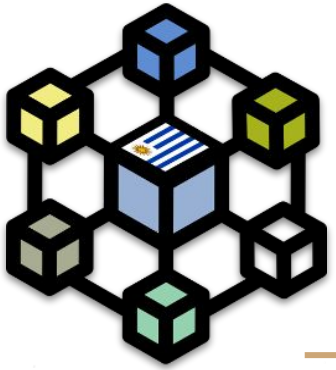
Centro nacional de supercomputación

-- cluster.uy --



Aspectos Avanzados

(a.k.a. La Parte 2)

Tipo de trabajo

- Secuencial
- Multihilado
- Distribuido
- Con GPU

Trabajo secuencial

Ejecuta procesos de forma secuencial. Ejemplo:

```
#!/bin/bash -l
#SBATCH --ntasks=1
#SBATCH --mem=1G
#SBATCH --time=6:00:00
#SBATCH --output=salida.out
#SBATCH --partition=normal
#SBATCH --qos=normal
cd ~/my_greetings
./hello_world
./bye_world
```

Una única tarea

1Gb de RAM

6 hs. de ejecución máx.

Salida estándar y de error a **`salida.out`**

“Procesos” que se van a ejecutar en el trabajo

Trabajo multihilado

Ejecuta **procesos** de forma secuencial pero (*al menos*) un proceso tiene más de un hilo de ejecución (i.e. puede usar **más de un núcleo de CPU**). Ejecuta

```
#!/bin/bash -l
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=16
#SBATCH --mem-per-cpu=512M
#SBATCH --time=1-00:00:00
#SBATCH --partition=normal
#SBATCH --qos=normal
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
./openmp_hello_world
```

Una única tarea

16 CPUs por tarea

512MB de RAM por CPU

Un día de ejecución

Variable de SLURM con la cantidad de CPUs por tarea

Trabajo distribuido

Cuenta con **varios procesos** ejecutando simultáneamente, potencialmente en diferentes nodos. Los procesos deben utilizar la red para comunicarse. Cada proceso puede ser multihilado. Por ejemplo:

```
#!/bin/bash -l
#SBATCH --ntasks=16
#SBATCH --cpus-per-task=1
#SBATCH --mem=512M
#SBATCH --partition=normal
#SBATCH --qos=normal
./hello_world
./bye_world
```

Trabajo distribuido: usando srun

Ejecución de procesos homogéneos usando srun:

```
#!/bin/bash -l
#SBATCH --ntasks=16
#SBATCH --cpus-per-task=1
#SBATCH --mem=512M
#SBATCH --partition=normal
#SBATCH --qos=normal
srun ./hello_world
srun ./bye_world
```

Ejecuta 16 copias de hello_world, y luego 16 copias de bye_world

Trabajo distribuido: usando srun

Ejecución de procesos heterogéneos usando srun:

```
#!/bin/bash -l
#SBATCH --ntasks=16
#SBATCH --cpus-per-task=1
#SBATCH --mem=512M
#SBATCH --time=1-00:00:00
#SBATCH --partition=normal
#SBATCH --qos=normal
srun --ntasks=15 ./hello_world &
srun --ntasks=1 ./bye_world &
wait
```

Ejecuta 15 copias de hello_world y, ***simultáneamente***, 1 copia de bye_world

Trabajo distribuido: usando MPI

Ejemplo de script para lanzar un trabajo MPI:

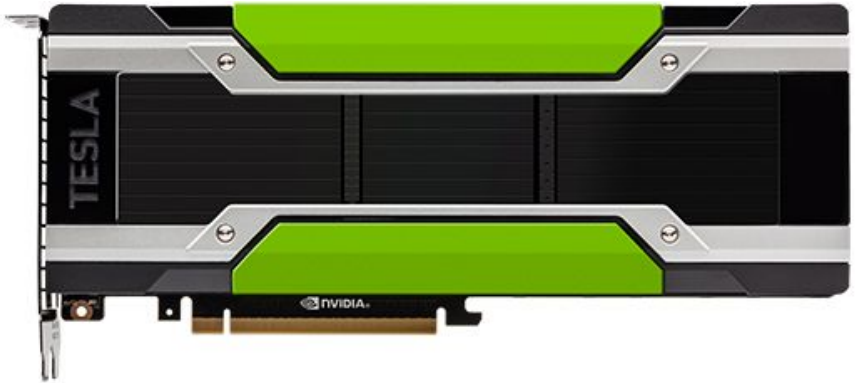
```
#!/bin/bash -l
#SBATCH --ntasks=16
#SBATCH --mem=512M
#SBATCH --time=1-00:00:00
#SBATCH --partition=normal
#SBATCH --qos=normal
module load mpi/mpich-3.2-x86_64
mpirun ./mpi_hello_world
```

