Proyecto: despliegue de Kubernetes sobre rkt

Francisco Jarana Camacho

Índice

1. Introducción

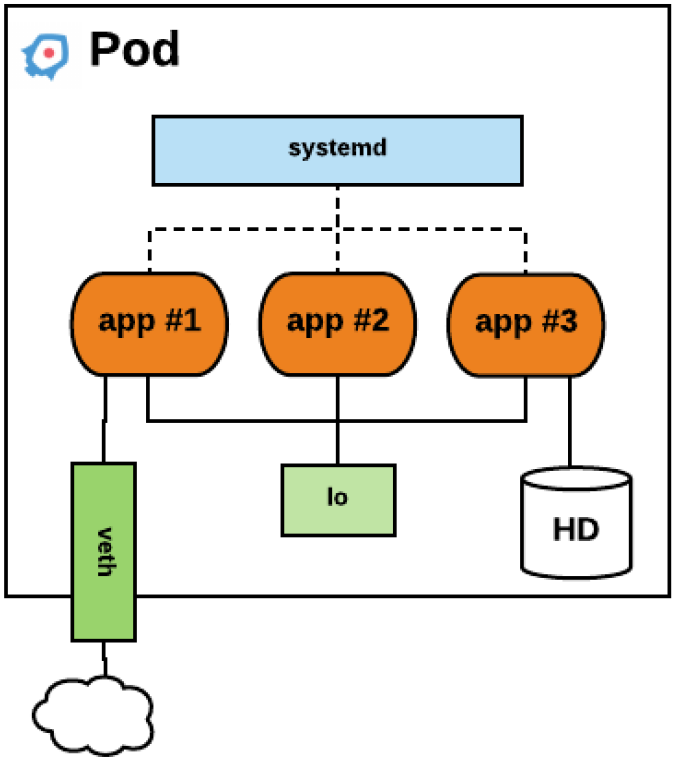
Los contenedores son una buena opción para ejecutar aplicaciones aisladas en un entorno determinado, sin necesidad de recurrir a virtualizar una máquina completa.

Vamos a hablar ahora de rkt (se pronuncia como “rock-it”), un gestor de contenedores que descarga las imágenes de las aplicaciones de internet, las verifica y las ejecuta dentro de un contenedor, el cual es un proceso aislado del resto dentro de la propia máquina.

El funcionamiento de rkt se basa en el uso de pods (concepto popularizado por Kubernetes), que son un grupo de uno o más contenedores que comparten los recursos. También se crean y se eliminan a la vez.

Rkt es una implementación de appc (App Container spec, una especie de “estándar” para la creación de contenedores) y utiliza el formato ACI (Application Contain Image) que viene definido por el appc. Este formato es un simple tarball comprimido de un directorio (rootfs) que contiene todos los ficheros necesarios para ejecutar una aplicación y otro directorio (manifest) el cual define el contenido del programa y cómo ejecutarlo.

Resumiendo, un pod es una agrupación de una o más imágenes de contenedores con metadatos adicionales y opcionales.



2. Comparación rkt - docker

Comenzaremos diciendo que, en comparación con Docker, rkt no trae consigo ningún demonio "iniciable"; esto implica que puedes lanzar contenedores directamente mediante comandos sin necesidad de depender de una API, haciéndolo compatible con systemd. Cuando lanzas un contenedor rkt, se ejecutará directamente en el proceso con el cual lo iniciaste.

Como dato adicional, decir que rkt puede lanzar contenedores docker.

3. Seguridad con rkt

Por defecto, rkt verifica la firma de las imágenes y la integridad de los datos de la misma. También restringe sus capacidades y aplica la filosofía unix de la separación de tareas (cada programa realiza únicamente la tarea para la cuál fue creado). No obstante, en ocasiones será necesario bajar los principios de seguridad de rkt.

