

Título: ==Nodo Bitcoin==
Fecha de creación: 10-26-2022 (mm/dd/aa)
Hora de creación: 11h06/utc-04
tags: ['#nodo', '#bitcoin']
alias: [nodo completo btc]

Last Update: martes 8 noviembre 2022 23:51:43

Nodo Bitcoin

Para participar de la red [Bitcoin](#) como se desarrolló en [Infraestructura para Bitcoin](#) es necesario tener un ordenador con un cliente de Bitcoin-Core.

Se diferencian dos clases de participantes:

1. Usuarios que mediante un nodo (propio o de un tercero) interactúan con transacciones.
2. Mineros que adicional al nodo tienen un poder de hash para resolver la prueba de trabajo. Esto les genera un beneficio en Bitcoin pero también consume mucha energía eléctrica.

Mantener un nodo no es minar. No tiene un beneficio otorgado por la Red.

Sin embargo encontramos su necesidad por otras razones:

- Mantener la red Bitcoin descentralizada.
- Verificar la información. No solo se descarga la base de datos sino que se verifica que sea correcta en cada bloque.
- Privacidad. Si se desea realizar una transacción o consultar un historial se recomienda hacerlo sin

intermediarios (por ejemplo usando el buscador de google). Esto para no ceder datos personales.

- Realizar análisis de data sin pagar subscripciones o confiar en bases de datos de terceros. Obtener información directamente de la Red.
- Participar de la red **lightning**. Abrir canales para explorar el potencial de un layer2.

Consideraciones Hardware

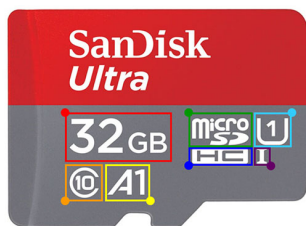
Se detalla el equipo a usarse:

1. Raspberrypi4 4 Gb Ram.
2. Disco duro externo HDD 1 Tera.

Adicionalmente tomamos en consideración los siguientes puntos:

- El RaspberryPi arranca el sistema desde una tarjeta SD. Se recomienda una memoria mayor a 16 Gb que soporte mas de 10 MB/s de lectura/escritura.

LECTURA DE UNA TARJETA SD



- 1: Capacidad de almacenamiento
- 2: Clase 10 de velocidad mínima (10MB/s)
- 3: Certificado de velocidad mínima constante tipo A1
- 4: Formato de tarjeta Micro SD
- 5: Velocidad Ultra High Speed tipo U1 (10MB/s)
- 6: versión de tarjeta SD High Capacity
- 7: Interfaz de Ultra High Speed I (entre 50MB/s y 104MB/s)

- El micro computador puede arrancar sistema con una fuente de 700 mA, en nuestro caso el conectar un disco duro externo por USB hace necesaria una alimentación mayor para no tener cortes y posibles fallas, lo recomendable es disponer una fuente de alimentación que entregue 5 V y más de 2 A.

- Disco Duro Externo. El disco duro puede ser HDD (con USB 3.0) con una velocidad mayor a 50 MB/s para tener algo aceptable. Esto se puede verificar por terminal como veremos luego. Para optimizar el performance en la de sincronización se puede usar un disco duro SSD por su mayor velocidad de lectura/escritura.
- Conexión a Internet. Se usa una conexión mediante cable para tener mayor estabilidad y velocidad.

Para optimizar todos los recursos instalaremos un sistema operativo LITE sin modo gráfico. Instalaremos uno a uno los paquetes a usar evitando software innecesario (en la versión Desktop vienen varias herramientas para mejorar la experiencia del usuario que en este caso de uso no son justificables) y procurando no usar clientes Bitcoin-core como Umbrel, myNode, etc. Sino compilando y verificando propiamente el cliente Bitcoin-Core.

Pre Instalación

En la siguiente página del proyecto Oficial de [Raspberrypi4](https://www.raspberrypi.org/) se puede descargar el software necesario para instalar y configurar un Rpi.

Descargamos dos archivos:

1. Raspberry Pi Imager.

<https://www.raspberrypi.com/software/>

Este programa copia el sistema operativo a la tarjeta SD para ser usado luego en el dispositivo.

Install Raspberry Pi OS using Raspberry Pi Imager

Raspberry Pi Imager is the quick and easy way to install Raspberry Pi OS and other operating systems to a microSD card, ready to use with your Raspberry Pi. [Watch our 45-second video](#) to learn how to install an operating system using Raspberry Pi Imager.

Download and install Raspberry Pi Imager to a computer with an SD card reader. Put the SD card you'll use with your Raspberry Pi into the reader and run Raspberry Pi Imager.

[Download for Windows](#)



2. El sistema operativo Raspbian.

<https://www.raspberrypi.com/software/operating-systems/>

Buscamos el sistema de 64 bits Lite.

