sif
output = script_mio.out
error = script_mio.err
log = script_mio.log
queue
```

Utilizzare un container esistente

Se un utente o un membro del suo gruppo ha già creato il file .sif di Apptainer, o sta utilizzando un container proveniente da fonti affidabili, può seguire questi passaggi per utilizzarlo in un job HTCondor.

1. Aggiungere il file .sif del container al file di submission: Se il file .sif si trova in una directory /home:

   ```
   container_image = path/to/containermio.sif
   ```

   Se il file .sif si trova in una directory /staging:

   ```
   container_image = file:///staging/path/to/containermio.sif
   ```

2. Testare il job con il container Si consiglia di inviare un singolo job di test e verificare che il job si comporti come previsto. Se si verificano problemi con il job, potrebbe essere necessario modificare l'eseguibile o addirittura (ri)costruire il container.

Costruire il proprio container Apptainer I

Se è necessario creare un container per il software desiderato, seguire questi passaggi.

1. Creare un file di definizione: Il file di definizione (.def) contiene le istruzioni su quale software installare durante la costruzione del container. Esistono moltissimi file di esempio online dai quali partire le prime volte.
2. Avviare un job interattivo per la costruzione: Avviare un job interattivo di costruzione (un esempio di file di invio chiamato build.sub è fornito qui sotto). Assicurarsi di includere il file .def nella linea `transfer_input_files`, oppure crearne uno una volta avviato il job interattivo, utilizzando un editor di linea di comando.

Costruire il proprio container Apptainer II

Submission file job interattivo per container Apptainer:

```
universe = vanilla
use_container = true
container_image = nome_del_container.sif
initialdir = $(PWD)
executable = /bin/bash
arguments = -i
output = nomecontainer.out
error = nomecontainer.err
log = nomecontainer.log
request_cpus = 1
request_memory = 1 GB
queue
```

Costruire il proprio container Apptainer III

Poi inviare il job interattivo di costruzione con:

```
condor_submit -i build.sub
```

3. **Costruire il container:** Durante il job interattivo, eseguire il comando:

   ```
   apptainer build containermio.sif image.def
   ```

   Se la costruzione del container termina con successo, verrà creato il file immagine del container (.sif). Questo file viene utilizzato per eseguire effettivamente il container.

4. **Testare il container:** Sempre all'interno del job interattivo, eseguire il comando:

   ```
   apptainer shell -e containermio.sif
   ```

   Questo comando avvia il container e consente di testare i comandi del software.
   Quando si é terminato di testare il container, uscire dal container con:

   ```
   exit
   ```

Costruire il proprio container Apptainer IV

5. Spostare il file .sif del container in staging: Una volta verificato che il container è stato costruito correttamente, copiare il file .sif nella directory di staging:

```
mv containermio.sif /staging/$USER
```

Dopo aver trasferito il file, uscire dal job interattivo con:

```
exit
```

Una volta che il container è stato costruito, seguire le istruzioni per utilizzarlo nel job HTCondor.

Creare un file di definizione I

Per creare un container con Apptainer, è necessario creare un file di definizione (.def).
Il processo generale per creare un file di definizione per un container personalizzato è il seguente:

1. Consultare la documentazione del software: Determinare i requisiti per l'installazione del software che si vuole utilizzare. In particolare, verificare (a) i sistemi operativi compatibili e (b) le librerie o i pacchetti necessari.
2. Scegliere un container di base: Il container di base dovrebbe almeno utilizzare un sistema operativo compatibile con il software. Idealmente, il container scelto dovrebbe già includere molte delle librerie/pacchetti richiesti.
3. Creare il file di definizione: Il file di definizione contiene i comandi di installazione necessari per configurare il software.

Creare un file di definizione II

Esempio di file Apptainer .def

```
Bootstrap: docker
From: ubuntu:24.04

%post
        apt-get update && apt-get install -y python3
        cp /my_script.py /app/script_mio.py

%startscript
        exec python3 /app/script_mio.py
```

Un file `.def` che usa un container esistente, installa Python, copia uno script python e rende quest'ultimo eseguibile all'avvio:

Avviare un job interattivo di costruzione

Il file `.def` contiene le istruzioni per creare il container, ma non è il container stesso. Per utilizzare il software definito nel file `.def`, è necessario prima "costruire" il container e creare il file `.sif`.

La costruzione di un container può essere un processo intensivo a livello computazionale. Per questo motivo, è auspicabile che la costruzione dei container avvenga durante un job interattivo. Nel sistema High Throughput, è possibile utilizzare il seguente file di invio `build.sub`:

Una volta creato il file di invio, il job interattivo può essere inviato con il comando:

```
condor_submit -i build.sub
```

Esempio di file .sub

```
universe = vanilla
log = build.log

# Nelle ultime versioni di HTCondor i job interattivi richiedono un eseguibile.
# Se non si dispone di un eseguibile esistente usare un comando generico Linux.
executable = /usr/bin/hostname

# Altri file nella directory /home necessari al container, vanno aggiunti
# alla linea transfer_input_files separati da virgole.
transfer_input_files = image.def

+IsBuildJob = true
request_cpus = 4
request_memory = 16GB
request_disk = 16GB

queue
```

Costruire il container I

Una volta avviato il job interattivo di costruzione, confermare che il file `image.def` sia stato trasferito nella directory corrente.
Per costruire il container, Apptainer utilizza le istruzioni nel file `.def` per creare un file `.sif`. Il file `.sif` è la raccolta compressa di tutti i file che compongono il container.
Per costruire il container, eseguire questo comando:

```
apptainer build containermio.sif image.def
```

Il nome del file `.sif` può essere scelto liberamente; ho utilizzato `containermio.sif`.
Durante l'esecuzione del comando, verranno stampate diverse informazioni sul terminale riguardanti il processo di costruzione del container. Una volta completato, il messaggio `INFO: Build complete: containermio.sif` dovrebbe apparire.
Utilizzando il comando `ls`, si dovrebbe vedere il file del container `containermio.sif`.
Se la costruzione fallisce, esaminare l'output per eventuali messaggi di errore che potrebbero spiegare il motivo del fallimento. Tipicamente si riscontrano problemi legati all'installazione di un pacchetto, come un errore di battitura o una

Costruire il container II

dipendenza mancante. A volte un errore durante l'installazione di un pacchetto non provoca il fallimento immediato della costruzione, ma quando si testa il container, si potrebbe notare un problema con quel pacchetto.
Se ci sono difficoltà nel trovare il messaggio di errore, modificare il file di definizione e rimuovere (o commentare) i comandi che seguono il pacchetto problematico. Successivamente, ricostruire l'immagine, in modo che i messaggi di errore pertinenti siano visibili alla fine dell'output di costruzione.
Una volta che l'immagine è stata costruita, è fondamentale testarla per assicurarsi che tutti i software, pacchetti e librerie siano stati installati correttamente.

Testare il container I

Una volta che il container è stato costruito correttamente, si consiglia vivamente di testarlo immediatamente durante la sessione di build interattiva.
Per testare il container, utilizzare il comando:

```
apptainer shell –e containermio.sif
```

Il prompt dei comandi dovrebbe cambiare in `Apptainer>`.
Il comando `shell` consente di accedere al terminale "interno" del container, con accesso alle librerie, pacchetti e programmi installati nel container in base alle istruzioni nel file `image.def`. (L'opzione `-e` impedisce che il terminale utilizzi i programmi del sistema host.)
Durante l'accesso al container, è consigliato eseguire i programmi installati. Tipicamente, si può iniziare provando a stampare l'help del programma, ad esempio: `program –help`. Se si utilizza un linguaggio di programmazione come python3 o R, provare a avviare una sessione interattiva di codice e caricare i pacchetti installati.

Testare il container II

Se il programma è stato installato in una posizione personalizzata, si può verificare la posizione dei file usando il comando `ls`. Potrebbe essere necessario impostare manualmente la variabile d'ambiente `PATH` per indicare la posizione dei file binari eseguibili del programma. Ad esempio:

```
export PATH=/opt/my-program/bin:$PATH
```

Quando si è finito di eseguire i comandi all'interno del container, utilizzare il comando `exit` per uscire dal container.

Spostare il file .sif del container in Staging Poiché i file .sif di Apptainer sono spesso superiori a 1 GB, si consiglia di spostare il file `containermio.sif` nella directory `/staging`. È più semplice spostare direttamente il file del container in staging mentre si è ancora nella sessione di build interattiva:

```
mv containermio.sif /staging/$USER
```

Utilizzare un container Apptainer nei job HTC

Una volta che l'immagine del container è stata salvata come file .sif, può essere utilizzata come ambiente per eseguire i job HTCondor. Nel file di submit, specificare il file immagine utilizzando il comando `container_image`. HTCondor trasferirà automaticamente il file .sif ed eseguirà il file eseguibile all'interno del container, senza necessità di includere comandi apptainer nell'eseguibile.
Se il file .sif si trova sul server di login, utilizzare:

```
container_image = containermio.sif
```

Se il file è posizionato in staging si utilizza:

```
container_image = file:///staging/path/to/containermio.sif
```

Il file di submit completo è mostrato in seguito.
Poi utilizzare `condor_submit` con il nome del file di submit:

```
condor_submit apptainer.sub
```

Esempio di file .sub

```
# apptainer.sub

# Fornisce a HTCondor il nome del file .sif e informazioni sull'universo
container_image = containermio.sif

executable = eseguibilemio.sh

log = esegibilemio.log
error = eseguibilemio.err
output = eseguibilemio.out

request_cpus = 1
request_memory = 4GB
request_disk = 10GB

queue
```

Considerazioni I

Dal punto di vista dell'utente, un job con container è praticamente identico a un job normale. L'unica differenza è che, invece di essere eseguito sul sistema operativo predefinito del punto di esecuzione, il job verrà eseguito all'interno del container.
Quando si invia un job a HTCondor usando un file di submit con `container_image` impostato, HTCondor gestisce automaticamente il processo di recupero ed esecuzione del container. Il processo si svolge più o meno come segue:

1. Reclama una macchina che soddisfi i requisiti del file di submit
2. Scarica (o trasferisce) l'immagine del container
3. Trasferisce i file di input e l'eseguibile nella directory di lavoro
4. Esegue lo script eseguibile all'interno del container, come utente del submit, con le directory chiave montate
5. Trasferisce i file di output indietro al server di submit.

Considerazioni II

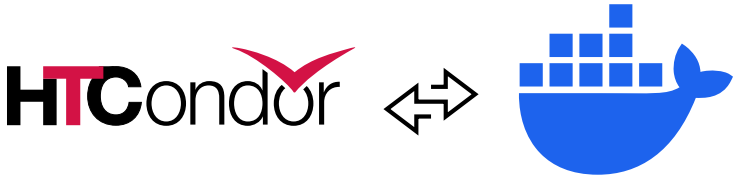
Per scopi di test, è possibile replicare il comportamento di un job con container con il seguente comando. Prima, avviare un job interattivo. Poi eseguire questo comando, sostituendo `containermio.sif` e `myExecutable.sh` con i nomi dei file .sif e .sh utilizzati:

Considerazioni III

Il comando Apptainer eseguito dietro le quinte da HTCondor

```
apptainer exec \
        --scratch /tmp \
        --scratch /var/tmp \
        --workdir $(pwd) \
        --pwd $(pwd) \
        --bind $(pwd) \
        --no-home \
        --containall \
        containermio.sif \
        /bin/bash myExecutable.sh 1> job.out 2> job.err
```

I container Docker in HTCondor



Nella modalità base il software necessario per i job in HTCondor viene installato o compilato localmente dagli utenti e poi trasferito a ogni job.
Un'alternativa è l'uso di un sistema di container, dove il software è incluso all'interno di un'immagine del container.
Utilizzare un container per gestire il software può essere vantaggioso in caso di:

1. numerose dipendenze
2. necessità di installazioni in una posizione specifica
3. percorsi "hard-coded" nell'installazione.

HTCondor è in grado di accedere, avviare container Docker ed eseguire job al loro interno.

Un esempio per chi ha ancora pazienza...

Submission file per Docker

