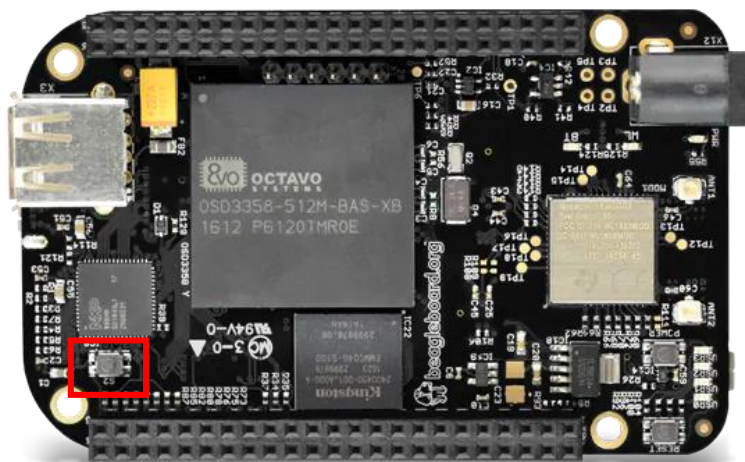


GUÍA DE INICIO PARA DESARROLLO DE EMBEBIDOS CON BEAGLEBONE BLACK

Principalmente debemos bootear nuestra tarjeta con una imagen iso de sistema operativo Linux reciente, podemos conocer las versiones actualizadas para desarrollo de embebido en BeagleBone Black en el siguiente enlace: <https://beagleboard.org/latest-images>. Es necesario verificar cuál de las imágenes del sistema operativo requerimos para el desarrollo a realizar.

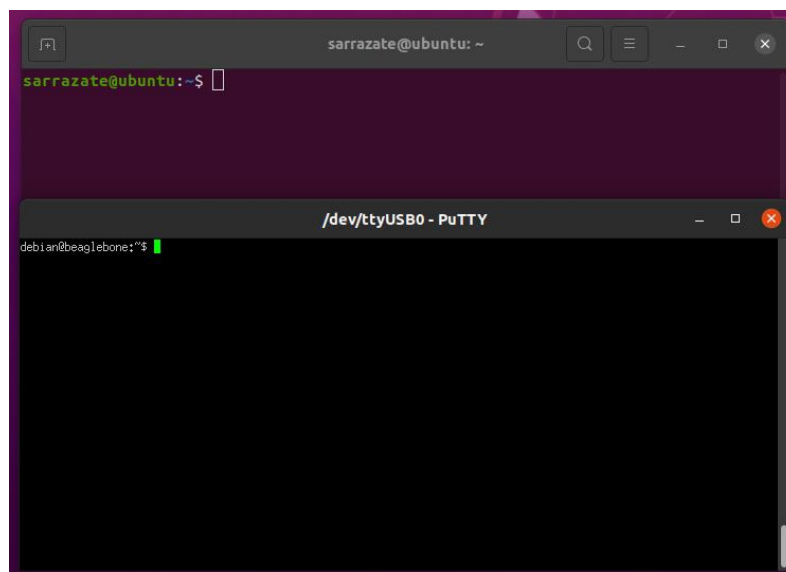
Flasheamos una memoria SD con la versión de Linux que hayamos decidido utilizar para el desarrollo.

Para arrancar la tarjeta desde la memoria SD, es necesario utilizar un cable de corriente directa 12v para energizar la tarjeta, una vez hayamos conectado la tarjeta, el sistema arrancará automáticamente con desde eMMC con el sistema operativo precargado. Para poder arrancar desde la SD previamente flasheada, al momento de conectar a la corriente la tarjeta hay que mantener presionado el botón encerrado en el recuadro rojo en la siguiente imagen hasta que los 4 leds indicadores de la placa dejen de parpadear, esto arrancará el sistema desde la SD.



Para comenzar a trabajar con nuestra placa necesitamos hacer como mínimo una conexión SSH, USB o serial. Para ello existen diferentes programas que nos ayudan, para estos ejercicios utilizaremos PuTTY, la cual debemos descargar e instalar desde el siguiente enlace: <https://www.putty.org/>

Es recomendable trabajar con una computadora con sistema operativo Linux ya que es un poco más sencillo de realizar las distintas configuraciones que mas adelante iremos mencionando, por lo que, para las instrucciones siguientes supondremos que se está trabajando sobre Linux, más específicamente sobre Debian 10.



Como podemos observar en la imagen anterior, estamos haciendo uso de una terminal de Linux en la parte superior y la terminal inferior es la terminal de PuTTY en la cual estamos haciendo una conexión serial a través del puerto /ttyUSB0. Con esto, podemos observar que nuestra tarjeta ya está funcionando y de esta manera podremos comenzar a trabajar sobre ella.

NOTA: debido a que este tipo de tarjetas cuentan con un espacio de almacenamiento mínimo, el desarrollo no lo podemos hacer directamente sobre esta por que pronto nos quedamos sin memoria y esto nos impide hasta crear una carpeta vacía (problema por el cual pasamos durante el proceso de prueba), para evitar esto existen diferentes formas como lo son las siguientes.