Carga el módulo MPICH 3.2

Trabajo con GPU

Características:

- Debe utilizar al menos un núcleo de CPU por nodo.
- La GPU se asigna de forma exclusiva (no se puede compartir).
- Se encuentra disponible CUDA y OpenACC.



Trabajo con GPU

Ejemplo:

```
#!/bin/bash -l
#SBATCH --ntasks=1
#SBATCH --mem=512M
#SBATCH --time=12:00:00
#SBATCH --partition=normal
#SBATCH --qos=gpu
#SBATCH --gres=gpu:1
module load cuda/11.0
./hello_cuda
```

Necesario para usar GPUs

Pide 1 GPU por nodo

Carga el CUDA 11.0

Trabajo interactivo con GPU

Necesario para compilar:

```
login$ interactivo -gpu

node14$ module load cuda/11.0

node14$ nvcc -o hello_cuda hello_cuda.cu
```

Trabajo interactivo con 1 GPU por 30 minutos

Carga CUDA 11.0

Modo de ejecución

- Normal
  - Recursos no son expropiables
  - Acceso limitado a los recursos
- Mejor esfuerzo
  - Recursos son expropiables
  - Acceso total a los recursos

Modo de ejecución: normal

```
#!/bin/bash -l
…
#SBATCH --partition=normal
#SBATCH --qos=normal
…
```

Sin GPU

```
#!/bin/bash -l
…
#SBATCH --partition=normal
#SBATCH --qos=gpu
…
```

Con GPU

Modo de ejecución: mejor esfuerzo

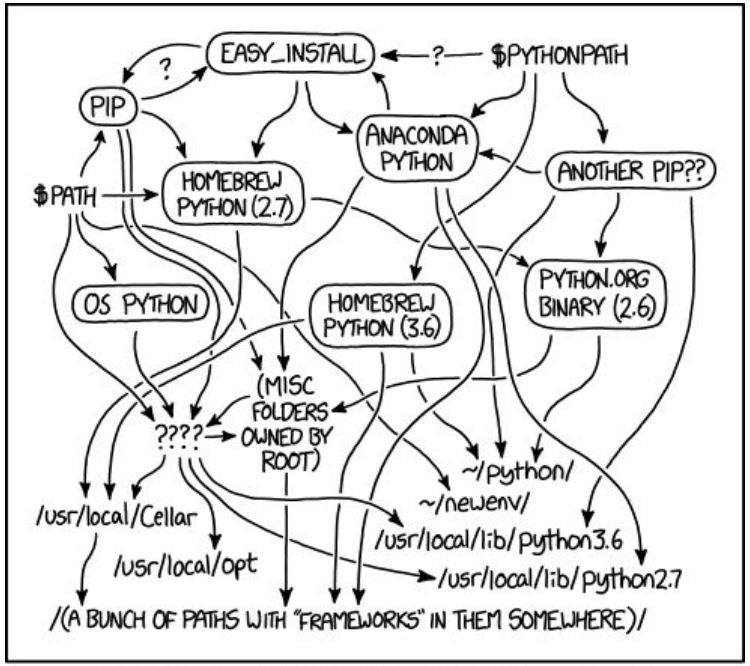
```
#!/bin/bash -l
…
#SBATCH --partition=besteffort
#SBATCH --qos=besteffort
…
```

Sin GPU

```
#!/bin/bash -l
…
#SBATCH --partition=besteffort
#SBATCH --qos=besteffort_gpu
…
```

Con GPU

Entornos virtuales de Python

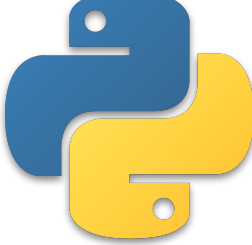


MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

Pip Installs Packages (pip)

- **`pip`** es un manejador de paquetes para Python.
- Facilita muchísimo la instalación de paquetes.
- Más de **195 mil** paquetes disponibles en https://pypi.org/
- *Peeero...* en general requiere acceso **`root`**.

