Laboratorio de Sistemas Operativos y Redes

Guía de Instalación y uso de:

Rodrigo García
Agustín Tupilojón

INDICE DE CONTENIDOS

INDICE DE FIGURAS

1. INTRODUCCIÓN

1.1. ¿QUÉ ES COREOS?

Recientemente adquirido por Red Hat y rebautizado como Container Linux, CoreOS es un sistema operativo ligero de código abierto basado en el kernel de Linux y diseñado para proporcionar la infraestructura para los despliegues en clúster, mientras se centra en la automatización, facilidad de despliegue de aplicaciones, seguridad, fiabilidad y escalabilidad.

Como sistema operativo, CoreOS ofrece sólo las funcionalidades mínimas necesarias para la implementación de aplicaciones dentro de contenedores de software, junto con mecanismos incorporados para el descubrimiento de servicios y el intercambio de configuración.

Los mecanismos principales utilizados son:

- etcd.

  Administración y sincronismo de clústers.

- fleetd y kubletd.

  Provisionador de aplicaciones para clústers.

- systemd.

  Sistema para administrar el inicio y la terminación de los procesos, tanto del kernel CoreOS como de las aplicaciones en containers.

- docker y rkt.

  Plataformas para la administración de containers.

1.2. CONTAINERS

CoreOS no ejecuta ningún servicio o aplicación de manera directa, sino que, a diferencia del resto de los sistemas operativos, lo hace a través de containers administrados ya sea por **Docker** o por **RKT**. Esta característica es lo que los hace un sistema operativo liviano y, en consecuencia, confiable y estable.

La adopción de containers viene ganando muchos adeptos para la virtualización de aplicaciones y hasta otros sistemas operativos (basados en Linux) ya que mejora el rendimiento de los recursos del **host** (hardware utilizado para 'montar' los servicios). Las ventajas de la containerización es que evita la reserva de recursos del **host** (RAM, disco, CPU) compartiéndola de manera aislada (procesos independientes) entre las aplicaciones dándole mayor escalabilidad en cuanto a la cantidad de servicios. Por otro lado, el tiempo de creación y disponibilidad de una aplicación es incomparablemente superior, y, además, son portables.

A continuación, se muestra la comparativa gráfica entre ambos mecanismos de virtualización.

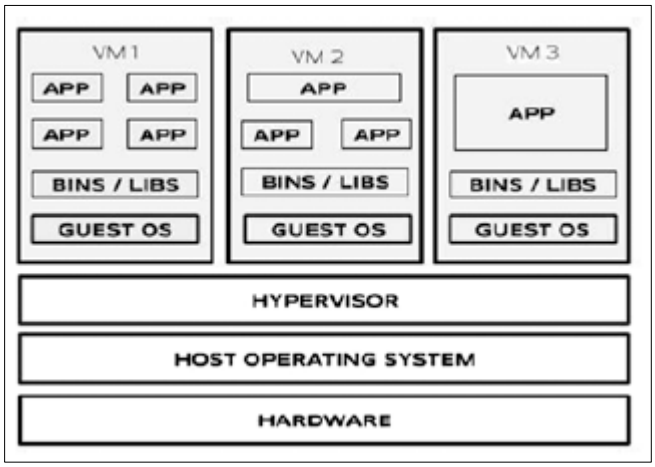


Figura 1 - Virtual Machines

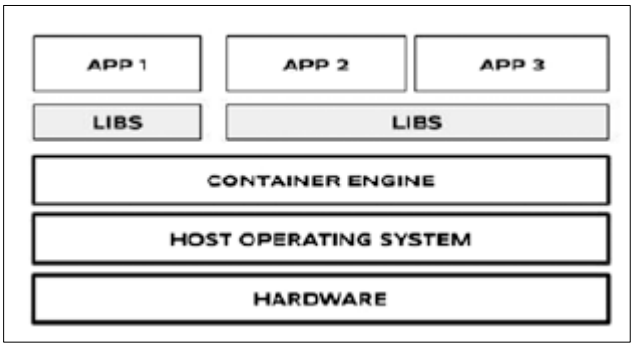


Figura 2 - Containers

CoreOS utiliza docker como contenedor para implementar servicios dentro del cluster, este contender permite asociar un servicio con su modulo en una imagen independiente del servicio. Esto permite evitar cualquier construcción de un entorno de compilación y reconstrucción de imagen en la plataforma y así cierra la brecha entre desarrollo e implementación de un servicio

Nuevamente, la ventaja de este sistema comparada con otras distribuciones de linux es que al no tener softwares instalados ni bibliotecas como apt, yum, etc, lo hace un sistema super ligero y liviano.

El mercado está siendo monopolizado por Docker (por sobre otras plataformas de administración de containers como por ejemplo LXC y su complemento LXD). Sin embargo, a medida que es robustecido por los requerimientos del mercado por el otro lado la visión de los diseñadores de CoreOS los llevó a crear una nueva plataforma denominada **rkt** que mejora el rendimiento y la aislación de los procesos.

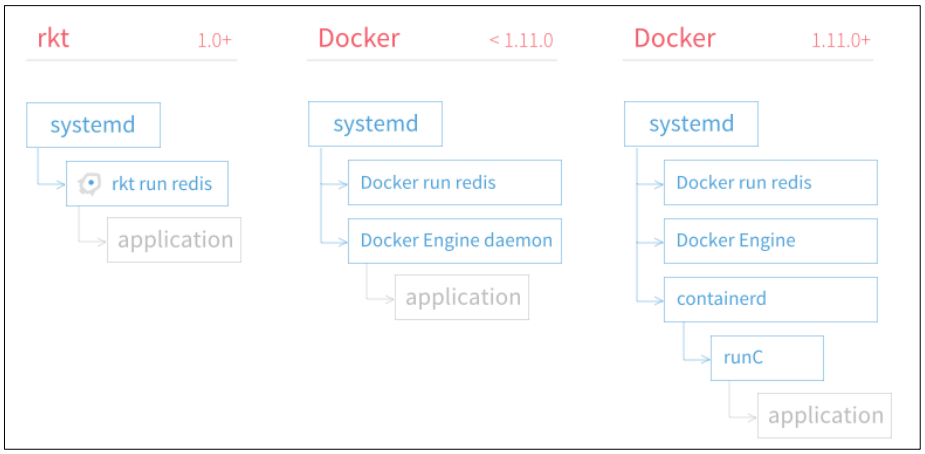


Figura 3 - Comparativa motor de contanerización

1.3. SYTEMD

El concepto de systemd es una utilidad del sistema (presente en la mayoría de las distribuciones basadas en Linux) diseñado para ejecutar múltiples operaciones en paralelo y se utiliza para detener, iniciar y reiniciar cualquiera de los servicios del sistema (denominados “units”) o programas de usuario durante el booteo.

Está que van desde las mencionadas hasta la administración de volúmenes, disco, dispositivos, etc.

systemd es el primero proceso que se inicia al booteo de coreOS y tiene dos terminologías o conceptos principales: unit y target.

- Unit es un archivo de configuración que describe las propiedades del servicio que se iniciará.

- Target es un mecanismo que permite agrupar múltiples servicios que se iniciarán al mismo tiempo.

Además, coreOS utiliza systemd para la administración del ciclo de vida de los containers y es totalmente customizable

1.4. CLUSTER

El concepto de clúster es la distribución de servicios en distintos nodos y presentarlos como un único front-end de manera de ofrecer entornos redundantes ante falla de alguna de los nodos.

Es imprescindible para su implementación, que todos los nodos se encuentren sincronizados, de manera que la pérdida de alguno de ellos no conlleve a consecuente pérdida de información.

El sistema coreOS esta agrupado en distintos nodos, en donde uno de ellos será nombrado como nodo "master" por el servicio etcd. Todos los nodos "slave" (el resto de los nodos del clúster) deben proporcionarle al nodo “master” la lista de servicios en ejecución, mediante un comando "float" que lleva esta información hasta el destino (nodo master).

A continuación, se mostrará una imagen con la arquitectura donde hay un clúster y todos comparten el mismo ID. Además de coreOS, cada nodo también posee servicios de systemd y etcd en ejecución.

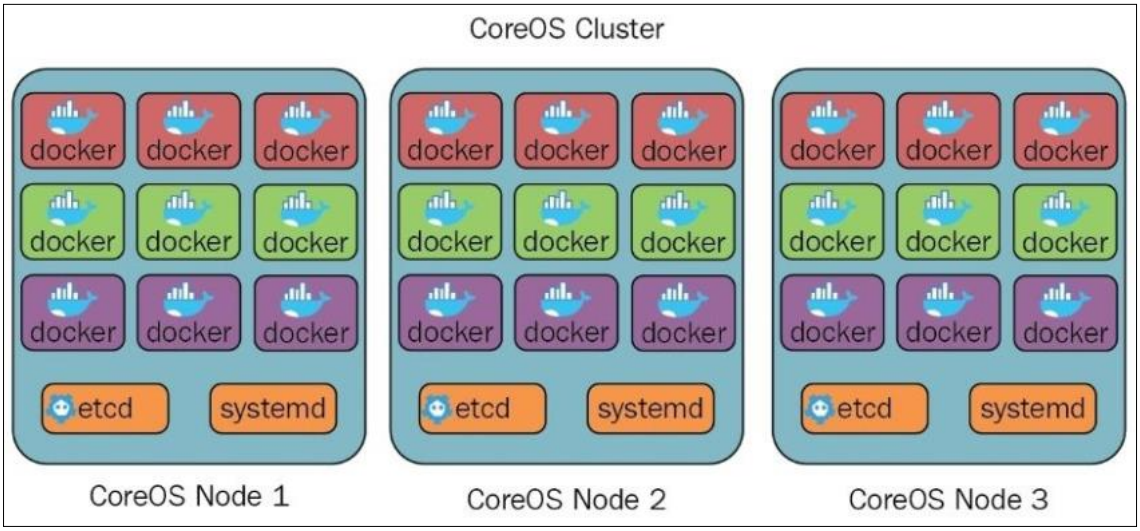


Figura 4 - Clustering

1.4.1. FLEET

El término **fleet** hace referencia a una herramienta que emula todos los nodos del clúster para que formen parte de un único sistema, y controla el servicio **systemd** a nivel de clúster, no en el nodo individual.

