

MPI para Python Utilizando Docker.

Profesor: Lic. Pautsch Germán Andrés.

Lic. Martini Esteban.

Estudiante: Britez Ezequiel Agustín

Legajo:LS00743.

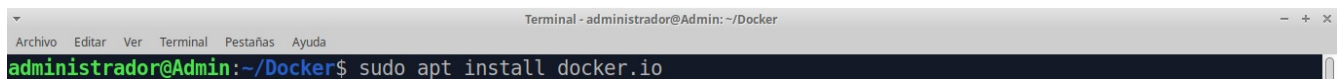
Utilizaremos la libreria mpi4py que nos permite utilizar la Interfaz de Paso de Mensaje (MPI) entre los procesos de una manera sencilla utilizando el lenguaje de programación **Python**.
En este caso utilizaremos la distribución de Xubuntu de Linux.

Herramientas Necesarias:
_Visual Studio Code (VS).
_Docker.

Instalando Docker en linux.

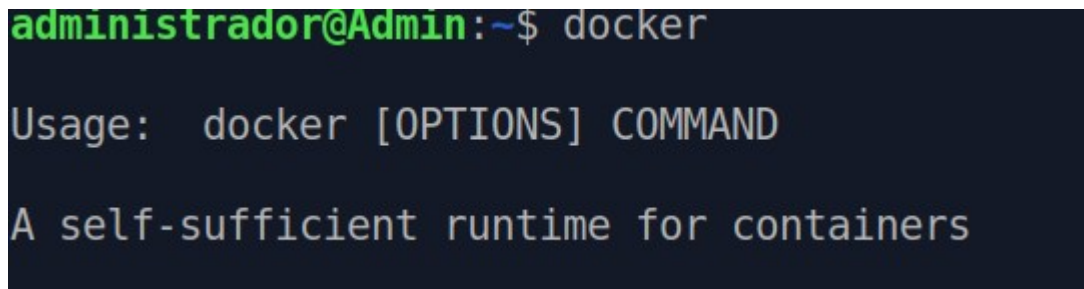
1) Primero abrimos la consola de comando y ejecutamos el siguiente comando:

sudo apt install docker.io

A screenshot of a terminal window titled 'Terminal - administrador@Admin: ~/Docker'. The terminal shows the command 'administrador@Admin:~/Docker\$ sudo apt install docker.io' being entered and executed. The window has a menu bar with 'Archivo', 'Editar', 'Ver', 'Terminal', 'Pestañas', and 'Ayuda'.

2) verificamos que ha sido instalado con el comando:

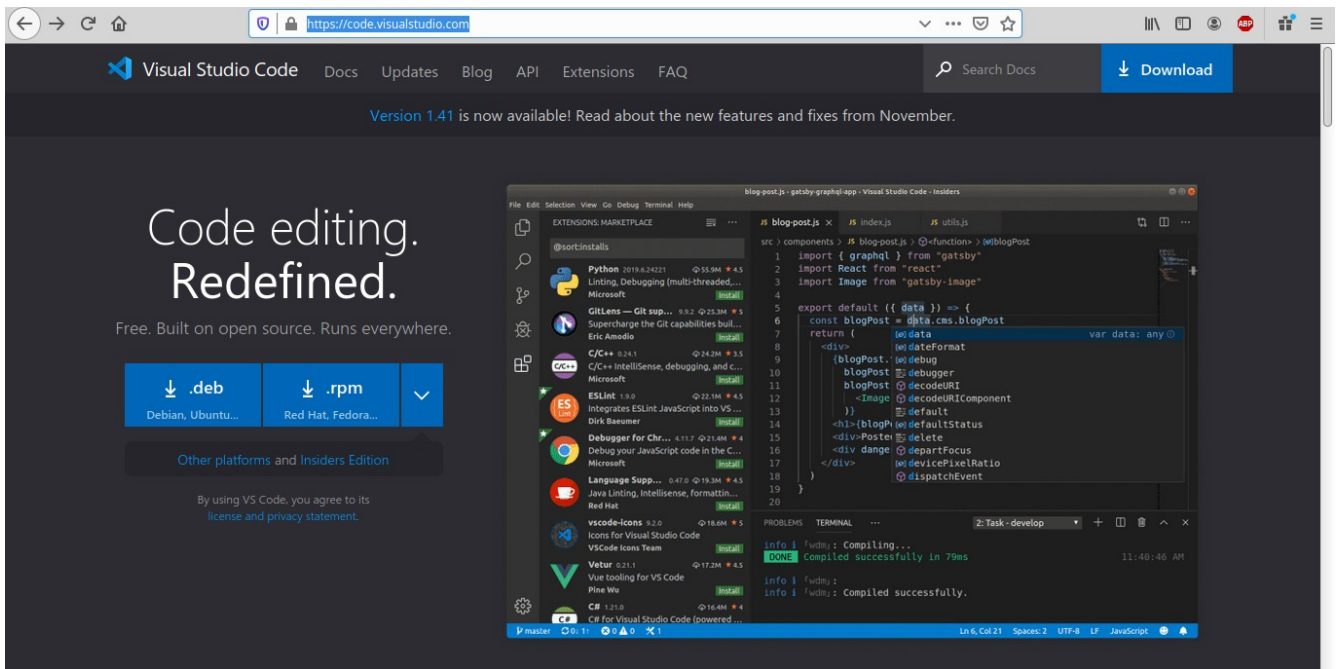
docker

A screenshot of a terminal window showing the output of the 'docker' command. The prompt is 'administrador@Admin:~\$ docker'. The output is: 'Usage: docker [OPTIONS] COMMAND' followed by 'A self-sufficient runtime for containers'.

