Developing Container Applications with VMware vSphere Integrated Containers Engine

vSphere Integrated Containers Engine 0.5.5

vmware

Table of Contents

Developing Container Applications with vSphere Integrated Containers Engine

Developing Container Applications with vSphere Integrated Containers Engine provides information about how to use vSphere Integrated Containers Engine as the endpoint for Docker container application development.

Product version: 0.5.5

NOTE This book is a work in progress.

Intended Audience

This information is intended for container application developers who's Docker environment uses vSphere Integrated Containers Engine as its endpoint. Knowledge of container technology and Docker is assumed.

VMware, Inc. 3401 Hillview Ave. Palo Alto, CA94304

www.vmware.com

Supported Docker Commands

vSphere Integrated Containers Engine supports Docker 1.11.2. The supported version of the Docker API is 1.23. If you are using a more recent version of the Docker client, see Docker Commands Fail with a Docker API Version Error.

General Container Operations

Container	Docker Reference	Supported
Docker login	Log into a Docker registry	Yes, use to access private repository
Docker start	Start a container	Yes
Docker version	Docker version information	Yes. vSphere Integrated Containers Engine version provided
Docker run	Composite command of create, start, inspect, attach, rm, resize, wait, kill	Yes. `docker run -c` and `docker run -m` parameters are supported. Container search using prettyname-ID `docker run -name` is supported. Mapping a random host port to the container when the host port is not specified is supported. Running images from private and custom registries is supported.
Docker create	Create a container	Yes
Docker inspect	Inspect a container Inspect an image	Yes
Docker port	Obtain port data	Yes. Displays port mapping data. Mapping a random host port to the container when the host port is not specified is supported.
Docker attach	Attach to a container Attach to a container websocket	Yes
Docker rm	Remove a container	Yes, only the `name` parameter is supported. `force` and `v` are a future implementation. Also removes associated volumes.
Docker info	Docker system information	Yes, docker-specific data, basic capacity information, list of configured volume stores, virtual container host information. Does not reveal vSphere datastore paths that might contain sensitive vSphere information
Docker container resize	Resize a container	Yes
Docker stop	Stop a container Stop	Yes. Powers down the VM
Docker images	Images list-images	Yes
Docker ps	Show running containers	Yes

Docker logs	Get container logs	Yes, except for the `docker logs --timestamps` (`-t`) and `--since` options, which are not supported.
Docker restart	Restart a container Restart	Yes
Docker kill	Kill a container Kill	Yes. Docker must wait for the container to shut down.
Docker Container list	List Containers	Yes
Docker pull	Pull an image or repository from a registry	Yes, pulling from insecure and custom registries is supported

Network Operations

For more information about network operations, see Network Port Use Cases.

Network	Docker Reference	Supported
Network create	Create a network	Yes. See the use case to connect to an external network in vSphere Integrated Container for vSphere Administrators. Bridge is also supported.
Network ls	List networks/	Yes
Network inspect	Inspect a network	Yes
Network connect	Connect to a network	Yes
Network rm	Remove a network	Yes. Network name and network ID are supported

Volume Operations

For more information about volume operations, see Using Volumes with vSphere Integrated Containers Engine.

Volume	Docker Reference	Supported
Docker volume create	Create a volume	The driver option is ignored even if you specify it. You must include `--opt VolumeStore= --Capacity=` as these are direct vSphere arguments. VIC does not assign random names during a volume create, but only for anonymous volumes.
Docker volume ls	List volumes	Yes
Docker volume rm	Remove a volume	No

Other Operations

Commands	Docker Reference	Supported
Link	Link	Future release
Docker export	Export a container	Future release
Docker save	Save images	Future release
Docker stats	Get container stats based on resource usage Stats	Future release
Docker update	Update a container Update	Future release
Docker rename	Rename a container Rename	Future release
Docker pause	Pause processes in a container Pause	Future release
Docker unpause	Unpause processes in a container Unpause	Future release
Docker cp	Copy files or folders in a container Copy	Future release
Wait	Wait for a container Wait	Future release

Use and Limitations of Containers in vSphere Integrated Containers Engine

vSphere Integrated Containers Engine currently includes the following capabilities and limitations:

- Container VMs only support root user.
- You can resolve the symbolic names of a container from within another container except for the following:
  - aliases
  - IPv6 support
  - service discovery
- Containers are capable of acquiring DHCP addresses if they are on a network that has DHCP.