Raspberry Pi OS Lite

Release date: September 22nd 2022

System: 64-bit

Kernel version: 5.15

Debian version: 11 (bullseye)

Size: 289MB

[Show SHA256 file integrity hash:](#)

[Release notes](#)

[Download](#)

[Download torrent](#)

[Archive](#)

Con sistema operativo que descargamos debemos verificar que coincide con la firma SHA256 que el fabricante nos entrega. En el caso del ejemplo es:

```
72c773781a0a57160eb3fa8bb2a927642fe60c3af62bc980827057bce  
cb7b98b
```

Este procedimiento se hace para asegurarnos que el sistema operativo no ha sido manipulado por un tercero malicioso inyectando malware. En la terminal aplicamos el siguiente comando.

```
~/Desktop
$ sha256sum 2022-09-22-raspbian-bullseye-arm64-lite.img.xz
72c773781a0a57160eb3fa8bb2a927642fe60c3af62bc980827057bcecb7b98b *2022-09-22-ras
pbian-bullseye-arm64-lite.img.xz
```

Podemos verificar que la firma obtenida coincide con la firma que nos da el fabricante.

Raspberry Pi Imager

Una vez instalado se configura el boot en el SD cuidando los siguientes detalles:

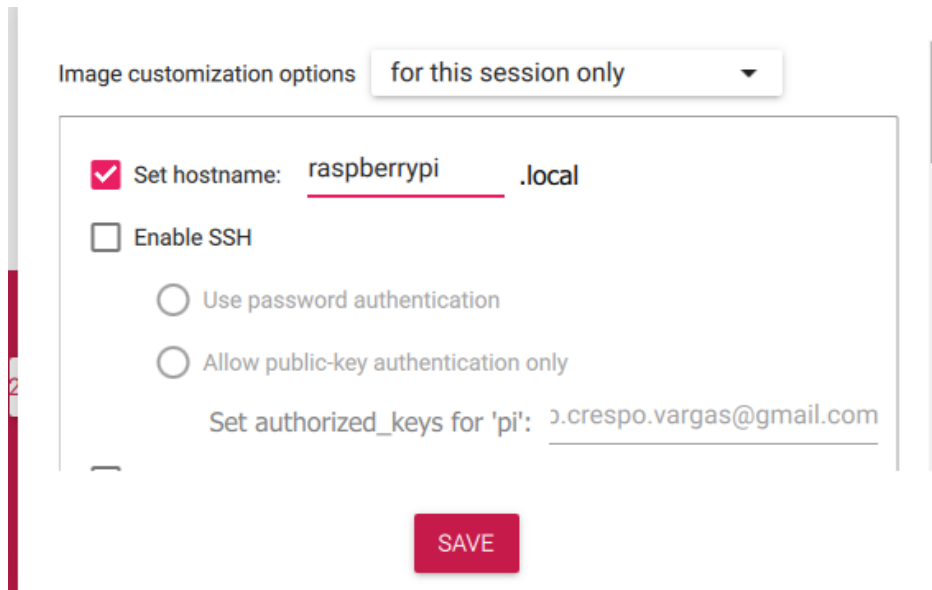


Image customization options for this session only

☒ Set hostname: raspberrypi.local

☐ Enable SSH

☐ Use password authentication

☐ Allow public-key authentication only

Set authorized_keys for 'pi': j.crespo.vargas@gmail.com

SAVE

1. Nombre. Para acceder remotamente facilita el acceso.
2. Enable SSH. Para acceder a una terminal por ssh, para evitar el uso de un teclado y pantalla y controlar remotamente el dispositivo.
3. Conexión a Internet. En el caso del nodo es cableado pero de usarse Wifi se puede configurar antes de iniciar el sistema.

Una vez culminado el proceso de escritura se debe conectar el Raspberry a una fuente de alimentación y esta listo para arrancar.

Configuración Post Instalación Una vez conectado el Raspberry podemos acceder usando SSH. En una terminal **Linux** accedemos con el siguiente comando.

```
ssh user@raspberrypi.local
```

Una vez se introduce la contraseña tenemos la siguiente pantalla.

```
Warning: Permanently added the ECDSA host key for IP address '192.168.1.6' to the list of known hosts.
ghost@onepi.local's password:
Linux onepi 5.15.61-v8+ #1579 SMP PREEMPT Fri Aug 26 11:16:44 BST 2022
aarch64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Oct 26 22:32:28 2022 from fe80::4555:11ce:f2fe:e2ac%wlan0
ghost@onepi:~ $
```

El primer paso es actualizar el sistema con los comandos.

```
sudo apt update && sudo apt upgrade
```

Instalamos algunas dependencias que usaremos para desplegar el repositorio principal.

```
sudo apt install git python3-venv
```