Instalando Visual Studio en linux.

1) Descargamos el archivo “*.deb*” desde el sitio web oficial.

<https://code.visualstudio.com/>

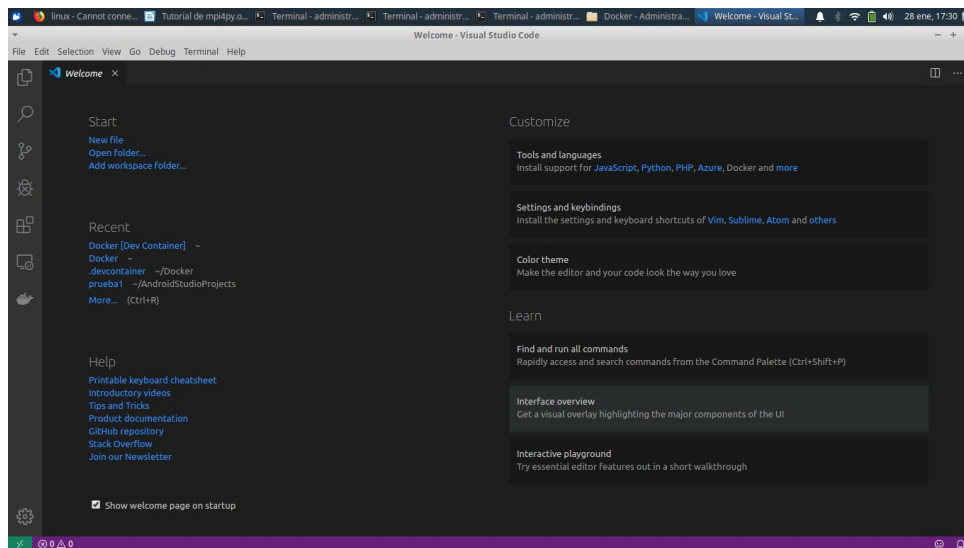
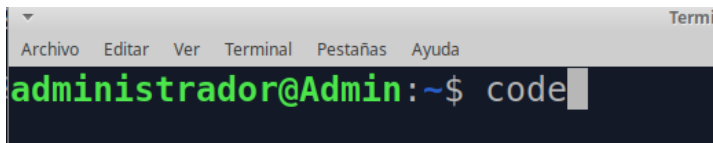


2) Ir a la ubicación del archivo descargado y ejecutar el siguiente comando.

Sudo dpkg -i <Nombre del archivo>

3) Ejecutamos en la consola.

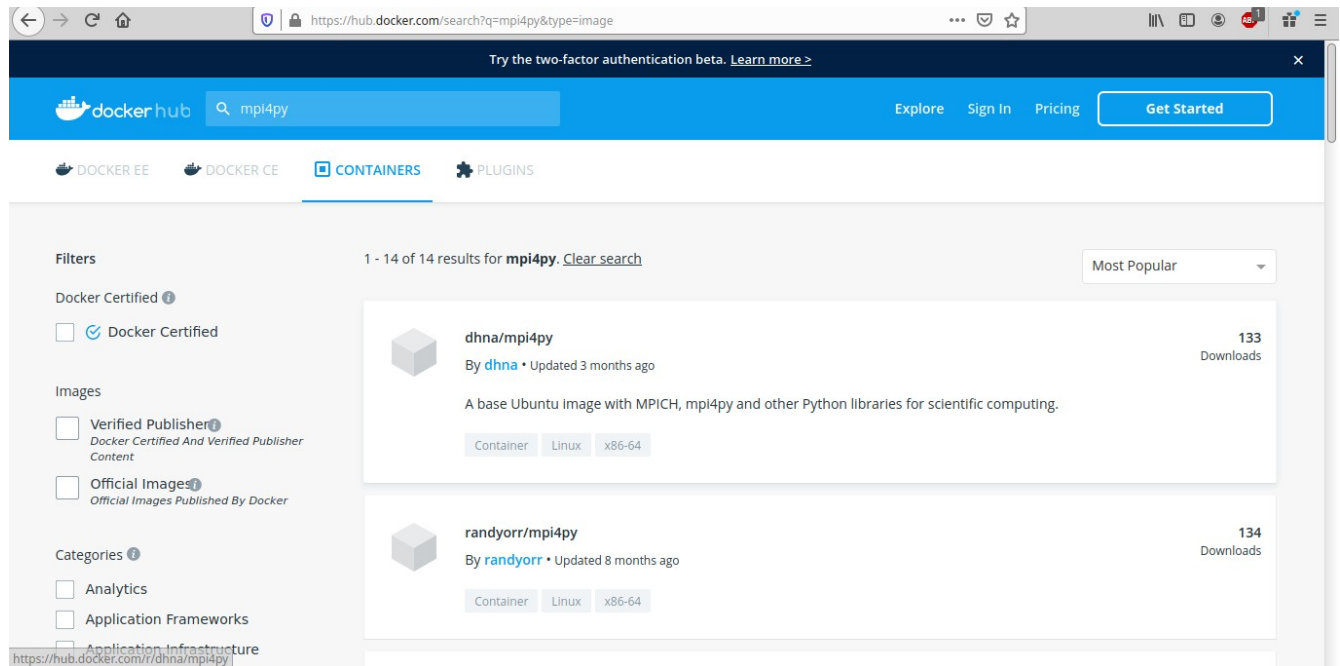
Code (Abrirá el visual estudio code)



Pasos para utilizar MPI en Python.