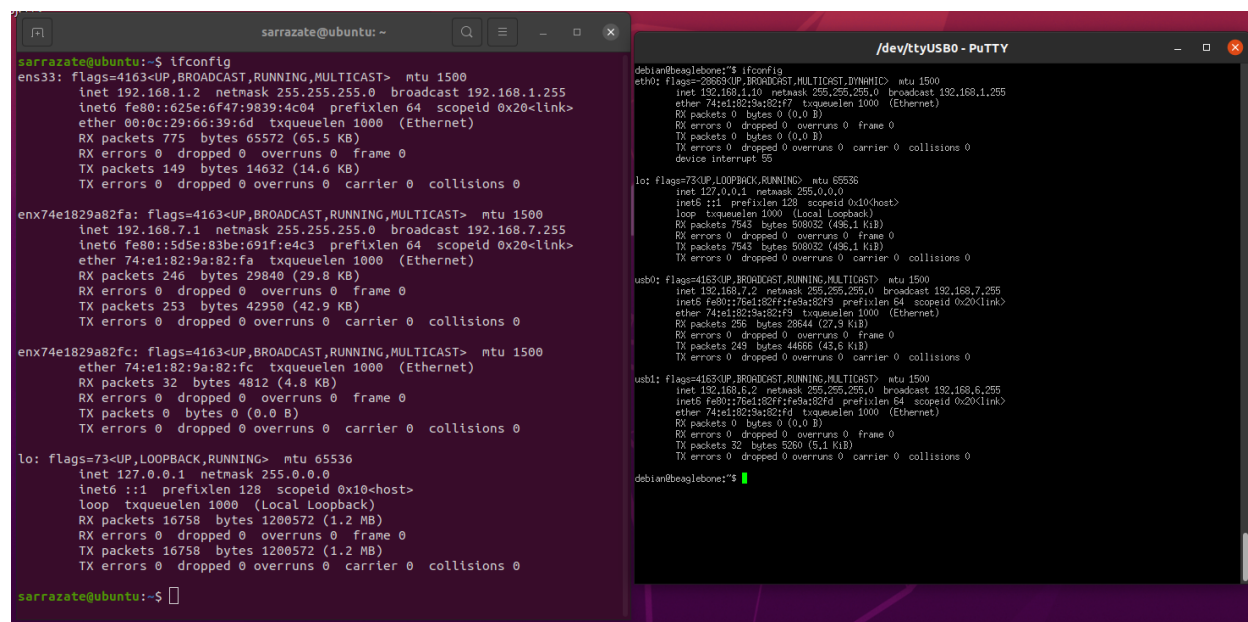
Es posible trabajar con U-boot, con este podemos cargar un sistema de archivos en nuestra PC y arrancar nuestra BeagleBone haciendo uso de este sistema de archivos externo como servidor.

Para esta guía utilizaremos el método de trabajo a partir de carpeta compartida con NFS, usando como servidor nuestra PC (máquina virtual Linux) y configuramos la BeagleBone Black como cliente para hacer uso de la carpeta compartida en la cual almacenaremos los archivos y programas desarrollados para pruebas sin necesidad de que ocupen espacio en la tarjeta físicamente.

Antes de comenzar con las configuraciones, es necesario conocer las direcciones IP con las que estaremos trabajando. Para hacer uso de NFS necesitamos comunicación de red por lo que el direccionamiento IP es indispensable.

Generalmente esto se hace a través de una red local con un cable ethernet, por motivos de complejidad, se decidió trabajar a través de una conexión directa USB entre la BeagleBone Black y la maquina virtual. Para conocer las direcciones de ambos equipos (las cuales utilizaremos en la configuración de NFS) basta con ejecutar el siguiente comando y fijarnos en la interfaz llamada usb:

Ifconfig



```
sarrazate@ubuntu: ~  
$ ifconfig  
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500  
    inet 192.168.1.2 netmask 255.255.255.0 broadcast 192.168.1.255  
    inet6 fe80::625e:6f47:9839:4c04 prefixlen 64 scopeid 0x20<link>  
    ether 00:0c:29:66:39:6d txqueuelen 1000 (Ethernet)  
    RX packets 775 bytes 65572 (65.5 KB)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 149 bytes 14632 (14.6 KB)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
  
enx74e1829a82fa: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500  
    inet 192.168.7.1 netmask 255.255.255.0 broadcast 192.168.7.255  
    inet6 fe80::5d5e:83be:691f:e4c3 prefixlen 64 scopeid 0x20<link>  
    ether 74:e1:82:9a:82:fa txqueuelen 1000 (Ethernet)  
    RX packets 246 bytes 29840 (29.8 KB)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 253 bytes 42950 (42.9 KB)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
  
enx74e1829a82fc: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500  
    ether 74:e1:82:9a:82:fc txqueuelen 1000 (Ethernet)  
    RX packets 32 bytes 4812 (4.8 KB)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 0 bytes 0 (0.0 B)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
  
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536  
    inet 127.0.0.1 netmask 255.0.0.0  
    inet6 ::1 prefixlen 128 scopeid 0x10<host>  
    loop txqueuelen 1000 (Local Loopback)  
    RX packets 16758 bytes 1200572 (1.2 MB)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 16758 bytes 1200572 (1.2 MB)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
  
sarrazate@ubuntu: ~  
$
```

```
/dev/ttyUSB0 - PuTTY  
debian@beaglebone:~$ ifconfig  
eth0: flags=20669<UP,BROADCAST,MULTICAST,DYNAMIC> mtu 1500  
    inet 192.168.1.40 netmask 255.255.255.0 broadcast 192.168.1.255  
    ether 74:e1:82:9a:82:f7 txqueuelen 1000 (Ethernet)  
    RX packets 0 bytes 0 (0.0 B)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 0 bytes 0 (0.0 B)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
    device interrupt 55  
  
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536  
    inet 127.0.0.1 netmask 255.0.0.0  
    inet6 ::1 prefixlen 128 scopeid 0x10<host>  
    loop txqueuelen 1000 (Local Loopback)  
    RX packets 7543 bytes 508032 (496.1 KiB)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 7543 bytes 508032 (496.1 KiB)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
  
usb0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500  
    inet 192.168.7.2 netmask 255.255.255.0 broadcast 192.168.7.255  
    inet6 fe80::76e1:82cf:fe9a:82f9 prefixlen 64 scopeid 0x20<link>  
    ether 74:e1:82:9a:82:f9 txqueuelen 1000 (Ethernet)  
    RX packets 255 bytes 28544 (27.5 KiB)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 249 bytes 44666 (43.6 KiB)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
  
usb1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500  
    inet 192.168.6.2 netmask 255.255.255.0 broadcast 192.168.6.255  
    inet6 fe80::76e1:82cf:fe9a:82fd prefixlen 64 scopeid 0x20<link>  
    ether 74:e1:82:9a:82:fd txqueuelen 1000 (Ethernet)  
    RX packets 0 bytes 0 (0.0 B)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 32 bytes 5260 (5.1 KiB)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
  
debian@beaglebone:~$
```

