Developing Container Applications with VMware vSphere Integrated Containers Engine

vSphere Integrated Containers Engine 0.9

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 VMware vSphere® Integrated Containers™ Engine as the endpoint for Docker container application development.

For the full vSphere Integrated Containers Engine documentation set, go to https://vmware.github.io/vic-product/#getting-started.

Product version: 0.9

Intended Audience

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

Send Documentation Feedback

Help us to improve the vSphere Integrated Containers documentation.

- Send doc feedback to VMware by email
- Submit an issue in Github
- Send us a message on https://vmwarecode.slack.com/messages/vic-doc

---

VMware, Inc. 3401 Hillview Ave. Palo Alto, CA 94304

www.vmware.com

Overview of vSphere Integrated Containers Engine For Container Application Developers

vSphere Integrated Containers Engine is container engine that is designed to integrate of all the packaging and runtime benefits of containers with the enterprise capabilities of a vSphere environment. As a container developer, you can deploy, test, and run container processes in the same way as you would normally perform container operations.

The information in this topic is intended for container developers. For an extended version of this information, see Overview of vSphere Integrated Containers Engine for vSphere Administrators in *vSphere Integrated Containers Engine Installation*.

Differences Between vSphere Integrated Containers Engine and a Classic Container Environment

The main differences between vSphere Integrated Containers Engine and a classic container environment are the following:

- vSphere, not Linux, is the container host:
  - Containers are spun up *as* VMs, not *in* VMs.
  - Every container is fully isolated from the host and from the other containers.
  - vSphere provides per-tenant dynamic resource limits within a vCenter Server cluster
- vSphere, not Linux, is the infrastructure:
  - You can select vSphere networks that appear in the Docker client as container networks.
  - Images, volumes, and container state are provisioned directly to VMFS.
- vSphere is the control plane:
  - Use the Docker client to directly control selected elements of vSphere infrastructure.
  - Acontainer endpoint Service-as-a-Service presents as a service abstraction, not as IaaS

What Does vSphere Integrated Containers Engine Do?

vSphere Integrated Containers Engine allows the vSphere administrator to easily make the vSphere infrastructure accessible to you, the container application developer, so that you can provision container workloads into production.

Scenario 1: A Classic Container Environment

In a classic container environment:

- You raise a ticket and say, "I need Docker".
- The vSphere administrator provisions a large Linux VM and sends you the IP address.
- You install Docker, patch the OS, configure in-guest network and storage virtualization, secure the guest, isolate the containers, package the containers efficiently, and manage upgrades and downtime.

In this scenario, what the vSphere administrator has given you is similar to a nested hypervisor that you have to manage and which is opaque to them.

Scenario 2: vSphere Integrated Containers Engine

With vSphere Integrated Containers Engine:

- You raise a ticket and say, "I need Docker".
- The vSphere administrator identifies datastores, networking, and compute on a cluster that you can use in your Docker environment.
- The vSphere administrator uses a utility called `vic-machine` to install a small appliance. The appliance represents an authorization for you to use the infrastructure that the vSphere administrator has identified, into which you can self-provision container workloads.
- The appliance runs a secure remote Docker API, that is the only access that you have to the vSphere infrastructure.
- Instead of sending you a Linux VM, the vSphere administrator sends you the IP address of the appliance, the port of the remote Docker API, and a certificate for secure access.

In this scenario, the vSphere administrator has provided you with a service portal. This is better for you because you do not have to worry about isolation, patching, security, backup, and so on. It is better for the vSphere administrator because every container that you deploy is a container VM, that they can manage just like all of their other VMs.

If you discover that you need more compute capacity, in Scenario 1, the vSphere administrator has to power down the VM and reconfigure it, or give you a new VM and let you deal with the clustering implications. Both of these solutions are disruptive to you. With vSphere Integrated Containers Engine in Scenario 2, the vSphere administrator can reconfigure the VCH in vSphere, or redeploy it with a new configuration in a way that is completely transparent to you.

vSphere Integrated Containers Engine Concepts

The objective of vSphere Integrated Containers Engine is to take as much of vSphere as possible and layer whatever Docker capabilities are missing on top, reusing as much of Docker's own code as possible. The result should not sacrifice the portability of the Docker image format and should be completely transparent to a Docker client. The following sections describe key concepts and components that make this possible.