(Se instalará el contenedor en Docker que se encuentra en la dirección <https://hub.docker.com/r/dhna/mpi4py>.)

Si el enlace no existe hay que buscar la imagen en el siguiente enlace <https://hub.docker.com/> y buscar “mpi4py” que es la imagen que estamos usando.



1) Abrimos una terminal y ejecutamos.

docker pull dhna/mpi4py

(En este caso ya se encuentra instalado).

```
administrador@Admin:~$ docker pull dhna/mpi4py
Using default tag: latest
latest: Pulling from dhna/mpi4py
Digest: sha256:3ff9ab29249fb222cf639e6fac41a0e26888ad3b48bac730fd32b0cf43a95b89
Status: Image is up to date for dhna/mpi4py:latest
docker.io/dhna/mpi4py:latest
```

2) Podremos ver si instalo la imagen con el comando:

Docker images

```
administrador@Admin:~$ docker images
```

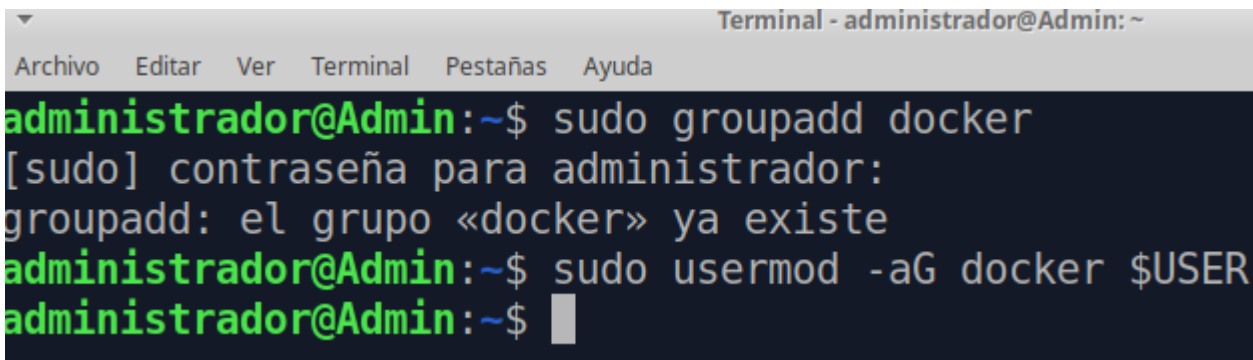
REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
cirrusci/flutter	latest	345e5d03df89	6 weeks ago	3.21GB
dhna/mpi4py	latest	1c912c836805	3 months ago	882MB

3) A continuación debemos ejecutar los siguientes comandos para dar los permisos suficiente al visual studio code.

Comandos en el siguiente orden.

sudo groupadd docker

sudo usermod -aG docker \$USER

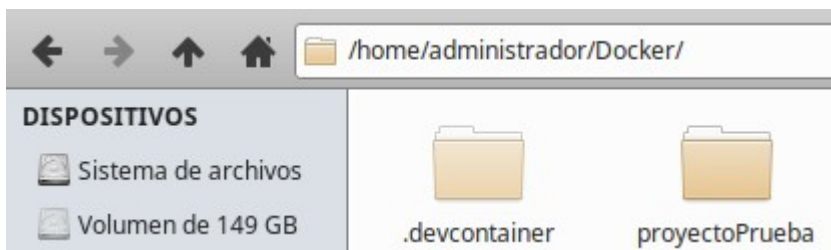


```
Terminal - administrador@Admin: ~
Archivo  Editar  Ver  Terminal  Pestañas  Ayuda

administrador@Admin:~$ sudo groupadd docker
[sudo] contraseña para administrador:
groupadd: el grupo «docker» ya existe
administrador@Admin:~$ sudo usermod -aG docker $USER
administrador@Admin:~$
```

Reiniciamos el Sistema Operativo.

4) Una vez iniciado el sistema creamos una carpeta llamada "Docker" en la ruta del HOME, y dentro de ella otra carpeta ".devcontainer".



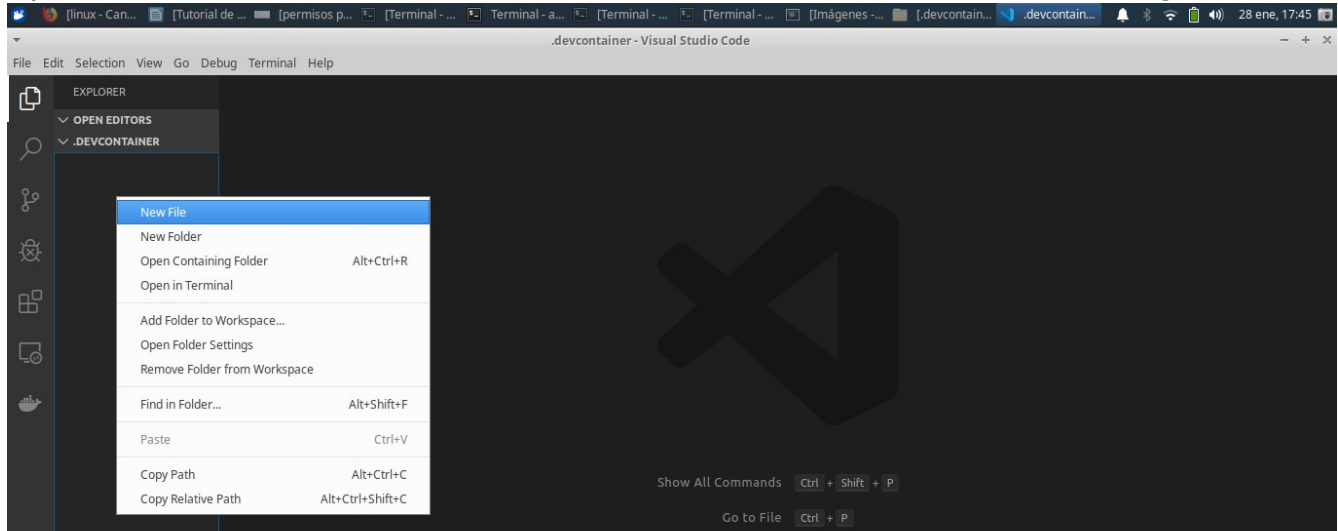
5) Abrimos el Visual Studio Code en la ruta de la carpeta .devcontainer .

