Architettura stateless con uso di Singularity per il porting degli applicativi WRF-4.1.5 e BOLAM

Fabio Massimo Grasso, Leopoldo Fazioli, Tony Christian Landi, Oxana Drofa

Maggio 2020

1 Introduzione

1.1 Architettura *Infrastructure As A Service* (IAAS)

Lo sviluppo tecnologico ha permesso un accrescimento notevole delle capacità di calcolo, di storage, di interconnessione e la creazione di nuove architetture dei data center per finalizzarli all'erogazione di servizi. Lo IAAS è un servizio cloud che consente l'accesso e l'utilizzo delle risorse di una struttura IT ad alta scalabilità interamente gestita da un provider. Nelle architetture cloud l'applicazione è disaccoppiata dall'hardware e il committente chiede le risorse in base alle necessità contingenti; ciò comporta che la stessa sia modulare poiché non si ha prontezza del calcolatore su cui funzionerà.

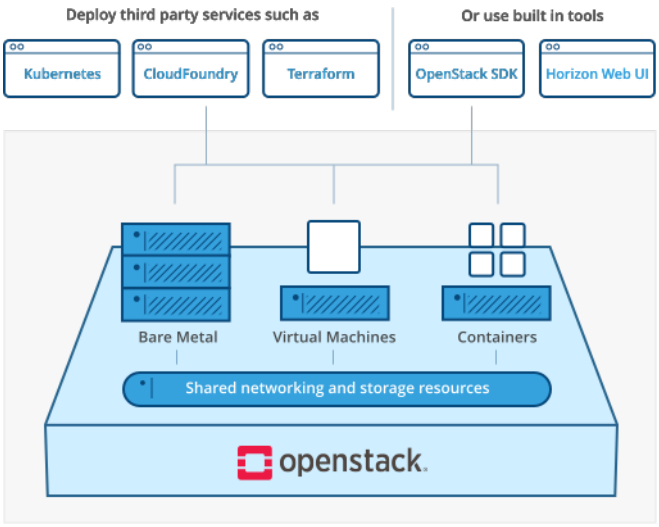

Figure 1: Overview dell'architettura IAAS (Infrastructure As A Service) Openstack® [1]

Volendo applicare il paradigma REST (REpresentational State Transfer), se immaginassimo l'applicazione come un motore che trasforma dei dati in ingresso in nuovi dati e volessimo esporre in rete le risorse che abbiamo generato, avremmo bisogno di un indirizzo con cui presentarle e un set di metodi per la fruizione; tutto ciò è già standardizzato nel protocollo HTTP e lo si può usare a patto di progettare l'applicazione in modo che non abbia uno stato, cioè una memoria dell'esperienza intrapresa dall'utente. Per operare un trasferimento tecnologico dei prodotti prototipali in servizi che richiedano risorse in base all'utilizzo, è necessaria una infrastruttura software che offra vantaggi di:

- scalabilità
- modularità
- contenimento dei costi di manutenzione e gestione
- sicurezza

Il progettista deve sviluppare una architettura che sia stateless e modulare, in cui l'ecosistema applicativo è suddiviso in container; ogni container contiene uno o più programmi che fanno parte dell'applicazione, il sistema operativo e i software strettamente necessari all'esecuzione dei programmi; questi container devono poter essere instanziati e distrutti nell'infrastruttura senza alcun problema e possono essere gestiti tramite appositi software di coordinamento.

1.2 Container

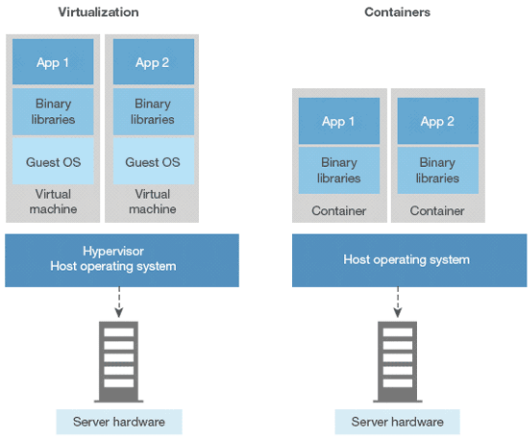

Figure 2: Confronto fra VM e container [2]