Container VMs

The container VMs that vSphere Integrated Containers Engine creates have all of the characteristics of software containers:

- An ephemeral storage layer with optionally attached persistent volumes.
- A custom Linux guest OS that is designed to be "just a kernel" and that needs images to be functional.
- A mechanism for persisting and attaching read-only binary image layers.
- A PID 1 guest agent *tether* extends the control plane into the container VM.
- Various well-defined methods of configuration and state ingress and egress
- Automatically configured to various network topologies.

The provisioned container VM does not contain any OS container abstraction.

- The container VM boots from an ISO that contains the Photon Linux kernel. Note that container VMs do not run the full Photon OS.
- The container VM is configured with a container image that is mounted as a disk.
- Container image layers are represented as a read-only VMDK snapshot hierarchy on a vSphere datastore. At the top of this hierarchy is a read-write snapshot that stores ephemeral state.
- Container volumes are formatted VMDKs that are attached as disks and indexed on a datastore.
- Networks are distributed port groups that are attached as vNICs.

Virtual Container Hosts

A virtual container host (VCH) is the functional equivalent of a Linux VM that runs Docker, but with some significant benefits. A VCH represents the following elements:

- A clustered pool of resource into which to provision container VMs.
- A single-tenant container namespace.
- A secure API endpoint.
- Authorization to use and configure pre-approved virtual infrastructure.

A VCH is functionally distinct from a traditional container host in the following ways:

- It naturally encapsulates clustering and dynamic scheduling by provisioning to vSphere targets.
- The resource constraints are dynamically configurable with no impact on the containers.
- Containers do not share a kernel.
- There is no local image cache. This is kept on a datastore in the cluster that the vSphere administrator specified when they deployed a VCH.
- There is no read-write shared storage

Supported Docker Commands

vSphere Integrated Containers Engine 0.9 supports Docker 1.13. The supported version of the Docker API is 1.25. If you are using a more recent version of the Docker client, see Docker Commands Fail with a Docker API Version Error.

Docker Management Commands

Command	Docker Reference	Supported
`dockerd`	Launch the Docker daemon	No
`info`	Docker system information	Yes. Provides Docker-specific data, basic capacity information, lists configured volume stores, and virtual container host information. Does not reveal vSphere datastore paths that might contain sensitive vSphere information.
`inspect`	Inspect a container or image	Yes. Includes information about the container network.
`version`	Docker version information	Yes. vSphere Integrated Containers Engine version provided.

Image Commands

Command	Docker Reference	Supported
`build`	Build an image from a Dockerfile	No
`commit`	Create a new image from a container's changes	No
`history`	Show the history of an image	No
`images`	Images	Yes. Supports `--filter` , `--no-trunc` , and `--quiet`
`import`	Import the contents from a tarball to create a filesystem image	No
`load`	Load an image from a tar archive or STDIN	No
`rmi`	Remove a Docker image	Yes. Does not yet support `tag` or `untag` operations, or any options.
`save`	Save images	No
`tag`	Tag an image into a repository	Yes

Container Commands

Command	Docker Reference	Supported
`attach`	Attach to a container	Yes
`container list`	List Containers	Yes
`container resize`	Resize a container	Yes
`cp`	Copy files or folders between a container and the local filesystem	No
`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 Engine, 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. `--memory` Minimum memory is 512MB, maximum unlimited. If unspecified, default is 2GB. Supports the `--attach`, `--cpuset-cpus`, `--env`, `--ip`, `--memory`, `--interactive`, `--link`, `--label`, `--network`, `--tty`, and `--volume` options.
`diff`	Inspect changes on a container's filesystem	No
`events`	Get real time events from the server	Yes. Supports passive Docker events for containers and images. Does not yet support events for volumes or networks.
`exec`	Run a command in a running container	No
`export`	Export a container	No
`kill`	Kill a running container	Yes. Docker must wait for the container to shut down.
`logs`	Get container logs	Yes. Does not support `docker logs --timestamps` (`-t`) and `--since` options.
`pause`	Pause processes in a container	No
`port`	Obtain port data	Yes. Displays port mapping data. Supports mapping a random host port to the container when the host port is not specified.
`ps`	Show running containers	Yes. Supports the `-a/--all`, `-f/--filter`, `--no-trunc`, and `-q/--quiet` options. Filtering by network name is supported, but filtering by network ID is not supported.
`rename`	Rename a	No