Por seguridad generamos claves ssh para conectarnos con repositorios en Github.

```
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

Generando el archivo `id_rsa.pub` en la dirección `'~/.ssh'`.

Hasta este punto tenemos una configuración básica para un Raspberry. Tanto para desplegar un Nodo con Bitcoin-Core como para usar en análisis de datos o automatismos con Bots.

Preparando Nodo Bitcoin

Una vez que el nodo Raspberry Pi esta listo para funcionar por primera vez conectamos el disco duro por un puerto USB 3.0 (azul) disponible. Al darle energía el nodo toma un par de minutos en estar disponible.

Ping al Nodo

Para constatar que el nodo esta funcionando correctamente y esta conectado a Internet se puede hacer un ping desde otro punto. Para lo cual se necesita saber que IP tiene asignado el Nodo o convenientemente usar el alias `'raspberrry.local'` que se configuro en la Pre Instalación.

Usando el comando `ping` se puede saber si el Nodo esta conectado y funcionando.

```
ping -c raspberrry.local
```

Se muestra cuando el Nodo esta funcionando correctamente y cuando deja de estar operando.

```
gh@pi:~$ ping -c 5 192.168.1.33
PING 192.168.1.33 (192.168.1.33) 56(84) bytes of data.
64 bytes from 192.168.1.33: icmp_seq=1 ttl=64 time=35.4 ms
64 bytes from 192.168.1.33: icmp_seq=2 ttl=64 time=6.29 ms
64 bytes from 192.168.1.33: icmp_seq=3 ttl=64 time=17.6 ms
64 bytes from 192.168.1.33: icmp_seq=4 ttl=64 time=7.81 ms
64 bytes from 192.168.1.33: icmp_seq=5 ttl=64 time=6.97 ms

--- 192.168.1.33 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4007ms
rtt min/avg/max/mdev = 6.289/14.812/35.417/11.093 ms
gh@pi:~$ ping -c 5 192.168.1.33
PING 192.168.1.33 (192.168.1.33) 56(84) bytes of data.

--- 192.168.1.33 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4075ms
```

Una vez se accede por SSH se instala los siguientes paquetes.

```
sudo apt install wget curl gpg git --install-recommends
```

Medimos el Performance

Para asegurarnos que tendremos un buen funcionamiento analizamos el funcionamiento del disco externo. Instalamos el paquete hddparm

```
sudo apt install hddparm
```

Para ver si el disco externo a sido montado correctamente en el sistema usamos el comando lsblk

```
lsblk -pli
```

Retornando la dirección del punto donde el disco externo fue montado.

NAME	MAJ:MIN	RM	SIZE	RO	TYPE	MOUNTPOINT
/dev/sda	8:0	0	931.5G	0	disk	
/dev/sda1	8:1	0	931.5G	0	part	
/dev/mmcblk0	179:0	0	29G	0	disk	
/dev/mmcblk0p1	179:1	0	256M	0	part	/boot
/dev/mmcblk0p2	179:2	0	28.7G	0	part	/

Con la dirección del disco podemos medir la velocidad de lectura/escritura

```
sudo hdparm -t --direct /dev/sda
```

```
/dev/sda1:
Timing O_DIRECT disk reads: 306 MB in 3.00 seconds = 101.86 MB/sec
```

Si la velocidad medida es mayor a < 50 MB/sec es suficiente para usarlo con el Blockchain de Bitcoin.

Si la velocidad en USB 3.0 no llega a ser aceptable se recomienda verificar los drivers, conectores y puertos.

Directorio de Almacenamiento

El almacenamiento de la base de datos del Blockchain se guarda en un directorio dedicado `/data/` que debe estar en un lugar distinto de `/home/`, precisamente en el punto donde se monta el disco externo.

Por lo general en un sistema Linux el montaje del disco externo es automático. Para hacerlo manualmente primero vemos que ha sido reconocido el periférico.

```
sudo fdisk -l
```

Una vez encontrado el disco duro y su dirección, en el caso del ejemplo `/dev/sda1`, se crea una carpeta para que el disco duro sea montado. Luego montamos el disco en esta dirección recién creada.

```
sudo mkdir /media/blockchain
sudo mount /dev/sda1 /media/blockchain
```

Para que este montaje sea automático con cada arranque del sistema (cómo cuando se reinicia inesperadamente el sistema) usamos `fstab`. Necesitamos el ID que tiene el disco externo, lo obtenemos con el siguiente comando.

```
sudo blkid
```

Se copia la variable `PARTUUID` del disco y editamos el archivo de configuración `sudo nano /etc/fstab` y agregamos la siguiente línea

```
PARTUUID=b47b5397-0670-41a3-b38f-2bc64dd2cc18 /media/blockchain ext4
```

Al reiniciar el Nodo el montaje es automático y se puede constatar con el comando `sudo lsblk -pli`

En el punto de montaje `/media/blockchain` es que se crea la carpeta `/data/` donde se almacena el Blockchain. Se le otorgan permisos.