Nei sistemi Linux la memoria è divisa in due parti: kernel space e user space. Nella prima è eseguito il sistema operativo che gestisce l'hardware sottostante

e fornisce i suoi servizi ai processi dell'utente eseguiti nello user space; uno dei compiti del sistema operativo è prevenire conflitti fra i processi che possono accedere al kernel tramite chiamate di sistema e ottenere l'accesso I/O alle periferiche ovvero la creazione di ulteriori processi. Nei container lo user space diventa un componente *swappable* in cui programmi, configurazioni e variabili di ambiente, sono indipendenti dal sistema operativo sottostante. Con questo strumento gli sviluppatori possono replicare i loro stack sul sistema operativo che meglio corrisponde alle loro necessità e distribuire l'ambiente di runtime liberando gli utenti finali dalla preoccupazione dei requisiti e delle dipendenze, che spesso confliggono con quelli offerti dal sistema operativo ospitante.

2 Singularity®

Figure 3: Logo di Singularity ®

Un container Singularity® è un singolo file costruito per contenere tutti i programmi, librerie, dati e script necessari alla portabilità e riproducibilità dell'ambiente di sviluppo/produzione in maniera indipendente dalla versione di Linux che gestisce l'hardware. Il container può essere costruito su una normale workstation, come il laptop del ricercatore, e copiato sull' HPC . Sviluppato inizialmente presso il Lawrence Berkeley National Laboratory, dal 2017 è velocemente diventato popolare presso il mondo accademico ed è supportato in importanti centri HPC come il CINECA[3]. Ciò che differenzia questo prodotto dai competitor [4] e lo centra sul target della ricerca è:

- security: lo user all'interno del container è lo stesso al suo esterno, cioè nell'host, e non vi può ottenere privilegi aggiuntivi; questa è una caratteristica molto importante perché negli HPC non si può girare un codice con privilegi di root
- i container possono essere criptografati e firmati digitalmente, la decriptazione è effettuata in memoria
- isolamento con un efficace e semplice uso di network e file system

- portabilità del container in un unico file *.sif* che può essere facilmente condiviso

2.1 Formati del container

Il container Singularity è una immagine monolitica dell'ambiente portabile; quando l'utente entra e ci lavora, è situato fisicamente al suo interno [6]. Questo oggetto incapsula l'ambiente del sistema operativo e tutte le applicazioni da cui dipende il workflow applicativo. Nel caso in cui debba essere replicato, è sufficiente copiarlo. Possono essere prodotti due tipologie di container:

- SIF (Singularity Image File): un file unico che può essere compresso e in sola lettura, adatto per la produzione
- Sandbox: è una directory (ch)root, scrivibile, per lo sviluppo del workflow.

2.1.1 Creazione del container

Il container si crea con il comando **build** con i privilegi di root specificando un input che può essere di diversi tipi:

- URI con *library://* per costruire da un container in libreria
- URI con *docker://* per costurire da un hub Docker
- URI con *shub://* per costruire da un hub Singularity
- percorso ad un container già esistente in locale
- percorso ad una directory per costruire da una *sandbox*
- percorso ad un *definition file*; questi file, aventi estensione .def, sono una rappresentazione schematica di come Singularity costruirà il container, partendo dal sistema operativo di base, le variabili d'ambiente il software propedeutico al funzionamento del workflow applicativo [5]. Si possono copiare dei dati dall'host di partenza al container e questo può essere sfruttato per incapsulare dei dati scientifici, in un'ottica di replicabilità di un esperimento, o per compilare delle librerie o programmi del workflow operativo.
  Inoltre, è anche possibile definire i metadati del container. Un elenco esaustivo delle sezioni che costituiscono il definition file è in [5].

Ad esempio per creare un container SIF partendo dal recipe del modello WRF posto in appendice, bisogna dare il comando:

```
sudo singularity build wrf.sif wrf-4.1.5_and_wps.def
```

Volendo ottenere una sandbox in cui testare l'ambiente di sviluppo, il comando diventa:

```
sudo singularity build --sandbox sandbox-path wrf-4.1.5_and_wps.def
```