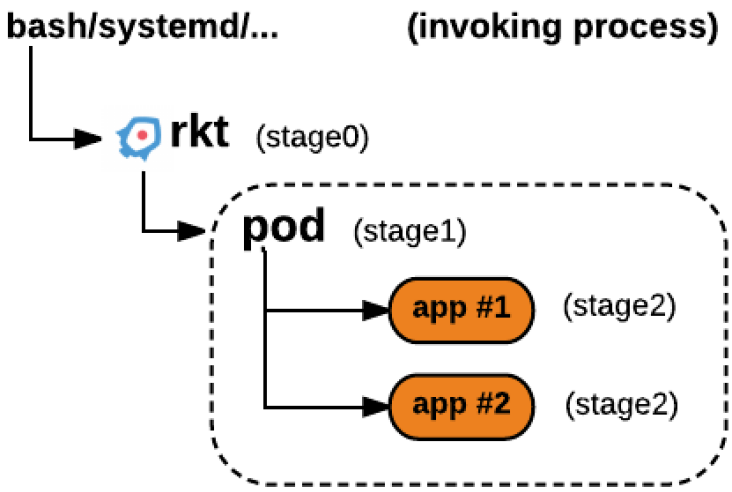
4. Escenarios de rkt

Rkt se crea entorno a una arquitectura basada en "escenarios" (stages), el "stage 0" es el comando rkt en sí mismo, el cual invoca al "stage1" el cual ejecuta el exec().

A través del "stage 1" se puede obtener la verdadera aplicación, la cual sería "stage 2".

Se puede llamar a rkt a través de la línea de comandos (bash), por systemd, por kubelet, etc.

El mecanismo por defecto del "stage 1" es muy similar a los cgroups y los namespaces de Linux; también es parecido a la forma en la que Docker utiliza los contenedores, haciendo uso de systemd-nspawn para lanzar contenedores. Otra forma válida de "stage 1" es LKVM, el cual usa una máquina virtual ligera que puede levantarse en milisegundos.



5. Instalando rkt

Existe un paquete compilado para debian por CoreOS de una versión bastante reciente (la 1.25), que es la que vamos a descargar siguiendo estos sencillos pasos:

- Obtenemos la clave pública para la verificación del paquete.

gpg --recv-key 18AD5014C99EF7E3BA5F6CE950BDD3E0FC8A365E

- Nos descargamos el paquete .deb junto con el certificador.

wget https://github.com/coreos/rkt/releases/download/v1.25.0/rkt_1.25.0-1_amd64.deb
wget https://github.com/coreos/rkt/releases/download/v1.25.0/rkt_1.25.0-1_amd64.deb.asc

- Comprobamos el fichero.