CONFIGURACIÓN DE NFS PC SERVIDOR (Máquina virtual Debian 10)

Para instalar las herramientas necesarias en el ordenador que actuará como servidor, ejecutaremos la siguiente orden:

```
sudo apt install nfs-kernel-server
```

El siguiente paso es definir los directorios que queremos compartir. Para esto editamos el archivo **/etc/exports**. Así por ejemplo si queremos compartir el directorio **compartir**, tendríamos que configurarlo de la siguiente forma:

```
/home/atareao/compartir 192.168.1.0/24(rw,no_subtree_check,async)
```

Con esto hemos compartido el directorio en cuestión con toda nuestra red (192.168.1.0/24). Indicamos que es de lectura y escritura (rw). No realiza comprobación de subdirectorios (no_subtree_check) lo que en ocasiones mejora el rendimiento. No es necesario que todas las escrituras se completen (async) aunque esto puede conllevar una pérdida de datos en caso de que se produzca un corte de conexión.

En caso de que queramos compartir la carpeta con un solo equipo específico, simplemente debemos cambiar la dirección IP de red que es la que se utiliza en el ejemplo anterior por la dirección exacta en este caso de nuestra tarjeta.

Después de cualquier cambio sobre el archivo **/etc/exports** tenemos que ejecutar la siguiente orden para actualizar la tabla de NFS:

```
sudo exportfs -arv
```

La opción **-a** exporta todos los directorios, la opción **-r** eliminará las entradas antiguas, mientras que la opción **-v** nos mostrará el resultado de la ejecución.

Una vez configurado el servidor, es necesario reiniciar el servicio para que funcione correctamente,

```
sudo systemctl enable nfs-kernel-server
```

```
sudo systemctl start nfs-kernel-server
```

CONFIGURACIÓN DE NFS CLIENTE (BeagleBone Black)

Ahora nos toca instalar y configurar el cliente en nuestro equipo Ubuntu. Esto es tan sencillo, como ejecutar la siguiente orden en un emulador de terminal. Lo primero instalar el cliente.

```
sudo apt install nfs-common
```

Únicamente nos resta montar el directorio remoto en nuestra tarjeta, esto se hace conociendo la dirección IP de nuestro servidor y accediendo a los archivos de configuración de NFS de nuestra tarjeta para colocar las reglas que necesitamos.

Lo primero que debemos hacer es crear el directorio que queremos montar en nuestra tarjeta, en nuestro caso manejamos la carpeta **/mnt/compartidos**, para esto ejecutaremos la siguiente orden en nuestra terminal:

```
sudo mkdir -p /mnt/compartidos
```

A continuación, solo nos queda ejecutar la siguiente orden en un emulador de terminal, para poder acceder fácilmente a ese directorio.

```
sudo mount 192.168.7.1:/home/sarrazate/Desktop/Compartidos /mnt/compartidos
```

El cambio que hemos hecho no es permanente. Es decir, cada vez que reiniciemos el cliente tendremos que volver a montar el directorio remoto. Para añadir el directorio remoto de forma permanente, tenemos que añadir la siguiente línea en el archivo **/etc/fstab** en la máquina cliente.

```
192.168.7.1:/home/sarrazate/Desktop/Compartidos /mnt/compartidos/ nfs rw,async 0 0
```

Una vez guardados los cambios en **/etc/fstab** tienes que desmontar **/mnt/compartidos** con la orden **umount /mnt/cloudstation/**. Posteriormente toca montarlo todo. Para ello ejecuta **mount -a** que intentará montar todo lo que está en **/etc/fstab** y que no esté montado, claro.

COMPILACIÓN CRUZADA

Se escribe un programa en su computadora y compila un archivo ejecutable en su computadora. Finalmente, este archivo ejecutable debe descargarse a su placa de desarrollo y ejecutarse. El entorno en el que finalmente se ejecuta el programa ha cambiado, por ejemplo, su placa de desarrollo está basada en Arm, el programa se edita y compila en X86 y finalmente se ejecuta en otro chip Arm con una arquitectura completamente diferente a la de X86.

Sin embargo, hay un problema: los conjuntos de instrucciones de los chips X86, Arm, MIPS, RISC-V están diseñados por diferentes organizaciones o empresas y no son compatibles entre sí: las CPU de Arm y MIPS no pueden ejecutarse con el conjunto de instrucciones X86. Así que tenemos que compilar un programa que se pueda ejecutar en Arm en una computadora X86.

arm-none-eabi-gcc se aplica generalmente a las plataformas Arm Cortex-M / Cortex-R y utiliza la biblioteca newlib.

arm-linux-gnueabi-gcc y **aarch64-linux-gnu-gcc** son adecuados para chips de la serie Arm Cortex-A. El primero es para chips de 32 bits y el segundo para chips de 64 bits. Utiliza la biblioteca glibc. Se puede utilizar para compilar u-boot, kernel de Linux y programas de aplicación.

VERIFICAR COMPILADOR CRUZADO EN MÁQUINA VIRTUAL (Ubuntu)