`rename`	container	No
`restart`	Restart a container	Yes
`rm`	Remove a container	Yes. Removes associated anonymous and regular volumes. Supports the `--force` option and the `name` parameter. `docker rm -v` and `docker rm -f` are not supported. To view volumes attached to a container that is removed, use `docker volume ls` and `docker volume inspect <id>`. If you continually invoke `docker create` to make more anonymous volumes, those volumes are left behind after each subsequent removal of that container.
`run`	Run a command in a new container	Yes. Supports container search by using prettyname-ID with `docker run --name`. Supports the `--detach`, `--detach-keys`, and `--dns` options. Supports mapping a random host port to the container when the host port is not specified. Supports running images from private and custom registries. `docker run --net=host` is not supported. You can specify a container network by using the `--container-network` option when you deploy a virtual container host.
`start`	Start a container	Yes
`stats`	Get container stats based on resource usage	No
`stop`	Stop a container	Yes. Attempts to politely stop the container. If that fails, powers down the VM.
`top`	Display the running processes of a container	No
`unpause`	Unpause processes within a container	No
`update`	Update a container	No
`wait`	Wait for a container	Yes

Hub and Registry Commands

Command	Docker Reference	Supported
`login`	Log into a registry	Yes
`logout`	Log out from a registry	Yes
`pull`	Pull an image or repository from a registry	Yes. Supports pulling from secure or insecure public and private registries.
`push`	Push an image or a repository to a registry	No
`search`	Search the Docker hub for images	No

Network and Connectivity Commands

Command	Docker Reference	Supported
`network connect`	Connect to a network	Yes. Not supported for running containers.
`network create`	Create a network	Yes. See the use case to connect to an external network in Container Networking with vSphere Integrated Containers Engine. Bridge is also supported. The `--label`, `--internal`, and `--ipam` options are currently not supported.
`network disconnect`	Disconnect a network	No
`network inspect`	Inspect a network	Yes
`network ls`	List networks/	Yes
`network rm`	Remove a network	Yes. Network name and network ID are supported.

Shared Data Volume Commands

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

Command	Docker Reference	Supported
`volume create`	Create a volume	Yes
`volume inspect`	Information about a volume	Yes. Use with docker compose.
`volume ls`	List volumes	Yes
`volume rm`	Remove or delete a volume	Yes

Docker Compose Commands

vSphere Integrated Containers Engine 0.9 supports Docker Compose version 1.9.0.

For more information about using Docker Compose with vSphere Integrated Containers Engine, see Creating a Containerized Application with vSphere Integrated Containers Engine.

For information about Docker Compose file support, see Supported Docker Compose File Options.

Command	Docker Reference	Supported
`build`	Build or rebuild service	No. Depends on `docker build`.
`bundle`	Generate a Distributed Application Bundle (DAB) from the Compose file	No
`config`	Validate and view the compose file	Yes
`create`	Create services	Yes
`down`	Stop and remove containers, networks, images, and volumes	Yes
`events`	Receive real time events from containers	No. Depends on `docker events`.
`exec`	Run commands in services	No. Depends on `docker exec`.
`help`	Get help on a command	Yes
`kill`	Kill containers	No, but `docker kill` works.

`logs`	View output from containers	Yes
`pause`	Pause services	No. Depends on `docker pause`.
`port`	Print the public port for a port binding	Yes
`ps`	List containers	No. Depends on `docker ps --filter`.
`pull`	Pulls service images	Yes
`push`	Pushes images for service	No. Depends on `docker push`
`restart`	Restart services	Yes
`rm`	Remove stopped containers	Yes
`run`	Run a one-off command	Yes
`scale`	Set number of containers for a service	No. Depends on `docker ps --filter`.
`start`	Start services	Yes
`stop`	Stop services	Yes
`unpause`	Unpause services	No. Depends on `docker unpause`.
`up`	Create and start containers	Conditionally supported. Does not work if there are orphaned containers. Depends on `docker rename` and `docker ps --filter`.
`version`	Show Docker Compose version information	Yes

Swarm Commands

This version of vSphere Integrated Containers Engine does not support Docker Swarm.

Supported Docker Compose File Options

vSphere Integrated Containers Engine 0.9 supports Docker Compose file version 2 and 2.1.

This topic provides information about the Docker Compose file options that vSphere Integrated Containers Engine 0.9 supports.