Using Volumes with vSphere Integrated Containers Engine

vSphere Integrated Containers Engine supports the use of container volumes. When you create or the vSphere Administrator creates a virtual container host, you or the Administrator specify the datastore to use to store container volumes in the `vic-machine create --volume-store` option. For information about how to use the `vic-machine create --volume-store` option, see the section on `volume-store` in Virtual Container Host Deployment Options in *vSphere Integrated Containers Engine Installation and Configuration*.

Obtain the List of Available Volume Stores

To obtain the list of volume stores that are available on a virtual container host, run `docker info`.

```
docker -H virtual_container_host_address:2376 --tls info
```

The list of available volume stores for this virtual container host appears in the `docker info` output under `VolumeStores`.

```
[...]
Storage Driver: vSphere Integrated Containers Backend Engine
VolumeStores: volume_store_1 volume_store_2 ... volume_store_n
vSphere Integrated Containers Backend Engine: RUNNING
[...]
```

Create a Volume in a Volume Store

When you use the `docker volume create` command to create a volume, you can optionally provide a name for the volume by specifying the `--name` option. If you do not specify `--name`, `docker volume create` assigns a random UUID to the volume.

- If the volume store label is anything other than `default`, you must specify the `--opt VolumeStore` option and pass the name of an existing volume store to it. If you do not specify `--opt VolumeStore`, `docker volume create` searches for a volume store named `default`, and returns an error if no such volume store exists.

  ```
  docker -H virtual_container_host_address:2376 --tls volume create
  --opt VolumeStore=volume_store_label
  --name volume_name
  ```

- If you or the vSphere Administrator set the volume store label to `default` when running `vic-machine create`, you do not need to specify `--opt VolumeStore`.

  ```
  docker -H virtual_container_host_address:2376 --tls volume create
  --name volume_name
  ```

- If you intend to create anonymous volumes by using `docker create -v`, a volume store named `default` must exist. In this case, you include the path to the destination at which you want to mount an anonymous volume in the `docker create -v` command. Docker creates the volume in the `default` volume store, if it exists.

  ```
  docker -H virtual_container_host_address:2376 --tls create
  -v destination_path_for_anonymous_volume busybox
  ```

  NOTE: If you use `docker create -v`, vSphere Integrated Containers Engine only supports the `-r` and `-rw` options.

- You can optionally set the capacity of a volume by specifying the `--opt Capacity` option when you run `docker volume create`. If you do not specify the `--opt Capacity` option, the volume is created with the default capacity of 1024MB.

  If you do not specify a unit for the capacity, the volume is created with a capacity in megabytes.

  ```
  docker -H virtual_container_host_address:2376 --tls volume create
  --opt VolumeStore=volume_store_label
  --opt Capacity=2048
  --name volume_name
  ```

- To create a volume with a capacity in gigabytes or terabytes, include `GB`, or `TB` in the value that you pass to `--opt Capacity`. The unit is case insensitive.

  ```
  docker -H virtual_container_host_address:2376 --tls volume create
  --opt VolumeStore=volume_store_label
  --opt Capacity=10GB
  --name volume_name
  ```

NOTE: When using a vSphere Integrated Containers Engine virtual container host as your Docker endpoint, the storage driver is always the vSphere Integrated Containers Engine Backend Engine. If you specify the `docker volume create --driver` option, it is ignored.

Obtain the List of Available Volumes

To obtain a list of volumes that are available on a virtual container host, run `docker volume ls`.

```
docker -H virtual_container_host_address:2376 --tls volume ls

DRIVER          VOLUME NAME
vsphere         volume_1
vsphere         volume_2
[...]           [...]
vsphere         volume_n
```

Delete a Named Volume from a Volume Store

To delete a volume, run `docker volume rm` and specify the name of the volume to delete.

```
docker -H virtual_container_host_address:2376 --tls
volume rm volume_name
```

NOTE: In the current builds, `docker volume rm` is not yet supported.

Network Port Use Cases

These are some use cases of containers using network ports to communicate with each other.

Container with a Published Port

Launch a container and expose a port: `run -p`

Connect the container with the external mapped port on the external surface of the vSphere Container Host.

```
$ docker run -p 8080:80 --name test1 my_container my_app
```

Outcome

You can access Port 80 on test1 from the external network interface on the virtual container host at port 8080.

Simple Bridge Network

Create a new non-default bridge network and set up two containers on the network. Verify that the containers can locate and communicate with each other.

```
$ docker network create -d bridge my-bridge-network
$ docker network ls
...
NETWORK ID          NAME                DRIVER
615d565d498c        my-bridge-network   bridge
...
$ docker run -d --net=my-bridge-network \
                --name=server my_server_image server_app
$ docker run -it --name=client --net=my-bridge-network busybox
/ # ping server
PING server (172.18.0.2): 56 data bytes
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.073 ms
64 bytes from 172.18.0.2: seq=1 ttl=64 time=0.092 ms
64 bytes from 172.18.0.2: seq=2 ttl=64 time=0.088 ms
```

Outcome

Server and Client can ping each other by name.