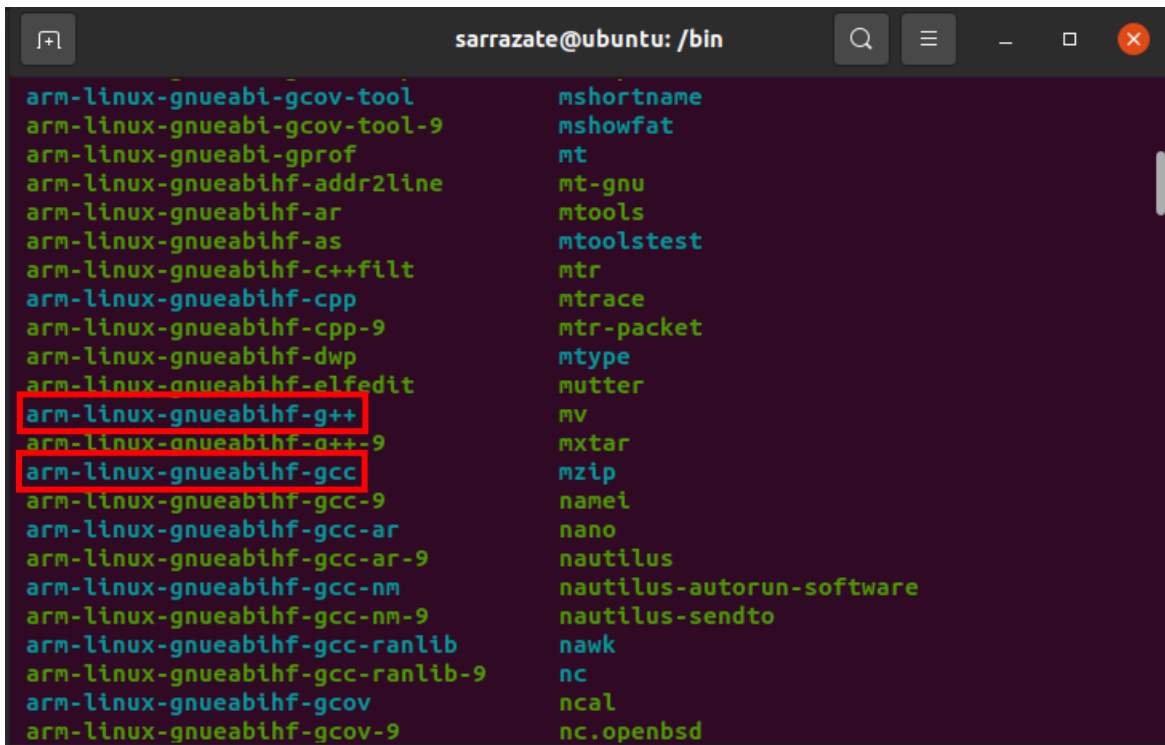
Generalmente los sistemas operativos ya vienen con compiladores cargados, podemos ver esto de una manera muy sencilla que es checando las variables de entorno, ya que cada compilador tendrá su variable correspondiente y haciendo uso de ellas podemos lograr compilar.

Para ver estas variables de entorno podemos acceder a la carpeta /bin dentro de nuestra maquina virtual de la siguiente manera:

```
Cd /bin
```

Este comando nos desplegará una lista de variables, en este punto debemos buscar nuestro compilador que en este caso será **arm-linux-gnueabi-gcc** para C y **arm-linux-**

gnueabi-hf-g++ para C++. La imagen siguiente muestra un ejemplo de la lista de variables en donde encontraremos la de nuestro compilador:



```
sarrazate@ubuntu: /bin
arm-linux-gnueabi-gcov-tool      mshortname
arm-linux-gnueabi-gcov-tool-9    mshowfat
arm-linux-gnueabi-gprof          mt
arm-linux-gnueabi-hf-addr2line   mt-gnu
arm-linux-gnueabi-hf-ar           mtools
arm-linux-gnueabi-hf-as           mtoolstest
arm-linux-gnueabi-hf-c++filt      mtr
arm-linux-gnueabi-hf-cpp          mtrace
arm-linux-gnueabi-hf-cpp-9        mtr-packet
arm-linux-gnueabi-hf-dwp          mtype
arm-linux-gnueabi-hf-elfedit      mutter
arm-linux-gnueabi-hf-g++          mv
arm-linux-gnueabi-hf-g++-9        mxtar
arm-linux-gnueabi-hf-gcc          mzip
arm-linux-gnueabi-hf-gcc-9        namei
arm-linux-gnueabi-hf-gcc-ar        nano
arm-linux-gnueabi-hf-gcc-ar-9      nautilus
arm-linux-gnueabi-hf-gcc-nm        nautilus-autorun-software
arm-linux-gnueabi-hf-gcc-nm-9      nautilus-sendto
arm-linux-gnueabi-hf-gcc-ranlib     nawk
arm-linux-gnueabi-hf-gcc-ranlib-9   nc
arm-linux-gnueabi-hf-gcov           ncal
arm-linux-gnueabi-hf-gcov-9         nc.openbsd
```

COMPILAR EN CONSOLA

Para realizar una compilación cruzada desde nuestra terminal, nos posicionamos en donde sea que tengamos nuestro archivo .c o .cpp para compilar, en nuestro caso se encuentra en la carpeta compartida NFS en la ruta /home/sarrazate/Desktop/Compartidos.

A continuación, ya podemos compilar nuestro programa de la siguiente manera:

Para C:

```
arm-linux-gnueabi-hf-gcc NombreArchivo.c -o NombreDeSalida
```

Para C++:

```
arm-linux-gnueabi-hf-g++ NombreArchivo.c -o NombreDeSalida
```

EJECUTAR COMPILADO DESDE BEAGLEBONE BLACK

Para ejecutar nuestro programa desde nuestra BeagleBone Black solo debemos posicionarnos en la carpeta compartida desde nuestro servidor NFS, en este caso **/mnt/compartidos**.

Si ejecutamos el comando **ls** dentro de esta carpeta podremos ver que ya se encuentra nuestro archivo binario que debemos ejecutar. Para esto basta con hacer **./nombrearchivo**.



```
debian@beaglebone: /mnt/compartidos
debian@beaglebone:/mnt/compartidos$ ls
holamundoarm  holamundo.cpp  hola.txt  hola86  prueba1.txt
debian@beaglebone:/mnt/compartidos$ ./holamundoarm
Hola Mundo!debian@beaglebone:/mnt/compartidos$
```

The image shows a terminal window titled 'debian@beaglebone: /mnt/compartidos'. The first command is 'ls', which lists the files in the directory: 'holamundoarm', 'holamundo.cpp', 'hola.txt', 'hola86', and 'prueba1.txt'. The second command is './holamundoarm', which outputs 'Hola Mundo!'. Red boxes highlight the 'ls' command, the file list, and the './holamundoarm' command. A red arrow points to the output 'Hola Mundo!'.