Dentro de la carpeta ejecutamos:

code . (se abrirá el VS en esta dirección).

```
Archivo  Editar  Ver  Terminal  Pestañas  Ayuda
administrador@Admin:~/Docker/.devcontainer$ code .
```

6) Creamos un archivo nuevo desde VS, llamado "devcontainer.json".



7) En el archivo ingresamos los siguientes datos de configuración.

```
{
"name": "Docker mpi4py",

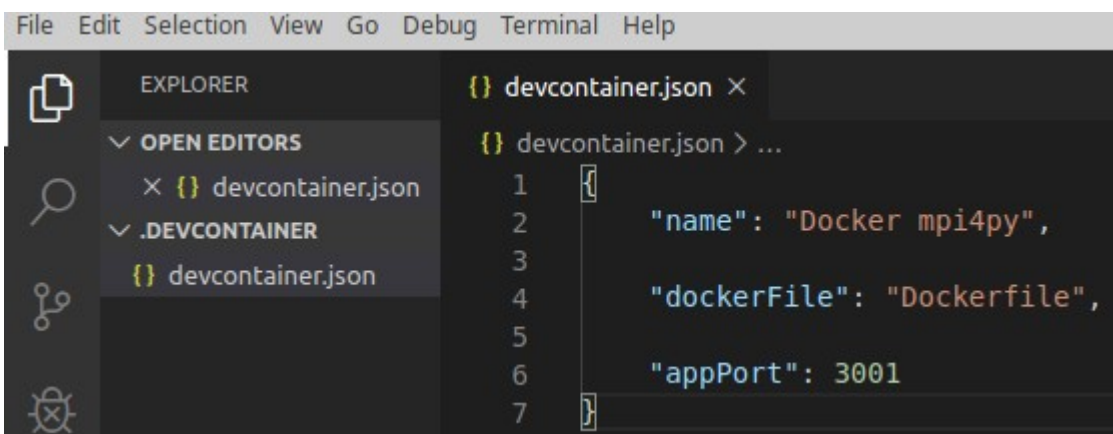
"dockerFile": "Dockerfile",

"appPort": 3001
}
```

name (El nombre del contenedor)

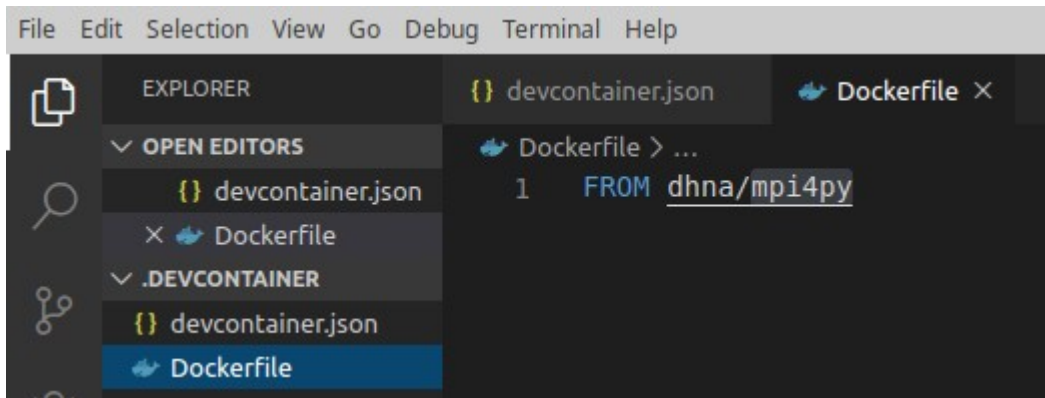
dockerFile (El nombre del archivo que nos dice las imágenes que usaremos)

appPort (el puerto que se utilizara para el contenedor).

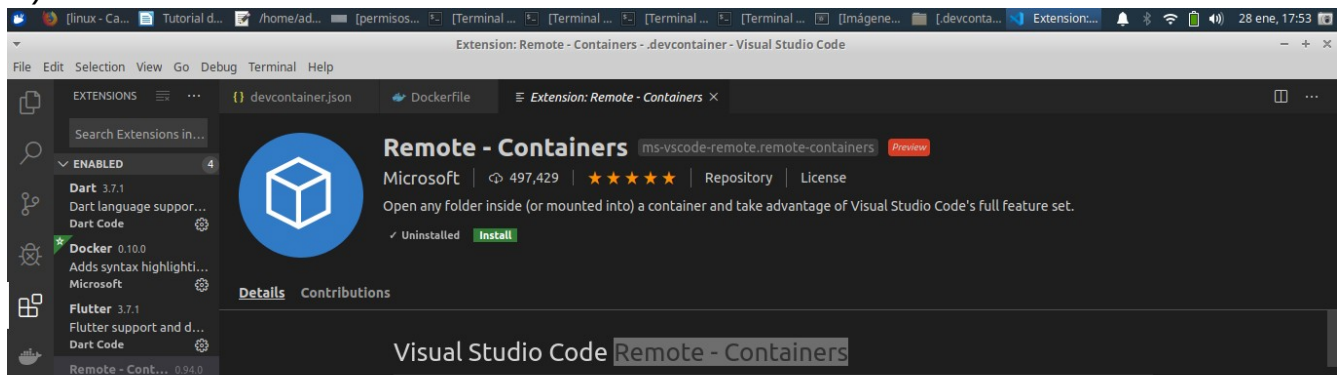


8) Creamos otro archivo que se llamara "Dockerfile" en él ingresaremos.

```
FROM dhna/mpi4py
```