Bridged Containers with Exposed Port

Connect two containers on a bridge network and set up one of the containers to publish a port via the virtual container host. Assume server_app binds to port 5000.

```
$ docker network create -d bridge my-bridge-network
$ docker network ls
...
NETWORK ID          NAME                DRIVER
615d565d498c        my-bridge-network   bridge
...
$ docker run -d -p 5000:5000 --net=my-bridge-network \
                --name=server my_server_image server_app
$ docker run -it --name=client --net=my-bridge-network busybox
/ # ping -c 3 server
PING server (172.18.0.2): 56 data bytes
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.073 ms
64 bytes from 172.18.0.2: seq=1 ttl=64 time=0.092 ms
64 bytes from 172.18.0.2: seq=2 ttl=64 time=0.088 ms
/ # telnet server 5000
GET /

Hello world!Connection closed by foreign host
$ telnet vch_external_interface 5000
Trying 192.168.218.137...
Connected to 192.168.218.137.
Escape character is '^]'.
GET /

Hello world!Connection closed by foreign host.
```

Outcome

Server and Client can ping each other by name. You can connect to the server on port 5000 from the client container and to port 5000 on the virtual container host external interface.

Containers using External Network

Configure two external networks in vSphere: `default-external` is `10.2.0.0/16` with gateway `10.2.0.1` `vic-production` is `208.91.3.0/24` with gateway `208.91.3.1`

Associate a virtual container host, then set up the virtual container host to the default external network.

Attach the virtual container host to the default-external network at 08.91.3.2.

`docker network ls` shows:

```
$ docker network ls
NETWORK ID          NAME                DRIVER
e2113b821ead        none                null
37470ed9992f        default-external    bridge
ea96a6b919de        vic-production      bridge
b7e91524f3e2        bridge              bridge
```

You have a container providing a web service to expose outside of the vSphere Integrated Containers Engine environment.

Output of `docker network inspect default-external` :

```
[
    {
        "Name": "default-external",
        "Id": "37470ed9992f6ab922e155d8e902ca03710574d96ffbfde1b3faf541de2a701f",
        "Scope": "external",
        "Driver": "bridge",
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "10.2.0.0/16",
                    "Gateway": "10.2.0.1"
                }
            ]
        },
        "Containers": {},
        "Options": {}
    }
]
```

Output of `docker network inspect vic-production`:

```
[
    {
        "Name": "vic-production",
        "Id": "ea96a6b919de4ca2bd627bfdf0683ca04e5a2c3360968d3c6445cb18fab6d210",
        "Scope": "external",
        "Driver": "bridge",
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "208.91.3.0/24",
                    "Gateway": "208.91.3.1"
                }
            ]
        },
        "Containers": {},
        "Options": {}
    }
]
```

Set up a server on the vic-production network:

```
$ docker run -d --expose=80 --net=vic-production --name server my_webapp
$ docker inspect --format='{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' server
208.91.3.2
$ telnet 208.91.3.2 80
Trying 208.91.3.2...
Connected to 208.91.3.2.
Escape character is '^]'.
GET /

Hello world!Connection closed by foreign host.
```

NOTE: You can also use `-p 80` or `-p 80:80` instead of `--expose=80`. If you try to map to different ports with `-p`, you get a configuration error.

Outcome

The server container port is exposed on the external network vic-production.

Docker Commands Fail with a Docker API Version Error

After a successful deployment of a vSphere Integrated Containers Engine virtual container host, attempting to run a Docker command fails with a Docker version error.

Problem

When you attempt to run a Docker command from a Docker client that is connecting to a virtual container host, the command fails with the error `Error response from daemon: client is newer than server (client API version: 1.24, server API version: 1.23)`.

Cause

vSphere Integrated Containers Engine supports Docker 1.11, that includes version 1.23 of the Docker API. You are using version 1.12 of the Docker client, that uses version 1.24 of the Docker API, which is incompatible.

Solution

1. Open a Docker client terminal.
2. Set the Docker client API to the same version as is used by vSphere Integrated Containers Engine.

   ```
   export DOCKER_API_VERSION=1.23
   ```

3. Check that your Docker client can now connect to the virtual container host by running a Docker command.

   ```
   docker -H virtual_container_host_address:2376 --tls info
   ```

   The `docker info` command should succeed and you should see information about the virtual container host.