```
sudo mkdir data
sudo chown admin:admin data/
```

Configuración SWAP

En los sistemas Linux además de la memoria RAM es usual que se asigne un espacio extra del disco duro como memoria **swap**. Este tipo de memoria se asigna como una especie de memoria RAM extra (no tan rápida obviamente). En el RaspberryPi esta

configurada por defecto. Para incrementar la velocidad y esencialmente la estabilidad del sistema es que dejamos sin efecto esta asignación.

Editamos el archivo.

```
sudo nano /etc/dphys-swapfile
```

Comentando la siguiente línea únicamente.

```
# comment or delete the CONF_SWAPSIZE line. It will then be created d  
#CONF_SWAPSIZE=100
```

Finalmente reiniciamos la configuración y el servicio para que tenga efecto.

```
sudo dphys-swapfile install  
sudo systemctl restart dphys-swapfile.service
```

Seguridad

Acceso con llaves SSH

Firewall UFW

En la configuración por defecto el acceso a terminal por SSH solicita un password. Cualquier persona que conozca esta contraseña puede acceder a una terminal siendo una brecha de seguridad.

Una mejor opción es deshabilitar el acceso por password y que únicamente se pueda acceder si se tiene un certificado SSH. Para esto se debe copiar la llave pública generada al Raspberry Pi.

Desde la dirección `/.ssh/` copiamos la llave pública al Raspberry.

```
ssh-copy-id admin@raspberrypi.local
```

Finalmente deshabilitamos que se pueda acceder por SSH mediante contraseña.

```
sudo nano /etc/ssh/sshd_config
```

Cambiamos las siguientes líneas.

```
PasswordAuthentication no  
ChallengeResponseAuthentication no
```

Finalmente reiniciamos el servicio ssh.

```
sudo systemctl restart sshd  
exit
```

Firewall

Un firewall controla que tipo de tráfico desde afuera acepta el ordenador y que aplicaciones tienen la salida de datos.

Muchas redes tienen los puertos disponibles por default para distintas entradas en nuevas conexiones. Inhabilitar puertos innecesarios cierra una brecha de seguridad, considerando que Bitcoin-Core se conecta mediante la red de TOR y no necesita ningún puerto de entrada extra.

Usamos un firewall amigable en su configuración.

```
sudo apt install ufw  
sudo ufw default deny incoming
```

```
sudo ufw default allow outgoing
sudo ufw allow ssh
sudo ufw logging off
sudo ufw enable
```

Se habilita a UFW para que inicie automáticamente con cada inicio de sistema.

```
sudo systemctl enable ufw
```

Se puede consultar el estado con el siguiente comando.

```
sudo ufw status
```

En el caso del nodo tenemos esta respuesta.

```
g c:~ $ sudo ufw status
Status: active

To Action From
--
22/tcp ALLOW Anywhere
22/tcp (v6) ALLOW Anywhere (v6)
```

Fail2ban

El ingreso por SSH mediante contraseña se ha deshabilitado, pero suponiendo que el atacante logra tener acceso a la llave privada necesita la contraseña para tener un acceso no permitido. Esta se puede obtener por fuerza bruta.

Para evitar este ataque este software banea el ip donde tenga una contraseña inválida una cantidad x de veces por una cantidad x de tiempo.

Basta con instalar para que fail2ban se active.

```
sudo apt install fail2ban
```

Open File Limit

El nodo puede ser inundado de request (honestos o maliciosos) y dar un error del tipo 'can't accept connection: too many open file'. Esto se soluciona aumentando el límite. Se crea un archivo en la dirección `/etc/security/limits.d/90-limits.conf` con las siguientes líneas.

```
*      soft nfile 128000
*      hard nfile 128000
root soft nfile 128000
root hard nfile 128000
```

Y aumentamos la siguiente línea antes del comentario final en los archivos siguientes.

```
sudo nano /etc/pam.d/common-session
session required                                pam_limits.so
```

```
sudo nano /etc/pam.d/common-session-noninteractive
session required                                pam_limits.so
```

NGINX Reverse Proxy

NGINX es un servidor web open source de alta performance. Muchos componentes en el nodo están expuestos por el puerto de comunicación, como el explorador de bloques.

Si bien estos servicios estarán en la red local es buena práctica que estén encriptados. NGINX se usa para encriptar la comunicación con SSL/TLS. Esta configuración se llama 'reverse proxy'.

Instalamos NGINX.