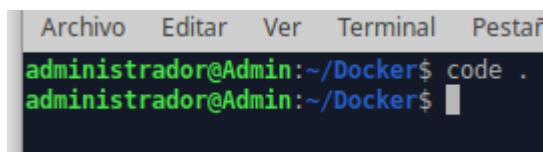
9) Instalaremos una extensión en el VS llamado "Remote - Containers"



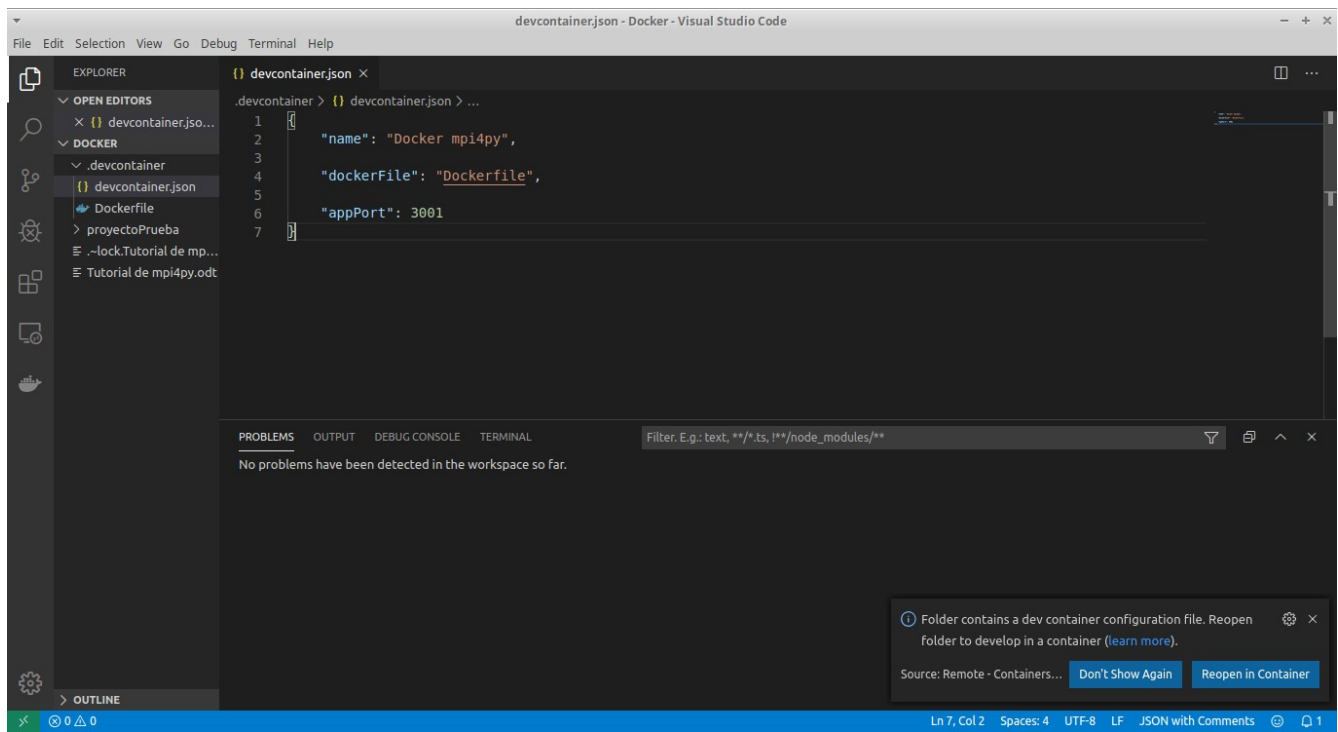
Al finalizar cerramos el Visual Studio Code.

10) ingresamos nuevamente a la carpeta "Docker" y ejecutamos el comando:

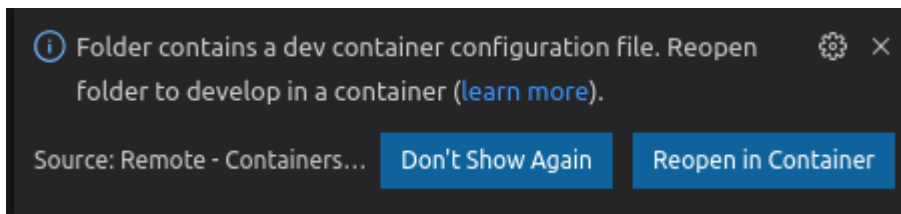
```
CODE .
```



11) El comando anterior abrirá el VS y en la parte inferior derecha saldrá la siguiente ventana.