gpg --verify rkt_1.25.0-1_amd64.deb.asc

- Instalamos rkt con dpkg -i

sudo dpkg -i rkt_1.25.0-1_amd64.deb

```
enrickk@alatreon:~/Descargas$ wget https://github.com/coreos/rkt/releases/download/v1.25.0/rkt_1.25.0-1_amd64.deb.asc
--2017-12-03 19:30:32--  https://github.com/coreos/rkt/releases/download/v1.25.0/rkt_1.25.0-1_amd64.deb.asc
Resolviendo github.com (github.com)... 192.30.253.112, 192.30.253.113
Conectando con github.com (github.com)[192.30.253.112]:443... conectado.
Petición HTTP enviada, esperando respuesta... 301 Moved Permanently
Localización: https://github.com/rkt/rkt/releases/download/v1.25.0/rkt_1.25.0-1_amd64.deb.asc [siguiendo]
--2017-12-03 19:30:33--  https://github.com/rkt/rkt/releases/download/v1.25.0/rkt_1.25.0-1_amd64.deb.asc
Reutilizando la conexión con github.com:443.
Petición HTTP enviada, esperando respuesta... 302 Found
Localización: https://github-production-release-asset-2e65be.s3.amazonaws.com/26509369/c9361588-f78c-11e6-9cf9-2148dbaf0b56?X-Amz-Algorithm=AWS4-HMAC-
SHA256&X-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2F20171203%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20171203T183033Z&X-Amz-Expires=300&X-Amz-Signature=
45f17a721a19f22562ca35a09338e96528983ebc31172f9fabe087b29ec2efe3&X-Amz-SignedHeaders=host&actor_id=0&response-content-disposition=attachment%3B%20file
name%3Drkt_1.25.0-1_amd64.deb.asc&response-content-type=application%2Foctet-stream [siguiendo]
--2017-12-03 19:30:33--  https://github-production-release-asset-2e65be.s3.amazonaws.com/26509369/c9361588-f78c-11e6-9cf9-2148dbaf0b56?X-Amz-Algorithm
=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2F20171203%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20171203T183033Z&X-Amz-Expires=300&X-Amz
-Signature=45f17a721a19f22562ca35a09338e96528983ebc31172f9fabe087b29ec2efe3&X-Amz-SignedHeaders=host&actor_id=0&response-content-disposition=attachmen
t%3B%20filename%3Drkt_1.25.0-1_amd64.deb.asc&response-content-type=application%2Foctet-stream
Resolviendo github-production-release-asset-2e65be.s3.amazonaws.com (github-production-release-asset-2e65be.s3.amazonaws.com)... 54.231.48.248
Conectando con github-production-release-asset-2e65be.s3.amazonaws.com (github-production-release-asset-2e65be.s3.amazonaws.com)[54.231.48.248]:443...
 conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: 488 [application/octet-stream]
Grabando a: “rkt_1.25.0-1_amd64.deb.asc”

rkt_1.25.0-1_amd64.deb.asc            100%[=====================================================================>]     488  --.-KB/s    in 0s

2017-12-03 19:30:34 (5,22 MB/s) - “rkt_1.25.0-1_amd64.deb.asc” guardado [488/488]
```

```
enrickk@alatreon:~$ cd Descargas/
enrickk@alatreon:~/Descargas$ gpg --recv-key 18AD5014C99EF7E3BA5F6CE950BDD3E0FC8A365E
gpg: key 50BDD3E0FC8A365E: "CoreOS Application Signing Key <security@coreos.com>" not changed
gpg: Total number processed: 1
gpg:              unchanged: 1
enrickk@alatreon:~/Descargas$ wget https://github.com/coreos/rkt/releases/download/v1.25.0/rkt_1.25.0-1_amd64.deb
--2017-12-03 19:29:00--  https://github.com/coreos/rkt/releases/download/v1.25.0/rkt_1.25.0-1_amd64.deb
Resolviendo github.com (github.com)... 192.30.253.113, 192.30.253.112
Conectando con github.com (github.com)[192.30.253.113]:443... conectado.
Petición HTTP enviada, esperando respuesta... 301 Moved Permanently
Localización: https://github.com/rkt/rkt/releases/download/v1.25.0/rkt_1.25.0-1_amd64.deb [siguiendo]
--2017-12-03 19:29:01--  https://github.com/rkt/rkt/releases/download/v1.25.0/rkt_1.25.0-1_amd64.deb
Reutilizando la conexión con github.com:443.
Petición HTTP enviada, esperando respuesta... 302 Found
Localización: https://github-production-release-asset-2e65be.s3.amazonaws.com/26509369/c9231aaa-f78c-11e6-8158-b1b459ee9b5e?X-Amz-Algorithm=AWS4-HMAC-
SHA256&X-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2F20171203%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20171203T182901Z&X-Amz-Expires=300&X-Amz-Signature=
b65bb4df55005a7671b1663a6ea859c5204072a5ee6206f886cf3a7c43801df9&X-Amz-SignedHeaders=host&actor_id=0&response-content-disposition=attachment%3B%20file
name%3Drkt_1.25.0-1_amd64.deb&response-content-type=application%2Foctet-stream [siguiendo]
--2017-12-03 19:29:01--  https://github-production-release-asset-2e65be.s3.amazonaws.com/26509369/c9231aaa-f78c-11e6-8158-b1b459ee9b5e?X-Amz-Algorithm
=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2F20171203%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20171203T182901Z&X-Amz-Expires=300&X-Amz
-Signature=b65bb4df55005a7671b1663a6ea859c5204072a5ee6206f886cf3a7c43801df9&X-Amz-SignedHeaders=host&actor_id=0&response-content-disposition=attachmen
t%3B%20filename%3Drkt_1.25.0-1_amd64.deb&response-content-type=application%2Foctet-stream
Resolviendo github-production-release-asset-2e65be.s3.amazonaws.com (github-production-release-asset-2e65be.s3.amazonaws.com)... 52.216.128.115
Conectando con github-production-release-asset-2e65be.s3.amazonaws.com (github-production-release-asset-2e65be.s3.amazonaws.com)[52.216.128.115]:443..
. conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: 104335568 (100M) [application/octet-stream]
Grabando a: “rkt_1.25.0-1_amd64.deb”

rkt_1.25.0-1_amd64.deb                100%[=====================================================================>]  99,50M  3,59MB/s    in 35s

2017-12-03 19:29:37 (2,81 MB/s) - “rkt_1.25.0-1_amd64.deb” guardado [104335568/104335568]
```

```
enrickk@alatreon:~/Descargas$ gpg --verify rkt_1.25.0-1_amd64.deb.asc
gpg: assuming signed data in 'rkt_1.25.0-1_amd64.deb'
gpg: Signature made lun 20 feb 2017 17:47:44 CET
gpg:                using RSA key 5B1053CE38EA2E0FEB956C0595BC5E3F3F1B2C87
gpg: Good signature from "CoreOS Application Signing Key <security@coreos.com>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 18AD 5014 C99E F7E3 BA5F  6CE9 50BD D3E0 FC8A 365E
     Subkey fingerprint: 5B10 53CE 38EA 2E0F EB95  6C05 95BC 5E3F 3F1B 2C87
```