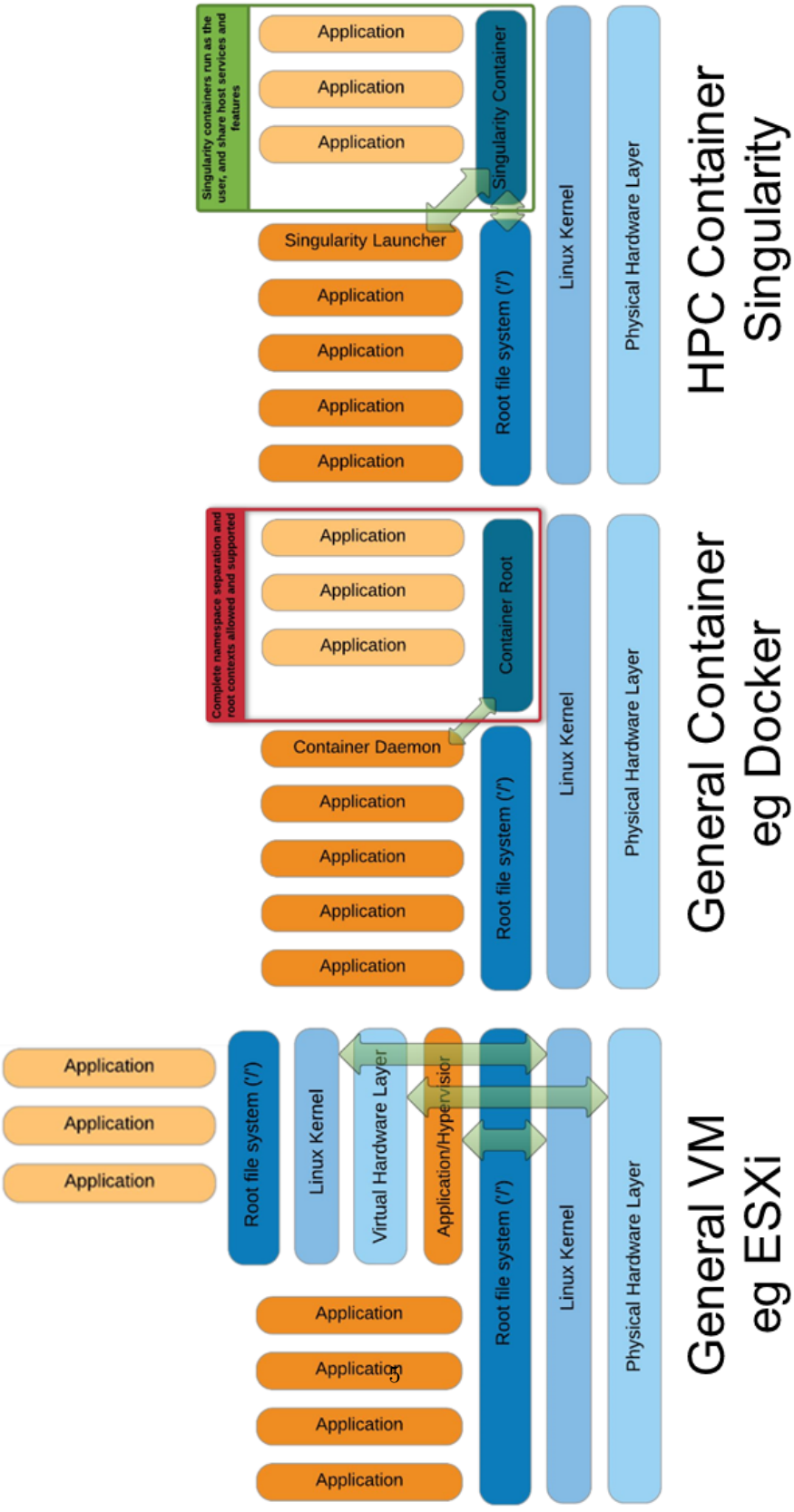

Figure 4: VM vs Docker vs Singularity

2.1.2 Interagire con il container

I container nel formato *sif* sono dei file eseguibili che possono essere lanciati direttamente da riga di comando, oppure seguendo il loro nome al comando *singularity run*; in questi modi vengono eseguiti i comandi elencati nella sezione *%runscript* del file delle definizioni.
Volendo invece lanciare un eseguibile contenuto nel container ma non elencato nella sezione *%runscript*, lo si può fare eseguendo il comando *singularity exec*, seguendo il suo percorso a quello del container:

```
singualarity exec name.sif /path/name.exe
```

Una immagine sif non è modificabile ma si può entrare nel container con il comando shell:

```
singualarity shell name.sif
```

Questo comando consente l'interazione col container attraverso l'apertura di una shell al suo interno come fosse una macchina virtuale. Le modifiche non sono consentite.
Nel caso in cui si abbia una sandbox il discorso cambia poiché questo strumento è pensato per modificare il workflow operativo ed è scrivibile; ci si accede con il comando:

```
sudo singualarity shell --writable sandbox-path
```