Entornos virtuales al rescate



- Crea y administra entornos de Python totalmente aislados.
- Facilita la instalación de paquetes en el **`HOME`** de cada usuario usando **`pip`** (no precisa **`root`**).
- Crear entorno virtual en **Python 2.7.5** o **Python 3.4.9**:

```
$ virtualenv-2 [nombre del entorno]
```

```
$ virtualenv-3 [nombre del entorno]
```

- Se crea un directorio con el nombre del entorno.

```
$ source [nombre del entorno]/bin/activate
```

Entornos virtuales al rescate

Por ejemplo:

```
login$ interactivo -g
node10$ virtualenv-3 ~/mitesis
node10$ source ~/mitesis/bin/activate
(mitesis) node10$ pip install pandas
(mitesis) node10$ deactivate
node10$
```

Conda: alternativa a entornos virtuales

Conda es un manejador de paquetes avanzado. Permite instalar Python como un paquete.

Instalación de miniconda:

```
login$ interactivo -g
node18$ wget
https://repo.anaconda.com/miniconda/Miniconda3-latest
-Linux-x86_64.sh
node18$ ./Miniconda3-latest-Linux-x86_64.sh
```

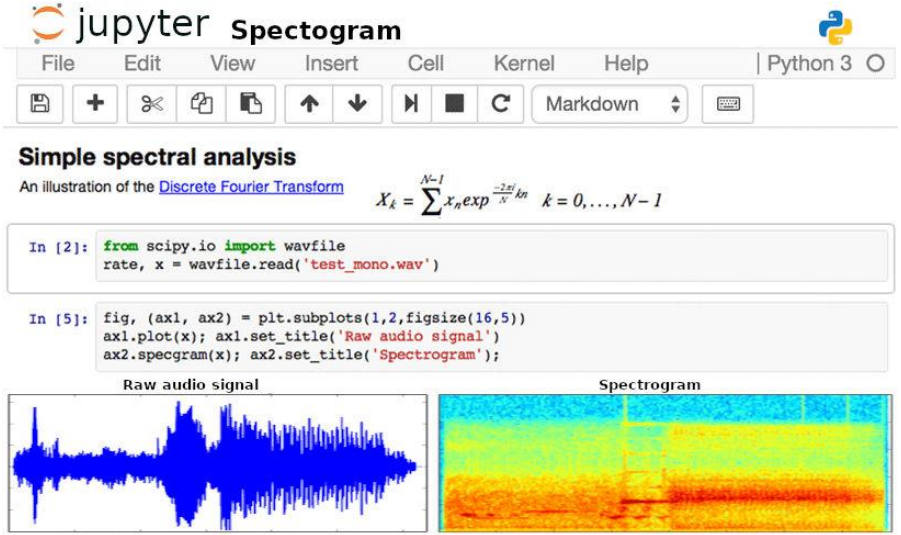
Conda: alternativa a entornos virtuales

Por ejemplo:

```
login$ interactivo -g
node22$ source miniconda3/etc/profile.d/conda.sh
node22$ conda create -n mitesis python=3.9
node22$ conda env list
…
node22$ conda activate mitesis
(mitesis) node22$ conda install pandas
(mitesis) node22$ conda deactivate
node22$
```

Jupyter Notebook usando Conda

Un documento de Jupyter Notebook contiene una lista ordenada de celdas de entrada/salida que pueden contener código, texto (usando Markdown), matemáticas, gráficos y texto enriquecidos.