Este paquete requiere las siguientes dependencias para poder instalarse: adduser, dbus, libc6, systemd, iptables.

Además, creará en el directorio /usr/lib/rkt/stage1-images, donde contendrá los ficheros .aci, que serán los diferentes motores para levantar los contenedores. También creará un enlace al binario de rkt en el directorio /usr/bin, añadirá la documentación a man, y creará los grupos de usuarios rkt y rkt-admin.

6. Ejecutando rkt

Vamos ahora a ejecutar un primer contenedor de ejemplo (siguiendo las instrucciones de la página oficial); el comando siguiente busca un contenedor en el repositorio de quay.io, verifica su firma, lo inicia y nos introduce dentro del mismo.

```
enrickk@alatreon:~$ rkt run quay.io/coreos/alpine-sh --interactive
image: using image from local store for image name coreos.com/rkt/stage1-coreos:
1.13.0
image: using image from local store for image name quay.io/coreos/alpine-sh
networking: loading networks from /etc/rkt/net.d
networking: loading network default with type ptp
/ #
```

También podemos lanzar contenedores docker directamente desde repositorio, pero en este caso no podremos verificar el contenedor, por lo que tendremos que añadir la opción –insecure-options;

```
enrickk@alatreon:~$ rkt run docker://alpine --insecure-options=image --exec=/bin
/sh
image: using image from local store for image name coreos.com/rkt/stage1-coreos:
1.13.0
image: remote fetching from URL "docker://alpine"
Downloading sha256:2fdfe1cd78c [==============================] 2.06 MB / 2.06 MB
networking: loading networks from /etc/rkt/net.d
networking: loading network default with type ptp
```

7. Networking con rkt

Rkt utiliza los pliguns de CNI (interfaz de red para contenedores) para crear las redes de los contenedores. La manera más sencilla de conectar un pod rkt sería compartiendo la red del host con él, lo cual se puede hacer añadiendo el argumento –net=host. El pod adquirirá el namespace de la red en el proceso que está invocando rkt.