REDIMENSIONAR PARTICIÓN DE TARJETA MICROSD DE BEAGLEBONE

Es común que cuando flasheamos nuestra memoria microSD con el sistema operativo que utilizaremos en la BeagleBone Black seleccionemos todas las opciones por defecto que el programa de flasheo que utilicemos, el principal problema que esto nos podrá presentar es que la dimensión de la partición que nos hará en la microSD por defecto será muy pequeña lo que nos deja un espacio de memoria muy restringido para trabajar con nuestra tarjeta.

Para solucionar este problema existe un software en Linux llamado GParted, el cual podemos instalar desde terminal de la siguiente manera:

Ubuntu:

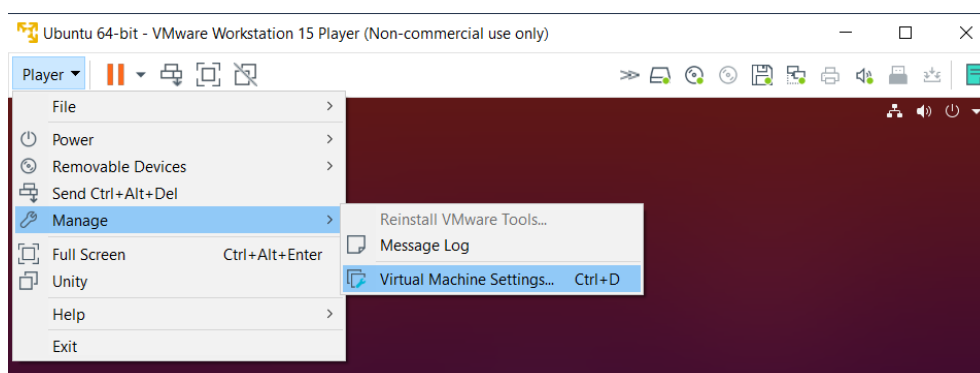
- `sudo apt install gparted`

Debian:

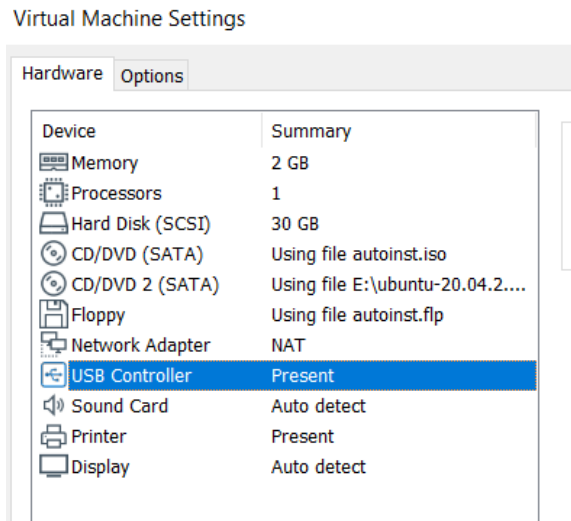
- `sudo apt-get install gparted`

Para realizar la redimensión de la partición es necesario conectar directamente nuestra microSD a nuestra computadora (recomiendo hacerlo mediante un adaptador microSD a USB). En este punto es posible que te encuentres con algunos problemas para conectar la memoria a la máquina virtual si estás haciendo uso del virtualizador VMware ya que probablemente no te reconozca el dispositivo y por lo tanto no puedas manipularlo para redimensionar las particiones. Si este es el caso sigue los siguientes pasos para solucionarlo:

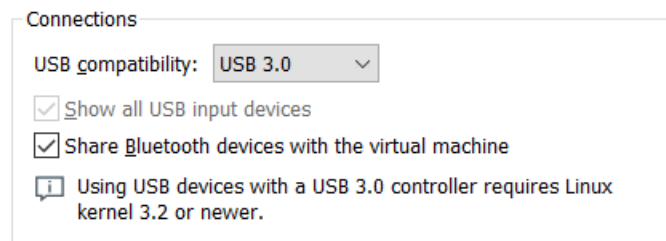
1. Ingresa a las configuraciones de VMware.



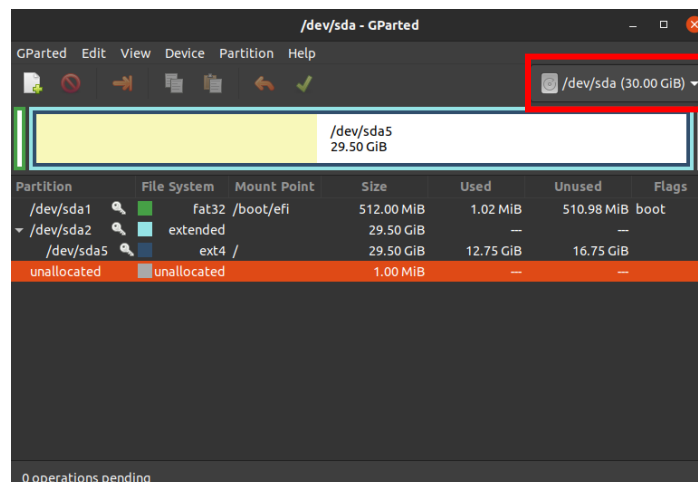
2. En las configuraciones de hardware, dirígete a la opción de USB controller.