Service Configuration Options

Option	Compose File Reference	Supported
`build`	Options applied at build time	No
`cap_add` , `cap_drop`	Add or drop container capabilities	No. Depends on `docker run --cap-add` and `docker run --cap-drop`
`command`	Override the default command	Yes
`cgroup_parent`	Specify an optional parent `cgroup` for the container.	No; need docker run --cgrop_parent
`container_name`	Specify a custom container name	Yes
`devices`	List of device mappings	No. Depends on `docker create --device` .
`depends_on`	Express dependency between services	Yes
`dns`	Custom DNS servers	Yes
`dns_search`	Custom DNS search domains	No. Depends on `docker run --dns-search` .
`tmpfs`	Mount a temporary file system inside the container	No. Depends on `docker run --tmpfs` .
`entrypoint`	Override the default entry point	No. Depends on `docker run --entrypoint` .
`env_file`	Add environment variables from a file	Yes
`environment`	Add environment variables	Yes
`expose`	Expose ports without publishing them to the host machine	No. Depends on `docker run --expose` .
`extends`	Extend another service	Yes
`external_links`	Link to containers started outside this YML	No. Depends on `docker rename` .
`extra_hosts`	Add hostname mappings	No. Depends on `docker run --add-host` .
`group_add`	Specify additional groups for the user inside the container	Yes
`healthcheck`	Check container health	No. Depends on `docker run --health-cmd` .
`image`	Specify container image	Yes
`isolation`	xxx	No. Depends on `docker run --isolation` .
`labels`	Add metadata by using labels	Yes
`links`	Link to containers in another service	Yes
`logging` , `log_driver` ,		

`logging` , `log_driver` , `log_opt`	Logging configuration	`log-opt` .
`net`	Network mode (version 1)	Yes
`network_mode`	Network mode (version 2)	Yes
`networks`	Networks to join	Yes
`aliases`	Aliases for this service on the network	Yes
`ipv4_address` , `ipv6_address`	Static IP address for containers	Yes for IPv4. vSphere Integrated Containers Engine does not support IPv6.
`link_local_ips`	List of link-local IPs	No. Depends on `docker run --link-local-ip`
`pid`	Sets PID mode	No. Depends on `docker run --pid` .
`ports`	Expose ports	Yes
`userns_mode`	Disables the user namespace	No
`volumes` , `volume_driver`	xxx	Yes
`volumes_from`	Mount volumes from another service or container	No

The following Docker run options are supported if their `docker run` counterpart is supported: `security_opt` , `stop_grace_period` , `stop_signal` , `sysctls` , `ulimits` , `userns_mode` , `cpu_shares` , `cpu_quota` , `cpuset` , `domainname` , `hostname` , `ipc` , `mac_address` , `mem_limit` , `memswap_limit` , `oom_score_adj` , `privileged` , `read_only` , `restart` , `shm_size` , `stdin_open` , `tty` , `user` , `working_dir` .

Volume Configuration Options

vSphere Integrated Containers 0.9 does not support any volume configuration options, pending the implementation of shared volume support.

Network Configuration Options

Option	Compose File Reference	Supported
`driver`	Specify driver to use for this network	Yes
`driver_opts`	Specify options to pass to the driver for this network	No
`enable_ipv6`	Enables IPv6	No. vSphere Integrated Containers Engine does not support IPv6.
`ipam`	Specify custom IPAM configuration	No. Depends on `docker network create --ipam` .
`internal`	Create an externally isolated overlay network	No. Depends on `docker network create --internal` .
`labels`	Add metadata to containers	No. Depends on `docker network --label` .
`external`	Specify that network has been created outside of Compose	Yes

Use and Limitations of Containers in vSphere Integrated Containers Engine

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

Supported Docker Features

This version of vSphere Integrated Containers Engine supports these features:

- Docker Compose (basic)
- Registry pull from docker hub and private registry
- Named Data Volumes
- Anonymous Data Volumes
- Bridged Networks
- External Networks
- Port Mapping
- Network Links/Alias

Limitations

vSphere Integrated Containers Engine includes these limitations:

- Container VMs only support root user.
- When you do not configure a PATH environment variable, or create a container from an image that does not supply a PATH, vSphere Integrated Containers Engine provides a default PATH.
- You can resolve the symbolic names of a container from within another container, except in the following cases:
  - Aliases
  - IPv6
  - Service discovery
- Containers can acquire DHCP addresses only if they are on a network that has DHCP.

Unsupported Docker Features

This version of vSphere Integrated Containers Engine does not support these features:

- Pulling images via image digest
- Pushing a registry
- Sharing concurrent data volume between containers
- Mapping a local host folder to a container volume
- Mapping a local host file to a container
- Docker build
- Docker copy files into a container, both running and stopped
- Docker container inspect does not return all container network for a container

For limitations of using vSphere Integrated Containers with volumes, see Using Volumes with vSphere Integrated Containers Engine.

Obtain a VCH

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

When you or the vSphere administrator use `vic-machine create` to deploy a VCH, the VCH 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 VCH endpoint VM when you run Docker commands.