Recientemente, por ello que aún merece un título, fleet ha sido dejado de ser recomendado para coreOS y, en su lugar, se recomienda la utilización de **Kubernetes** o **Tectonic**, este último desarrollado por el equipo coreOS pero en versión comercial.

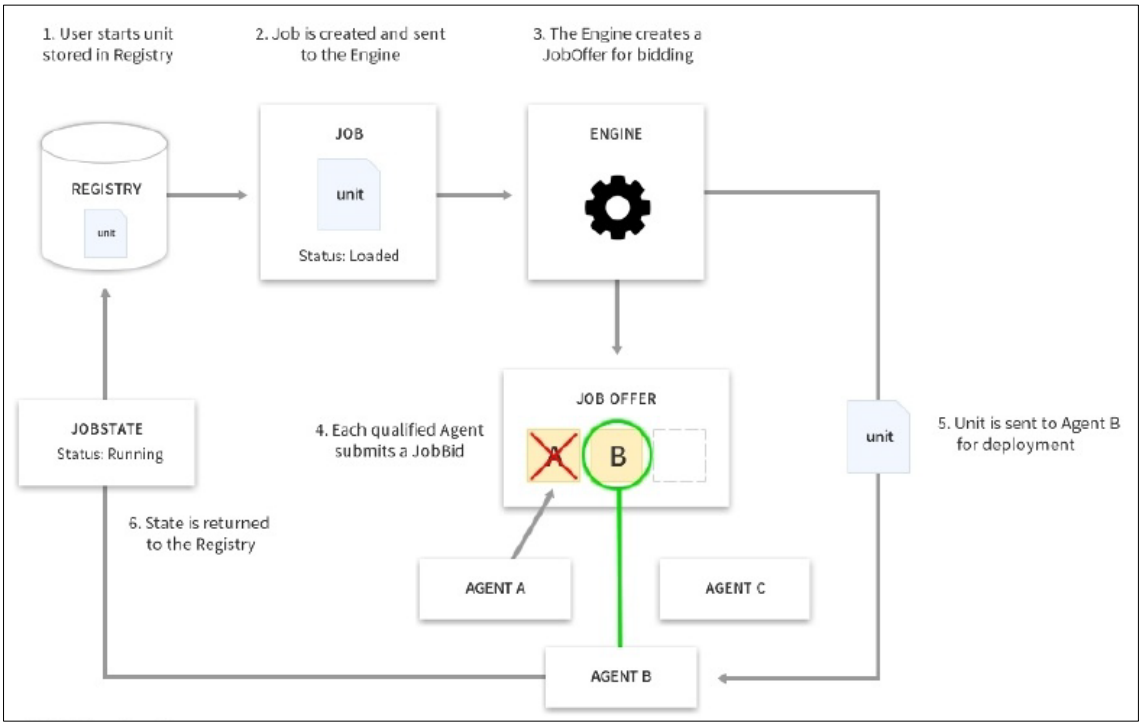


Figura 5 - Flujo de trabajo en Fleet

1.4.2. KUBERNETES

Kubernetes ofrece una infraestructura de orquestación para el despliegue de containers o grupo de containers (pod) en un clúster como si fuere un único servicio.

Es de tipo open source y proviene de la mejora de la herramienta de Google denominada Borg. Como nota de color, muchos de los desarrolladores de Borg participan ahora activamente en Kubernetes.

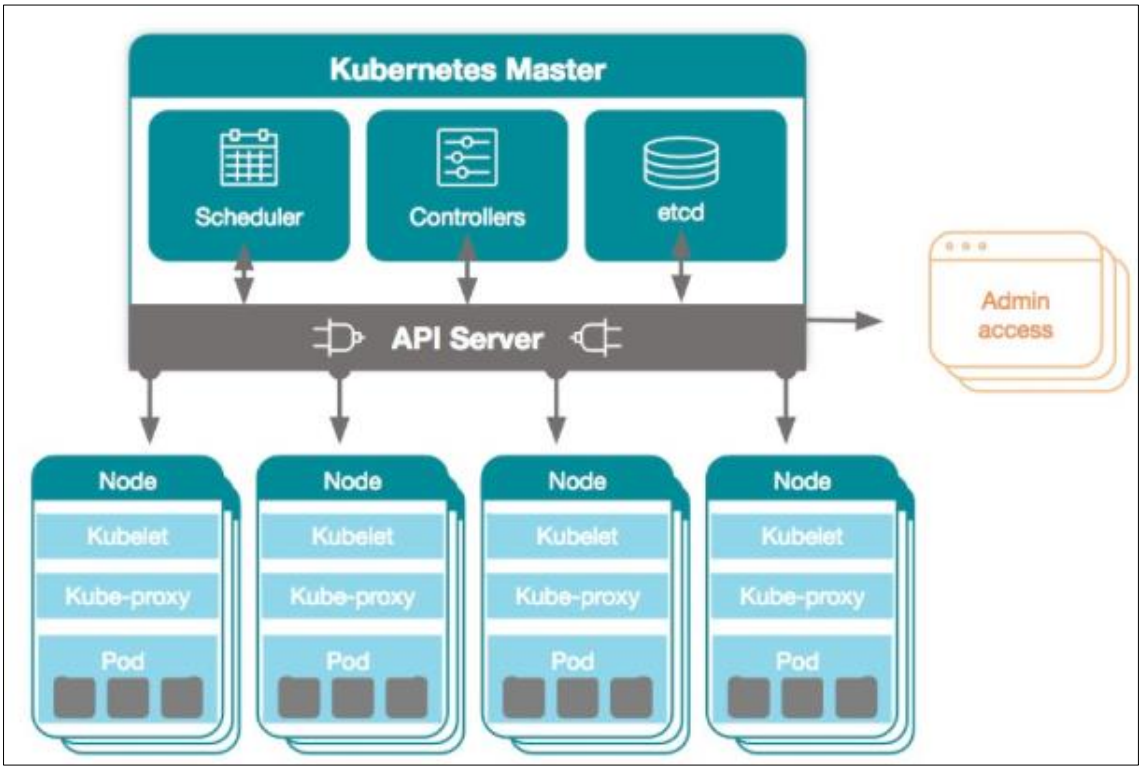


Figura 6 - Arquitectura Kubernetes

2. INSTALACIÓN

2.1. PLATAFORMA

Por practicidad (fácil de romper, loguear y volver a probar) se realizó la instalación de coreOS en PC Windows 64bits (**host**) utilizando **VirtualBox**.

2.2. PRIMEROS PASOS

2.2.1. VIRTUAL BOX

En la página de la apliación hay opciones de descarga los sistemas operativos más populares. E nuestro caso, como estamos usando Windows, descargamos el .exe.

https://download.virtualbox.org/virtualbox/6.0.8/VirtualBox-6.0.8-130520-Win.exe

2.2.2. COREOS

Utilizaremos para la instalación la última versión (current) estable (stable, lista para ser puesta en produccón) publicada en el repositorio de coreOS.

https://stable.release.core-os.net/amd64-usr/current/coreos_production_iso_image.iso

Existen, además, versiones Alpha (versiones de testing seguidas activamente por los desarrolladores) y Beta (versiones Alpha promovidas para ser lanzadas a producción)[1].

2.3. PROCEDIMIENTO MANUAL

El procedimiento de instalación de coreOS consta de dos pasos: **booteo** e **instalación**.

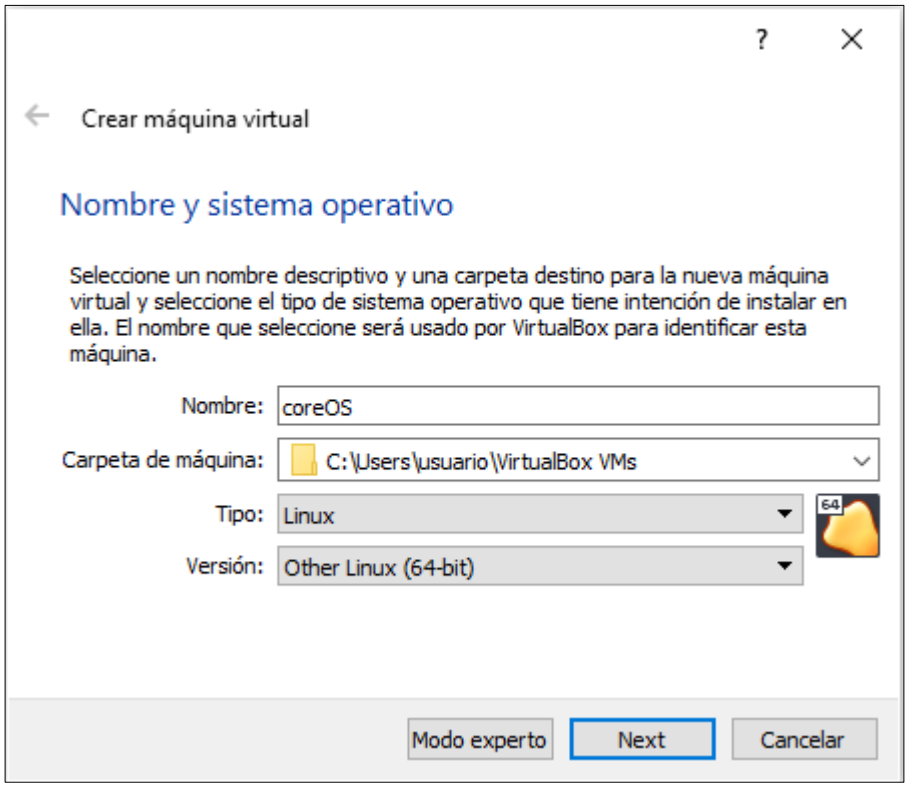
Decimos que es el procedimiento "manual" porque es similar (con sus particularidades) al que aplicaríamos en otras distribuciones de Linux.