Al termine delle modifiche o di una fase di sviluppo potrebbe essere vantaggioso incapsulare la sandbox in un container non sovrascrivibile, dunque eseguire un cambio di formato, con il comando: max width=

```
singularity build new-sif sandbox-path
```

2.2 Uso di librerie MPI

Message Passing Interface è uno standard ampiamente usato nell'ambito HPC in applicazioni parallele poichè consente di distribuire il dominio applicativo sui nodi computazionali, facendogli scambiare dei messaggi. Le due implementazioni più diffuse di questo standard sono le librerie MPICH e OpenMPI, entrambe supportate da Singularity. Ci sono due modalità per usare le librerie MPI in Singularity:

- Hybrid model: si sfruttano l'installazione MPI dell'host e quella del container che devono essere compatibili
- Mounting nel container di un volume dell'host contenente l'installazione MPI e l'uso esclusivo di questo volume. Questo approccio richiede una condivisione del file system fra host e container che potrebbe non essere permessa nell' HPC poichè necessita di operazioni privilegiate.

Nei casi d'uso è stato utilizzato il primo approccio, il modello ibrido; per questo i container sono stati costruiti con la stessa versione di OpenMPI installata nell'host, la 3.1.6 .
Il diagramma del flusso che si viene a creare fra host e container è il seguente:

- Il launcher di MPI, e.g. *mpirun* viene chiamato dal resource manager, e.g. PBS
- OpenMPI chiama il process management daemon (ORTED) che lancia il container
- Singularity crea in memoria il container e il namespace environment
- Singularity lancia l'applicazione MPI contenuta nel container
- L'applicazione MPI lancia le librerie OpenMPI
- Le librerie OpenMPI si connettono a ORTED tramite il Process Management Interface (PMI)

In appendice è riportato lo script PBS per lanciare il caso d'uso WRF e si può notare che il comando è il seguente:

```
$ mpirun -n <#_RANKS> singularity exec <IMAGE/PATH> \
    </PATH/TO/BINARY/WITHIN/CONTAINER>
```

3 Casi d'uso

3.1 WRF-4.1.5

ARW-WRF (Advanced Research for Weather Research and Forecasting model) è un sistema di previsione numerica di mesoscala concepito sia per la ricerca che per le previsioni operative dello stato del tempo, sviluppato originariamente dal National Center for Atmospheric Research (NCAR)[8] [7]. Ad oggi il WRF dispone di una comunità molto ampia di sviluppatori in tutto il mondo, che testa periodicamente il software con nuove routine e moduli dedicati ad applicazioni specifiche prima di rilasciare periodicamente le versioni più aggiornate. La versione del modello testata è il WRF-4.1.5, (rilasciata il 10 marzo 2020) utilizzata per l'implementazione di un sistema operativo di previsione del tempo, MoSarT (Models of Sardinia Toolkit), presso l'istituto CNR-ISAC di Cagliari. In figura.5 viene rappresentato schematicamente il workflow del sistema di previsione *MoSarT*.

Il sistema MoSarT consiste attualmente di tre domini innestati (d01, d02, d03) con griglie regolari di 9, 3 e 1km (figura.6) e produce ogni giorno previsioni orarie fino 4 giorni.

I dati di input per ogni previsione sono i dati geografici statici (topografia, dati di uso del suolo) per la generazione dei domini di interesse(*geogrid*) ed i dati meteorologici globali GFS (Global Forecasting System) a 0.25x0.25, provenienti

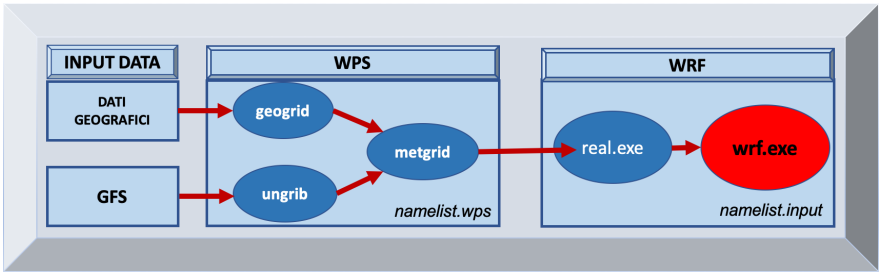

Figure 5: Workflow del sistema di simulazione numerica dello stato del tempo, WRF.