```
sudo apt install nginx
```

Creamos un certificado SSL/TLS (válido por 10 años)

```
sudo openssl req -x509 -nodes -newkey rsa:4096 -keyout /etc/ssl/private
```

NGINX es un servidor web completo. Para usarlo solo como un reverse proxy usamos la siguiente configuración en `nginx.conf`

```
$ sudo mv /etc/nginx/nginx.conf /etc/nginx/nginx.conf.bak
$ sudo nano /etc/nginx/nginx.conf
```

```
user www-data;
worker_processes 1;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events {
    worker_connections 768;
}

stream {
    ssl_certificate /etc/ssl/certs/nginx-selfsigned.crt;
    ssl_certificate_key /etc/ssl/private/nginx-selfsigned.key;
    ssl_session_cache shared:SSL:1m;
    ssl_session_timeout 4h;
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_prefer_server_ciphers on;

    include /etc/nginx/streams-enabled/*.conf;

}
```

Creamos un nuevo directorio para futuras configuraciones.

```
sudo mkdir /etc/nginx/streams-enabled
```

Testeamos la configuración.

```
sudo nginx -t  
> nginx: the configuration file /etc/nginx/nginx.conf syntax is ok  
> nginx: configuration file /etc/nginx/nginx.conf test is successful
```

Desactivar conexiones inalámbricas

El Raspberry Pi 4 tiene soporte para conexiones Wifi y Bluetooth. Deshabilitamos estas pues el nodo tendrá conexión a Internet por ethernet.

```
sudo nano /boot/config.txt
```

Añadimos las líneas. Se guarda y al siguiente reinicio están desactivadas.

```
dtoverlay=disable-bt  
dtoverlay=disable-wifi
```

Privacidad usando TOR

Al correr un Nodo Bitcoin se expone una dirección IP. Para tener mayor privacidad se recomienda usar un VPN y evitar exponer el IP directamente.

Otra manera es usar TOR project para anonimizar el tráfico y ocultar la dirección IP.

Se instala con el comando.

```
sudo apt install apt-transport-https
```

Creamos un archivo llamado `tor.list` y agregamos las siguientes líneas.

```
sudo nano /etc/apt/sources.list.d/tor.list
```



```
deb      [arch=arm64 signed-by=/usr/share/keyrings/tor-archive-keyring
deb-src  [arch=arm64 signed-by=/usr/share/keyrings/tor-archive-keyring
```

Se accede a `root` para añadir las llaves gpg.

```
sudo su
wget -qO- https://deb.torproject.org/torproject.org/A3C4F0F979CAA22CD
exit
```

Instalamos el keyring

```
sudo apt update
sudo apt install tor deb.torproject.org-keyring
```

Podemos verificar que Tor se instaló correctamente con el comando.

```
tor --version
> Tor version 0.4.7.10.
```

El servicio de Tor debe escuchar por default por los puertos `9050` y `9051`.

Para habilitar el `9051` configuramos la línea en el archivo `/etc/tor/torrc`

```
ControlPort 9051
```

Reiniciamos el servicio

```
sudo service tor restart
```

Y constatamos que esta correctamente configurado.

```
sudo ss -tulpn | grep tor | grep LISTEN
```

Con una salida esperada.

```
root@kali:~# sudo ss -tulpn | grep tor | grep LISTEN
tcp    LISTEN 0      4096      127.0.0.1:9050      0.0.0.0:*    users:(("tor",
pid=12743,fd=6))
tcp    LISTEN 0      4096      127.0.0.1:9051      0.0.0.0:*    users:(("tor",
pid=12743,fd=7))
```

Configurando TOR

Bitcoin-Core se comunica directamente con el demonio Tor para enrutar todo el tráfico mediante esa red. Necesitamos habilitar a Tor para que acepte instrucciones mediante el puerto de control con su propia autenticación.

Modificamos el archivo `/etc/tor/torrc` removiendo el # de las siguientes líneas.

```
# uncomment:
ControlPort 9051
CookieAuthentication 1

# add:
CookieAuthFileGroupReadable 1
```

Recargamos el servicio Tor para activar las modificaciones.

```
sudo systemctl reload tor
```

Revisamos el systemd journal para ver la salida de logs en tiempo real.

```
sudo journalctl -f -u tor@default
```

SSH con TOR

Una forma de controlar el nodo mediante una terminal estando lejos es usando un servicio oculto SSH mediante TOR.

Servidor SSH

Se habilita el servicio en el archivo `torrc`

```
sudo nano /etc/tor/torrc
```

Modificando las siguientes líneas en la sección "location-hidden services",

```
##### This section is just for location-hidden services ###  
# Hidden Service SSH server  
HiddenServiceDir /var/lib/tor/hidden_service_sshd/  
HiddenServiceVersion 3  
HiddenServicePort 22 127.0.0.1:22
```

Se recarga el servicio Tor y se toma nota de la dirección que se genera.

```
$ sudo systemctl reload tor  
$ sudo cat /var/lib/tor/hidden_service_sshd/hostname  
> abcdefg.....xyz.onion
```

Cliente SSH

Desde otro ordenador instalamos netcat

```
sudo apt install netcat tor
```

y accedemos sabiendo el host previamente anotado.

