Comenzar, Parte 5: Pilas

Tiempo estimado de lectura: 9 minutos

Requisitos previos

- Instale Docker versión 1.13 o superior (https://docs.docker.com/engine/installation/) .
- Obtenga Docker Compose (https://docs.docker.com/compose/overview/) como se describe en los requisitos previos de la Parte 3 (https://docs.docker.com/get-started/part3/#prerequisites) .
- Obtenga Docker Machine (https://docs.docker.com/machine/overview/) como se describe en los requisitos previos de la Parte 4 (https://docs.docker.com/get-started/part4/#prerequisites) .
- Lea la orientación en la Parte 1 (https://docs.docker.com/get-started/) .
- Aprenda a crear contenedores en la Parte 2 (https://docs.docker.com/get-started/part2/) .
- Asegúrese de haber publicado la `friendlyhello` imagen que creó empujándola a un registro (https://docs.docker.com/get-started/part2/#share-your-image) . Usaremos esa imagen compartida aquí.
- Asegúrese de que su imagen funciona como un contenedor implementado. Ejecutar este comando, ranurado en su información de `username` , `repo` y `tag` : `docker run -p 80:80 username/repo:tag` , entonces visita `http://localhost/` .
- Tener una copia de la `docker-compose.yml` de la parte 3 (https://docs.docker.com/get-started/part3/) práctico.
- Asegúrese de que las máquinas configuradas en la parte 4 (https://docs.docker.com/get-started/part4/) están funcionando y listas. Ejecutar `docker-machine ls` para verificar esto. Si las máquinas se detienen, ejecute `docker-machine start myvm1` para arrancar el gestor, seguido por `docker-machine start myvm2` arrancar el trabajador.
- Haga que el enjambre que creó en la parte 4 se (https://docs.docker.com/get-started/part4/) ejecute y esté listo. Ejecutar `docker-machine ssh myvm1 "docker node ls"` para verificar esto. Si el enjambre está arriba, ambos nodos reportarán un `ready` estado. Si no, reinicialice el enjambre y únase al trabajador como se describe en Configurar su enjambre (https://docs.docker.com/get-started/part4/#set-up-your-swarm) .

Introducción

En la parte 4 (https://docs.docker.com/get-started/part4/) , aprendió a configurar un enjambre, que es un grupo de máquinas que ejecutan Docker, e implementó una aplicación en él, con contenedores ejecutándose en concierto en varias máquinas.

Aquí en la parte 5, llegará a la parte superior de la jerarquía de aplicaciones distribuidas: la **pila** . Una pila es un grupo de servicios interrelacionados que comparten dependencias, y pueden ser orquestados y escalados juntos. Una sola pila es capaz de definir y coordinar la funcionalidad de una aplicación completa (aunque las aplicaciones muy complejas pueden querer usar pilas múltiples).

Una buena noticia es que técnicamente ha estado trabajando con pilas desde la parte 3, cuando creó un archivo de Compose y lo utilizó `docker stack deploy` . Pero eso era una pila de servicio único que se ejecuta en un único host, que no suele ser lo que ocurre en la producción. Aquí, usted tomará lo que ha aprendido, hará que los servicios múltiples se relacionen entre sí y los ejecute en varias máquinas.

Lo estás haciendo muy bien, este es el tramo de casa!

Agregar un nuevo servicio y reasignar

Es fácil agregar servicios a nuestro `docker-compose.yml` archivo. En primer lugar, vamos a agregar un servicio de visualizador gratuito que nos permite ver cómo nuestro enjambre está programando los contenedores.