Sin este argumento, por defecto, rkt creará un par de interfaces ethernet, una en el lado del host y otra en el pod. El pod tendrá una IP privada del rango 172.16.28.0 y usará la interfaz del host como puerta de enlace.

Por defecto, solo se permiten 253 pods por host. Si no se quiere que los pods tengan acceso al exterior pero sí entre ellos, utilizando el argumento –net=default-restricted o cualquier otro nombre que no sea “host”, “none” o “default”. Se pueden añadir múltiples redes separándolas por comas.

Por último, se puede aislar un pod de cualquier otra red con –net=none, así el pod se creará únicamente con la interfaz de loopback.

Si se quiere que el pod pueda resolver nombres a través de un DNS, hay que añadir el argumento –dns 8.8.8.8 (o el servidor que se quiera utilizar).

También se puede definir el puerto en el que se quiera que escuche el contenedor, declarando después de la IP con el argumento –port.

8. Kubernetes

Google comenzó el proyecto de Kubernetes en 2014 como una plataforma de código abierto diseñada para automatizar despliegues, escalados y aplicaciones sobre contenedores.

Kubernetes define un conjunto de bloques de construcción que, conjuntamente, proveen los mecanismos para el despliegue, mantenimiento y escalado de aplicaciones. Los componentes que forman Kubernetes están diseñados para estar débilmente acoplados pero al mismo tiempo ser extensibles para que puedan soportar una gran variedad de flujos de trabajo. La extensibilidad es provista en gran parte por la API de Kubernetes, que es utilizada por componentes internos así como extensiones y contenedores ejecutándose sobre Kubernetes.

- **Pods**

Como hemos hablado anteriormente, Kubernetes utiliza “pods” para sus despliegues. Estos agregan un nivel de abstracción más elevado a los componentes en contenedores. Cada pod en Kubernetes es asignado a una única dirección IP dentro del clúster que permite a las aplicaciones utilizar puertos sin riesgos de conflictos. Un pod puede definir un volumen y exponerlo a los contenedores dentro del pod.

- **Etiquetas y selectores**

Kubernetes permite a los clientes vincular pares clave-valor llamados etiquetas (labels) a cualquier objeto API en el sistema, como pods o nodos. Las etiquetas y los selectores son el mecanismo principal de agrupamiento en Kubernetes, y son utilizados para determinar los componentes sobre los cuales aplica una operación.

- **Controladores**

Un controlador es un bucle de reconciliación que lleva al estado real del clúster hacia el estado deseado. Hace esto mediante la administración de un conjunto de pods. Un tipo de controlador es un “Replication Controller”, que se encarga de la replicación y escala mediante la ejecución de un número especificado de copias de un pod a través de un clúster. También se encarga de crear pods de reemplazo si un nodo subyacente falla. Otros controladores que forman parte del sistema central de Kubernetes incluye al “DaemonSet Controller” para la ejecución de exactamente un pod en cada máquina, y un “Job Controller” para ejecutar pods que ejecutan hasta su finalización. El conjunto de pods que un controlador administra está determinado por los selectores de etiquetas que forman parte de la definición del controlador.

- **Servicios**

Un servicio Kubernetes es un conjunto de pods que trabajan al unísono , como una capa más de una aplicación multicapas. El conjunto de pods que constituyen un servicio está definido por el selector de etiquetas. Kubernetes provee de un servicio de descubrimiento y enrutamiento de pedidos mediante la asignación de una dirección IP estable y un nombre DNS al servicio, y balancea la carga de tráfico en un estilo round-robin hacia las conexiones de red de las direcciones IP entre los pods que verifican el selector. Por defecto un servicio es expuesto dentro de un clúster, pero un servicio puede ser expuesto también hacia afuera del mismo.

9. Arquitectura de Kubernetes

Kubernetes sigue una arquitectura maestro-esclavo. Los componentes de Kubernetes pueden ser divididos en aquellos que administran un nodo individual y aquellos que son partes de un panel de control.