[1] https://coreos.com/releases/

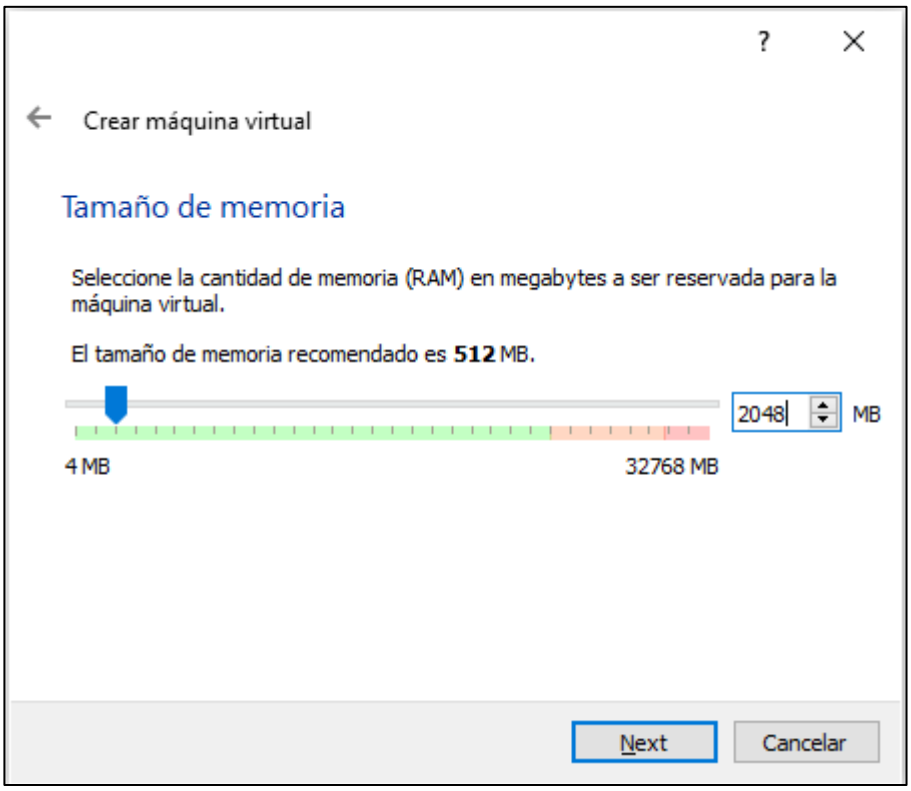
2.3.1. BOOTEO

El primer paso consiste en utilizar la imagen .iso previamente descargada para tener acceso al shell.

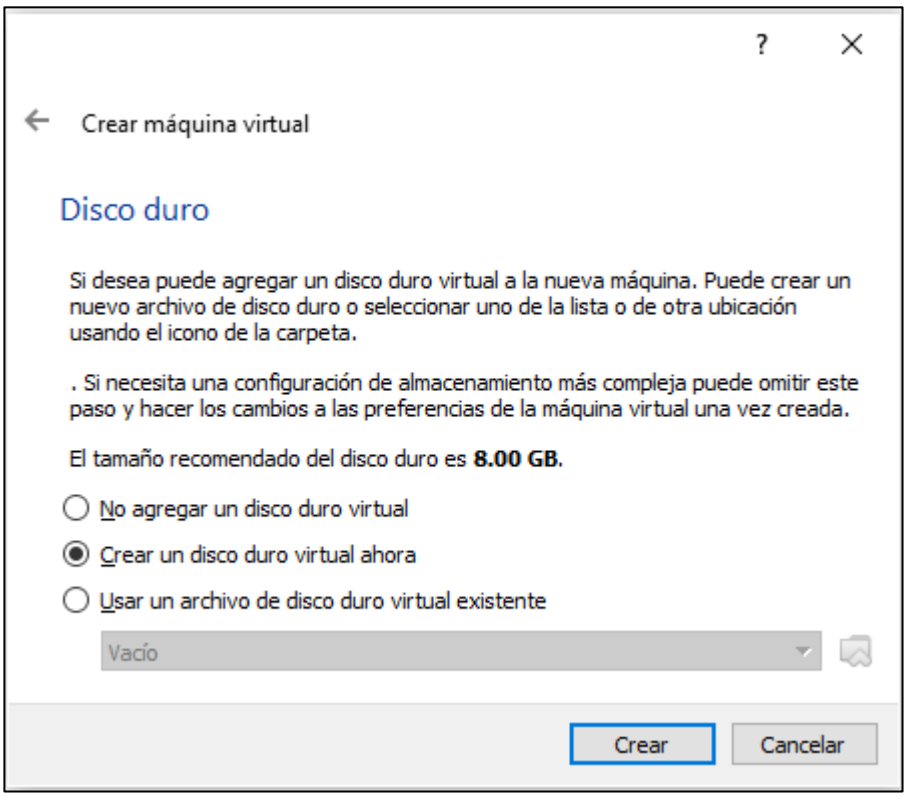
Creamos una VM nueva y le asignamos, en este caso, el nombre coreOS.



Se recomiendan 2GB RAM.

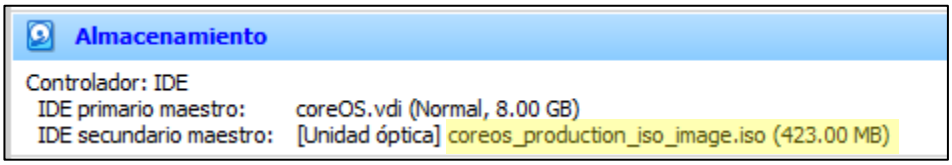


A los fines del tutorial, no existen mayores requerimientos de Disco.

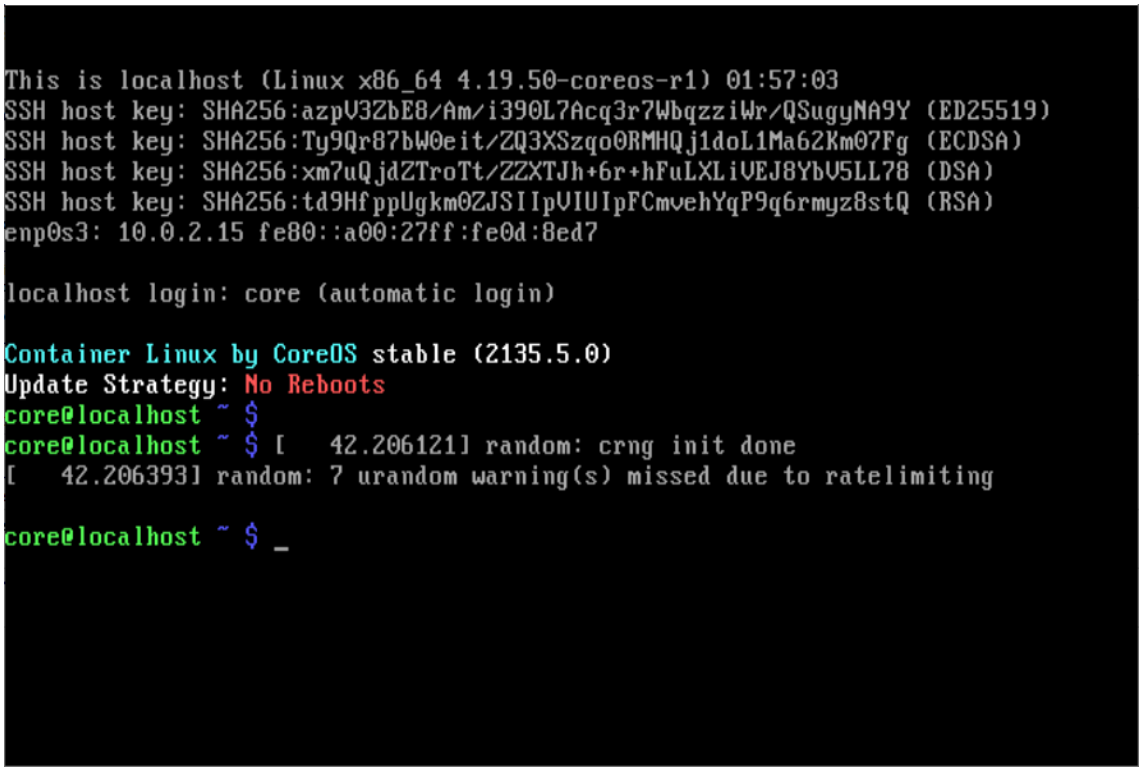


Le damos continuar con las opciones por defecto hasta que se obtenga la VM creada.

