**Cambios frecuentes**

**Agregar un datanode**

Para agregar un datanode se tienen que realizar 5 pasos:

1. Modificar archivo **cluster\_start.sh** (intercambiar llaves con el nuevo datanode)
2. Modificar archivo **workers** (agregar el nuevo datanode)
3. Reconstruir la imagen de hadoop
4. Modificar archivo **docker-compose.yml** (agregar el nuevo datanode)
5. Crear el servicio completo.

A continuación se detallará cada uno de los pasos.

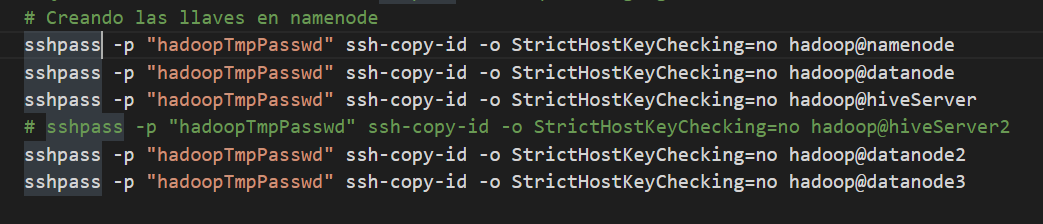
1. **Modificar archivo cluster\_start.sh (intercambiar llaves con el nuevo datanode)**

Hadoop utiliza SSH para comunicarse con datanodes, es por esto que en este archivo se realiza un intercambio de llaves de todos los contenedores involucrados, por lo que tendremos que agregar a cada uno de los intercambios de llaves una nueva línea.

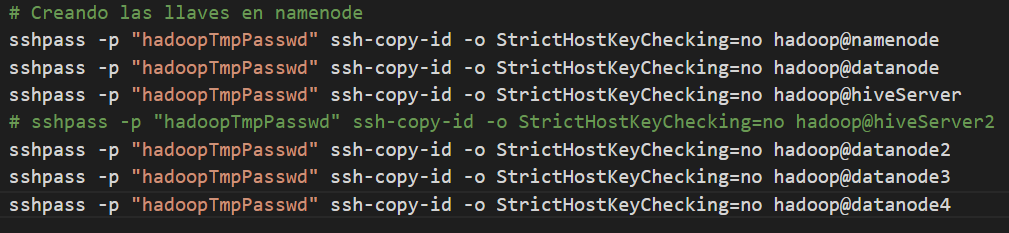
Ubica la sección de intercambio de llaves y en ella ubica el intercambio por cada uno de los contenedores (namenode, datanodes, hiveServer, etc)



Por cada una de los espacios destinados al intercambio por contenedor, copia la última línea de código y pégala, asegurándote de cambiar **hadoop@datanode3** por **hadoop@datanode4** (o el número de datanode correspondiente). Entonces si el espacio de intercambio de llaves del namenode antes estaba así:

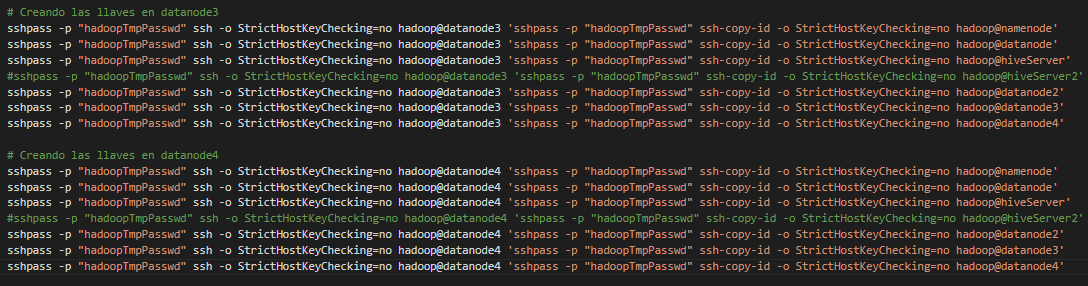


Después del cambio quedaría:

****

\*\* Nota: Recuerda realizar esto por cada uno de los contenedores que están listados en esta sección de intercambio.