Jupyter Notebook usando Conda


```
login$ interactivo -g
node23$ source miniconda3/etc/profile.d/conda.sh
(mitesis) node23$ conda install notebook
(mitesis) node23$ jupyter-notebook --no-browser
--port=8000 --ip=0.0.0.0
```

```
Jupyter Notebook 6.5.2 is running at:
http://127.0.0.1:8000/?token=cc5a499fdb24c1d1c751b3047
69336d22ed42b3a5de5009d
```

Jupyter Notebook usando Conda

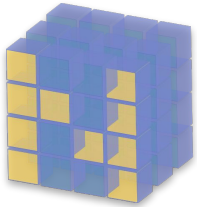
Para acceder desde nuestro navegador es necesario establecer un túnel ssh. Desde nuestra máquina local ***en otra terminal*** debemos establecer el túnel ssh con los parámetros indicados:

```
$ ssh -L 8000:node23:8000 -N <usuario>@cluster.uy
```

Luego desde nuestro navegador accedemos a la dirección del notebook:

```
Jupyter Notebook 6.5.2 is running at:
http://127.0.0.1:8000/?token=cc5a499fdb24c1d1c751b304
769336d22ed42b3a5de5009d
```

Algunos paquetes gestionados por Conda



NumPy

seaborn

TensorFlow

pandas

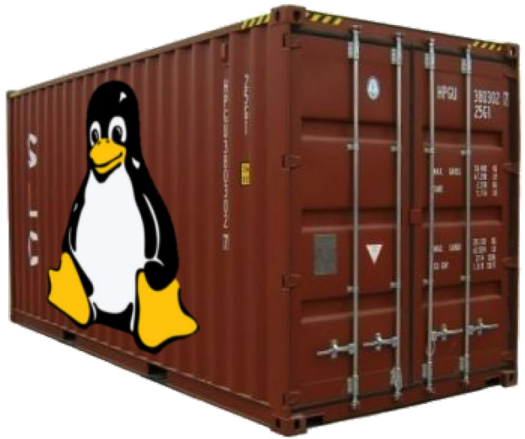
$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$

SciPy

Keras

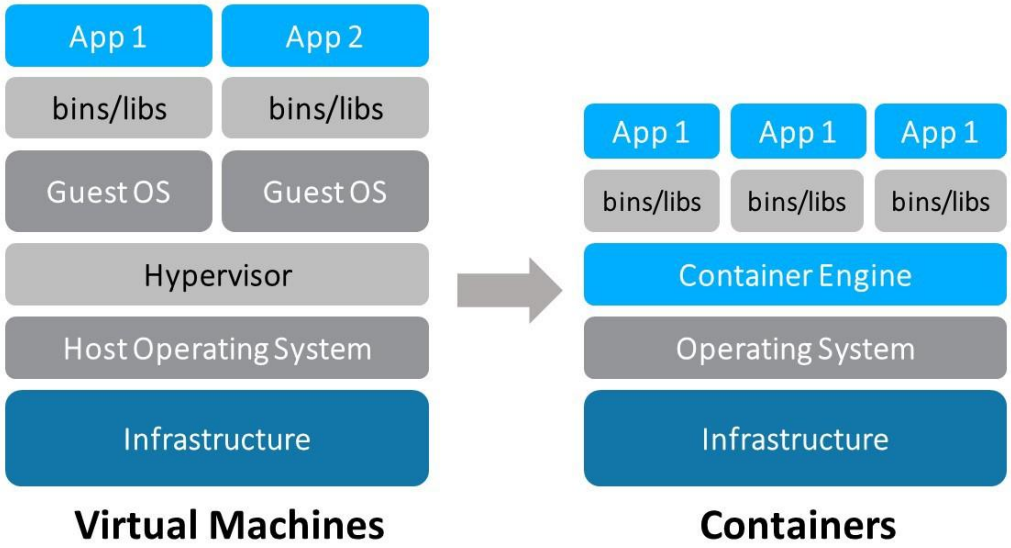
jupyter

Contenedores

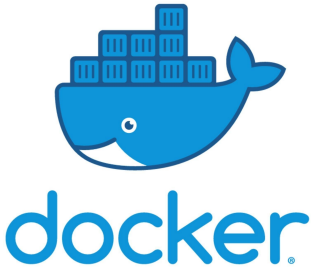


¿Qué es un contenedor?

- Es un método de virtualización a nivel de sistema operativo.
  - El sistema operativo **no es virtualizado**.



¿Por qué usar un contenedor?