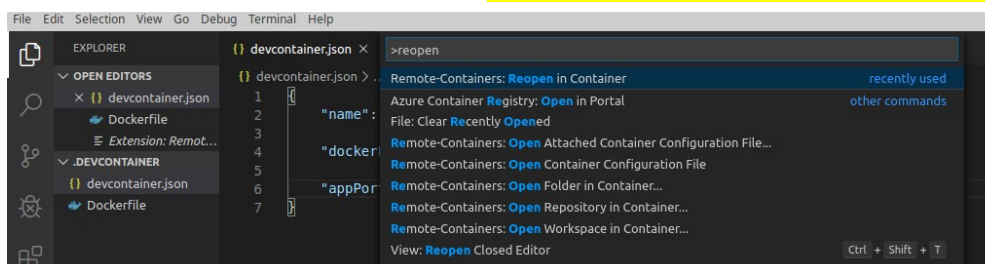
Esta ventana Seleccionaremos “Reopen in Container”



Reiniciará el VS y estaremos en el contenedor de **mpi4py** donde se encuentra instalado todo lo necesario para desarrollar en python utilizando esta librería.

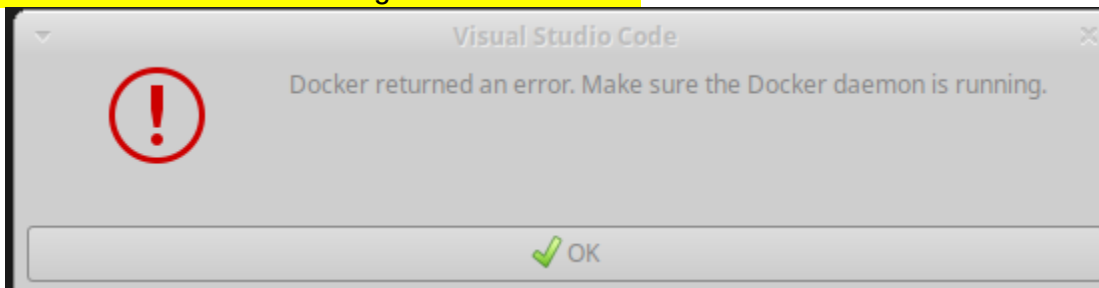
Otra forma de abrir el container desde VS en vez de cerrar y esperar a que aparezca la ventana emergente.

Podemos presionar los botones Ctrl+Shift+P. Escribimos “**reopen**” y seleccionamos la opción “**Remote-Containers: Reopen in Container**”.



Con esto finalizamos la instalación cada vez que queremos utilizar la librería debemos entrar a la carpeta .devcontainer desde el VS y ejecutar el comando. "Remote-Containers: Reopen in Container".

Nota: Si le muestra el siguiente error.



Debe ejecutar el Comando:

sudo dockerd

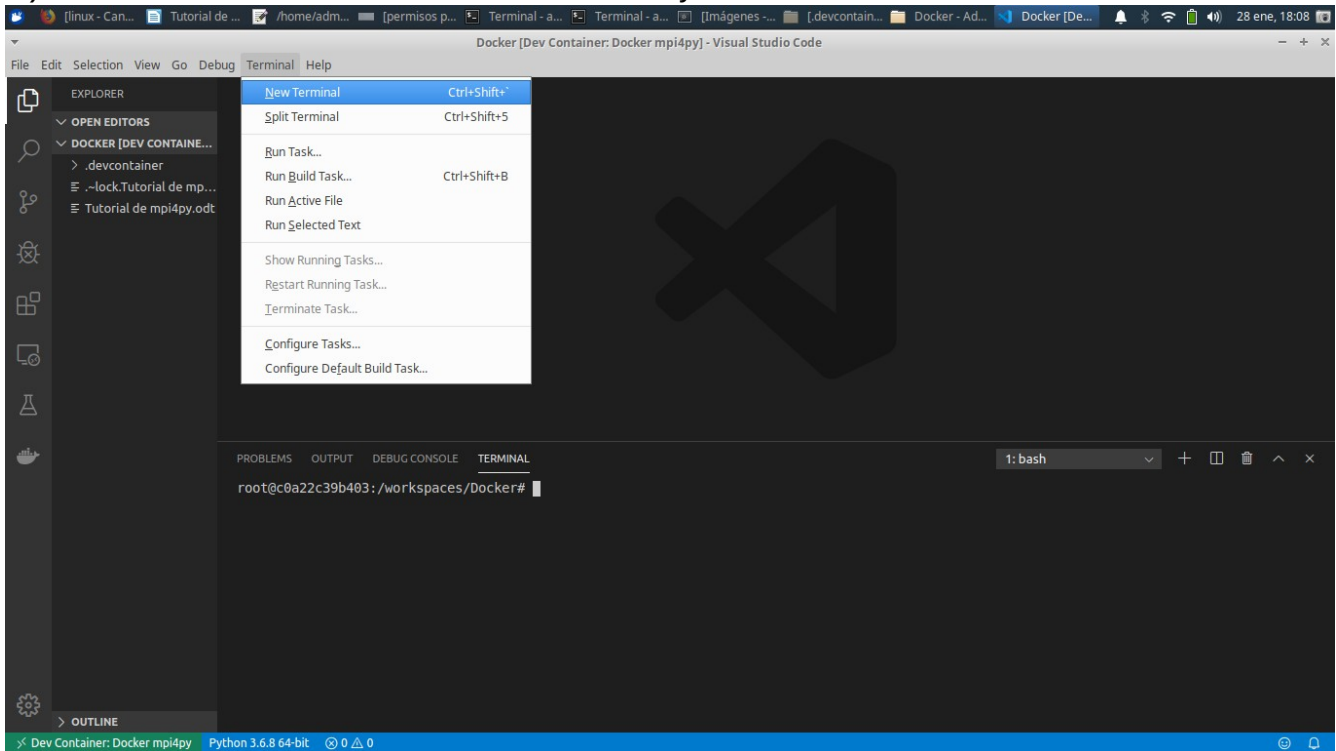
(Dejar la terminal abierta.) y en otra terminal volver abrir el VS CODE.

```
Terminal - administrador@Admin: ~/Docker
Archivo  Editar  Ver  Terminal  Pestañas  Ayuda
administrador@Admin:~/Docker$ sudo dockerd
[sudo] contraseña para administrador:
INFO[2020-02-04T13:17:22.896342203-03:00] Starting up
INFO[2020-02-04T13:17:22.899580195-03:00] detected 127.0.0.53 nameserver, assumi
ng systemd-resolved, so using resolv.conf: /run/systemd/resolve/resolv.conf
INFO[2020-02-04T13:17:23.014791250-03:00] parsed scheme: "unix"
      module=grpc
INFO[2020-02-04T13:17:23.014838007-03:00] scheme "unix" not registered, fallback
to default scheme  module=grpc
INFO[2020-02-04T13:17:23.014881560-03:00] ccResolverWrapper: sending update to c
c: {[{unix:///run/containerd/containerd.sock 0 <nil>}] }  module=grpc
INFO[2020-02-04T13:17:23.014911135-03:00] ClientConn switching balancer to "pick
_first"  module=grpc
INFO[2020-02-04T13:17:23.015754386-03:00] pickfirstBalancer: HandleSubConnStateC
hange: 0xc000769800, CONNECTING  module=grpc
INFO[2020-02-04T13:17:23.015840507-03:00] blockingPicker: the picked transport i
s not ready, loop back to repick  module=grpc
INFO[2020-02-04T13:17:23.019690507-03:00] pickfirstBalancer: HandleSubConnStateC
hange: 0xc000769800, READY  module=grpc
INFO[2020-02-04T13:17:23.031051307-03:00] parsed scheme: "unix"
      module=grpc
INFO[2020-02-04T13:17:23.031153830-03:00] scheme "unix" not registered, fallback
to default scheme  module=grpc
```