- **Etcd**

Un servidor etcd es un almacén de datos persistente, liviano, distribuido de clave-valor desarrollado por CoreOS que almacena de manera confiable los datos de configuración de un clúster, representando el estado general del clúster en un momento específico. Otros componentes escuchan por cambios en este almacén para avanzar al estado deseado.

- **Servidor de API**

El servidor API es un componente central y sirve a la API de kubernetes utilizando json sobre HTTP, que proveen la interfaz interna y externa de Kubernetes. El servidor API procesa y valida las peticiones REST y actualiza el estado de los objetos API en etcd, lo que permite a los clientes configurar los flujos de trabajo y contenedores a través de los nodos esclavos.

- **Planificador**

El planificador es el componente enchufable que selecciona sobre qué nodo un pod sin planificar deberá correr basado en la disponibilidad de recursos. Para este propósito, el planificador debe conocer los requerimientos de recursos, la disponibilidad de los mismos y una variedad de restricciones y políticas directivas como quality-of-service (QoS), requerimiento de afinidad, localización de datos entre otros.

- **Administrador del controlador**

El administrador de controlador es el proceso sobre el cual el núcleo de los controladores Kubernetes como DaemonSet y Replication se ejecuta. Los controladores se comunican con el servidor API para crear, actualizar y eliminar recursos que ellos manejan, como pods, servicios, etc.

- **Nodo Kubernetes**

El nodo, también conocido como esclavo o worker, es la máquina (física o virtual) donde los contenedores (flujos de trabajos) son desplegados. Cada nodo en el clúster debe ejecutar la rutina de tiempo de ejecución, así como también los componentes mencionados más abajo, para comunicarse con el maestro para la configuración en red de estos contenedores.

- **Kubelet**

Kubelet es responsable por el estado de ejecución de cada nodo (se asegura de que todos los contenedores del nodo se encuentran saludables). Se encarga del inicio, la detención y el mantenimiento de los contenedores de aplicaciones (organizados como pods) como es indicado por el panel de control.

Kubelet monitorea el estado de un pod y, si no se encuentra en el estado deseado, el pod será desplegado nuevamente al mismo nodo

- **Kube-Proxy**

Kube-Proxy es la implementación de un proxy de red y balanceador de carga soportando la abstracción del servicio jutno con otras operaciones de red. Es responsable del enrutamiento del tráfico hacia el contenedor basado en la dirección IP y el número de puerto indicados en el pedido.

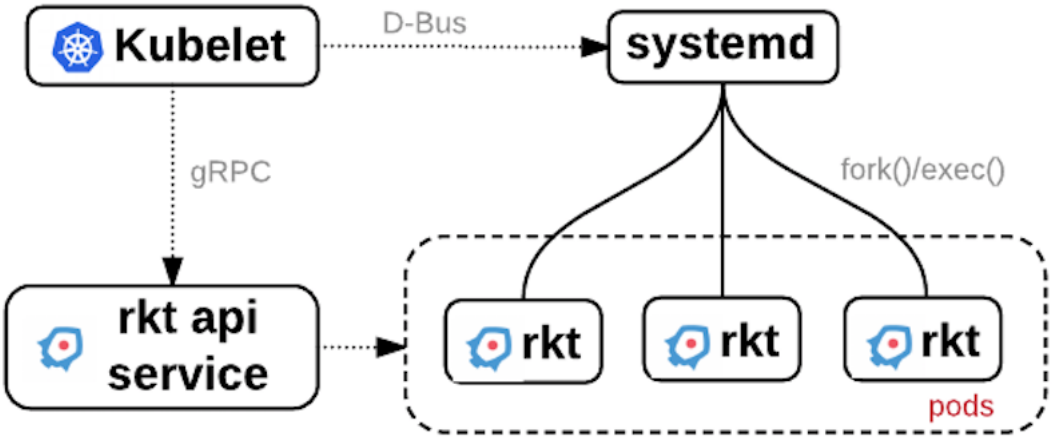
- **cAdvisor**

cAdvisor es un agente que monitorea y recoge métricas de utilización de recursos y rendimiento como CPU, memoria, uso de archivos y red de los contenedores en cada nodo.