1. Abrir `docker-compose.yml` en un editor y reemplazar su contenido con lo siguiente. Asegúrese de reemplazar `username/repo:tag` con los detalles de su imagen.

```
version: "3"
services:
  web:
    # replace username/repo:tag with your name and image details
    image: username/repo:tag
    deploy:
      replicas: 5
      restart_policy:
        condition: on-failure
      resources:
        limits:
          cpus: "0.1"
          memory: 50M
    ports:
      - "80:80"
    networks:
      - webnet
  visualizer:
    image: dockersamples/visualizer:stable
    ports:
      - "8080:8080"
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock"
    deploy:
      placement:
        constraints: [node.role == manager]
    networks:
      - webnet
networks:
  webnet:
```

Lo único nuevo aquí es el servicio de pares `web` , llamado `visualizer` . Verá dos cosas nuevas aquí: una `volumes` clave, que le da al visualizador acceso al archivo de socket del host para Docker, y una `placement` clave, asegurando que este servicio sólo se ejecuta en un administrador de enjambre - nunca un trabajador. Esto se debe a que este contenedor, construido a partir de un proyecto de código abierto creado por Docker (https://github.com/ManoMarks/docker-swarm-visualizer) , muestra los servicios de Docker que se ejecutan en un enjambre en un diagrama.

Hablaremos más sobre las restricciones y los volúmenes de colocación en un momento.

2. Copie este `docker-compose.yml` archivo nuevo en el gestor de enjambre `myvm1` :

```
docker-machine scp docker-compose.yml myvm1:~
```

3. Vuelva a ejecutar el `docker stack deploy` comando en el administrador y los servicios que necesite actualizar se actualizarán:

```
$ docker-machine ssh myvm1 "docker stack deploy -c docker-compose.yml getstartedlab"
Updating service getstartedlab_web (id: angi1bf5e4to03qu9f93trnxm)
Updating service getstartedlab_visualizer (id: l9mnwkeq2jiononb5ihz9u7a4)
```

4. Echa un vistazo al visualizador.

Se vio en el archivo de composición que se `visualizer` ejecuta en el puerto 8080. Obtenga la dirección IP de uno de sus nodos ejecutándose `docker-machine ls` . Vaya a cualquiera de las direcciones IP en el puerto 8080 y verá el visualizador en ejecución:

La copia única `visualizer` se ejecuta en el gestor como se espera, y las cinco instancias de `web` se distribuyen a través del enjambre. Puede corroborar esta visualización ejecutando `docker stack ps <stack>` :

```
docker-machine ssh myvm1 "docker stack ps getstartedlab"
```

El visualizador es un servicio autónomo que puede ejecutarse en cualquier aplicación que lo incluya en la pila. No depende de nada más. Ahora vamos a crear un servicio que *no* tiene una dependencia: el servicio Redis que proporcionará un contador de visitas.

Persiste los datos

Vamos a pasar por el mismo flujo de trabajo una vez más para agregar una base de datos Redis para almacenar datos de la aplicación.

1. Guarde este nuevo `docker-compose.yml` archivo, que finalmente agrega un servicio Redis. Asegúrese de reemplazar `username/repo:tag` con los detalles de su imagen.

```
version: "3"
services:
  web:
    # replace username/repo:tag with your name and image details
    image: username/repo:tag
    deploy:
      replicas: 5
      restart_policy:
        condition: on-failure
      resources:
        limits:
          cpus: "0.1"
          memory: 50M
    ports:
      - "80:80"
    networks:
      - webnet
  visualizer:
    image: dockersamples/visualizer:stable
    ports:
      - "8080:8080"
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock"
    deploy:
      placement:
        constraints: [node.role == manager]
    networks:
      - webnet
  redis:
    image: redis
    ports:
      - "6379:6379"
    volumes:
      - ./data:/data
    deploy:
      placement:
        constraints: [node.role == manager]
    networks:
      - webnet
networks:
  webnet:
```

Redis tiene una imagen oficial en la biblioteca de Docker y se le ha concedido el `image` nombre corto de just `redis` , así que no hay `username/repo` notación aquí. El puerto Redis, 6379, ha sido preconfigurado por Redis para ser expuesto desde el contenedor al host, y aquí en nuestro archivo de Compose lo exponemos del host al mundo, para que pueda ingresar el IP para cualquiera de sus Nodos en Redis Desktop Manager y administrar esta instancia Redis, si así lo desea.

Lo más importante, hay un par de cosas en la `redis` especificación que hacen que los datos persisten entre las implementaciones de esta pila:

- `redis` Siempre se ejecuta en el gestor, por lo que siempre está utilizando el mismo sistema de archivos.
- `redis` Accede a un directorio arbitrario en el sistema de archivos del host como `/data` dentro del contenedor, donde Redis almacena los datos.