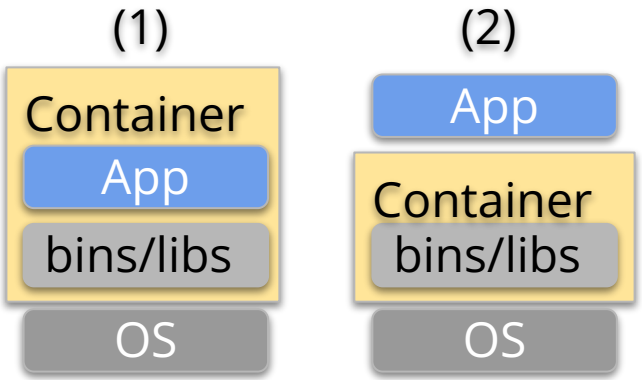
- Permite múltiples espacios de usuario **aislados**.
  - Pueden mantenerse diferentes versiones de bibliotecas.
- Comparado con una VM:
  - Rápida instanciación y apagado .
  - Muy poco overhead de memoria y procesador.
    - **Sin penalización** (casi) en el desempeño.
- Generalmente se utiliza un contenedor individual por aplicación.
  - Se empaquetan todas las dependencias de esa aplicación.
  - Permite que la aplicación sea portable, compartible y reproducible.

¿Que es Singularity?

- Solución de contenedorización de código abierto.
- Enfocada a infraestructuras de alto desempeño.
- Compatible con Docker.

Sistema de archivos del contenedor

- El contenedor mantiene su propio sistema de archivos virtualizado y comparte algunos directorios del sistema de archivos real.
- Por defecto los directorios `~`, `/tmp` y el directorio *actual* al iniciar el contenedor son compartidos con el sistema de archivos real.
- Opciones de empaquetado:
  - (1) Dependencias y aplicación.
  - (2) Sólo dependencias.



¿Cómo crear un contenedor?

- Es necesario partir de un contenedor **pre-armado**.
- Recomendamos utilizar https://hub.docker.com
- Dos posibles escenarios:
  - Ya **existe** un contenedor pre-armado con todo el entorno que precisa mi aplicación :-)
  - Es necesario **adaptar** un contenedor para ejecutar mi aplicación :-(

Descargar un contenedor pre-armado

- Se usa **`build`** para descargar y convertir un contenedor Docker.
- Por ejemplo: descarga del contenedor **`godlovedc/lolcow`**

```
login$ interactivo -g
node13$ singularity build lolcow.simg
docker://godlovedc/lolcow
```