Depending on the nature of your organization, you might deploy VCHs yourself, or you might request a VCH 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 VCH 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, by using the vRealize Automation interface or APIs to request VCHs. At the end of the provisioning process, vRealize Automation would communicate the VCH endpoint VM address to you.

Using Docker Environment Variables

If you or the vSphere administrator deploy VCHs with TLS authentication, `vic-machine create` generates a file named `vch_address.env`. The `env` file contains Docker environment variables that are specific to the VCH. You can use the contents of the `env` file to set environment variables in your Docker client. A self-provisioning service such as vRealize Automation could potentially provide the env file at the end of the provisioning process for VCHs.

Connecting to the VCH

How you connect to your VCH depends on the security options with which you or the vSphere administrator deployed the VCH.

- If the VCH uses TLS authentication, either by using server certificates or by using mutual authentication with client and server certificates, you connect to the VCH at *vch_address*:2376.
- If the VCH uses mutual authentication with client and server certificates, you must configure the Docker client appropriately with one of the following options:
  - By using the following `--tlsverify`, `--tlscert`, and `--tlskey` Docker options, adding `tlscacert` if a custom CA was used to sign the server certificate.
  - By setting `DOCKER_CERT_PATH=/path/to/client/cert.pem` and `DOCKER_TLS_VERIFY=1`.
- If the VCH uses server certificates without client authentication, you run Docker commands with the `--tls` option. The `DOCKER_TLS_VERIFY` environment variable must not be set. Note that setting `DOCKER_TLS_VERIFY` to 0 or `false` has no effect.
- If TLS is completely disabled on the VCH, you connect to the VCH at *vch_address*:2375 and do not need to specify any additional Docker options.

Using Volumes with vSphere Integrated Containers Engine

vSphere Integrated Containers Engine supports the use of container volumes.

IMPORTANT: To use container volume capabilities with vSphere Integrated Containers Engine, you or the vSphere administrator must configure the virtual container host (VCH) appropriately at the moment of deployment of the VCH. When you create or the vSphere administrator creates a VCH, you or the administrator must specify the datastore to use to store container volumes in the `vic-machine create --volume-store` option. You cannot currently add volume stores, and therefore volume capabilities, to a VCH after its initial deployment. For information about how to use the `vic-machine create --volume-store` option, see the section on `volume-store` in VCH Deployment Options in *vSphere Integrated Containers Engine Installation*.

For simplicity, the examples in this topic assume that the VCHs 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 VCH, run `docker info`.

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

The list of available volume stores for this VCH 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 VCH, 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 you or the vSphere administrator created the VCH with one or more volume stores, but none of the volume stores are named `default`, you must specify the name of an existing volume store in the `--opt VolumeStore` option. 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 created the VCH with a volume store named `default`, you do not need to specify `--opt VolumeStore` in the `docker volume create` command. If you do not specify a volume store name, the `docker volume create` command automatically uses the `default` volume store if it exists.

  ```
  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 mount that volume in a container by running either of the following commands:

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

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

In the examples above, Docker mounts the volume `volume_name` to `/folder` in the container.

NOTE: When using a vSphere Integrated Containers Engine VCH 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. A specified 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 with a New 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 VCH.

NOTES:

- vSphere Integrated Containers Engine does not support mounting folders 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 and mount new volumes on them, vSphere Integrated Containers Engine only supports the `-r` and `-rw` options.

Create a Container with a New Anonymous Volume

To create an anonymous volume, you include the path to the destination at which you want to mount the anonymous volume in the `docker create -v` command. Docker creates the anonymous volume in the `default` volume store, if it exists. The VCH mounts the anonymous volume on the container.

The `docker create -v` example below performs the following actions:

- Creates a busybox container that uses an anonymous volume in the `default` volume store.
- Mounts the volume to `/volumes` in the container.

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

Create a Container with a Named Volume

To create a container with a new named volume, you specify a volume name in the `docker create -v` command. When you create containers that with named volumes, the VCH checks whether the volume exists in the volume store, and if it does not, creates it. The VCH mounts the existing or new volume on the container.

The `docker create -v` example below performs the following actions:

- Creates a busybox container
- Creates volume named `volume_1` in the `default` volume store.
- Mounts the volume to the `/volumes` folder in the container.

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

Mount an Existing Volume on a Container

Mounting existing volumes on containers is subject to the following limitations:

- vSphere Integrated Containers Engine currently supports mounting a volume on only one container at a time.
- Docker does not support unmounting a volume from a container, whether that container is running or not. When you mount

a volume on a container by using `docker create -v`, that volume remains mounted on the container until you remove the container. When you have removed the container you can mount the volume onto a new container.
- If you intend to create and mount a volume on one container, remove that container, and then mount the same volume on another container, use a named volume. It is possible to mount an anonymous volume on one container, remove that container, and then mount the anonymous volume on another container, but it is not recommended to do so.

The `docker create -v` example below performs the following operations:

- Creates a container named `container1` from the `busybox` image.
- Mounts the named volume `volume1` to the `myData` folder on that container, starts the container, and attaches to it.
- After performing operations in `volume1:/myData`, stops and removes `container1`.
- Creates a container named `container2` from the Ubuntu image.
- Mounts `volume1` to the `myData` folder on `container2`.