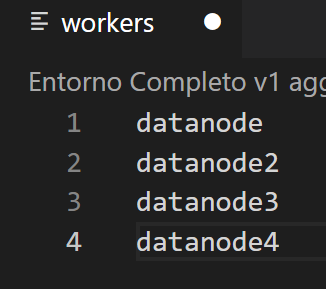
Después copia y pega todas las líneas del datanode3 y cambia la primera referencia de **hadoop@datanode3** por **hadoop@datanode4** en cada una de ellas:



\*Nótese que es una copia la sección de debajo de la de arriba, únicamente cambia en cada línea la referencia del datanode4 en lugar del 3.

1. **Modificar archivo workers (agregar el nuevo datanode)**

Únicamente se agrega una nueva línea con el nuevo datanode.

****

**3. Reconstruir la imagen de hadoop**

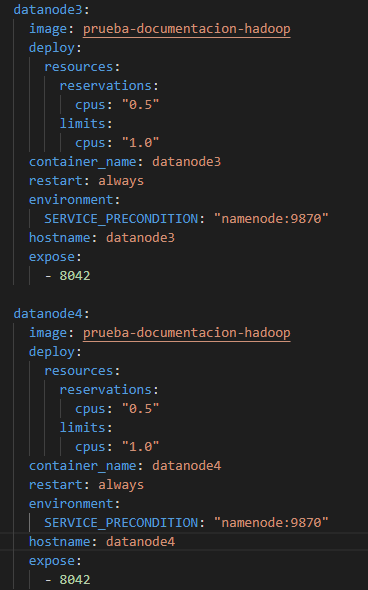
Una vez que los archivos estén en los directorios correspondientes, reconstruir la imagen mediante

docker build –t nombreImagen .

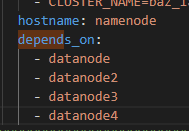
Con esto se reconstruirá la imagen a partir de la copia del archivo WORKERS

**4. Modificar archivo docker-compose.yml (agregar el nuevo datanode)**

Simplemente copiar y pegar una sección de datanode y reemplazar las referencias a datanode4. Ejemplo:

****

Localizar la sección de **namenode** y agregar en el apartdado de **deponds on** el datanode4



**5. Crear el servicio completo.**

Posicionarse en la carpeta donde está el **docker-compose.yml** y ejecutar:

docker stack deploy –c docker-compose.yml NOMBRE\_SERVICIO

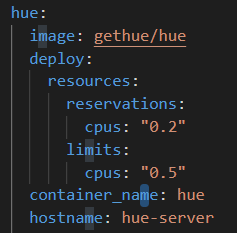
En donde reemplazarás NOMBRE\_SERVICIO por el nombre del servicio que quieras.

**Cambio de límite y reserva de CPU.**

En docker existe el concepto de límite y de reserva. El límite es lo máximo que un contenedor puede utilizar, mientras que la reserva como su nombre lo indica, es una reserva de recursos para un contenedor, una garantía de que ese recurso estará disponible.

Para hacer cambio en los límites o reservas se tiene que modificar el **docker-compose.yml**

Supongamos que queremos asignar el 20% de un CPU como reserva (como garantía) al contenedor de hue y que máximo pueda usar un 50% de un procesador. Esto lo haríamos de la siguiente forma:

****

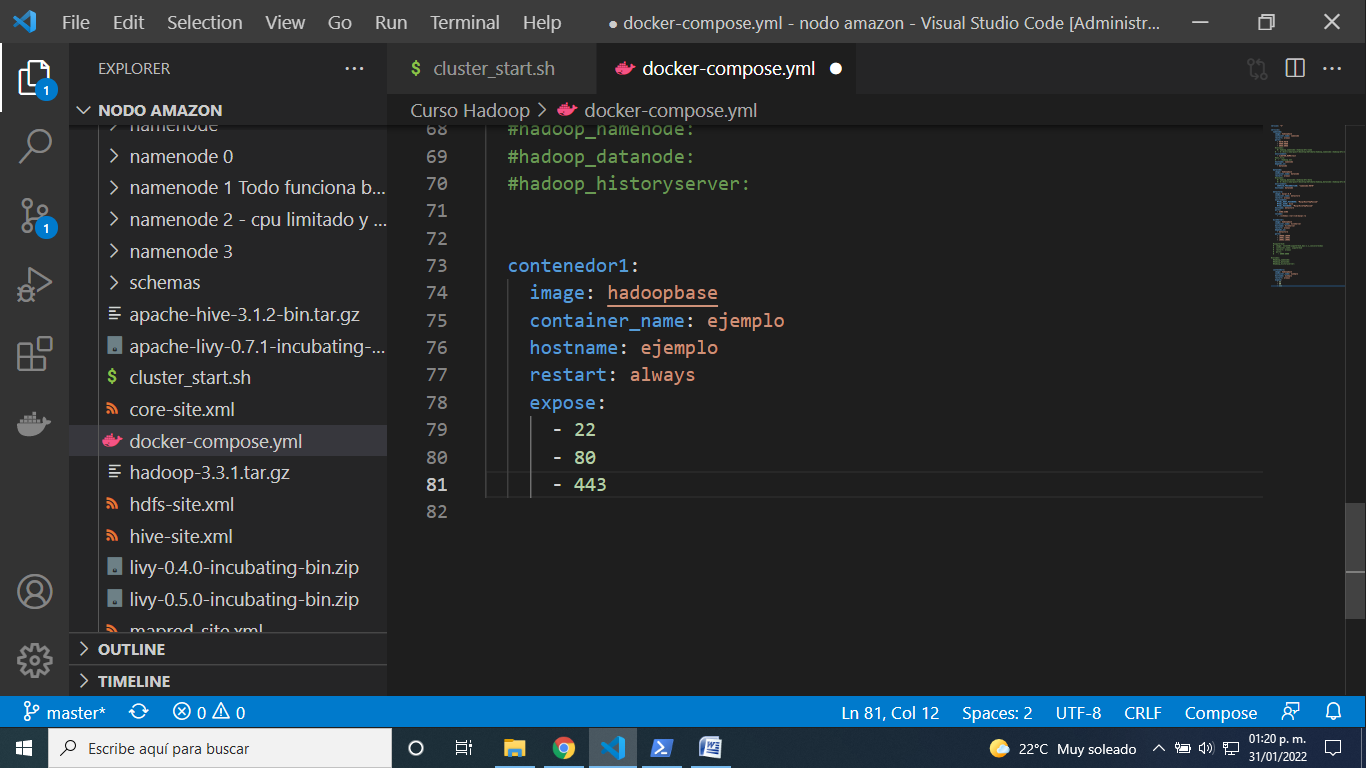
Las cantidades se manejan en decimal, es decir, si quisiéramos un procesador completo sería 1.0, 9 procesadores y medio serían 9.5, etc.

**Cambio de puertos**

En los contenedores de docker existen 2 opciones para abrir los puertos. EXPOSE y PORT.

La primera, es decir **expose**, es para dejar expuestos los puertos a los contenedores de la misma red, así que ahí se podrá hacer una lista de los puertos que se quiere estén visibles.