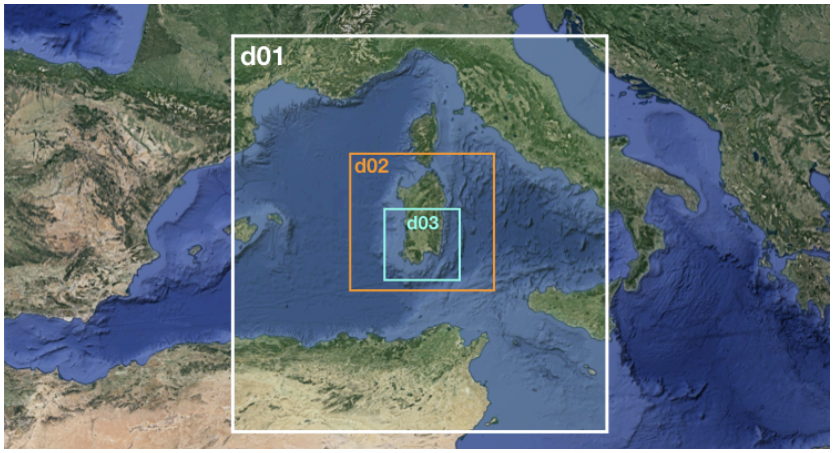

Figure 6: Tre domini annidatati del sistema di previsione MoSarT.

dal Centro di Previsione Americano NCEP (National Center for Enviromental Prediction). I dati GFS in formato grib2 vengono dapprima scompattati(*ungrib*) e poi interpolati(*metgrid*) sulle griglie dei modelli come come impostato nella *namelist.wps*. I dati così pre-processati passano al modulo principale di WRF che grazie alle impostazioni della *namelist.input* sulla fisica, sulla struttura verticale e sulla stabilità numerica, prepara dapprima le condizioni iniziali(*wrfinput*) e al contorno(*wrfbdy*) attraverso il programma *real.exe* ed infine esegue il codice principale *wrf.exe* per l'integrazione numerica in MPI.

In questo scenario il container prende il posto del modello compilato e si colloca al fianco degli elementi al contorno di modo che possa essere un elemento sostituibile nel workflow operativo, a patto che le nuove versioni di WRF non implichino una nuova architettura di questo.

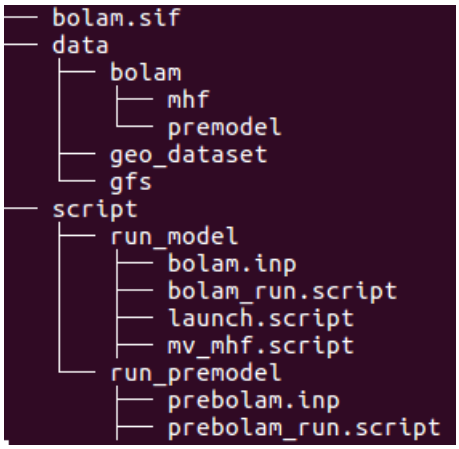

Figure 7: Architettura del caso d'uso Bolam

3.2 BOLAM

BOLAM è il primo modello sviluppato presso l'ISAC, a partire dagli anni '90. E' un modello idrostatico ad area limitata che integra le equazioni primitive, con parametrizzazione della convezione atmosferica. Esso attualmente viene integrato con un passo di griglia di 0.075 gradi (8.3 km), in coordinate geografiche ruotate, con 60 livelli atmosferici e 7 di suolo.
In particolare, BOLAM è basato principalmente su tre elementi:

- Container *bolam.sif* contenente gli applicativi Bolam e Prebolam
- Dati:
  - *geo_dataset*: dati stazionari usati in più simulazioni.
  - *gfs*: dati di input di Prebolam, possno essere i dati dell'analisi e di previsione del modello globale GFS (Global Forecasts System).
  - *bolam*: in questa cartella trovano posto i dati di input del modello, che a loro volta sono l'output del Prebolam (cartella *premodel*) e la cartella *mhf* con l'output del modello.
- Script: impostazioni delle variabili d'ambiente del wokflow operativo, che sono correlate con quelle della simulazione; questa cartella contiene gli script per lanciare il Prebolam e Bolam, quest'ultimo attraverso lo scheduler PBS.