```
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
```

NOTE: vSphere Integrated Containers does not support running `docker rm -v` to remove volumes that are associated with a container.

Using Private Registry Servers with vSphere Integrated Containers Engine

If your development environment includes private registry servers for container images, you or the vSphere administrator must correctly configure virtual container hosts (VCHs) to allow them to connect to the private registry servers.

You can use vSphere Integrated Containers Engine with either secure or insecure private registry servers.

Secure Private Registry Servers

If the private registry server is configured with TLS, the VCH must be able to validate the registry's certificate. If the registry's server certificate was signed by a custom CA, you must provide that CAto the VCH by using the `--registry-ca` option. If the registry server has a certificate signed by a public CAthen it should function without any additional configuration.

For information about how to configure a VCH to use private registry server CAcertificates, see the section on `--registry-ca` in VCH Deployment Options in *vSphere Integrated Containers Engine Installation*.

Insecure Private Registry Servers

If you set up a private registry that does not use certificates, you or the vSphere administrator must deploy the VCH with the `vic-machine create --insecure-registry` option. Setting the `insecure-registry` option on a VCH informs that VCH that it is authorized to pull images from the designated insecure private registry server.

If you authorize a VCH to connect to an insecure private registry server, the VCH attempts to access the registry server via HTTP if access via HTTPS fails. VCHs always use HTTPS when connecting to registry servers for which you have not authorized insecure access. Insecure private registries are not recommended in production environments.

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

Pull a Container Image from a Private Registry Server

To pull a container image from a 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
```

These commands will only work in the following circumstances:

- The private registry server at *registry_server_address* is secured by CAcertificates, and you or the vSphere administrator passed the appropriate certificates to the VCH during deployment by using the `--registry-ca` option.
- The private registry server at *registry_server_address* is not secured by certificates, and you or the vSphere administrator authorized access to this registry server by using the `--insecure-registry` option during VCH deployment.

NOTE: In the examples, the Docker commands specify `--tls` . This is to specify that the connection between the Docker client and the VCH is secured by TLS. The level of security of the connection between the Docker client and the VCH is completely independent from the level of security of the connection between the VCH and the private registry server. The connection to the private registry server can be insecure when the connection between the client and the VCH is secure, and the reverse.

Using vSphere Integrated Containers Engine with vSphere Integrated Containers Registry (Harbor)

This example illustrates using a deployed Virtual Container Host (VCH) with Harbor as a private registry with the assumption that a VCH has been set up using either static IP or FQDN. It also assumes there is access to standard Docker that has been updated with the CAcertificate used to sign the Harbor instance's server certificate and server private key.

Workflow

1. Develop or obtain a docker container image on a computer or terminal using standard docker. Tag the image for Harbor and push the image to the server.
2. Pull down the image from Harbor to a deployed VCH and use it.

Push a Container Image to Harbor Using Standard Docker

1. Pull the busybox container image from the docker hub to your machine, which you have updated with the CAcertificate earlier. See Deploy a VCH with vSphere Integrated Containers Registry (Harbor) for more information on updating certificates.
2. Tag the image for uploading to your Harbor registry and push the image up to it.