Luego, cargamos la imagen en la unidad de almacenamiento.



Iniciamos la máquina con pantalla.



El sistema operativo booteó correctamente.

Estamos en un shell de bash logueados con un usuario root (core) sin posibilidad de setearle una password.

2.3.2. INSTALACIÓN

La instalación que vamos a utilizar se la conoce como **cloud-init**. No es la opción recomendada por coreOS ya que implica un segundo reboot que con la opción **ignition** puede evitarse.

Si fuere más cómodo para el administrador, es posible crear usuarios en esta instancia como en cualquier otra distro (con `useradd`) para luego ingresar a la VM por SSH.

En nuestro caso nos manejaremos por la consola visual con el usuario `core`.

Para la instalación vamos a agregar un usuario y un hostname. Para ello primero tenemos que generar hash de password[2] y lo guardamos en la misma ubicación en la que estamos parados (`/home/core`) un archivo que denominaremos cloud-init-config.

En el ejemplo, el hash corresponde a la password **test**.

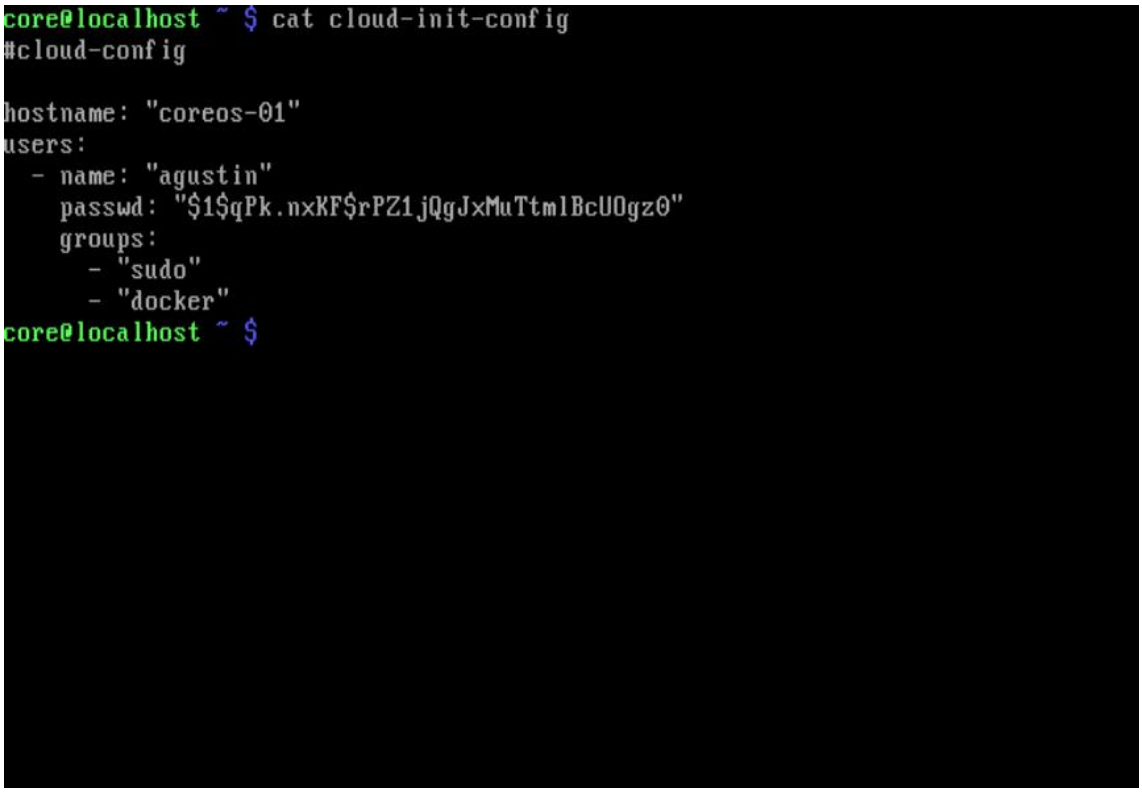
```
core@localhost ~ $ openssl passwd -1 > cloud-init-config
Password:
Verifying - Password:
core@localhost ~ $ cat cloud-init-config
$1$qPk.nxKF$rPZ1jQgJxMuTtmlBcUOgz0
core@localhost ~ $ _
```

[2] Puede ser generada en otro dispositivo pero por practicidad la generamos en el mismo sistema y la copiamos al archivo que luego utilizaremos.

El archivo cloud-init-config es el que se utilizará para la configuración del sistema operativo cuando se instale y se escribe en formato YAML[3].

Editamos el archivo con VIM, que es el editor de coreOS. El archivo siempre debe comenzar con `#cloud-config`.

```
vim cloud-init-config
```



Lo guardamos con `ctrl+c` y luego `:wq!`

Creado el archivo de configuración, procedemos a la instalación.

```
sudo coreos-install -d /dev/sda -C stable -c cloud-config-file
```

Comienza la descarga de la versión indicada (stable).

[3] Opciones tipificada: https://coreos.com/os/docs/latest/cloud-config.html

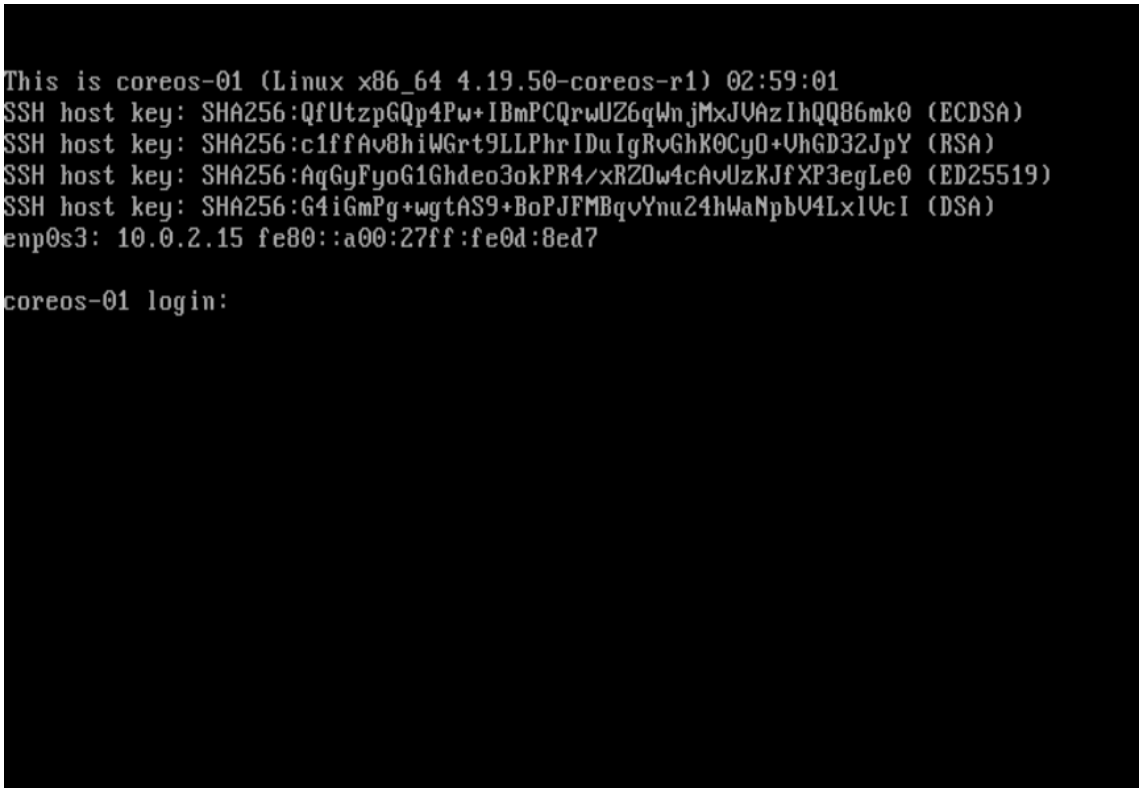
```
core@localhost ~ $ sudo coreos-install -d /dev/sda -C stable -c cloud-init-confi
g
2019/07/12 02:45:03 Checking availability of "local-file"
2019/07/12 02:45:03 Fetching user-data from datasource of type "local-file"
Current version of CoreOS Container Linux stable is 2135.5.0
Downloading the signature for https://stable.release.core-os.net/amd64-usr/2135.
5.0/coreos_production_image.bin.bz2...
2019-07-12 02:45:05 URL:https://stable.release.core-os.net/amd64-usr/2135.5.0/co
reos_production_image.bin.bz2.sig [566/566] -> "/tmp/coreos-install.0rTZ1FqwYI/c
oreos_production_image.bin.bz2.sig" [1]
Downloading, writing and verifying coreos_production_image.bin.bz2...
_
```

Esta acción tarda unos minutos. Al finalizar retiramos la imagen de la unidad de almacenamiento y reiniciamos la VM.

```
gpg: key 50E0885593D2DCB4 marked as ultimately trusted
gpg: checking the trustdb
gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
gpg: Good signature from "CoreOS Buildbot (Offical Builds) <buildbot@coreos.com>
" [ultimate]
[ 3094.799066] GPT:Primary header thinks Alt. header is not at the end of the di
sk.
[ 3094.799541] GPT:9289727 != 16777215
[ 3094.799797] GPT:Alternate GPT header not at the end of the disk.
[ 3094.800346] GPT:9289727 != 16777215
[ 3094.800712] GPT: Use GNU Parted to correct GPT errors.
[ 3094.801182]  sda: sda1 sda2 sda3 sda4 sda6 sda7 sda9
[ 3095.031103] GPT:Primary header thinks Alt. header is not at the end of the di
sk.
[ 3095.031856] GPT:9289727 != 16777215
[ 3095.032229] GPT:Alternate GPT header not at the end of the disk.
[ 3095.032744] GPT:9289727 != 16777215
[ 3095.033115] GPT: Use GNU Parted to correct GPT errors.
[ 3095.033594]  sda: sda1 sda2 sda3 sda4 sda6 sda7 sda9
[ 3095.769750] EXT4-fs (sda9): mounted filesystem with ordered data mode. Opts:
(null)
Installing cloud-config...
Success! CoreOS Container Linux stable 2135.5.0 is installed on /dev/sda
core@localhost ~ $ _
```

```
sudo reboot
```

Al volver la consola ya vemos que cambió el hostname.