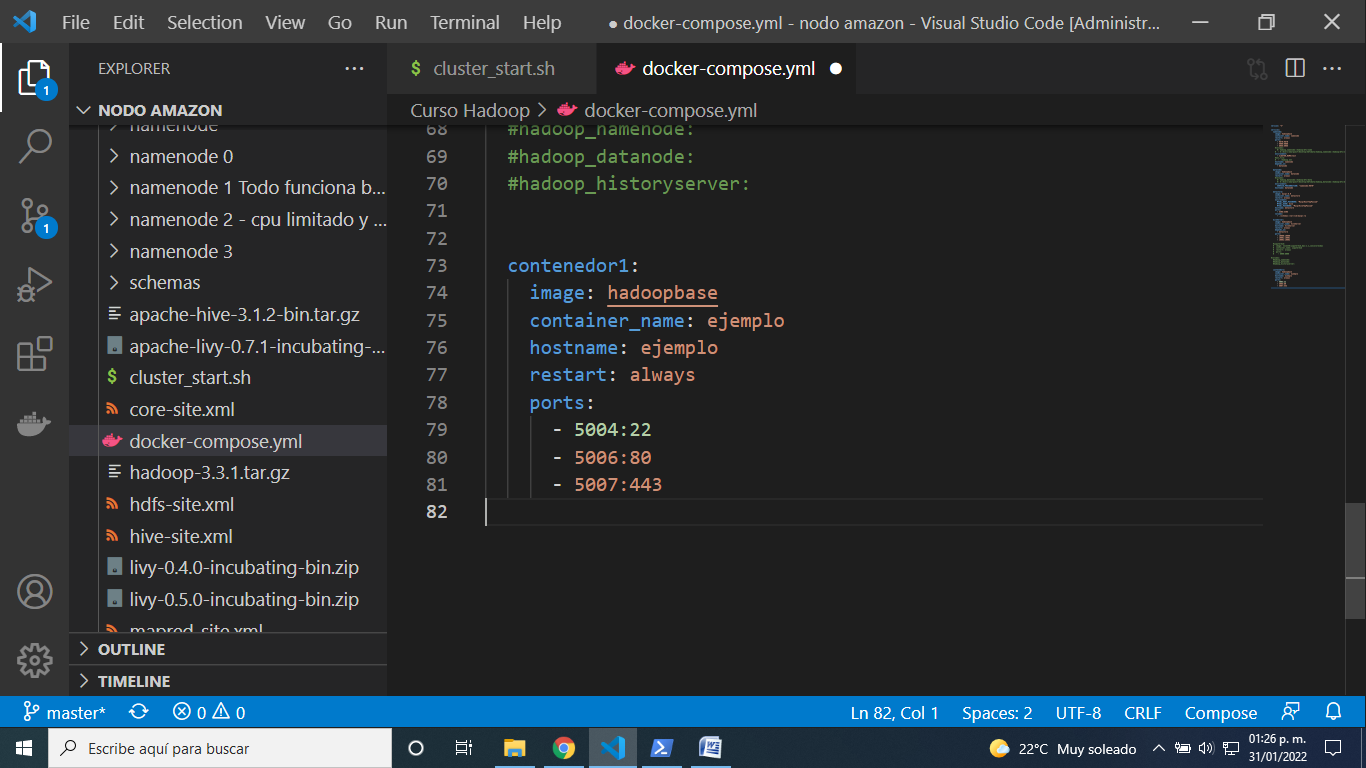
Ejemplo. Se quiere que el contenedor1 tenga expuestos los puertos 22, 80 y 443 para los contenedores de la misma red. En el **docker-compose.yml** tendría que ir lo siguiente:



Si en cambio se quiere hacer un forward del host a un contenedor (es decir, que un puerto del host corresponda a un puerto del contenedor) entonces tendremos que usar la segunda opción.

Para utilizar la opción **port** de igual manera hay que especificarlo en el **docker-compose.yml** antes de arrancar los servicios.

Ejemplo. Con el mismo contenedor1 del ejemplo pasado. Solo tenemos disponibles los puertos 5004, 5006, 5007 del host. Queremos redireccionar dichos puertos al 22, 80 y 443 respectivamente para poder acceder a ellos desde fuera del ambiente de los contenedores. Entonces:



\*\*Nota importante: ports no modifica la comunicación interna entre los contenedores. Es decir, si un contenedor de la misma red quiere comunicarse al contenedor1 a través de SSH tendrá que ir al puerto 22 y no al 5004 (como se tendría que hacer fuera de la red de contenedores)

