Developing Container Applications with VMware vSphere Integrated Containers Engine

vSphere Integrated Containers Engine 0.7.0

vmware

Table of Contents

Introduction	0
Overview of vSphere Integrated Containers Engine For Container Application Developers	1
Supported Docker Commands	2
Use and Limitations of Containers in vSphere Integrated Containers Engine	3
Obtain a Virtual Container Host	4
Using Volumes with vSphere Integrated Containers Engine	5
Using Insecure Private Registry Servers with vSphere Integrated Containers Engine	6
Network Port Use Cases	7
Docker Commands Fail with a Docker API Version Error	8

Developing Container Applications with vSphere Integrated Containers Engine

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

Product version: 0.7.0

NOTE This book is a work in progress.

For an introduction to vSphere Integrated Containers Engine and descriptions of its main components, see *vSphere Integrated Containers Engine for vSphere Installation*.

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.

Copyright © 2016 VMware, Inc. All rights reserved. Copyright and trademark information. Any feedback you provide to VMware is subject to the terms at www.vmware.com/community_terms.html.

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

www.vmware.com

Overview of vSphere Integrated Containers Engine For Container Application Developers

vSphere Integrated Containers Engine is a Docker interface for containers with a vSphere back-end. As a container developer, you can deploy, test, and run container processes in the same environment as you would normally perform container operations.

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 attach	Attach to a container Attach to a container websocket	Yes
Docker container list	List Containers	Yes
Docker container resize	Resize a container	Yes
Docker create	Create a container	Yes. `--cpuset-cpus` in Docker specifies CPUs the container is allowed to use during execution (0-3, 0,1). In vSphere Integrated Containers, this parameter specifies the number of virtual CPUs to allocate to the container VM. Minimum CPU count is 1, maximum is unlimited. Default is 2. `--ip` allows you to set a static IP on the container. By default, the virtual container host manages the container IP. `-m --memory` Minimum memory is 512MB, maximum unlimited. If unspecified, default is 2GB.
Docker images	Images list-images	Yes
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 inspect	Inspect a container Inspect an image	Yes. Includes information about the container network.
Docker kill	Kill a container Kill	Yes. Docker must wait for the container to shut down.
Docker login	Log into a Docker registry	Yes, use to access private repository
Docker logs	Get container logs	Yes, except for the `docker logs --timestamps` (`-t`) and `--since` options, which are not supported.
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 ps	Show running containers	Yes
Docker pull	Pull an image or repository from a registry	Yes, pulling from insecure and custom registries is supported
Docker restart	Restart a container Restart	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 rmi	Remove a Docker image	Yes
Docker run	Composite command of create, start, inspect, attach, rm, resize, wait, kill	Yes. 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 run --net=host` is not supported.
Docker start	Start a container	Yes
Docker stop	Stop a container Stop	Yes. Powers down the VM
Container wait	Wait for a container Wait	Yes
Docker version	Docker version information	Yes. vSphere Integrated Containers Engine version provided

Network Operations

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

Network	Docker Reference	Supported
Network connect	Connect to a network	Yes
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 inspect	Inspect a network	Yes
Network ls	List networks/	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. vSphere Integrated Containers does not assign random names during a volume create, but only for anonymous volumes.
Docker volume inspect	Information about a volume	Yes, use with docker compose
Docker volume ls	List volumes	Yes
Docker volume rm	Remove or delete a volume	Yes

Other Operations

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

Obtain a Virtual Container Host

vSphere Integrated Containers Engine does not currently provide an automated means of obtaining virtual container hosts.

When you or the vSphere Administrator use `vic-machine create` to deploy a virtual container host, the virtual container host endpoint VM obtains an IP address. The IP address can either be static or be obtained from DHCP. As a container developer, you require the IP address of the virtual container host endpoint VM when you run Docker commands.

Depending on the nature of your organization, you might deploy virtual container hosts yourself, or you might request a virtual container host from a different person or team. If you do not run `vic-machine create` yourself, your organization must define the process by which you obtain virtual container host addresses. This process can be as simple as an exchange of emails with a vSphere Administrator, or as advanced as a custom self-provisioning portal or API end-point. For example, your organization could use VMware vRealize® Automation™ to provide a self-provisioning service. In this case, you would use the vRealize Automation interface or APIs to request a virtual container host. At the end of the provisioning process, vRealize Automation would communicate the virtual container host endpoint VM address to you.