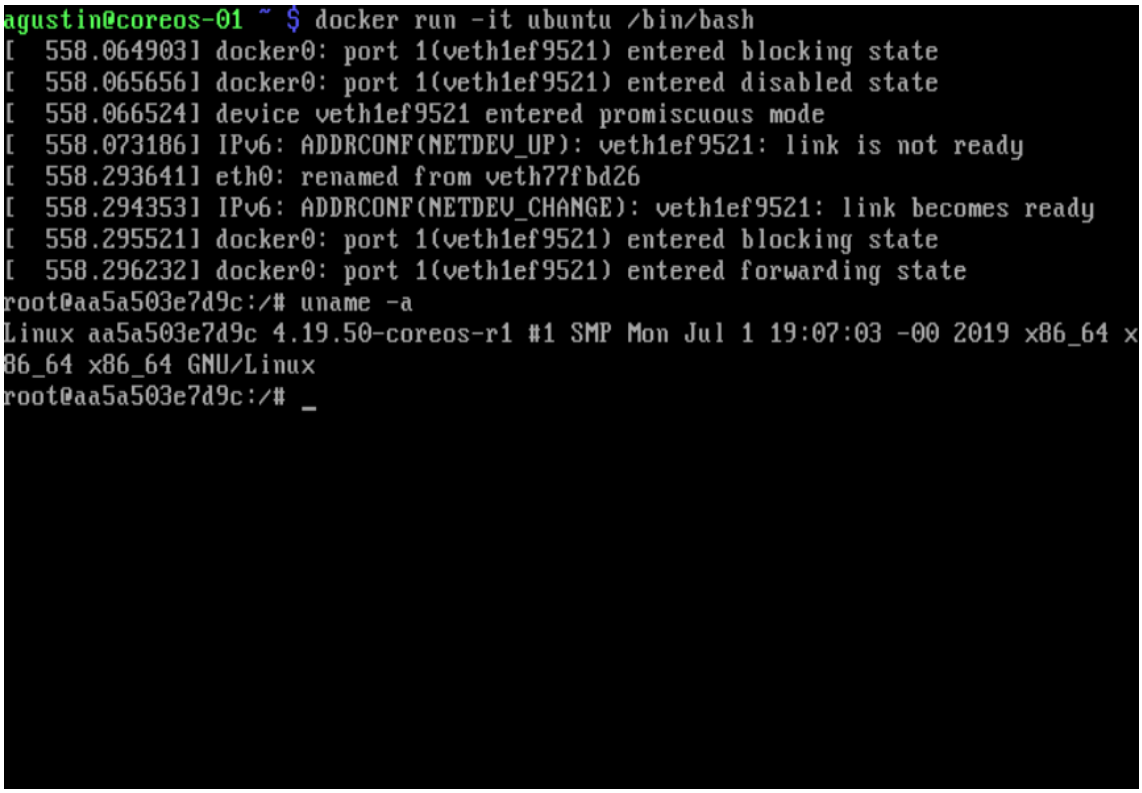


Nos logueamos con las credenciales **agustin/test** y verificamos permisos y grupos del usuario.

```
agustin@coreos-01 ~ $ whoami
agustin
agustin@coreos-01 ~ $ id
uid=1000(agustin) gid=1000(agustin) groups=1000(agustin),150(sudo),233(docker) c
ontext=system_u:system_r:kernel_t:s0
agustin@coreos-01 ~ $ pwd
/home/agustin
agustin@coreos-01 ~ $
```

Prueba de funcionalidad con un container de Ubuntu.

```
agustin@coreos-01 ~ $ docker pull ubuntu
[  251.605054] bridge: filtering via arp/ip/ip6tables is no longer available by
default. Update your scripts to load br_netfilter if you need this.
[  251.610056] Bridge firewalling registered
[  251.754526] Initializing XFRM netlink socket
[  251.840821] IPv6: ADDRCONF(NETDEV_UP): docker0: link is not ready
Using default tag: latest
latest: Pulling from library/ubuntu
5b7339215d1d: Pull complete
14ca88e9f672: Pull complete
a31c3b1caad4: Pull complete
b054a26005b7: Pull complete
Digest: sha256:9b1702dcfe32c873a770a32cfd306dd7fc1c4fd134adfb783db68defc8894b3c
Status: Downloaded newer image for ubuntu:latest
agustin@coreos-01 ~ $ _
```


agustin@coreos-01 ~ $ docker run -it ubuntu /bin/bash
[558.064903] docker0: port 1(veth1ef9521) entered blocking state
[558.065656] docker0: port 1(veth1ef9521) entered disabled state
[558.066524] device veth1ef9521 entered promiscuous mode
[558.073186] IPv6: ADDRCONF(NETDEV_UP): veth1ef9521: link is not ready
[558.293641] eth0: renamed from veth77fbd26
[558.294353] IPv6: ADDRCONF(NETDEV_CHANGE): veth1ef9521: link becomes ready
[558.295521] docker0: port 1(veth1ef9521) entered blocking state
[558.296232] docker0: port 1(veth1ef9521) entered forwarding state
root@aa5a503e7d9c:/# uname -a
Linux aa5a503e7d9c 4.19.50-coreos-r1 #1 SMP Mon Jul 1 19:07:03 -00 2019 x86_64 x
86_64 x86_64 GNU/Linux
root@aa5a503e7d9c:/# _

2.4. PROCEDIMIENTO AUTOMATIZADO

Otra forma, automatizada, de instalar coreOS supone la utilización de una herramienta denominada Vagrant (https://www.vagrantup.com/).

Se trata de un orquestador que facilita el deployment de, tanto una VM con coreOS, como de un cluster.

En nuestro ejemplo, haremos la misma implementación del procedimiento manual.

2.4.1. GIT

Necesitaremos utilizar la aplicación de Git para Windows.

https://github.com/git-for-windows/git/releases/download/v2.22.0.windows.1/Git-2.22.0-64-bit.exe

2.4.2. VAGRANT

Descargamos[4] e instalamos Vagrant en nuestro host Windows.

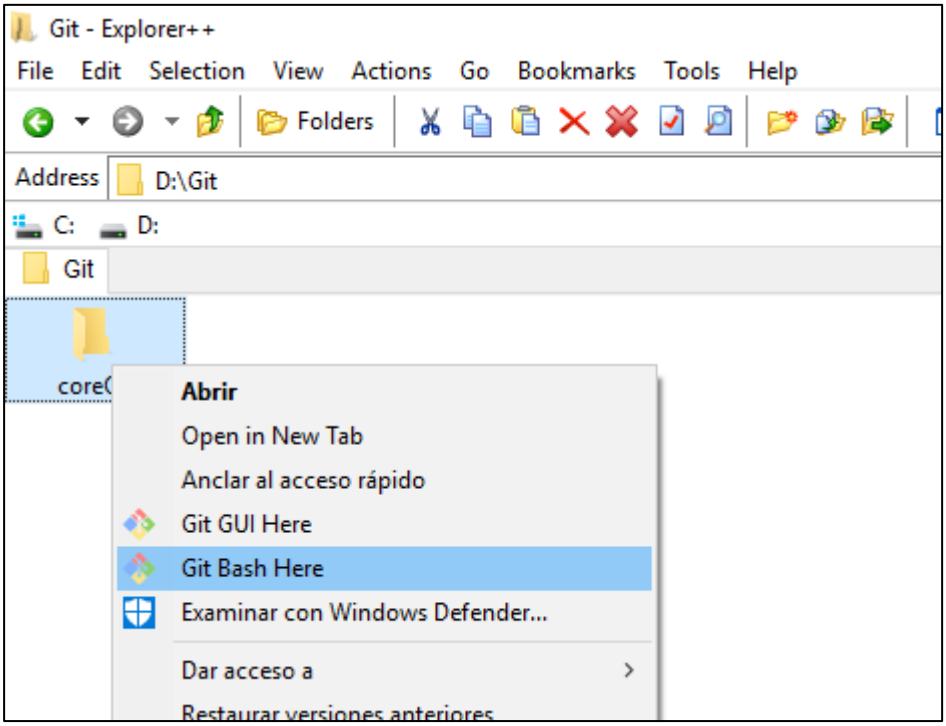
https://releases.hashicorp.com/vagrant/2.2.5/vagrant_2.2.5_x86_64.msi