```
# Fornisce a HTCondor il nome del container Docker
container_image = docker://user/repo:tag
universe = container

executable = eseguibilemio.sh
transfer_input_files = other_job_files

log = eseguibilemio.log
error = eseguibilemio.err
output = eseguibilemio.out

request_cpus = 1
request_memory = 4GB
request_disk = 2GB

queue
```

Integrazione con HTCondor

Quando il job viene avviato, HTCondor scaricherà l'immagine indicata da DockerHub e la utilizzerà per eseguire il job. Non sarà necessario eseguire manualmente alcun comando Docker.
Le altre parti del job (l'eseguibile e i file di input) saranno esattamente come in un job non-Docker.
L'unica differenza aggiuntiva potrebbe essere che l'eseguibile non dovrà più installare o decomprimere il software, poiché questo sarà già incluso nel container Docker.

Scegliere o Creare un'Immagine del container Docker I

Per eseguire un job Docker, è necessario avere accesso a un'immagine del container Docker che sia stata costruita e caricata su DockerHub. Esistono due modi principali per farlo:

1. Immagini Pre-esistenti Il modo più semplice per ottenere un'immagine del container Docker per eseguire un job è utilizzare un'immagine pubblica o pre-esistente su DockerHub. È possibile trovare immagini creando un account su DockerHub e cercando il software da utilizzare.
   Esempi di immagini: - Python - R - . . . e altre
   Un'immagine supportata da un gruppo sarà continuamente aggiornata, con le versioni indicate da "tag".

Scegliere o Creare un'Immagine del container Docker II

2. Creare la Propria Immagine È anche possibile creare un'immagine del container Docker personalizzata e caricarla su DockerHub. Per ulteriori informazioni sulla creazione di container, è possibile consultare questa guida o la documentazione ufficiale di Docker.
   Analogamente, si consiglia di utilizzare i tag per le immagini. È importante che, ogni volta che si apportano modifiche significative al container, venga utilizzato un nuovo nome di tag per garantire che i job ricevano una versione aggiornata dell'immagine, non una versione "vecchia" memorizzata nella cache di DockerHub.

Test

Per testare i container Docker, esistono due opzioni:

- sul proprio computer
- su HTCondor utilizzando un file di submit Docker e attivando l'opzione interattiva con `condor_submit`

Crea un file di sottomissione (`interactive.sub`). Eseguire il comando:

```
$ condor_submit -i interactive.sub
```

Questo avvierà una sessione all'interno del container Docker indicato e permetterà di connettersi tramite ssh. Digitando `exit` si terminerà il job interattivo.

Submission file per sesione interattiva di test:

```
universe = docker
docker_image = nome_dell_immagine
initialdir = $(PWD)
executable = /bin/bash
arguments = -i
output = output.txt
error = error.txt
log = condor.log
request_cpus = 1
request_memory = 1 GB
queue
```

Creare un'Immagine del container Docker

I container Linux sono un metodo per costruire ambienti autonomi che includono software, librerie e altri strumenti. HTCondor supporta l'esecuzione di job all'interno di container Docker.

In seguito si descrive come costruire un'immagine Docker da utilizzare per eseguire job su HTCondor.

I Dockerfile

Le immagini Docker possono essere create utilizzando un formato di file speciale chiamato “Dockerfile” contenente comandi che permettono di:

- Usare un’immagine Docker pre-esistente come base
- Aggiungere file all’immagine
- Eseguire comandi di installazione
- Impostare variabili d’ambiente

Successivamente, sarà possibile "costruire" un’immagine da questo file, testarla localmente e caricarla su DockerHub, da dove HTCondor potrà utilizzarla per creare container da eseguire nei job. Diverse versioni dell’immagine possono essere etichettate con "tag" differenti. I passaggi descritti di seguito devono essere eseguiti sul proprio computer, non su HTCondor.

Guida Passo per Passo I

1. Impostare Docker sul Proprio Computer Se non è già stato fatto, è necessario creare un account DockerHub e installare Docker sul proprio computer. Per farlo, bisogna cercare la Docker Community Edition compatibile con il sistema operativo in uso. Inizialmente, il lancio di Docker potrebbe richiedere del tempo, soprattutto la prima volta. Una volta avviato Docker, non verrà aperta una finestra; si vedrà solo l'icona di una balena e di un container sulla barra degli strumenti del computer. Per usare Docker, bisognerà aprire un terminale (Mac/Linux) o il prompt dei comandi (Windows) ed eseguire i comandi da lì.
2. Esplorare i container Docker (opzionale) Per chi non ha mai usato Docker, si consiglia di esplorare un container pre-esistente e testare i passaggi di installazione interattivamente prima di creare un Dockerfile.

Guida Passo per Passo II

3. Creare un Dockerfile Un Dockerfile è un file di testo semplice contenente parole chiave. Esistono molte parole chiave che possono essere utilizzate in un Dockerfile (documentate sul sito ufficiale di Docker). Verrà utilizzato un sottoinsieme di queste parole chiave per ottenere un modello di base contenente le seguenti informazioni:
   - Punto di partenza: Da quale immagine Docker partire?
   - Aggiunte: Cosa deve essere aggiunto? Cartelle? Dati? Software?
   - Ambiente: Quali variabili (se presenti) sono impostate durante l'installazione del software?

Se si prevede di creare più immagini per diverse fasi del flusso di lavoro, è consigliato creare una cartella separata per ogni nuova immagine, con il relativo Dockerfile.

Crea un'immagine Docker per la tua applicazione scrivendo un `Dockerfile` e costruendo l'immagine.

Esempio Dockerfile