Desarrollo utilizando la librería (MPI4PY)

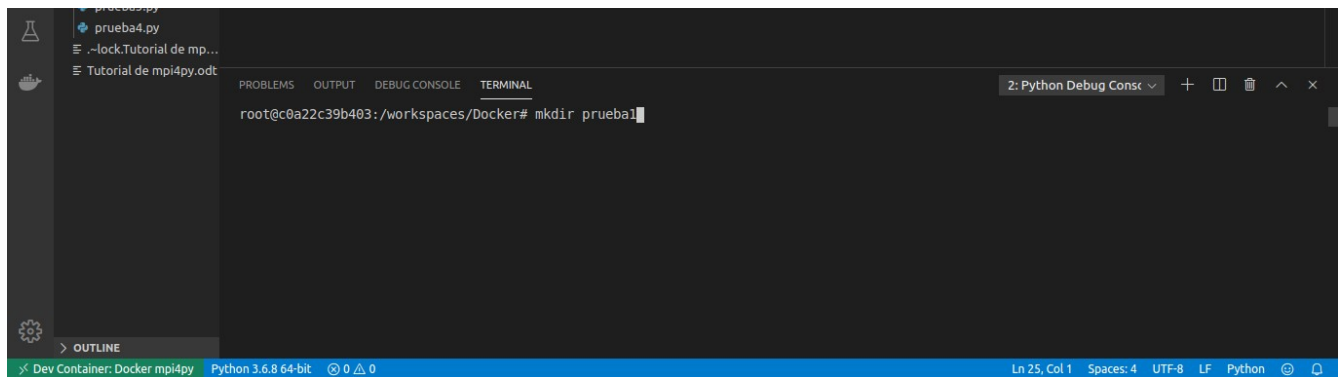
Ahora veremos como utilizar mpi en python.

1) Abrimos el vs con el contenedor y abrimos una nueva terminal.