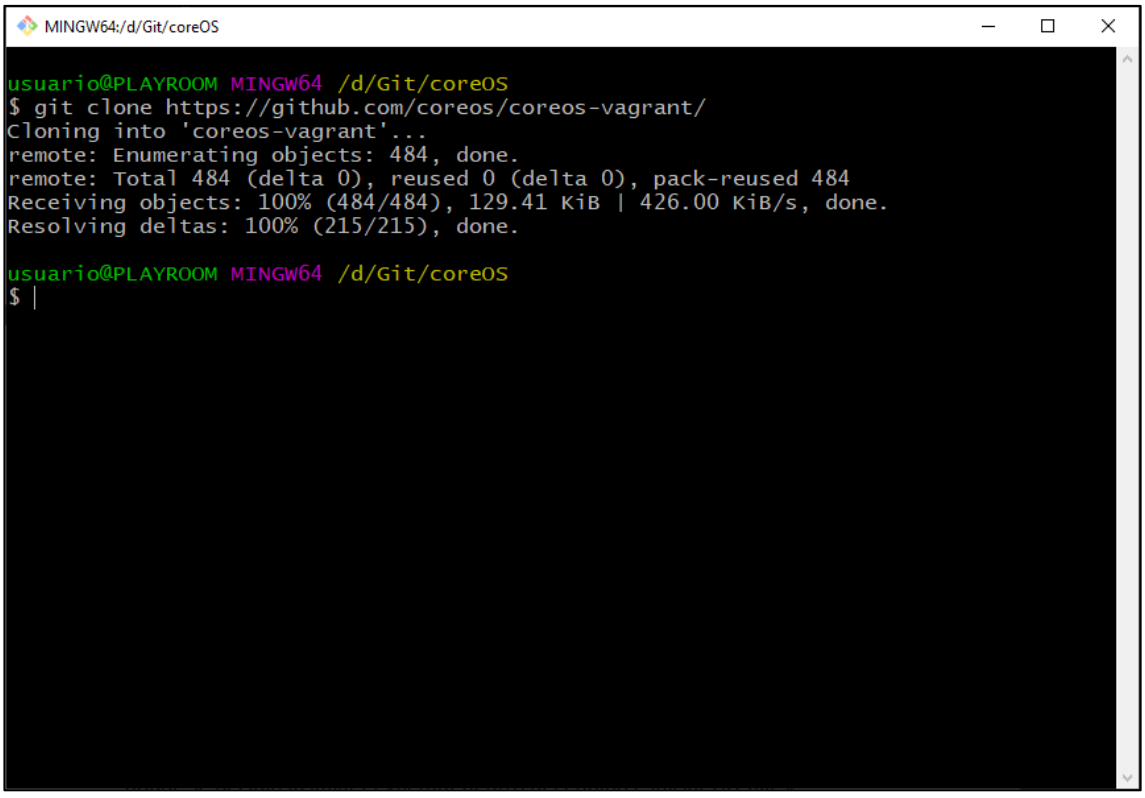
2.4.3. CLONE

El próximo paso consiste en clonar el repositorio de Vagrant para coreOS. Seleccionamos una carpeta, la abrimos con Git bash y clonamos el repositorio.

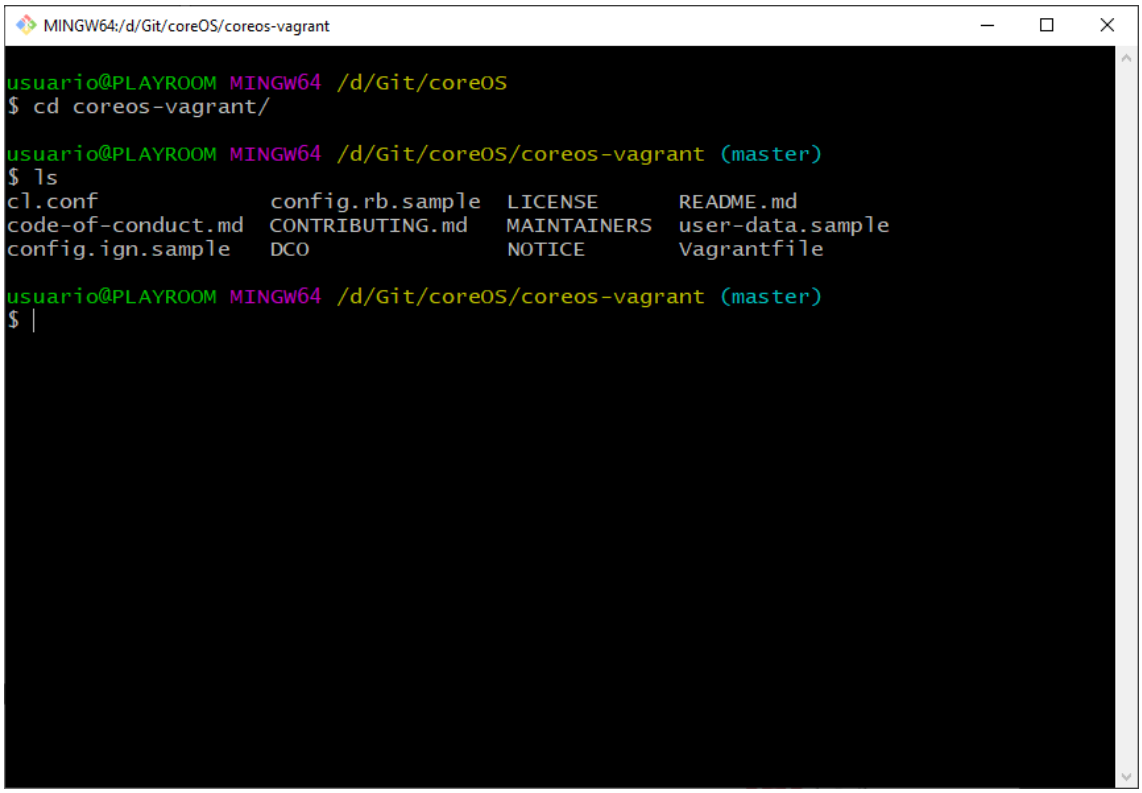
```
git clone https://github.com/coreos/coreos-vagrant/
```

[4] https://www.vagrantup.com/downloads.html





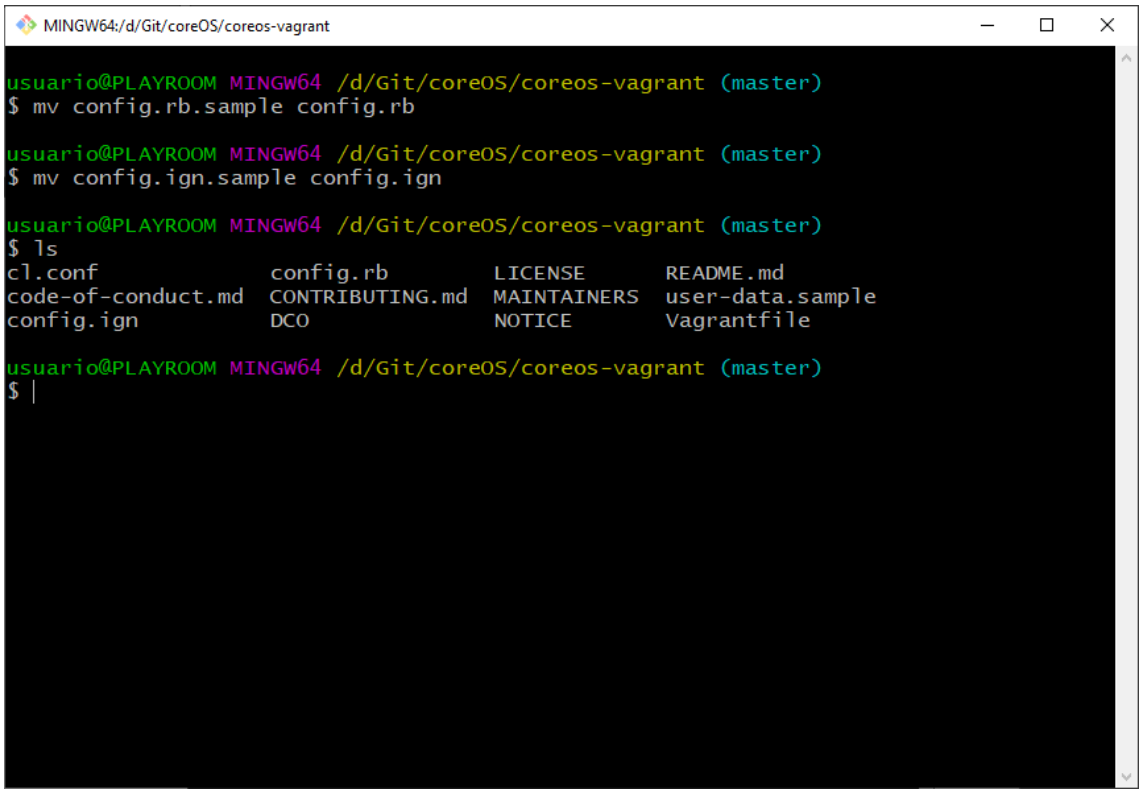
Verificamos la carpeta y archivos clonados.



Los archivos que utiliza Vagrant para el setup de la VM son: config.ign y config.rb. Al clonar el repositorio aparecen unas versiones .sample. Las renombraremos ara obtener los archivos requeridos.