Juntos, esto está creando una "fuente de verdad" en el sistema de archivos físico de su anfitrión para los datos Redis. Sin esto, Redis almacenaría sus datos `/data` dentro del sistema de archivos del contenedor, que sería eliminado si ese contenedor se reasignaba.

Esta fuente de la verdad tiene dos componentes:

- La restricción de ubicación que coloca en el servicio Redis, asegurándose de que siempre utiliza el mismo host.
- El volumen que creó que permite que el contenedor acceda `./data` (en el host) como `/data` (dentro del contenedor Redis). Mientras los contenedores vienen y van, los archivos almacenados `./data` en el host especificado persistirán, lo que permitirá la continuidad.

Está listo para implementar su nueva pila Redis-using.

2. Cree un `./data` directorio en el administrador:

```
$ docker-machine ssh myvm1 "mkdir ./data"
```

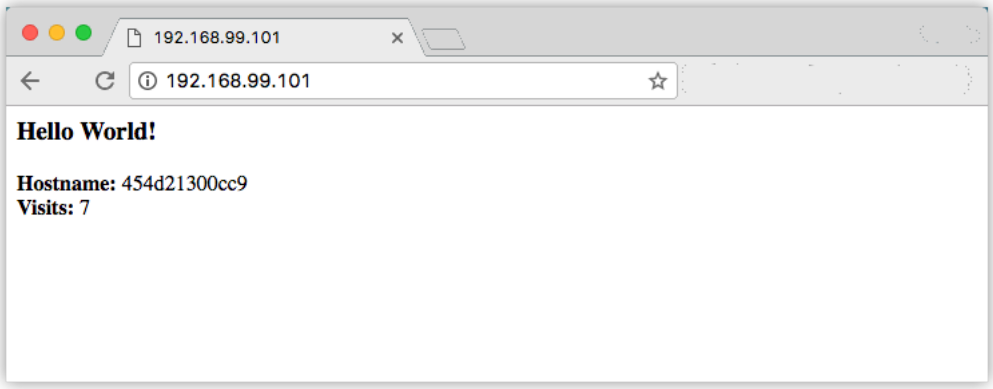
3. Copie el nuevo `docker-compose.yml` archivo con `docker-machine scp` :

```
$ docker-machine scp docker-compose.yml myvm1:~
```

4. Corre `docker stack deploy` una vez más.

```
$ docker-machine ssh myvm1 "docker stack deploy -c docker-compose.yml getstartedlab"
```

5. Compruebe la página web en uno de sus nodos (por ejemplo `http://192.168.99.101`) y verá los resultados del contador de visitantes, que ahora está en vivo y almacenando información sobre Redis.



Además, compruebe el visualizador en el puerto 8080 en la dirección IP de cada nodo y verá el `redis` servicio ejecutándose junto con los servicios `web` y `visualizer` .

En la parte 6 » (https://docs.docker.com/get-started/part6/)

Recapitulación (opcional)

Esta es una grabación terminal de lo que estaba cubierto en esta página (https://asciinema.org/a/113840) :

Powered by asciinema

Aprendiste que las pilas son servicios interrelacionados, todo funcionando en concierto, y que - sorpresa! - has estado usando pilas desde la tercera parte de este tutorial. Aprendiste que para agregar más servicios a tu stack, los insertas en tu archivo de Compose. Por último, aprendió que al utilizar una combinación de restricciones de ubicación y volúmenes, puede crear un alojamiento permanente para datos persistentes, de modo que los datos de su aplicación sobreviva cuando el contenedor se desmantela y se reubica.

Pila (https://docs.docker.com/glossary/?term=stack) , datos (https://docs.docker.com/glossary/?term=data) , persistir (https://docs.docker.com/glossary/?term=persist) , dependencias (https://docs.docker.com/glossary/?term=dependencies) , redis (https://docs.docker.com/glossary/?term=redis) , almacenamiento (https://docs.docker.com/glossary/?term=storage) , volumen (https://docs.docker.com/glossary/?term=volume) , puerto (https://docs.docker.com/glossary/?term=port)

Califica esta página:
118 11