Important You must log onto the Harbor server before pushing the image up to it.

```
user@Devbox:~/mycerts$ docker pull busybox
Using default tag: latest
latest: Pulling from library/busybox

56bec22e3559: Pull complete
Digest: sha256:digest
Status: Downloaded newer image for busybox:latest
user@Devbox:~/mycerts$
user@Devbox:~/mycerts$ docker tag busybox <Harbor FQDN or static
IP>/test/busybox

user@Devbox:~/mycerts$ docker login <Harbor FQDN or static IP>
Username: user
Password:
Login Succeeded

user@Devbox:~/mycerts$ docker push <Harbor FQDN or static IP>/test/busybox
The push refers to a repository [<Harbor FQDN or static IP>/test/busybox]
e88b3f82283b: Pushed
latest: digest: sha256:digest size: 527
```

Pull the Image from Harbor to the VCH

In another terminal, pull the image from Harbor to the VCH.

```
user@Devbox:~$ export DOCKER_HOST=tcp://<Deployed VCH IP>:2375
user@Devbox:~$ export DOCKER_API_VERSION=1.23
user@Devbox:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE

user@Devbox:~$ docker pull <Harbor FQDN or static IP>/test/busybox
Using default tag: latest
Pulling from test/busybox
Error: image test/busybox not found

user@Devbox:~$ docker login <Harbor FQDN or static IP>
Username: user
Password:
```

```
Login Succeeded

user@Devbox:~$ docker pull <Harbor FQDN or static IP>/test/busybox
Using default tag: latest
Pulling from test/busybox
56bec22e3559: Pull complete
a3ed95caeb02: Pull complete
Digest: sha256:digest
Status: Downloaded newer image for test/busybox:latest

user@Devbox:~$ docker images
REPOSITORY                                TAG        IMAGE ID      CREATED            SIZE
<Harbor FQDN or static IP>/test/busybox   latest     e292aa76ad3b        5 weeks ago         1.093 MB
user@Devbox:~$
```

Note that the first attempt to pull the image fails with a 'not found' error message. After you log into the Harbor server, the pull attempt succeeds.

Container Networking with vSphere Integrated Containers Engine

This section presents some examples of how to perform container networking operations when using vSphere Integrated Containers Engine as your Docker endpoint.

- For information about the default Docker networks, see https://docs.docker.com/engine/userguide/networking/.
- For an overview of the networks that vSphere Integrated Containers Engine uses, see Networks Used by vSphere Integrated Containers Engine in *vSphere Integrated Containers Engine Installation*.

Publish a Container Port

Connect a container to an external mapped port on the public network of the virtual container host (VCH):

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

Result

You can access Port 80 on `test1` from the public network interface on the VCH at port 8080.

Add Containers to a New 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
```

Result

The `server` and `client` containers 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 VCH. Assume that `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_public_interface 5000
Trying 192.168.218.137...
Connected to 192.168.218.137.
Escape character is '^]'.
GET /

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

Result

The `server` and `client` containers can ping each other by name. You can connect to `server` on port 5000 from the `client` container and to port 5000 on the VCH public network.

Deploy Containers on Multiple Bridge Networks

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

Run a container that is connected to two networks:

```
docker run -it --net net1 --net net2 busybox
```

Run containers that can each only reach one of the networks:

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

Run a container that can reach both networks:

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

Result

- `n1` and `n2` cannot communicate with each other
- `n12` can communicate with both `n1` and `n2`

Connect Containers to External Networks

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`

Deploy a VCH that uses the default network at 208.91.3.2 as its public network.

`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 that provides a Web service to expose outside of the vSphere Integrated Containers Engine environment.

Output of `docker network inspect default-external`:

```
[
    {
        "Name": "default-external",
        "Id": "id",
        "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": "id",
        "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.

Result

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

Deploy Containers That Use Multiple Container Networks

Create multiple container networks by using `vic-machine create --container-network`.

NOTE: The networks known as container networks in vSphere Integrated Containers Engine terminology correspond to external networks in Docker terminology.

Example:

```
./vic-machine-darwin create --target 172.16.252.131 --image-store datastore1 --name vic-demo --user root --password 'Vmware!23' --compute
```

`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 a container network, the traffic to and from that network does not go through the VCH.