A Recipe

A.1 wrf-4.1.5_and_wps.def

```
BootStrap: library
From: debian:latest

%setup

%files
  SOURCES/* /opt

%environment

%runscript
  /opt/WRF-4.1.5/test/em_real/wrf.exe

%startscript

%test

%labels
  Author f.grasso@isac.cnr.it
  Version 0.1
  exec WRF-4.1.5  and last released WPS
%help
  WRF-4.1.5 and WPS installed through:
  debian:latest
  automake-1.16
  libpng-1.2.50
  jasper-1.900.1
  openmpi-3.1.6
  hdf5-1.8.21
  netcdf-c-4.7.3
  netcdf-fortran-4.5.2

%post
  echo "*****************  Hello from inside the container"
  apt-get update
  apt-get install -y texinfo vim dialog locales libtool libtool-bin
      autoconf git flex gawk build-essential gfortran g++ zlib1g-dev
       csh libcurl4-gnutls-dev

  cd /opt
  tar xzvf automake-1.16.tar.gz
  cd /opt/automake-1.16
  sed -i 's:/\\\${:/\\\$\\{:' bin/automake.in
  autoreconf -i -f
  ./configure --prefix=/usr/local --docdir=/usr/share/doc/automake
      -1.16
  make
  make install

  cd /opt
  tar xzvf openmpi-3.1.6.tar.gz
  cd /opt/openmpi-3.1.6
  ./configure --prefix=/usr/local --enable-static
  make all install
  ldconfig

  cd /opt
```

```
tar xvf hdf5-1.8.21.tar
cd /opt/hdf5-1.8.21
CC=/usr/local/bin/mpicc ./configure --prefix=/usr/local --enable-
    fortran --enable-cxx
make  && make check && make install
ldconfig

cd /opt
tar xvf netcdf-c-4.7.3.tar.gz
cd netcdf-c-4.7.3
CC=mpicc  CFLAGS='-fPIC' LDFLAGS=-L/usr/local/lib CPPFLAGS=-I/usr/
    local/include ./configure --prefix=/usr/local --disable-dap-
    remote-tests
make
make check
make install
ldconfig

cd /opt
tar xzvf netcdf-fortran-4.5.2.tar.gz
cd /opt/netcdf-fortran-4.5.2
CPPFLAGS="-I/usr/local/include -DgFortran" LDFLAGS=-L/usr/local/
    lib FC=mpif90 F77=mpif90 CC=mpicc F77FLAGS="-O3 -fno-second-
    underscore" FCFLAGS="-O3 -fno-second-underscore" FFLAGS="-O3 -
    fno-second-underscore" ./configure --prefix=/usr/local --
    enable-shared
make check install
ldconfig

cd /opt
tar xzvf libpng-1.2.50.tar.gz
cd libpng-1.2.50
./configure
make test
make install
ldconfig

cd /opt
tar xzvf jasper-1.900.1.tar.gz
cd jasper-1.900.1
./configure
make
make install
ldconfig

cd /opt
tar xzvf WRF-4.1.5.tar.gz
cd /opt/WRF-4.1.5
mv /opt/configure.wrf .
mkdir netcdf_links
ln -s /usr/local/include netcdf_links/include
ln -s /usr/local/lib netcdf_links/lib
./compile em_real

cd /opt
git clone https://github.com/wrf-model/WPS
cd WPS
```

```
mv ../configure.wps .
mkdir netcdf_links
ln -s /usr/local/include netcdf_links/include
ln -s /usr/local/lib netcdf_links/lib
./compile
```

A.1.1 PBS

```
#!/bin/sh

### Nome del job

#PBS -N WRFV3:test_sing.exe

### Declare job non-rerunable

#PBS -r n

### Output files

#PBS -e wrf1.err

#PBS -o wrf1.out

### Inserire il proprio indirizzo email

#PBS -M fazioli@isac.cnr.it

#PBS -m ae

### Exports all environment variables to the job
#PBS -V

### Coda su cui lanciare il job

#PBS -q socrate

#PBS -l nodes=socrate14:ppn=15+socrate15:ppn=15+socrate16:ppn=15+
    socrate17:ppn=15

###cd $PBS_O_WORKDIR
cd /cantiere/grazioli
rm -f endwrf.txt

echo Running on host `hostname`

echo Time is `date`

echo Directory is `pwd`

echo This jobs runs on the following processors:

echo `cat $PBS_NODEFILE`

# Define number of processors

NPROCS=`wc -l < $PBS_NODEFILE`
```

```
echo This job has allocated $NPROCS nodes

/opt/openmpi-3.1.6/bin/mpirun -v -machinefile $PBS_NODEFILE -np
    $NPROCS singularity exec wrf-4.1.5_and_wps.sif /opt/WRF-4.1.5/
    main/wrf.exe > wrfout2.txt
```