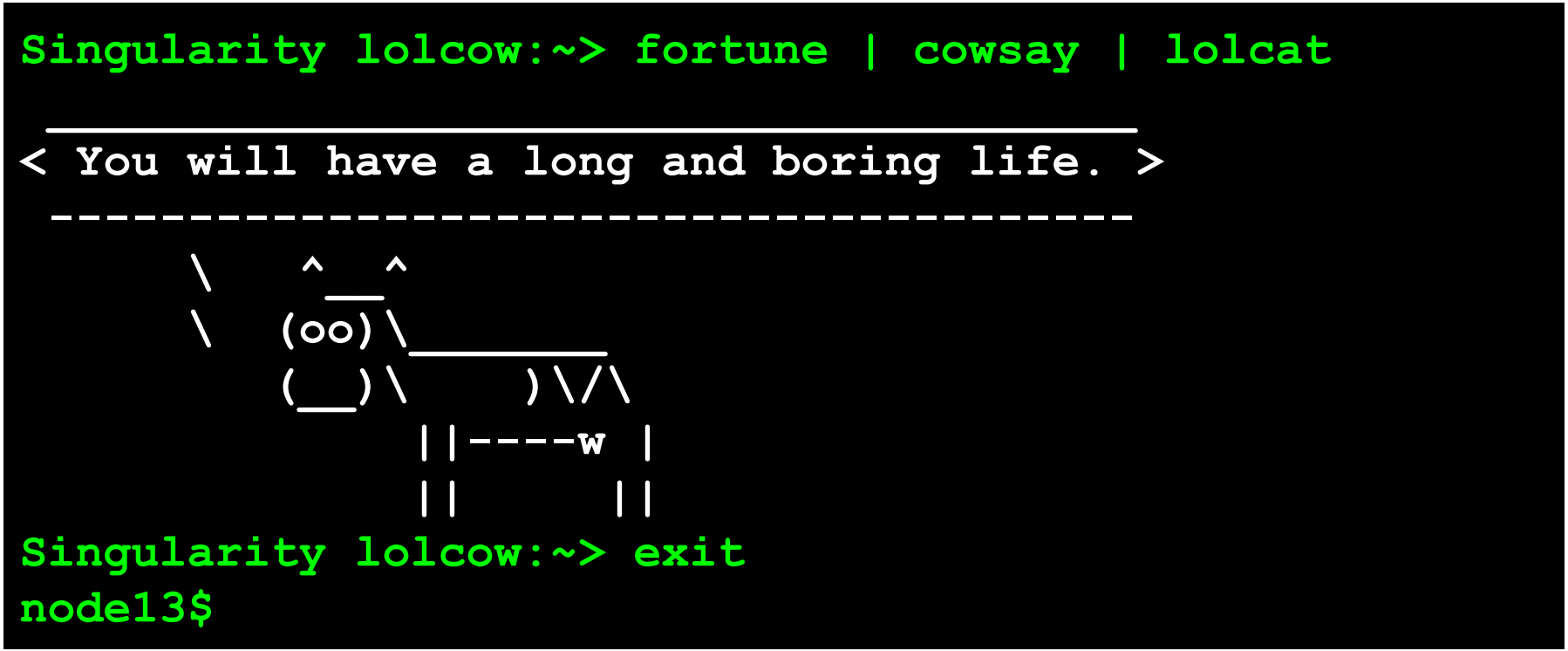
Instanciación de un contenedor

- Formas de instanciación:
  - Interactiva con **`shell`**.
  - Ejecución de un comando con **`exec`**.
- Ejemplo de instanciación interactiva:

```
node13$ singularity shell lolcow.simg
Singularity: Invoking an interactive shell within
container...

Singularity lolcow:~>
```

Ejemplo de ejecución interactiva



(Otro) Ejemplo de ejecución interactiva

```
node13$ echo “puts ‘Hello World!’” > hello.rb
node13$ singularity shell lolcow.simg
Singularity lolcow:~> ruby hello.rb
Hello world!
Singularity lolcow:~> exit
node13$
```

Ejecución de un comando (no interactivo)

- Ejecuta un único comando y luego detiene el contenedor.
- Se debe usar el comando **`exec`**:

```
singularity exec [contendor] [comando]
```

- Ejemplo:

```
node13$ singularity exec lolcow.simg ruby hello.rb
Hello world!
node13$
```

Ejemplo de integración con SLURM

```
#!/bin/bash
#SBATCH --job-name=helloruby
#SBATCH --ntasks=1
#SBATCH --time=05:00
#SBATCH --partition=normal
#SBATCH --qos=normal

singularity exec lolcow.simg ruby hello.rb
```

¿Y si tengo que adaptar un contenedor?

Voy a necesitar lo siguiente:

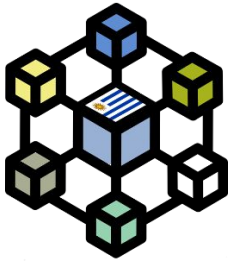
- Una máquina con Linux con acceso **`root`**.
  - Puede ser una máquina virtual (p.ej. Virtualbox).
- Tener Singularity instalado.
  - En particular la **versión 2.6.x**
  - Descargable desde: https://sylabs.io/singularity/

Adaptando un contenedor en 3 pasos “fáciles”

1. Descargar el contenedor en modo **`sandbox`**.
   a. **`singularity build --sandbox lolcow/ docker://godlovedc/lolcow`**
2. Iniciar un **`shell`** en el contenedor en modo escritura e instalar/modificar lo que uno desee.
   a. **`singularity shell --writable lolcow/`**
3. Empaquetar el contenedor en una imagen **`simg`** inmutable.
   a. **`singularity build lolcow-new.simg lolcow/`**

¡Es necesario ejecutar todos estos pasos como **`root`**!

Links de interés



- Ayuda de cluster.uy: https://cluster.uy/ayuda
- Ayuda de Singularity 2.6: https://sylabs.io/guides/2.6/user-guide/
- Docker Hub: https://hub.docker.com

¡Gracias por su atención!

soporte@cluster.uy

@clusteruy

2714 2714 int 12040

@clusterstatus_bot

t.me/clusteruy