10. Rktnetes (rkt + kubernetes)

Como hemos mencionado anteriormente, rkt es una alternativa a Docker para usar contenedores; también lo es cuando queremos usar una alternativa al demonio de Docker para Kubernetes. En lugar de esto, es Kubelet el demonio que se ejecuta en todos los nodos del cluster Kubernetes, delegándole a rkt todas las operaciones relacionadas con los contenedores. Rkt se encarga de descargar, comprobar y ejecutar las imágenes mientras que Kubelet se comunica con systemd para ejecutar los pods via rkt.

El stage1 usado cuando se lanza simplemente usa el syscall de chroot y no establece ningún namespace, por lo que Kubelet puede acceder a los ficheros del sistema y a la configuración de la red necesarias para ejecutar el contenedor dentro del cluster de Kubernetes. Estamos hablando ya de un proceso del sistema y no de un simple pod.



Para desplegar nuestro escenario, primero tenemos que instalar kubectl:

```
# curl -Lo kubectl https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl && chmod +x kubectl && sudo mv ./kubectl /usr/local/bin/
```

Seguidamente, descargamos el binario de minikube:

```
# curl -Lo minikube https://storage.googleapis.com/minikube/releases/v0.16.0/minikube-linux-amd64 && chmod +x minikube && sudo mv minikube /usr/local/bin/
```

Indicamos a minikube que utilice el driver de kvm para los contenedores:

```
enrickk@alatreon:~$ sudo minikube config set vm-driver kvm
These changes will take effect upon a minikube delete and then a minikube start
```

Levantamos el cluster de kubernetes con minikube:

```
enrickk@alatreon:~$ sudo minikube start --network-plugin=cni --container-runtime=rkt
Starting local Kubernetes cluster...
Kubectl is now configured to use the cluster.
```

Levantamos un servicio de ejemplo:

```
enrickk@alatreon:~$ sudo kubectl run webserver --image=nginx  --replicas=1 --port=80
deployment "webserver" created
```

Esperamos un tiempo a que se levante el servicio y comprobamos:

```
enrickk@alatreon:~$
enrickk@alatreon:~$ sudo kubectl get pods,deployments
NAME                            READY     STATUS    RESTARTS   AGE
po/webserver-1505803560-62hct   1/1       Running   0          4m

NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
deploy/webserver   1         1         1            1           4m
```

Accedemos al servicio:

```
enrickk@alatreon:~$ sudo curl $(sudo minikube service webserver --url)
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Por último, verificamos que efectivamente está usando rkt (si nos aparecen varias líneas, buscamos si existe alguna “running”, que es la que hemos creado):

A continuación mostraremos un ejemplo de creación de volúmenes; primero vamos a crear una carpeta “prueba” dentro del directorio /tmp y dentro de esta un fichero index.html con un “Hello world”. El siguiente paso es crear un volumen persistente; Kubernetes soporta rutas del propio host para el desarrollo y prueba de clústeres mononodos. Una ruta del host del volumen persistente usa un fichero o directorio del nodo para emular un almacenamiento asociado mediante una red. Vamos a crear un fichero task-pv-volume.yaml con el siguiente contenido:

```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: task-pv-volume
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/tmp/prueba"
```

Esta configuración especifica que el volumen se encuentra en /tmp/prueba dentro del nodo del clúster. La configuración especifica también un tamaño de 10 GB y permisos de lectura y escritura. Creamos el volumen persistente:

```
enrickk@alatreon:~/Plantillas$ sudo kubectl apply -f task-pv-volume.yaml --validate=false
persistentvolume "task-pv-volume" created
```

Comprobamos el estado del volumen:

```
enrickk@alatreon:~/Plantillas$ sudo kubectl get pv task-pv-volume
NAME             CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM     STORAGECLASS   REASON    AGE
task-pv-volume   10Gi       RWO            Retain           Available                                      5m
```

El estatus “Available” indica que todavía no ha sido enlazado a ningún dispositivo que esté usando un espacio físico del disco (PersistentVolumeClaim). Por lo tanto nuestro próximo paso es solicitar una parte del espacio físico del nodo para poder tener acceso de lectura-escritura en el mismo. Para ello creamos otra plantilla llamada task-pv-claim.yaml con el siguiente contenido:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: task-pv-claim
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
```

```
    requests:
      storage: 3Gi
```

El siguiente paso es crear la solicitud a partir del fichero:

```
enrickk@alatreon:~/Plantillas$ sudo kubectl create -f task-pv-claim.yaml --validate=false
persistentvolumeclaim "task-pv-claim" created
```

Ahora Kubernetes buscará un volumen persistente que satisfaga los requisitos de la solicitud. Si lo encuentra, se enlazará a este. Podemos comprobarlo de la siguiente forma:

```
enrickk@alatreon:~/Plantillas$ sudo kubectl get pv task-pv-volume
NAME             CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS    CLAIM                   STORAGECLASS   REASON    AGE
task-pv-volume   10Gi       RWO            Retain           Bound     default/task-pv-claim                            1h
```

Como podemos observar, el estado ha cambiado a “Bound”, lo que significa que se ha enlazado exitosamente. También podemos comprobar el estado de la solicitud realizada:

```
enrickk@alatreon:~/Plantillas$ sudo kubectl get pvc task-pv-claim
NAME            STATUS    VOLUME           CAPACITY   ACCESS MODES   STORAGECLASS   AGE
task-pv-claim   Bound     task-pv-volume   10Gi       RWO                           13m
```

Ahora vamos a crear un pod que haga uso del volumen que acabamos de crear. Utilizaremos para ello el siguiente fichero de configuración (fichero task-pv-pod.yaml):

```
kind: Pod
apiVersion: v1
metadata:
  name: task-pv-pod
spec:
  volumes:
    - name: task-pv-storage
      persistentVolumeClaim:
      claimName: task-pv-claim
  containers:
    - name: task-pv-container
      image: nginx
      ports:
        - containerPort: 8090
          name: "http-server"
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: task-pv-storage
```

Si observamos bien, vemos que el fichero de configuración especifica la solicitud del volumen persistente, y no el volumen en sí; esto es porque desde el punto de vista del pod, la solicitud es un volumen en sí. Ahora creamos el pod a partir del fichero:

```
enrickk@alatreon:~/Plantillas$ sudo kubectl create -f task-pv-pod.yaml --validate=false
pod "task-pv-pod" created
enrickk@alatreon:~/Plantillas$
```

Esperamos a que se despliegue y comprobamos:

```
enrickk@alatreon:~/Plantillas$ sudo kubectl get pod task-pv-pod
NAME          READY     STATUS    RESTARTS   AGE
task-pv-pod   1/1       Running   0          16m
```

Entramos dentro del contenedor:

```
enrickk@alatreon:~/Plantillas$ sudo kubectl exec -it task-pv-pod -- /bin/bash
```

Dentro de la shell, instalamos curl para verificar que nginx está sirviendo el fichero index.html de la ruta del host:

```
root@task-pv-pod:/# apt-get update
```

```
root@task-pv-pod:/# apt-get install curl
```

```
root@task-pv-pod:/# curl localhost
Hello world
```

11. Conclusión

Como hemos podido comprobar, rkt no sólo es una alternativa viable frente a Docker, sino que además puede utilizarse sin problemas con Kubernetes sin ninguna limitación en cuanto a funcionalidades.

Además; actualmente ya se encuentra desarrollado containerd; una nueva herramienta utilizada por Docker para dejar de depender de la tecnología de Kubernetes; por lo que a lo mejor, en un futuro, sea más recomendable utilizar un gestor de contenedores diferente a Docker para crear escenarios con Kubernetes.