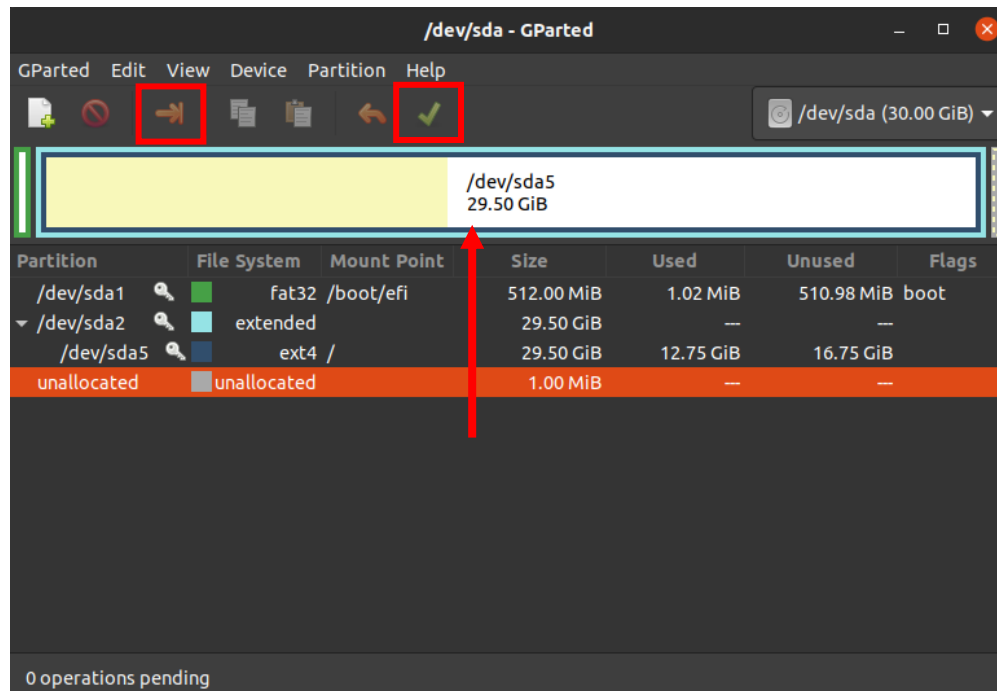
3. Del lado derecho aparecerá la opción de cambiar compatibilidad de USB para la cual debemos seleccionar 3.0.



Con estas configuraciones logramos que nuestra máquina virtual reconozca nuestra memoria y podamos realizar la redimensión de la partición. La realizamos de la siguiente manera.



En el recuadro rojo deberemos seleccionar nuestra microSD que queremos redimensionar.



Debemos seleccionar la opción para redimensionar (cuadro rojo de la izquierda), luego únicamente es necesario arrastrar el rectángulo de la partición hacia la derecha (señalado con flecha) y solo queda aceptar los cambios (cuadro rojo de la derecha).

COMPILAR MOSQUITTO FROM SOURCE

Primero debemos clonar el repositorio de mosquito desde GitHub en el siguiente enlace:

<https://github.com/eclipse/mosquitto>

Nos posicionamos en la carpeta de mosquito que acabamos de clonar en donde se encuentre el source code y ejecutamos el siguiente comando:

- `CC=arm-linux-gnueabi-gcc CXX=arm-linux-gnueabi-g++ AR=arm-linux-gnueabi-ar LD=arm-linux-gnueabi-ld make WITH_CJSON=no WITH_SRV=no WITH_TLS=no WITH_DOCS=no`

Luego ejecutamos el siguiente comando:

- `make install WITH_CJSON=no WITH_SRV=no WITH_TLS=no WITH_DOCS=no DESTDIR=/home/sarrazate/mosquito_build/`

Recuerda que en DESTDIR debemos escribir la dirección de destino para la instalación, en nuestro caso está en ~/mosquito_build/

CROSS COMPILAR APLICACIÓN CON LIBRERÍA MOSQUITTO PARA BEAGLEBONE

Una vez realizados los pasos anteriores con el repositorio de mosquito solo debemos cross compilar de la siguiente manera:

- `arm-linux-gnueabi-gcc -I/home/sarrazate/mosquito_build/usr/local/include -L/home/sarrazate/mosquito_build/usr/local/lib mqtt_pub.c -o test -lmosquitto`

Ahora podremos ejecutar nuestra aplicación desde nuestra tarjeta BeagleBone Black

COMO USAR MOSQUITTO

Para instalar mosquito en las principales distros de Linux solo debemos ejecutar los siguientes comandos:

- `sudo apt update`
- `sudo apt upgrade`
- `sudo apt-get install mosquitto mosquitto-clients`

Si queremos ejecutar mosquitto al arrancar el sistema, ejecutamos:

- `sudo systemctl enable mosquitto.service`

Mosquitto proporciona dos utilidades 'mosquitto_sub' y 'mosquitto_pub', con estas podemos comprobar la comunicación entre el cliente y el bróker. Por defecto la dirección del bróker es localhost y escucha el puerto 1883, no cuenta con usuario ni contraseña, aunque o es complicado agregarlos.

Para probar que nuestro bróker escucha y distribuye nuestros mensajes MQTT, abrimos dos consolas, en donde, una trabajará como publicador y la otra como suscriptor.

MOSQUITTO_SUB

`mosquitto_sub { [-h hostname] [-p port-number] [-u username] [-P password] -t message-topic }`

En donde:

- -u y -p son necesarios únicamente en caso de que se haya hecho configuración previa al bróker.

En este caso haremos la prueba suscribiéndonos al tópic "IoT/Pruebas" de la siguiente manera.

- `mosquitto_sub -h localhost -p 1883 -t "IoT/Pruebas"`

MOSQUITTO_PUB

mosquitto_pub { [-h hostname] [-p port-number] [-u username] [-P password] [-t message-topic] -m "mensaje" }

En donde:

- -u y -p son necesarios únicamente en caso de que se haya hecho configuración previa al bróker.