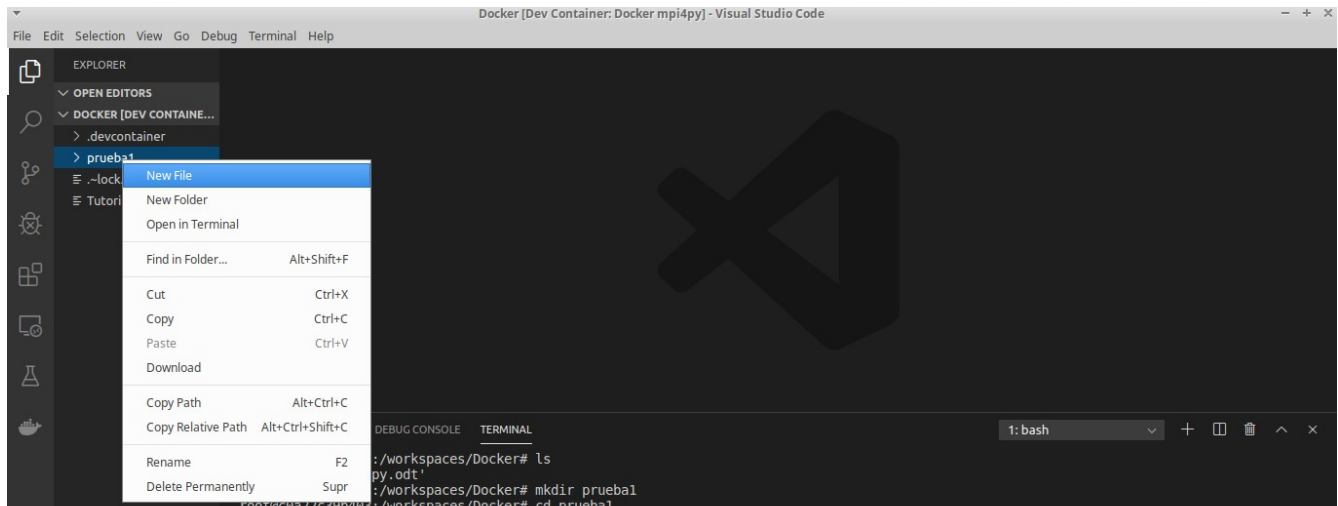


2) En la parte inferior del vs se encuentra la terminal iniciada en el container. En la direccion `/workspaces/` creamos una nueva carpeta para el desarrollo llamada "prueba1" con el comando:

`mkdir prueba1`



3) Ingresamos a la carpeta prueba1 y desde el vs en la parte inferior creamos un archivo "prueba1.py".



4) Dentro del archivo prueba1.py escribimos el código que deseamos ejecutar por ejemplo.

```
from mpi4py import MPI
#Creamos el comunicador Global.
comm = MPI.COMM_WORLD
#Obtenemos el id del proceso actual.
rank = comm.Get_rank()
#Obtenemos el ID del ultimo proceso
cantProc= comm.Get_size()-1
data2=-1

data = {'rank':rank}
if rank < cantProc:
comm.send(data, dest=rank+1, tag=11)
else:
#si es el utlimo proceso envia el mensaje al primero
comm.send(data, dest=0, tag=11)
```

```

if rank > 0:
data2 = comm.recv(source=rank-1, tag=11)
else:
#si es el primer proceso espera el mensaje del último
data2 = comm.recv(source=cantProc, tag=11)
print ("Soy el proceso "+ str(rank)+ " recibí: " +
str(data2))

```

5) En la terminal de la parte inferior ejecutamos el comando:

`mpiexec -n 10 python prueba1.py`

`-n= numero de procesos`

De esta manera realizamos la ejecución de los códigos fuentes con el código de ejemplo nos devuelve.

```

root@ec0a22c39b403: /workspaces/Docker/prueba1# mpiexec -n 10 python prueba1.py
Soy el proceso 6 recibí: {'rank': 5}
Soy el proceso 4 recibí: {'rank': 3}
Soy el proceso 7 recibí: {'rank': 6}
Soy el proceso 3 recibí: {'rank': 2}
Soy el proceso 5 recibí: {'rank': 4}
Soy el proceso 9 recibí: {'rank': 8}
Soy el proceso 0 recibí: {'rank': 9}
Soy el proceso 8 recibí: {'rank': 7}
Soy el proceso 1 recibí: {'rank': 0}
Soy el proceso 2 recibí: {'rank': 1}
root@ec0a22c39b403: /workspaces/Docker/prueba1#

```

6) No es necesario definir el inicio y fin de la ejecución en paralelo Solamente utilizamos la constante MPI.<metodo/atributo>

Algunos métodos:

Debemos crear una variable con el comunicador por ejemplo

`comm= MPI.COMM_WORLD.`

`Comm.send(self, buf, int dest, int tag=0)`

self= datos del que envia.

buf= el mensaje a enviar.

dest= numero del proceso destino.

tag= el numero de tag.

Comm.recv(self, buf=None, int source=ANY_SOURCE, int tag=ANY_TAG, Status status=None)

source= id del proceso que se espera.
tag=numero el tag que espera.
Status= estado del mensaje.

Comm.Isend(self, buf, int dest, int tag=0)
mismos datos que send(). Es sincronico.

Comm.recv(self, buf=None, int source=ANY_SOURCE, int tag=ANY_TAG, Status status=None) es sincronico

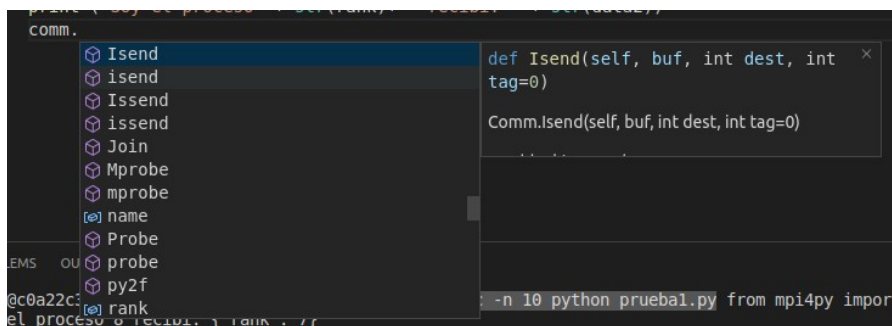
ejemplo de uso de los dos anteriores.

```
data={mensaje: "Hola"}  
req = comm.isend(data, dest=1, tag=11)  
req.wait()
```

```
req = comm.irecv(source=0, tag=11)  
data = req.wait()
```

req.wait()= Espera a recibir un mensaje.

Una manera sencilla de ver todos los métodos que podemos usar es con el VS.



O revisando la documentación oficial:

<https://mpi4py.readthedocs.io/en/stable/overview.html#communicating-python-objects-and-array-data>

Fuentes:

<https://mpi4py.readthedocs.io/en/stable/>

<https://www.youtube.com/watch?v=SjlTY4jfQy0>

<https://www.youtube.com/watch?v=CjLQz09KBQs>