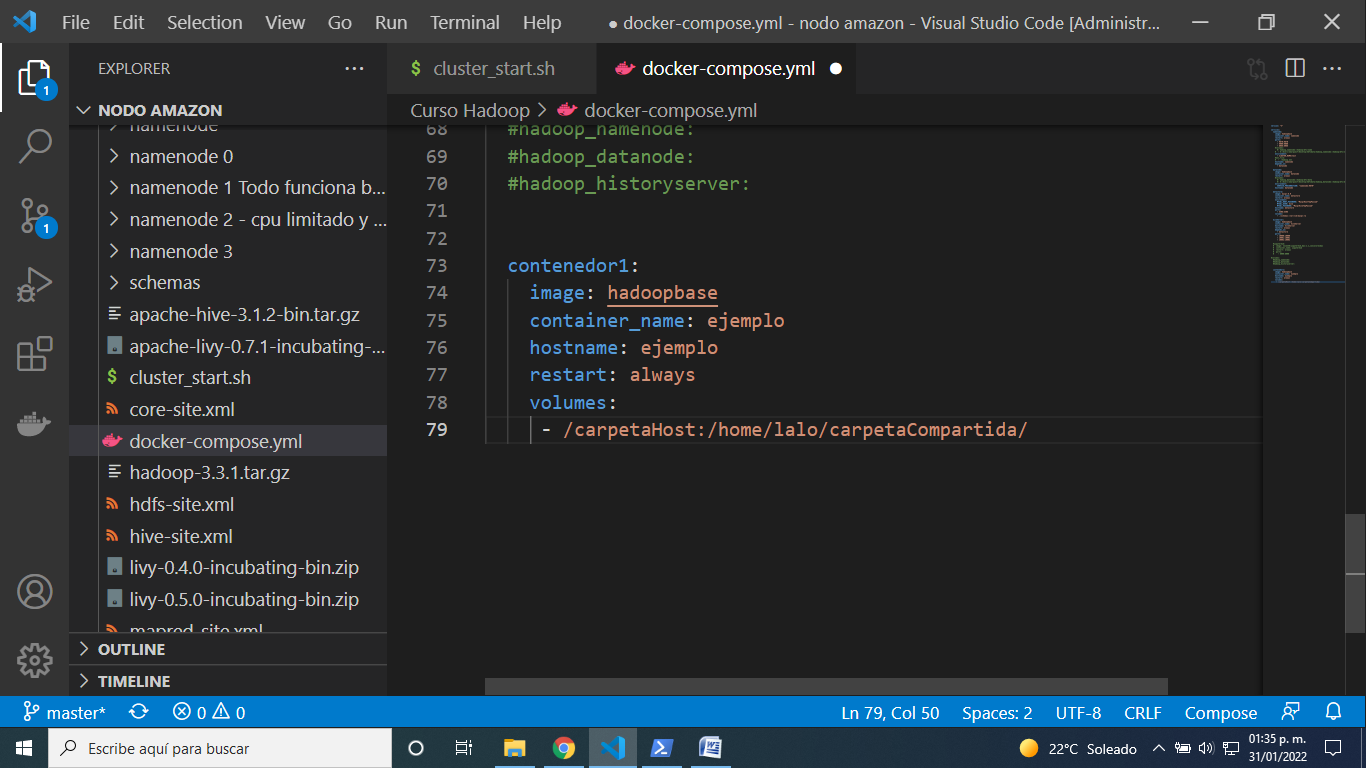
Los puertos que actualmente se están utilizando en el nodo son:

* 8001: **Laboratorio CYC**. Apuntando al 8000 del contenedor contenedorLalo.
* 8003: **ElasticSearch.**
* 8004: **Kibana.**
* 8007: **Hue.** Apuntando al 8888 del contenedor HueServer
* 8008: **Laboratorio DeepDive.** Apuntando al 8000 del contenedor contenedorLaloDeepDive.

**Cambio de volúmenes compartidos con el host**

Compartir un volumen sirve para que la información y datos que están en el host se vean reflejados en el contenedor y vise versa. Los cambios que se hagan al volumen dentro o fuera del contenedor lo afectarán.

Ejemplo. Se desea compartir la carpeta **/carpetaHost** del host con la carpeta **/home/lalo/carpetaCompartida/** dentro del contenedor1. Entonces:



\*\* Nota: Se recomienda el uso de rutas absolutas. Solo usar relativas en casos especiales.

Las rutas actualmente utilizadas para compartir volúmenes son:

* /data/data#/: Guarda la información almacenada en el HDFS. Corresponde a la carpeta /data#/ del datanode# (sustituir # por el número de datanode)
* /data/namenode/: Almacena los metadatos del hdfs
* /INTELIGENCIACYC/CLOUDERA/TABLES/: Todos los contenedores de hadoop comparten esta carpeta con el fin de que no haya duplicidad de información. Aquí se almacenan todas las tablas.
* /mysql\_general: Almacena las bases de datos del MySQL general, aquí se guardan los metadatos de hue.
* ~/hue.ini: Archivo de configuración de hue.