Para probar nuestro bróker y que nuestro suscriptor pueda recibir los mensajes enviaremos algo sobre el topic "IoT/Pruebas" de la siguiente forma:

- `mosquitto_pub -h localhost -p 1883 -t "IoT/Pruebas" -m "Hola Mundo"`

Veremos algo similar a lo siguiente si todo se realizó con éxito:

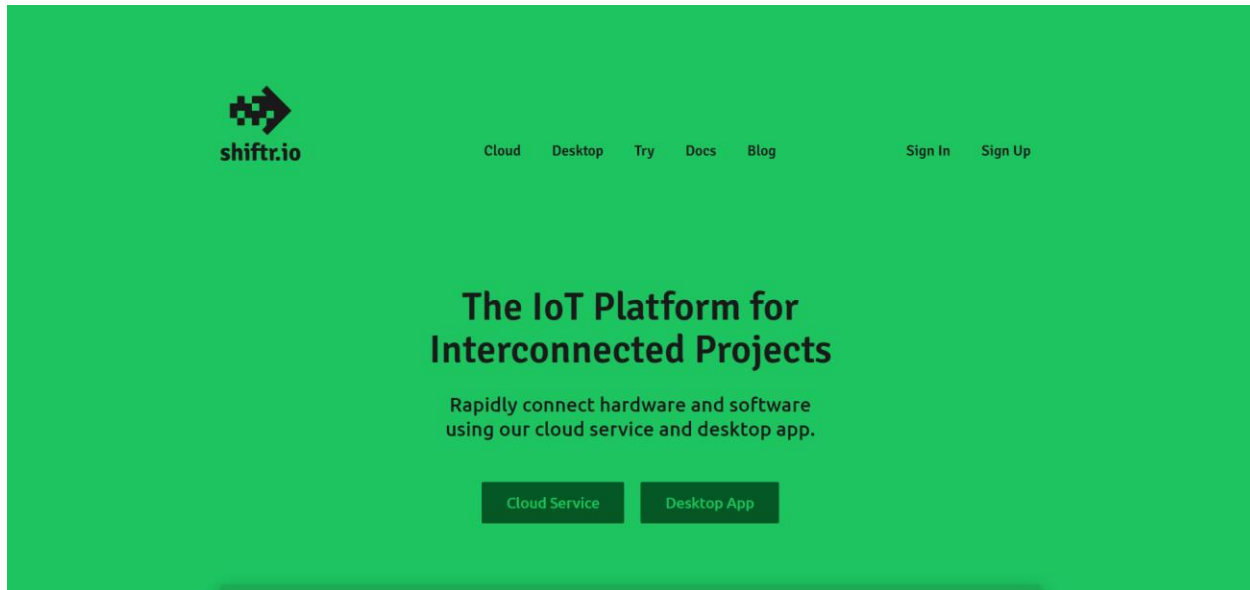
Publicador:

```
C:\Program Files\mosquitto>mosquitto_pub -d -h localhost -p 1883 -t "mimqtt/test" -m "Hola mundo"
Client mosqpub|700-BUSC-4P170 sending CONNECT
Client mosqpub|700-BUSC-4P170 received CONNACK (0)
Client mosqpub|700-BUSC-4P170 sending PUBLISH (d0, q0, r0, m1, 'mimqtt/test', ... (10 bytes))
Client mosqpub|700-BUSC-4P170 sending DISCONNECT
```

Suscriptor:

```
C:\Program Files\mosquitto>mosquitto_sub -d -h localhost -p 1883 -t "mimqtt/test"
Client mosqsub|12704-BUSC-4P170 sending CONNECT
Client mosqsub|12704-BUSC-4P170 received CONNACK (0)
Client mosqsub|12704-BUSC-4P170 sending SUBSCRIBE (Mid: 1, Topic: mimqtt/test, QoS: 0)
Client mosqsub|12704-BUSC-4P170 received SUBACK
Subscribed (mid: 1): 0
Client mosqsub|12704-BUSC-4P170 received PUBLISH (d0, q0, r0, m0, 'mimqtt/test', ... (10 bytes))
Hola mundo
```

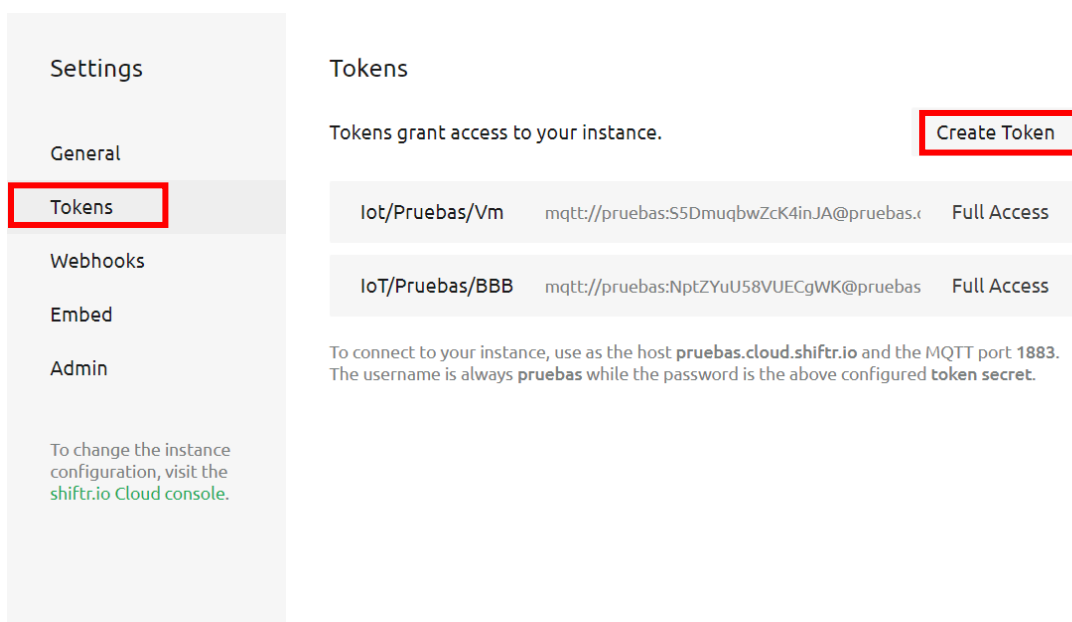
MQTT DESDE SERVIDOR WEB (SHIFTR.IO)



Es necesario crear una cuenta free para poder probar que nuestros mensajes MQTT sean escuchados y publicados por nuestro servidor de la plataforma.

Una vez creada la cuenta, debemos iniciar una instancia la cual es muy fácil de configurar. Lo importante es crear nuestros tokens que son a donde nos vamos a comunicar desde nuestra computadora para hacer las pruebas.