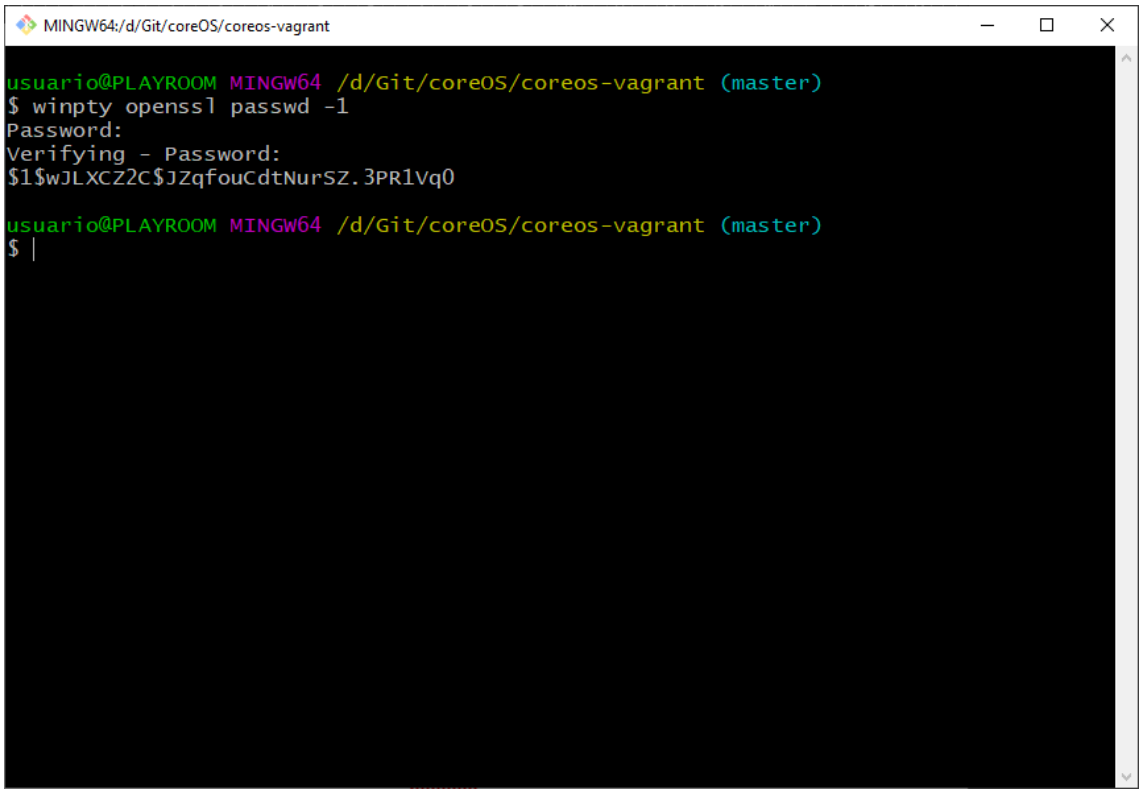
En este procedimiento, al tulizar Vagrant, automáticamente estamos pasando a una instalación mediante ignition.

Los archivos deberían verse de la siguiente forma.



Ahora bien, ignition utiliza formato JSON. Editamos las opciones para generar el usuario agustin/test.

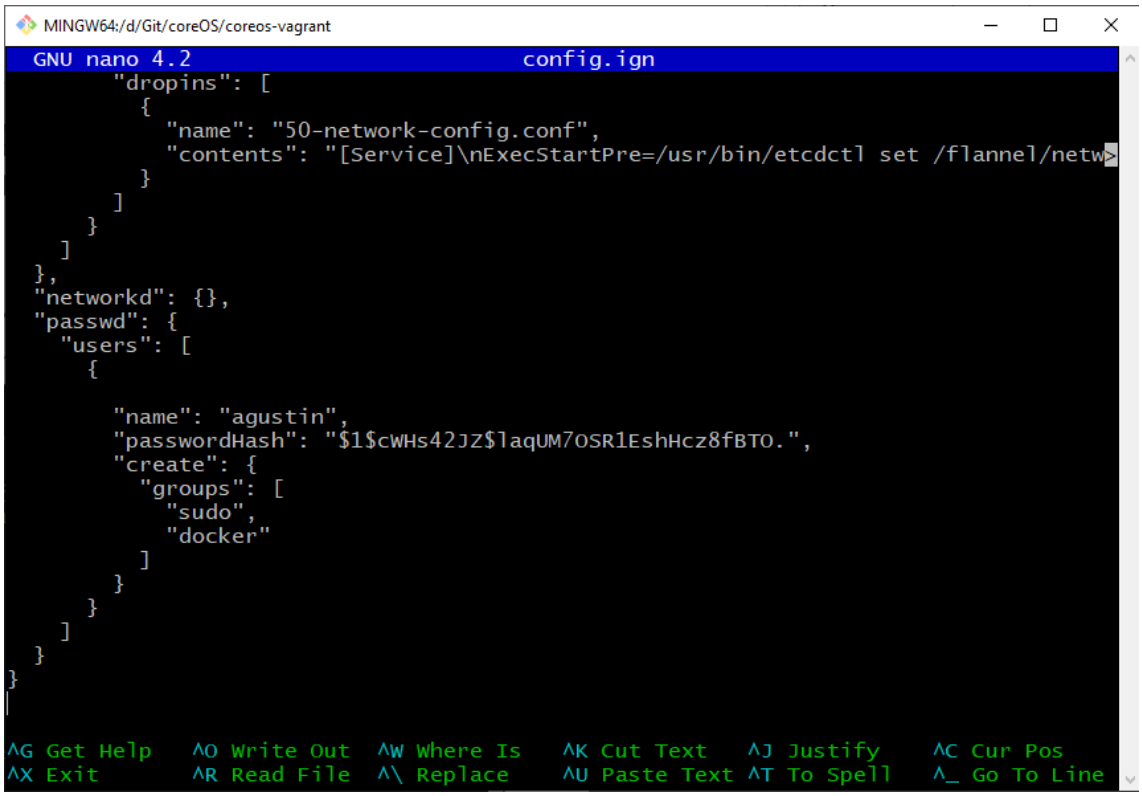
En Git bash, generamos el hash de la password.



Editamos el archi config.ign. Desde Git bash lo podemos hacer con nano.

En la línea passwd colocamos la información del usuario, mientras que el resto podemos mantenerlo como viene por defecto.

```
"passwd": {
    "users": [
      {
        "name": "agustin",
        "passwordHash": "$1$cWHs42JZ$laqUM7OSR1EshHcz8fBTO.",
        "create": {
          "groups": [
            "sudo",
            "docker"
          ]
        }
      }
    ]
  }
```



Como mencionamos anteriormente, el otro archivo importante para coreOS en Vagrant es **config.rb**. Allí podemos editar el hostname que se le impone a las VM.

El formato que usa es core-{index}, donde {index} es la instancia de la VM porque, recordemos, está pensado para el deployment de más de una VM a la vez. En nuestro caso, si no lo editamos, nos otorgará el hostname `core-01`.

Otra opción por modificar es la release que utilizará: tenemos que setear **stable**.

MINGW64:/d/Git/coreOS/coreos-vagrant

```
usuario@PLAYROOM MINGW64 /d/Git/coreOS/coreos-vagrant (master)
$ cat config.rb
# Size of the CoreOS cluster created by Vagrant
$num_instances=1

# Official CoreOS channel from which updates should be downloaded
$update_channel='stable'

# Used to fetch a new discovery token for a cluster of size $num_instances
$new_discovery_url="https://discovery.etcd.io/new?size=#{$num_instances}"

# Automatically replace the discovery token on 'vagrant up'

if File.exists?('user-data') && ARGV[0].eql?('up')
  require 'open-uri'
  require 'yaml'

  token = open($new_discovery_url).read

  data = YAML.load(IO.readlines('user-data')[1..-1].join)

  if data.key? 'coreos' and data['coreos'].key? 'etcd'
    data['coreos']['etcd']['discovery'] = token
  end

  if data.key? 'coreos' and data['coreos'].key? 'etcd2'
    data['coreos']['etcd2']['discovery'] = token
  end

  # Fix for YAML.load() converting reboot-strategy from 'off' to `false`
```