**Quitar el internet a un contenedor (sin quitar la comunicación con los demás**

**contenedores de la misma red)**

Docker utiliza internamente el firewall iptables para manejar el tráfico de los contenedores. Específicamente utiliza una cadena llamada FORWARD. Para dejar a un contenedor sin internet, este tiene que estar en la red “bridge” para comprobar si está ejecutamos el comando:

docker inspect CONTAINER\_NAME -f "{{json .NetworkSettings.Networks }}"

en donde se reemplazará **CONTAINER\_NAME** por el nombre del contenedor.

En caso de que no se encuentre “bridge” como una de las redes, deberemos agregar dicho contenedor a la red con el comando:

docker network connect bridge CONTAINER\_NAME

Después verificamos que IP tiene dicho contenedor, esto con el comando:

docker inspect -f "{{ .NetworkSettings.IPAddress }}" CONTAINER\_NAME

en donde se reemplazará **CONTAINER\_NAME** por el nombre del contenedor.

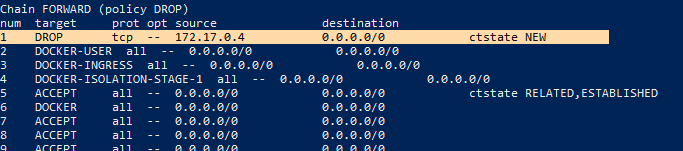
Supongamos que el contenedor que queremos restringir de internet tiene la IP **172.17.0.4**. Entonces el comando que tendríamos que aplicar sería:

iptables -I FORWARD -s 172.17.0.4 -i docker0 -p tcp -d 0/0 -m conntrack --ctstate NEW -j DROP

Para volver a darle internet a dicho contenedor, entonces habría que revisar las reglas de firewall de iptables y localizar la regla que acabamos de crear.

iptables -nL –-line-numbers

Hay que revisar el número de la regla. En este caso es la regla 1

****

Por lo que para eliminarla, el comando sería el siguiente:

iptables –D FORWARD 1

El número indica el número de regla que deseamos borrar.

**Aumentar o disminuir procesador o RAM a un usuario en Laboratorio de Jupyter**

En el laboratorio de jupyter existe un archivo de configuración llamado **/home/app/users.conf** el cual tiene las configuraciones de RAM y CPU de cada uno de los usuarios.

La estructura del archivo es en cada línea la configuración de un usuario y estas están separadas por “|”. Así que la estructura sería similar a:

Usuario|Memoria|CPU

Por ejemplo, tenemos un usuario **lalo** con **32GB** de ram y 7 procesadores y un usuario **chill** con **50GB** de ram y 2 procesadores. El archivo se vería así

lalo|32G|700  
chill|50G|200

Queremos que ambos usuarios tengan 65GB de ram y 9 procesadores, entonces se modifica el archivo users.conf para que se vea de la siguiente forma:

lalo|65G|700  
chill|65G|200

**Limitar el acceso a un contenedor solo desde una IP.**

iptables -I FORWARD '!' -s IP\_CON\_PERMISO -d IP\_CONTENEDOR -p tcp --dport PUERTO\_CONTENEDOR -j DROP

**Agregar OTRA IP a la lista de IPs permitidas (complemento del anterior)**

El orden en que se ponen las reglas importa, es por eso que primero se pone la regla de negación y después esta:

iptables -I FORWARD -s IP\_CON\_PERMISO -d IP\_CONTENEDOR -p tcp --dport PUERTO\_CONTENEDOR -j DROP

Ejemplo para no permitir que se pueda acceder al puerto 8000 del contenedor con IP 172.17.0.2 a menos que sea desde la IP 10.51.225.75 o la 18.219.34.90

iptables -I FORWARD '!' -s 10.51.225.75 -d 172.17.0.2 -p tcp --dport 8000 -j DROP

iptables -I FORWARD -s 18.219.34.90 -d 172.17.0.2 -p tcp --dport 8000 -j ACCEPT

En los comandos anteriores, los parámetros significan:

* -I: Insertar una regla nueva en la cadena X (en estos casos FORWARD). Esta regla será puesta con la mayor prioridad (para poner una regla con la menor prioridad utilizar –A)
* -s: Source. Es la IP fuente
* -d: Destination: La IP destino
* -p: Protocolo tcp o udb
* --dport: Puerto destino
* -j : Jump. La acción que se tomará, pueden ser DROP, REJECT o ACCEPT