```
FROM ubuntu:24.04
RUN apt-get update && apt-get install -y python3
COPY my_script.py /app/my_script.py
CMD ["python3", "/app/my_script.py"]
```

4 **Costruire, Nominare e Taggare l'Immagine** Finora, non è stata creata l'immagine, sono state elencate solo le istruzioni per costruirla nel Dockerfile. Ora si può costruire l'immagine!

Scegliere un nome per l'immagine e un tag. I tag sono importanti per tracciare quale versione dell'immagine è stata creata (e sarà utilizzata). Si consiglia di utilizzare un sistema di tag che abbia senso per l'utente, ma un sistema semplice potrebbe essere quello di usare numeri (ad esempio v0, v1, ecc.).

Per costruire e taggare l'immagine, aprire il terminale (Mac/Linux) o il prompt dei comandi (Windows) e navigare nella cartella contenente il Dockerfile:

```
$ cd directory
```

Sostituire `directory` con il percorso della cartella appropriata.

Assicurarsi che Docker sia in esecuzione e lanciare il comando:

```
$ docker build -t username/imagename:tag .
```

Sostituire `username/imagename:tag` con i valori scelti.

5 **Testare Localmente** Questa sezione descrive come interagire con la nuova immagine Docker sul proprio computer prima di provare a eseguire un job su HTCondor.

6 **Caricare su DockerHub** Una volta che l'immagine è stata costruita e testata con successo, si può caricarla su DockerHub per renderla disponibile per l'esecuzione dei job su HTCondor. Per fare ciò, eseguire il comando:

```
$ docker push username/imagename:tag
```

Il primo caricamento su DockerHub potrebbe richiedere il login, che può essere effettuato con:

```
$ docker login
```

Riproducibilità Se si possiede un account gratuito su DockerHub, le immagini del container caricate saranno programmate per essere rimosse se non vengono utilizzate (tirate) almeno una volta ogni 6 mesi. Per garantire una buona prassi, si consiglia di archiviare l'immagine del container e il Dockerfile in un archivio di lunga durata e backup.
Per creare un archivio di un'immagine del container, utilizzare il comando:

```
$ docker save --output archive-name.tar username/imagename:tag
```

È anche buona prassi archiviare una copia del Dockerfile insieme all'archivio dell'immagine del container.

7 **Esecuzione dei Job** Una volta che l'immagine Docker è su DockerHub, è possibile utilizzarla per eseguire i job sul sistema HTCondor.

Un semplice submission file per un job in un container Docker

```
universe = docker
docker_image = nome_dell_immagine
executable = /app/script_mio.py
output = script_mio.out
error = script_mio.err
log = script_mio.log
queue
```

Esempio I: Installazione di un pacchetto Python personalizzato da GitHub

Si suppone che un pacchetto Python personalizzato sia ospitato su GitHub, ma non disponibile su PyPI. Poiché pip può installare pacchetti direttamente dai repository git, il pacchetto può essere installato nel seguente modo:

I Esempio Dockerfile

```
FROM python:3.8

RUN pip3 install git+https://github.com/<Rep_Owner>/<Rep_Name>
```

In questo caso, <Rep_Owner> e <Rep_Name> dovranno essere sostituiti con i valori necessari.

Esempio II: Installazione di QIIME2 diretta

Questo Dockerfile installa QIIME2 seguendo delle istruzioni specifiche. Si presume che l'installer di miniconda a 64 bit per Linux sia stato scaricato nella directory contenente il Dockerfile.

Il Esempio Dockerfile