Using Docker Environment Variables

If you or the vSphere Administrator deploy the virtual container hosts with TLS authentication, either with trusted certificates or with untrusted self-signed certificates, `vic-machine create` generates a file named `vch_address.env`. The `env` file contains Docker environment variables that are specific to the virtual container host. You can use the contents of the `env` file to set environment variables in your Docker client so that it connects to the correct virtual container host and uses the appropriate level of authentication. Aself-provisioning service such as vRealize Automation could potentially provide the `env` file at the end of the provisioning process for virtual container hosts.

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*.

- Obtain the List of Available Volume Stores
- Obtain the List of Available Volumes
- Create a Volume in a Volume Store
- Creating Volumes from Images
- Create a Container and Attach it to an Anonymous or Named Volume
- Attach an Existing Volume to a Container
- Obtain Information About a Volume
- Delete a Named Volume from a Volume Store

For simplicity, the examples in this topic assume that the virtual container hosts implement TLS authentication with self-signed untrusted certificates, with no client verification.

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
[...]
```

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
```

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
  ```

- 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 default unit will be 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 megabytes, gigabytes, or terabytes, include `MB`, `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
  ```

After you create a volume by using docker volume create, you can attach it to a container by running either of the following commands:

```
docker -H virtual_container_host_address:2376 --tls
create -v /volume_name busybox
```

```
docker -H virtual_container_host_address:2376 --tls
run -v /volume_name busybox
```

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 an error stating that a bad driver has been selected will occur.

Creating Volumes from Images

Some images, for example, `mongo` or `redis:alpine`, contain volume bind information in their metadata. vSphere Integrated Containers Engine creates such volumes with the default parameters and treats them as anonymous volumes. vSphere Integrated Containers Engine treats all volume mount paths as unique, in the same way that Docker does. This should be kept in mind if you attempt to bind other volumes to the same location as anonymous or image volumes. Aspecified volume always takes priority over an anonymous volume.

If you require an image volume with a different volume capacity to the default, create a named volume with the required capacity. You can mount that named volume to the location that the image metadata specifies. You can find the location by running `docker inspect image_name` and consulting the `Volumes` section of the output. The resulting container has the required storage capacity and the endpoint.

Create a Container and Attach it to an Anonymous or Named Volume

If you intend to create named or anonymous volumes by using `docker create -v` when creating containers, a volume store named `default` must exist in the virtual container host. 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 anonymous volume in the `default` volume store, if it exists. The virtual container host attaches the anonymous volume to the container.

For example, to create a busybox container that is mounted to the `volumes` folder of an anonymous volume in the default volume store, run the following command:

```
docker -H virtual_container_host_address:2376 --tls
create -v /volumes busybox
```

You can create containers that are attached to named volumes by using `docker create -v` and specifying a volume name. When you create containers that are attached to named volumes, the virtual container host checks whether the volume exists in the volume store, and if it does not, creates it. The virtual container host attaches the existing or new volume to the container.

For example, to create a busybox container that is mounted to the `volumes` folder of a volume named `volume_1` in the default volume store with default capacity, run the following command:

```
docker -H virtual_container_host_address:2376 --tls
create -v volume_1:/volumes busybox
```

NOTES:

- vSphere Integrated Containers Engine does not support mounting directories as data volumes. A command such as `docker create -v /folder_name:/folder_name busybox` is not supported.
- If you use `docker create -v` to create containers that are attached to volumes, vSphere Integrated Containers Engine only supports the `-r` and `-rw` options.

Mount an Existing Volume on a Container

vSphere Integrated Containers Engine currently supports mounting a volume on only one container at a time. When you mount a volume on a container by using `docker create -v`, that volume remains mounted on the container until you remove that container. When you have removed the container you can mount the volume on a new container.