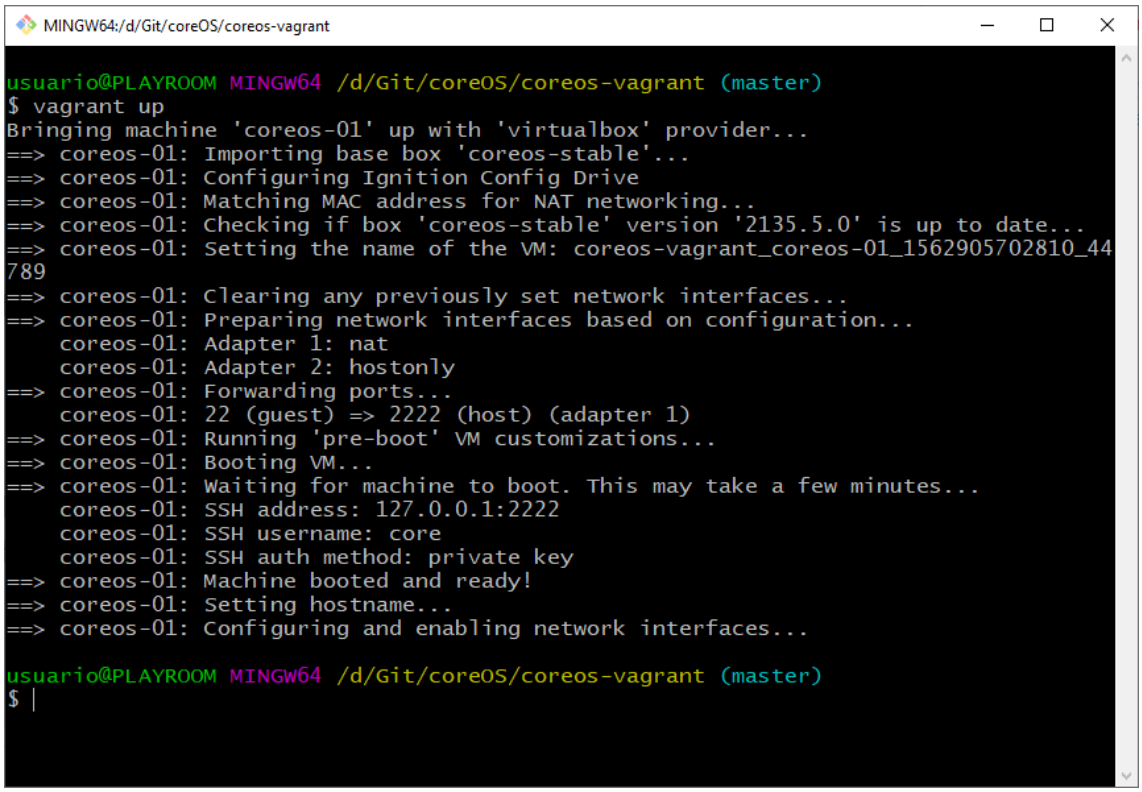
MINGW64:/d/Git/coreOS/coreos-vagrant

```
# options (global ruby variables) which are detailed below. To modify
# these options, first copy this file to "config.rb". Then simply
# uncomment the necessary lines, leaving the $, and replace everything
# after the equals sign..

# Change basename of the VM
# The default value is "core", which results in VMs named starting with
# "core-01" through to "core-${num_instances}".
$instance_name_prefix="coreos"

# Log the serial consoles of CoreOS VMs to log/
# Enable by setting value to true, disable with false
# WARNING: Serial logging is known to result in extremely high CPU usage with
# VirtualBox, so should only be used in debugging situations
#$enable_serial_logging=false

# Enable port forwarding of Docker TCP socket
# Set to the TCP port you want exposed on the *host* machine, default is 2375
# If 2375 is used, Vagrant will auto-increment (e.g. in the case of $num_instances >
 1)
# You can then use the docker tool locally by setting the following env var:
#   export DOCKER_HOST='tcp://127.0.0.1:2375'
#$expose_docker_tcp=2375

# Enable NFS sharing of your home directory ($HOME) to CoreOS
# It will be mounted at the same path in the VM as on the host.
# Example: /Users/foobar -> /Users/foobar
#$share_home=false

# Customize VMs
#$vm_gui = false
```

2.4.4. INSTALACIÓN

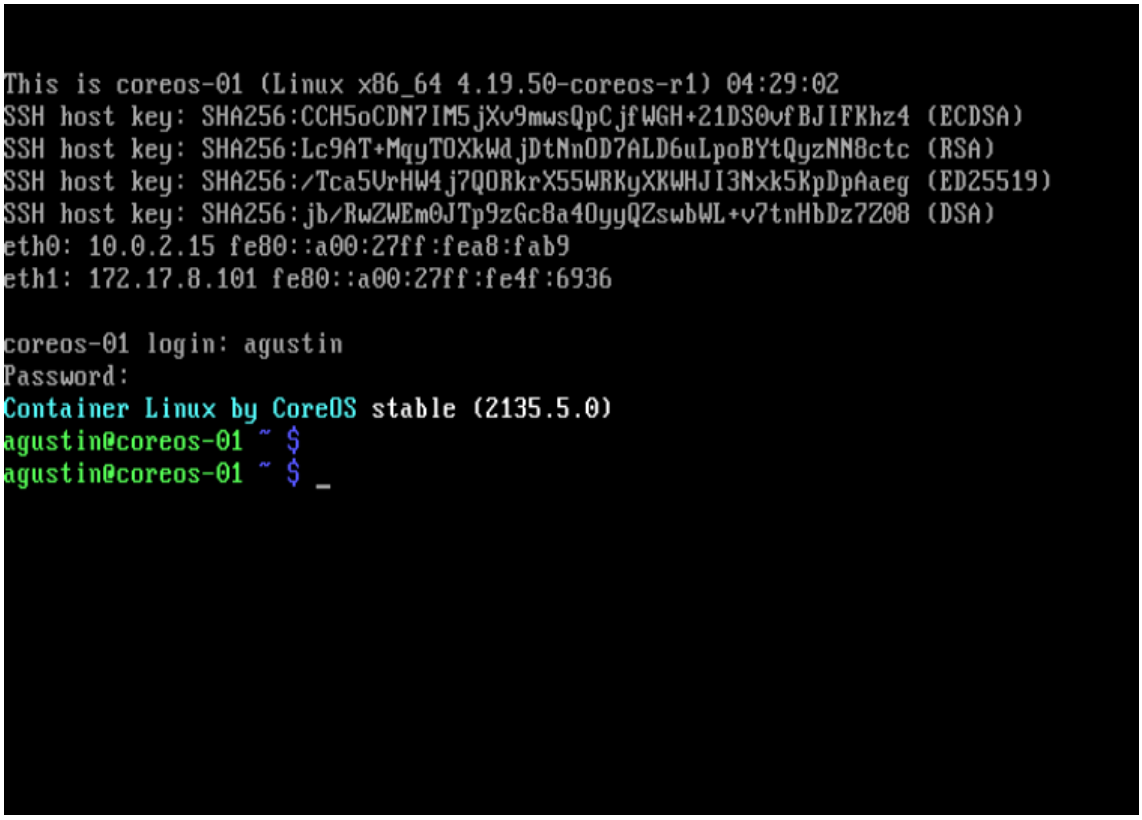
Una vez que tenemos todos los archivos configurados, desde Git bash ejecutamos Vagrant.

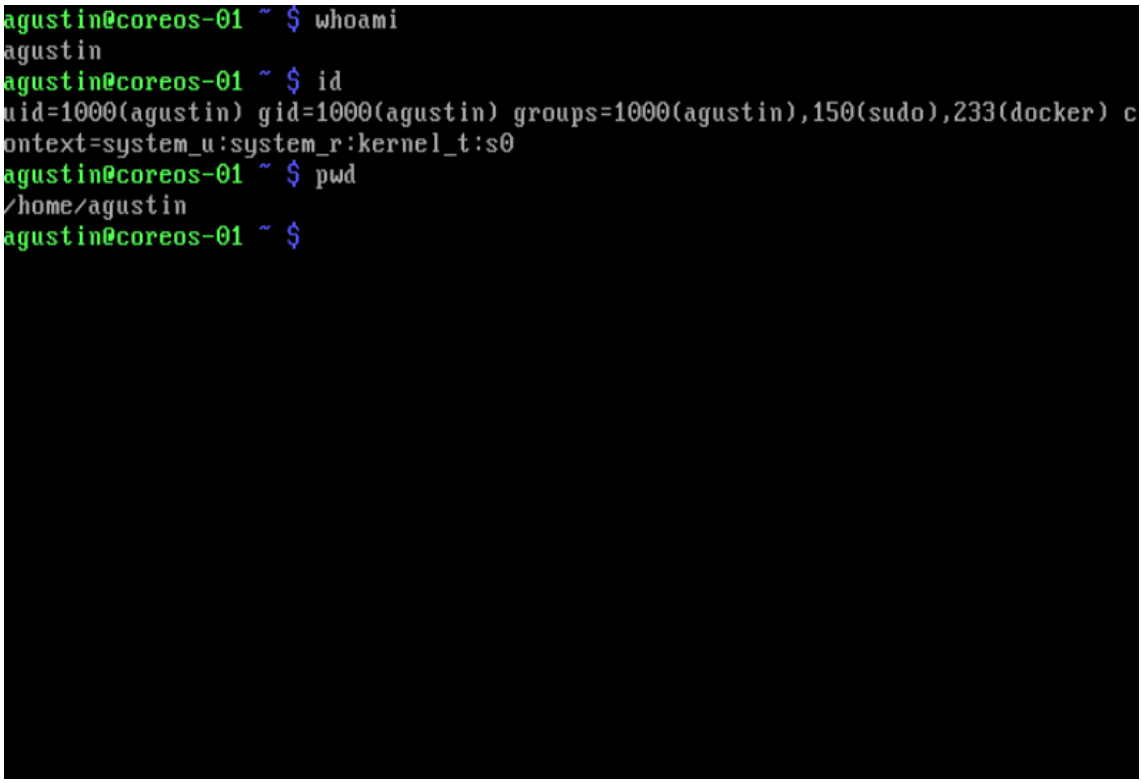
```
vagrant up
```



Podemos apreciar que existe una nueva VM en VirtualBox.

Abrimos la consola para el login con las credenciales configuradas por ignition así como la configuración del usuario.





Efectivamente está configurado como la primer parte.

2.5. ERRORES POSIBLES EN LA INSTALACIÓN

En windows es requerido si o si la instalacion del powerShell 3.0 .

https://www.microsoft.com/en-us/download/confirmation.aspx?id=34595&6B49FDFB-8E5B-4B07-BC31-15695C5A2143=1

Es necesario editar el archivo config.ign ya que de no hacerlo se logea sin usuario y no tendremos acceso al sistema operativo., ya que nos pide un usuario que no sabemos su contraseña. Además, el formato JSON no es muy intuitivo por lo que se utilizó un validador de configuración.

https://coreos.com/validate/