```
FROM python:3.6-stretch
COPY Miniconda3-latest-Linux-x86_64.sh /tmp
RUN mkdir /home/qiimeuser
ENV HOME=/home/qiimeuser
RUN cd /tmp \
&& ./Miniconda3-latest-Linux-x86\_64.sh -b -p /home/qiimeuser/minconda3 \
&& export PATH=/home/qiimeuser/minconda3/bin:$PATH \
&& conda update conda \
&& conda env create -n qiime2-tiny-2024.10 --file\
https://data.qiime2.org/distro/tiny/qiime2-tiny-2024.10-py310-linux-conda.yml
```

Convertire le immagini Docker in immagini Apptainer

Perché convertire un'immagine Docker in un'immagine Apptainer? Mentre HTCondor può prelevare immagini dei container da Docker Hub, è consigliato convertire le immagini Docker in immagini locali Apptainer (.sif) per disporre di una copia locale dell'immagine del container. Questo approccio riduce le richieste di prelievo da Docker Hub.

Per costruire l'immagine Apptainer, è possibile avviare un lavoro interattivo. Di seguito è riportato un esempio di un file di submit. Potrebbero essere necessari aggiustamenti nei requisiti di memoria o spazio su disco in base alle necessità specifiche per l'immagine.

Submission file per convertire Docker image in Apptainer image:

```
# build.sub
# Per costruire un container Apptainer

log = build.log

+IsBuildJob = true
request_cpus = 1
request_memory = 4GB
request_disk = 4GB

queue
```

Il lavoro interattivo va inviato tramite il comando:

```
$ condor_submit -i build.sub
```

Una volta avviato il lavoro interattivo, sarà possibile costruire l'immagine Apptainer con il comando `apptainer build`:

```
apptainer build container.sif docker://user/repo:tag
```

Dove `user/repo:tag` rappresenta un qualsiasi indirizzo valido di un registro Docker (ad esempio, `rocker/tidyverse:4.1.3` da DockerHub o `nvcr.io/nvidia/tensorflow:24.02-tf2-py3` dal NVIDIA Container Registry). Con questo comando si crea un'immagine Apptainer chiamata `container.sif`. Durante l'esecuzione, appariranno dei messaggi che indicano l'avanzamento del processo. Infine spostare l'immagine nella directory di staging se necessario.

Un esempio reale recente

Per rislvere un sistema di equazioni su basi di Groebner è necessario utilizzare un sat solver.
Identificato come strumento CryptoMiniSat SAT solver disponibile su Github:
https://github.com/msoos/cryptominisat
Si utilizza il Dockerfile disponibile sulle pagine di GitHub per creare un container Docker. Il primo tentativo fallisce e dopo aver consultato la documentazione viene realizzato un secondo Dockerfile. il secondo tentativo di costruzione riesce. Si decide di trasformare l'immagine Docker ottenuta in immagine Apptainer (.sif)
Ottenuto il file .sif lo si trasferisce su frontend HTCondor dove viene realizzato un file.sub (submission). Per lanciare i primi job di testo.

Prima Versione del Dockerfile (I Parte)

```
FROM ubuntu:16.04 as builder

LABEL maintainer="Mate Soos"
LABEL version="5.0"
LABEL Description="An advanced SAT solver"

# get curl, etc
RUN apt-get update && apt-get install --no-install-recommends -y \
        software-properties-common \
        && rm -rf /var/lib/apt/lists/*
RUN add-apt-repository -y ppa:ubuntu-toolchain-r/test \
        && rm -rf /var/lib/apt/lists/*
RUN apt-get update && apt-get install --no-install-recommends -y \
        libboost-program-options-dev gcc g++ make cmake zlib1g-dev wget \
        && rm -rf /var/lib/apt/lists/*
```

Prima Versione del Dockerfile (II Parte)

```
# set up build env
RUN groupadd -r solver -g 433
RUN useradd -u 431 -r -g solver -d /home/solver \
        -s /sbin/nologin -c "Docker image user" solver
RUN mkdir -p /home/solver/cms
RUN chown -R solver:solver /home/solver