A.2 bolam.def

```
BootStrap: library
From: debian:latest

%setup

%files
  SOURCES/* /opt

%environment

%runscript

%startscript

%test

%labels
  Author f.grasso@isac.cnr.it
  Version 0.1
%help
  Bolam installed through:
  debian:latest
  automake-1.16
  openmpi-3.1.6
  hdf5-1.8.21
  netcdf-c-4.7.3
  netcdf-fortran-4.5.2
  eccodes-2.17.0

%post
  echo "*****************  Hello from inside the container"
  apt-get update
  apt-get install -y texinfo vim dialog locales libtool libtool-bin
      autoconf git flex gawk build-essential gfortran g++ zlib1g-dev
       csh libcurl4-gnutls-dev libssl-dev wget subversion

  cd /opt
  tar xzvf automake-1.16.tar.gz
  cd /opt/automake-1.16
  sed -i 's:/\\\${:/\\\$\\{:' bin/automake.in
  autoreconf -i -f
  ./configure --prefix=/usr/local --docdir=/usr/share/doc/automake
      -1.16
  make
  make install

  cd /opt
  tar xzvf openmpi-3.1.6.tar.gz
```

```
cd /opt/openmpi-3.1.6
./configure --prefix=/usr/local --enable-static
make all install
ldconfig

cd /opt
tar xvf hdf5-1.8.21.tar
cd /opt/hdf5-1.8.21
CC=/usr/local/bin/mpicc ./configure --prefix=/usr/local --enable-
    fortran --enable-cxx
make  && make check && make install
ldconfig

cd /opt
tar xvf netcdf-c-4.7.3.tar.gz
cd netcdf-c-4.7.3
CC=mpicc  CFLAGS='-fPIC' LDFLAGS=-L/usr/local/lib CPPFLAGS=-I/usr/
    local/include ./configure --prefix=/usr/local --disable-dap-
    remote-tests
make
make check
make install
ldconfig

cd /opt
tar xzvf netcdf-fortran-4.5.2.tar.gz
cd /opt/netcdf-fortran-4.5.2
CPPFLAGS="-I/usr/local/include -DgFortran" LDFLAGS=-L/usr/local/
    lib FC=mpif90 F77=mpif90 CC=mpicc F77FLAGS="-O3 -fno-second-
    underscore" FCFLAGS="-O3 -fno-second-underscore" FFLAGS="-O3 -
    fno-second-underscore" ./configure --prefix=/usr/local --
    enable-shared
make check install
ldconfig

cd /opt
tar xzvf cmake-3.17.1.tar.gz
cd cmake-3.17.1
./bootstrap
make
make install

cd /opt
tar xzvf eccodes-2.17.0-Source.tar.gz
mkdir build
cd build
cmake ../eccodes-2.17.0-Source
make
ctest
make install
ldconfig

cd /opt/pacchetto_meteo/rad-ecmwf-new/package
./makeall

cd /opt/pacchetto_meteo
make
```

References

[1] https://www.openstack.org/software.

[2] https://www.zerounoweb.it/techtarget/searchdatacenter/storage-server-networking/software-container-come-si-possono-gestire-e-quali-sono-i-vantaggi-per-lutente/.

[3] http://www.hpc.cineca.it/software/singularity.

[4] https://sylabs.io/docs/.

[5] https://sylabs.io/guides/3.0/user-guide/definition$_f iles.html$.

[6] Bauer MW Kurtzer GM, Sochat V. Singularity: Scientific containers for mobility of compute. *PLoS ONE*, (12(5)).

[7] William Skamarock, J. Klemp, Jimy Dudhia, David Gill, Dale Barker, Wei Wang, and Jordan Powers. A description of the advanced research wrf version 2. Technical report, 06 2005.

[8] William C Skamarock, Joseph B Klemp, Jimy Dudhia, David O Gill, Dale M Barker, Wei Wang, and Jordan G Powers. A description of the advanced research wrf version 3. ncar technical note-475+ str. 2008.