This example performs the following operations:

- Creates a container named `container1` from the `busybox` image.
- Mounts the `myData` folder of a volume named `volume1` on that container, starts the container, and attaches to it.
- After performing operations in `volume1:/myData` then stopping and detaching `container1`, creates `container2` from the `ubuntu` image and mounts the `myData` folder of `volume1` on it.

```
docker -H virtual_container_host_address:2376 --tls
create --name container1 -v volume1:/myData busybox
docker start container1
docker attach container1

[Perform container operations and detach]

docker stop container1
docker rm container1
docker create -it --name container2 -v volume1:/myData ubuntu
docker start container2
docker attach container2

[Perform container operations with the same volume that was
previously mounted to container1]
```

Obtain Information About a Volume

To get information about a volume, run `docker volume inspect` and specify the name of the volume.

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

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
```

Using Insecure Private Registry Servers with vSphere Integrated Containers Engine

An insecure private registry server is a private registry server that is secured by self-signed certificates rather than by TLS.

If your Docker environment stores Docker images in an insecure private registry server, you or the vSphere administrator must have set the `vic-machine create --insecure-registry` option when creating the virtual container host. Setting the `insecure-registry` option on a virtual container host informs that virtual container host that it is permitted to pull images from the designated insecure registry server.

For information about how to use the `vic-machine create --insecure-registry` option, see the section on `insecure-registry` in Virtual Container Host Deployment Options in *vSphere Integrated Containers Engine Installation*.

Pull a Container Image from an Insecure Private Registry Server

For simplicity, the examples in this topic assume that the virtual container hosts implement TLS authentication with self-signed untrusted certificates, with no client verification.

To pull a container image from an insecure private registry server, run the following Docker command.

```
docker -H vch_address:2376 --tls
pull registry_server_address/path/to/image/image_name:image_version
```

If the private registry server listens for connections on a specific port, include the port number in the registry server URL.

```
docker -H vch_address:2376 --tls
pull registry_server_address:port_number/path/to/image/image_name:image_version
```

NOTE: The current builds of vSphere Integrated Containers do not yet support private registry servers that you secure by using TLS certificates.

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.

Container on a 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 on Multiple Bridge Networks

Create containers on multiple bridge networks by mapping ports through the Docker server. The virtual container host must have an IP address on the relevant bridge networks. To create bridge networks, use `network create`

Example: Run a container: `docker run -it --net net1 --net net2 busybox` For this container to reach both networks or containers connected to only those networks:

```
docker run -it --net net1 --name n1 busybox
docker run -it --net net2 --name n2 busybox
docker run -it --net net1 --net net2 --name n12 busybox
```

Outcome

n1 and n2 cannot talk to each other
n12 can talk to both n1 and n2

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 at 208.91.3.2.

`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.

Containers Using Multiple External Networks

Create external networks only using `vic-machine`.

Example:

```
./vic-machine-darwin create --target 172.16.252.131 --image-store datastore1 --name vic-demo --user root --password 'Vmware
```

pg1-3 are port groups on the ESX Server that are now mapped into docker network ls.

```
$ docker -H 172.16.252.150:2376 --tls network ls
NETWORK ID   NAME   DRIVER
903b61edec66 bridge bridge
95a91e11b1a8 pg1    external
ab84ba2a326b pg2    external
2df4101caac2 pg3    external
```

If a container is connected to an external network, the traffic to and from that network does not go through the appliance VM.

You also can create more bridge networks via the docker API. These are all backed by the same port group as the default bridge, but those networks are isolated via IP address management.

```
Example:
$ docker -H 172.16.252.150:2376 --tls network create mikes
0848ee433797c746b466ffeb57581c301d8e96b7e82a4d524e0fa0222860ba44
$ docker -H 172.16.252.150:2376 --tls network create bob
316e34ff3b7b19501fe14982791ee139ce98e62d060203125c5dbdc8543ff641
$ docker -H 172.16.252.150:2376 --tls network ls
NETWORK ID   NAME   DRIVER
316e34ff3b7b bob    bridge
903b61edec66 bridge bridge
0848ee433797 mikes  bridge
95a91e11b1a8 pg1    external
ab84ba2a326b pg2    external
2df4101caac2 pg3    external
```

Outcome

You can create containers with --net mikes or --net pg1 and be on the correct network. With docker you can combine them and attach multiple networks.

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.