# build CMS
USER root
COPY . /home/solver/cms
WORKDIR /home/solver/cms
RUN mkdir build
WORKDIR /home/solver/cms/build
RUN cmake -DSTATICCOMPILE=ON .. \
    && make -j6 \
    && make install \
    && rm -rf *
```

Prima Versione del Dockerfile (III Parte)

```
# set up for running
FROM alpine:latest
COPY --from=builder /usr/local/bin/cryptominisat5 /usr/local/bin/
ENTRYPOINT ["/usr/local/bin/cryptominisat5"]
```

modificato in:

Seconda Versione del Dockerfile (I Parte)

```
FROM ubuntu:24.04 AS builder

LABEL maintainer="Mate Soos"
LABEL version="5.0"
LABEL Description="An advanced SAT solver"

# get curl, etc
RUN apt-get update && apt-get install --no-install-recommends -y \
        software-properties-common && rm -rf /var/lib/apt/lists/*
RUN add-apt-repository -y ppa:ubuntu-toolchain-r/test \
        && rm -rf /var/lib/apt/lists/*
RUN apt-get update && apt-get install --no-install-recommends -y \
        libboost-program-options-dev gcc g++ \
        make cmake zlib1g-dev libgmp-dev wget \
        && rm -rf /var/lib/apt/lists/*
```

Seconda Versione del Dockerfile (II Parte)

```
# set up build env
RUN groupadd -r solver -g 433
RUN useradd -u 431 -r -g solver -d /home/solver \
        -s /sbin/nologin -c "Docker image user" solver
RUN mkdir -p /home/solver/cms
RUN chown -R solver:solver /home/solver

# build CMS
USER root
COPY . /home/solver/cms
WORKDIR /home/solver/cms
RUN mkdir build
WORKDIR /home/solver/cms/build
RUN cmake -DSTATICCOMPILE=ON .. \
    && make -j6 \
    && make install \
    && rm -rf *
```

Seconda Versione del Dockerfile (III Parte)

```
# set up for running
FROM alpine:latest
COPY --from=builder /usr/local/bin/cryptominisat5 /usr/local/bin/
ENTRYPOINT ["/usr/local/bin/cryptominisat5"]
```

Trasformazione del container Docker in container Apptainer

Docker -> Apptainer

```
$ apptainer build crmisa.sif docker-daemon:cryptominisat:latest
$ ll -h crmisa.sif
-rwxr-xr-x 1 matpauranoo matpauranoo 15M Nov 18 15:20 crmisa.sif
```

Spostato il file su frontend
Scritti i file .sub e .sh:

Submission file crmisa_container.sub

```
executable = crmisa_container_test.sh

log = crmisa.log
error = crmisa.err
output = crmisa.out

request_cpus = 250
request_memory = 128GB

queue
```

Submission file crmisa_container.sub

```
$ cat crmisa_container_test.sh
#!/usr/bin/env bash
./crmisa.sif -t 250 round4_eval200_sdeg2_aradiCNF.txt
```

Link Utili

- Biocontainers: Biocontainers homepage. URL: https://biocontainers.pro (visited on 23/11/2024).
- Docker Inc.: Docker documentation, URL: https://docs.docker.com (visited on 23/11/2024).
- Idem: Docker Github page, URL: https://github.com/docker (visited on 23/11/2024).
- Idem: Docker homepage, URL: https://www.docker.com (visited on 23/11/2024).
- Idem: Docker Hub homepage, URL: https://hub.docker.com (visited on 23/11/2024).
- HTCondor Team, Center for High Throughput Computing, Computer Sciences Department, University of Wisconsin-Madison, WI.: HTCondor documentation, URL: https://htcondor.org/documentation/htcondor.html (visited on 23/11/2024).
- Idem: HTCondor Github page, URL: https://github.com/htcondor (visited on 23/11/2024).
- Idem: HTCondor homepage, HTCondor Team, Center for High Throughput Computing, Computer Sciences Department, University of Wisconsin-Madison, WI., URL: https://htcondor.org (visited on 23/11/2024).

LF Projects, LLC: Apptainer documentation, URL: https://apptainer.org/documentation (visited on 23/11/2024).
Idem: Apptainer Github page, URL: https://github.com/apptainer (visited on 23/11/2024).
Idem: Apptainer homepage, URL: https://apptainer.org (visited on 23/11/2024).
Red Hat: Quay.io homepage, URL: https://quay.io (visited on 23/11/2024).