```
ssh -o "ProxyCommand /usr/bin/nc -X 5 -x 127.0.0.1:9050 %h %p" admin@
```

Si el puerto 9050 esta ocupado, Tor usa el puerto 9150 para el servicio ssh.

Instalando Bitcoin-Core



El cliente Bitcoin-core descarga el blockchain completo y valida cada transacción desde el primero en 2009 hasta el último (cada 10 min sale un nuevo bloque). La palabra 'validar' hace referencia a que revisa y constata que cada bloque es correcto mediante su firma hash y la dificultad de la prueba de trabajo POW. Si alguien intenta hacer pasar un bloque malicioso como verdadero el sistema revela que existe manipulación (tamper evident) y el nodo al verificar que es falso lo descarta.

El nodo siempre toma como verdadera la cadena con bloques válidos más larga. Por lo que es virtualmente imposible hacer modificaciones. Esta característica se llama tamper-proof, las transacciones son seguras e inmutables.

Esta validación toma un tiempo en verificar los mas de 700 000 bloques. El Raspberry Pi4 con sus limitaciones demora un par de semanas en realizarlo.

Instalando y compilando el Core

Se descarga la última version del binario del Bitcoin-Core y se compara esta con las firmas y el timestamp. Esta es una precaución para verificar que el Core no ha sido manipulado.

En una carpeta `/tmp/` descargamos la última versión disponible en: <https://bitcoincore.org/en/download/> (ARM Linux 64 bits), como también las firmas criptográficas.

```
# download Bitcoin Core binary
wget https://bitcoincore.org/bin/bitcoin-core-23.0/bitcoin-23.0-aarch64-linux-gnu.tar.gz

# download the list of cryptographic checksum
wget https://bitcoincore.org/bin/bitcoin-core-23.0/SHA256SUMS

# download the signatures attesting to validity of the checksums
wget https://bitcoincore.org/bin/bitcoin-core-23.0/SHA256SUMS.asc
```

Con el `SHA256SUMS` constatamos que el Core tenga la misma firma.

```
sha256sum --ignore-missing --check SHA256SUMS
> bitcoin-23.0-aarch64-linux-gnu.tar.gz: OK
```

Revisión de Firmas

Bitcoin-Core se lanza en cada versión con la firma de un número de desarrolladores donde cada uno usa su firma. Para verificar y validar cada una de estas firmas, primero descargamos las llaves públicas.

```
wget https://raw.githubusercontent.com/bitcoin/bitcoin/master/contrib/verifykeys.py
while read fingerprint keyholder_name; do gpg --keyserver hkps://keys.openpgp.org --recv-keys $fingerprint; done
```

Importamos una a una las firmas GPG.

```
gpg: key 7588242FBE38D3A8: public key "Gavin Andresen <gavinandresen@gmail.com>" imported
gpg: Total number processed: 1
gpg: imported: 1
gpg: key 8F617F1200A6D25C: public key "Gloria Zhao <gloriazhao@berkeley.edu>" imported
gpg: Total number processed: 1
gpg: imported: 1
gpg: key 410108112E7EA81F: public key "Hennadii Stepanov (GitHub key) <3296351@hebasto@users.noreply.github.com>" imported
gpg: Total number processed: 1
gpg: imported: 1
gpg: keyserver receive failed: No data
gpg: key 25F27A38A47AD566: public key "James O'Beirne <james.obeirne@pm.me>" imported
gpg: Total number processed: 1
gpg: imported: 1
gpg: key 2DA9C5A7FA81EA35: public key "Jarol Rodriguez <jarolrod@tutanota.com>" imported
gpg: Total number processed: 1
gpg: imported: 1
gpg: key 535B12980BB8A4D3: public key "Jeffri H Frontz <jeff.frontz@gmail.com>" imported
gpg: Total number processed: 1
gpg: imported: 1
gpg: key C5242A1AB3936517: public key "jl2012 <jl2012@xbt.hk>" imported
gpg: Total number processed: 1
gpg: imported: 1
gpg: key 796C4109063D4EAF: public key "Jon Atack <jon@atack.com>" imported
gpg: Total number processed: 1
gpg: imported: 1
gpg: key 29D4BCB6416F53EC: public key "Jonas Schnelli <dev@jonasschnelli.ch>" imported
gpg: Total number processed: 1
```

Se verifican las firmas con el archivo `SHA256SUMS.asc` poniendo énfasis en el siguiente texto.