CREANDO LOS TOKENS:

The image shows the "Tokens" settings page in the Shiftr.io interface. On the left is a sidebar with "Settings" at the top, followed by "General", "Tokens" (which is highlighted with a red box), "Webhooks", "Embed", and "Admin". Below the sidebar is a note: "To change the instance configuration, visit the shiftr.io Cloud console." The main content area is titled "Tokens" and contains the text "Tokens grant access to your instance." followed by a "Create Token" button (highlighted with a red box). Below this is a table with two rows of tokens:

lot/Pruebas/Vm	mqtt://pruebas:55DmuqbwZcK4inJA@pruebas.c	Full Access
IoT/Pruebas/BBB	mqtt://pruebas:NptZYU58VUECgWK@pruebas	Full Access

At the bottom of the main content area is a note: "To connect to your instance, use as the host pruebas.cloud.shiftr.io and the MQTT port 1883. The username is always pruebas while the password is the above configured token secret."

Settings

General

Tokens

Webhooks

Embed

Admin

To change the instance configuration, visit the [shifttr.io Cloud console](#).

New Token

Create a new token to grant access to your instance.

Description

IoT/Pruebas/PC

Secret

PZ9htJXaXp84MAY0

The secret can only be set during token creation.

Permission

Full Access

Create Token Cancel

En donde:

- Description es el Topic al que haremos referencia para comunicarnos con el bróker para publicar o suscribirnos en esa instancia.
- Secret será la contraseña que utilizaremos para autenticarnos y poder publicar y suscribirnos
- Permission con esto concedemos permisos solo de lectura o escritura o ambos en el caso de "FULL ACCES"
- El usuario que utilizaremos será el nombre que hayamos puesto al crear nuestra instancia al iniciar a utilizar la aplicación.

Ya que tenemos configurado nuestro Broker web, podemos publicar y suscribirnos de la siguiente manera:

Publicar:

- `mosquitto_pub -h pruebas.cloud.shifttr.io -p 1883 -u pruebas -P S5DmuqbwZcK4inJA -t IoT/Pruebas/Vm -m "Prueba Exitosa"`


```
sarrazate@ubuntu: ~  
sarrazate@ubuntu:~$ mosquitto_pub -h pruebas.cloud.shiftr.io -P S5DmuqbwZcK4inJA -t IoT/Pruebas  
-m "Prueba Exitosa"  
Error: Problem setting username and/or password.  
sarrazate@ubuntu:~$ mosquitto_pub -h pruebas.cloud.shiftr.io -p 1883 -u pruebas -P S5DmuqbwZcK4  
inJA -t IoT/Pruebas -m "Prueba Exitosa"  
sarrazate@ubuntu:~$ mosquitto_pub -h pruebas.cloud.shiftr.io -p 1883 -u pruebas -P S5DmuqbwZcK4  
inJA -t IoT/Pruebas -m "Prueba Exitosa"  
sarrazate@ubuntu:~$ mosquitto_pub -h pruebas.cloud.shiftr.io -p 1883 -u pruebas -P S5DmuqbwZcK4  
inJA -t IoT/Pruebas/Vm -m "Prueba Exitosa"  
sarrazate@ubuntu:~$ mosquitto_pub -h pruebas.cloud.shiftr.io -p 1883 -u pruebas -P S5DmuqbwZcK4  
inJA -t IoT/Pruebas/BBB -m "Prueba Exitosa"  
sarrazate@ubuntu:~$ mosquitto_pub -h pruebas.cloud.shiftr.io -p 1883 -u pruebas -P n1Xvngmhpum0  
ylhb -t IoT/Pruebas/Pc -m "Prueba Exitosa"  
sarrazate@ubuntu:~$ mosquitto_pub -h pruebas.cloud.shiftr.io -p 1883 -u pruebas -P S5DmuqbwZcK4  
inJA -t IoT/Pruebas/Vm -m "Prueba Exitosa"  
sarrazate@ubuntu:~$ mosquitto_pub -h pruebas.cloud.shiftr.io -p 1883 -u pruebas -P n1Xvngmhpum0  
ylhb -t IoT/Pruebas/Pc -m "Prueba Exitosa"  
sarrazate@ubuntu:~$ mosquitto_pub -h pruebas.cloud.shiftr.io -p 1883 -u pruebas -P S5DmuqbwZcK4  
inJA -t IoT/Pruebas/Vm -m "Prueba Exitosa"  
sarrazate@ubuntu:~$ mosquitto_pub -h pruebas.cloud.shiftr.io -p 1883 -u pruebas -P S5DmuqbwZcK4  
inJA -t IoT/Pruebas/Vm -m "Prueba Exitosa"  
sarrazate@ubuntu:~$
```

Suscribir:

- `mosquitto_sub -h pruebas.cloud.shiftr.io -p 1883 -u pruebas -P S5DmuqbwZcK4inJA -t IoT/Pruebas/#`

Nota: El símbolo # que ponemos al final del topic indica que escucharemos a todos los mensajes que se encuentren dentro de IoT/Pruebas sin importar la tercera parte del topic.

```
sarrazate@ubuntu: ~  
sarrazate@ubuntu:~$ mosquitto_sub -h pruebas.cloud.shiftr.io -p 1883 -u pruebas -P S5DmuqbwZcK4in  
JA -t IoT/Pruebas/#  
Prueba Exitosa  
Prueba Exitosa  
Prueba Exitosa  
Prueba Exitosa
```

Broker:

No es necesario hacer ninguna configuración extra al bróker, este ya se encuentra esperando los mensajes del publicador y esperando que un cliente se suscriba.



Posterior a esto podemos usar el bróker de distintas maneras, existen muchas formas y tecnologías para usar como necesitemos este protocolo. Por esta razón recomiendo tener muy en claro los objetivos que queremos lograr y con ello enfocar cosas como el lenguaje a utilizar, las plataformas de desarrollo a utilizar, en donde haremos el despliegue de nuestras aplicaciones, etc.