```
> gpg: Good signature from ...
> Primary key fingerprint: ...
```

Verificando Timestamp

El binario checksum es un archivo que tiene un timestamp en el Blockchain de Bitcoin. Usando el protocolo [OpenTimestamps](#) prueba que el archivo existía previamente al un punto en el tiempo.

Note

On your local computer, download the checksums file and its timestamp proof:

- <https://bitcoincore.org/bin/bitcoin-core-23.0/SHA256SUMS>
- <https://bitcoincore.org/bin/bitcoin-core-23.0/SHA256SUMS.ots>

- In your browser, open the [OpenTimestamps website](#)
- In the “Stamp and verify” section, drop or upload the downloaded SHA256SUMS.ots proof file in the dotted box
- In the next box, drop or upload the SHA256SUMS file
- If the timestamps is verified, you should see the following message. The timestamp proves that the checksums file existed on the [release date](#) of Bitcoin Core v23.0.

Una vez verificado de que el core es legítimo y seguro mediante checksum, firmas digitales y el timestamp procedemos a Instalarlo.

Instando el Core

Se descomprime el Core y se procede a instalarlo.

```
tar -xvf bitcoin-23.0-aarch64-linux-gnu.tar.gz  
sudo install -m 0755 -o root -g root -t /usr/local/bin bitcoin-23.0/b
```

Se puede ver que esta correctamente instalado al consultar la versión.

```
bitcoind --version  
> Bitcoin Core version v23.0.0
```

bitcoin user

La aplicación Core puede correr en segundo plano como un demonio y por razones de seguridad desde un nuevo usuario 'bitcoin' sin permisos de admin ni la posibilidad de cambiar archivos de configuración del sistema.

Creamos un nuevo usuario bitcoin

```
sudo adduser --gecos "" --disable-password bitcoin
```

Añadimos el usuario admin al grupo bitcoin

```
sudo adduser admin bitcoin
```

Permitimos al usuario bitcoin configurar Tor directamente al añadirlo al grupo debian-tor

```
sudo adduser bitcoin debian-tor
```

Data folder

El Bitcoin-Core descarga la data usando por defecto el directorio `.bitcoin` en `/home/`. En el Raspberry tenemos un disco externo específicamente para almacenar la data. Para lo cual creamos un directorio en el disco duro.

```
mkdir /media/blockchain/data/bitcoin  
sudo chown bitcoin: bitcoin /media/blockchain/data/bitcoin
```

Y desde el usuario `bitcoin` creamos un link simbólico que apunte al directorio que hemos creado.


```
sudo su - bitcoin
whoami
ln -s /data/bitcoin /home/bitcoin/.bitcoin
```

Para verificar que todo esta correctamente configurado usamos el comando `ls -la` que no debe mostrar error en su respuesta ('.bitcoin' en rojo indicaría que hay error).

```
g [redacted] c:~ $ ls -la
total 68
drwxr-xr-x 8 ghost ghost 4096 Nov  8 10:21 .
drwxr-xr-x 4 root  root  4096 Oct 28 10:58 ..
-rw----- 1 ghost ghost 8690 Nov  8 11:05 .bash_history
-rw-r--r-- 1 ghost ghost  220 Sep 21 22:51 .bash_logout
-rw-r--r-- 1 ghost ghost 3523 Sep 21 22:51 .bashrc
lrwxrwxrwx 1 ghost ghost   31 Oct 28 11:05 .bitcoin -> /media/blockchain/data/bi
tcoin/
drwxr-xr-x 3 ghost ghost 4096 Nov  8 06:24 .cache
```

Generando Credenciales de Acceso

Para otras instancias o programas que necesiten realizar consultas al Bitcoin Core, es que se crean credenciales de acceso adecuadas.

Bitcoin Core provee un simple programa en Python para generar un archivo de configuración.

En el directorio bitcoin descargamos RPCAuth (RPC remote procedure call).

```
cd .bitcoin
wget https://raw.githubusercontent.com/bitcoin/bitcoin/master/share/r
```

Al correr el script, se solicitan un username `raspiolt` y un password como argumentos.

Este password es para **Bitcoin RPC** únicamente.

```
python3 rpcauth.py raspiolt YourPasswordB
> String to be appended to bitcoin.conf:
> rpcauth=raspiolt:00d8682ce66c9ef3dd9d0c0a6516b10e$c31da4929b3d0e09
```

Se copia la última línea de `rpcauth`.

Configuración bitcoind

Si abrimos desde el usuario `bitcoin` el siguiente archivo, llenamos el espacio `rpcauth` con la última línea que copiamos anteriormente.

```
nano /home/bitcoin/.bitcoin/bitcoin.conf
```

```
# RaspiBolt: bitcoind configuration
# /home/bitcoin/.bitcoin/bitcoin.conf

# Bitcoin daemon
server=1
txindex=1

# Network
listen=1
listenonion=1
proxy=127.0.0.1:9050
bind=127.0.0.1

# Connections
rpcauth=<replace with your own auth line generated by rpcauth.py>
zmqpubrawblock=tcp://127.0.0.1:28332
zmqpubrawtx=tcp://127.0.0.1:28333
whitelist=download@127.0.0.1          # for Electrs

# Raspberry Pi optimizations
maxconnections=40
maxuploadtarget=5000
```

```
# Initial block download optimizations
dbcache=2000
blocksonly=1
```

Se le otorgan permisos para que solo el usuario bitcoin y otros miembros del grupo bitcoin puedan solo leerlo.

```
chmod 640 /home/bitcoin/.bitcoin/bitcoin.conf
```

Corriendo bitcoind

Aún con el usuario `bitcoin` iniciamos manualmente a bitcoind

```
bitcoind
```

Empieza a correr un log en pantalla que muestra que funciona. Se puede parar con `ctrl+c`. Se otorgan permisos de lectura dentro del grupo para el archivo del log.

```
chmod g+r ~/data/bitcoin/debug.log
exit
```

Como usuarios admin realizamos un link del archivo .bitcoin para usarlo directamente.

```
$ ln -s /data/bitcoin /home/admin/.bitcoin
```

Autoinicio en boot

El sistema necesita correr el demonio de bitcoin (demon o demonio es un término en Linux para referir a un proceso que corre en segundo plano) automáticamente, incluso si nadie se logea en el sistema.

Se usa a `systemd`, otro demonio que controla que el proceso se haya iniciado usando los archivos de configuración.

Creamos un archivo para configurar el demonio bitcoin.

```
$ sudo nano /etc/systemd/system/bitcoind.service
```

Con el siguiente contenido.

```
# Raspibolt: systemd unit for bitcoind
# /etc/systemd/system/bitcoind.service

[Unit]
Description=Bitcoin daemon
After=network.target

[Service]

# Service execution
#####

ExecStart=/usr/local/bin/bitcoind -daemon \
                                -pid=/run/bitcoind/bitcoind.pid \
                                -
conf=/home/bitcoin/.bitcoin/bitcoin.conf \
                                -datadir=/home/bitcoin/.bitcoin \
                                -startupnotify="chmod g+r
/home/bitcoin/.bitcoin/.cookie"

# Process management
#####
Type=forking
PIDFile=/run/bitcoind/bitcoind.pid
Restart=on-failure
TimeoutSec=300
RestartSec=30
```

```
# Directory creation and permissions
#####
User=bitcoin
UMask=0027

# /run/bitcoind
RuntimeDirectory=bitcoind
RuntimeDirectoryMode=0710

# Hardening measures
#####
# Provide a private /tmp and /var/tmp.
PrivateTmp=true

# Mount /usr, /boot/ and /etc read-only for the process.
ProtectSystem=full

# Disallow the process and all of its children to gain
# new privileges through execve().
NoNewPrivileges=true

# Use a new /dev namespace only populated with API pseudo devices
# such as /dev/null, /dev/zero and /dev/random.
PrivateDevices=true

# Deny the creation of writable and executable memory mappings.
MemoryDenyWriteExecute=true

[Install]
WantedBy=multi-user.target
```

Habilitamos el servicio y reiniciamos el sistema para empezar a correr.

```
sudo systemctl enable bitcoind.service
sudo reboot
```

Verificando bitcoind

Luego del reinicio ya debe empezar a sincronizar el blockchain para validarlo.

Para ver si el servicio esta funcionando usamos el comando siguiente.

```
sudo systemctl status bitcoind.service
> * bitcoind.service - Bitcoin daemon
>    Loaded: loaded (/etc/systemd/system/bitcoind.service; enabled;
>    Active: active (running) since Thu 2021-11-25 22:50:59 GMT; 7s
>    Process: 2316 ExecStart=/usr/local/bin/bitcoind -daemon -pid=/r
>    Main PID: 2317 (bitcoind)
>    Tasks: 12 (limit: 4164)
>    CPU: 7.613s
>    CGroup: /system.slice/bitcoind.service
>            └─2317 /usr/local/bin/bitcoind -daemon -pid=/run/bitco
>
```

Se debe tener cuidado especial en checar si el grupo `bitcoin` tiene los permisos correctos. La salida debe contener `-rw-r---` pues de lo contrario ninguna aplicación ejecutada por otro usuario podría acceder al Bitcoin-Core.

```
ls -la /home/bitcoin/.bitcoin/.cookie
> -rw-r----- 1 bitcoin bitcoin 75 Dec 17 13:48 /home/bitcoin/.bitcoin
```

Y ya podemos empezar a interactuar con el Core usando el `bitcoin-cli`

```
$ bitcoin-cli getblockchaininfo
```

En la siguiente imagen se muestra la salida un tiempo después de sincronizar el blockchain.

[illegible]

La sincronización será completa una vez que “verificationprogress” llegue a 1 (0.99999...)

Consideraciones Finales

Si se sigue paso a paso la configuración se obtiene un nodo completo sin usar clientes. Conlleva mucho tiempo y esfuerzo, requiriendo conocer trabajar mucho con la terminal.

En un futuro se buscará replicar este manual enfocados en montarlo sobre la red `testnet`.