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 mike
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 mike  bridge
95a91e11b1a8 pg1    external
ab84ba2a326b pg2    external
2df4101caac2 pg3    external
```

Result

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

Creating a Containerized Application with vSphere Integrated Containers Engine

The topics in this section provides guidelines for container developers who want to use vSphere Integrated Containers Engine to develop and deploy a containerized application. They include an example of using Docker Compose with vSphere Integrated Containers Engine.

Constraints of Using vSphere Integrated Containers Engine to Build Applications

There are some constraints on the types of containerized applications that you can deploy with this release of vSphere Integrated Containers Engine. For the lists of Docker features that this release of vSphere Integrated Containers Engine supports and does not support, see Use and Limitations of Containers in vSphere Integrated Containers Engine.

Building Container Images

This release of vSphere Integrated Containers Engine does not support the `docker build` or `push` commands. As a consequence, you must use regular Docker without vSphere Integrated Containers Engine to build a container image and to push it to the global hub or to your private registry server.

Sharing Configuration

This release of vSphere Integrated Containers Engine does not support data volume sharing or `docker copy`. As a consequence, providing configuration to a containerized application has some constraints.

An example of a configuration is the configuration files for a Web server. To pass configuration to a container when using vSphere Integrated Containers Engine, you can use the following workaround:

- Use command line arguments or environment variables.
- Add a script to the container image that ingests the command line argument or environment variable and passes the configuration to the contained application.

A benefit of using environment variables to transfer configuration is the containerized application closely follows the popular 12-factor application model.

Since vSphere Integrated Containers Engine does not support sharing volumes between containers, you have the following options for processes that must share files:

- Build the files into the same image and run them in the same container.
- When containers are on the same network, add a script to the container that mounts an NFS share:
  - Run the container with an NFS server that shares a data volume.
  - Mount the NFS share on the containers that need to share files.

Example of Building an Application with vSphere Integrated Containers Engine

The example in this topic modifies the voting app by Docker to illustrate how to work around the constraints that the current version of vSphere Integrated Containers Engine imposes. For information about the constraints, see Constraints of Using vSphere Integrated Containers Engine to Build Applications.

This example focuses on how to modify the Docker Compose YML file from the voting app to make it work with vSphere Integrated Containers. It does not describe the general function or makeup of the voting app.

Getting Started

1. Clone the Docker voting app repository from https://github.com/docker/example-voting-app.
2. Open the YML file for the simple Docker voting app, `docker-compose-simple.yml`.

```
version: "2"

services:
  vote:
    build: ./vote
    command: python app.py
    volumes:
     - ./vote:/app
    ports:
      - "5000:80"

  redis:
    image: redis:alpine
    ports: ["6379"]

  worker:
    build: ./worker

  db:
    image: postgres:9.4

  result:
    build: ./result
    command: nodemon --debug server.js
    volumes:
      - ./result:/app
    ports:
      - "5001:80"
      - "5858:5858"
```

This compose file uses two features that this version of vSphere Integrated Containers does not support:

- The `docker build` command
- Mapping of local folders to container volumes

To allow the voting app to work with vSphere Integrated Containers, you must modify it to work around these constraints.

Modify the Application

This version of vSphere Integrated Containers Engine does not support the `docker build`, `tag`, `push` commands. Use regular Docker without vSphere Integrated Containers Engine to perform the steps in this section.

NOTE: It is possible to build and tag an image in one step. In this example, building and tagging are in separate steps.

1. Build images for the different components of the application.

```
cd example-voting-app
docker build -t vote ./vote
docker build -t vote-worker ./worker
docker build -t vote-result ./result
```

2. Tag the the images to upload them to your private registry or to your personal account on Docker Hub.

   This example uses a Docker Hub account. Replace *dockerhub_username* with your own account name in the commands below.

```
docker tag vote dockerhub_username/vote
docker tag vote-worker dockerhub_username/vote-worker
docker tag vote-result dockerhub_username/vote-result
```

3. Push the images to the registry.

```
docker login
[Provide credentials]
docker push dockerhub_username/vote
docker push dockerhub_username/vote-worker
docker push dockerhub_username/vote-result
```

4. Open the `docker-compose-simple.yml` file in an editor and modify it to remove the operations that vSphere Integrated Containers does not support.
   - Remove local folder mapping
   - Remove all of the build directives
   - Update *dockerhub_username* to your Docker Hub account name
   - Save the modified file with the name `docker-compose.yml`.

The example below shows the YML file after the modifications:

```
version: "2"

services:
  vote:
    image: dockerhub_username/vote
    command: python app.py
    ports:
      - "5000:80"

  redis:
    image: redis:alpine
    ports: ["6379"]

  worker:
    image: dockerhub_username/vote-worker

  db:
    image: postgres:9.4
```

```
  result:
    image: dockerhub_username/vote-result
    command: nodemon --debug server.js
    ports:
      - "5001:80"
      - "5858:5858"
```

You can download the modified YML file from the vSphere Integrated Containers Engine repository on Github from https://github.com/vmware/vic/blob/master/demos/compose/voting-app.

Deploy the Application to a VCH

The steps in this section make the following assumptions:

- You have deployed a virtual container host (VCH).
- You deployed the VCH with a volume store named `default` by specifying `--volume-store datastore_name/path:default`.
- You deployed the VCH with the `--no-tls` option, to disable TLS authentication between the Docker client and the VCH.
- You are using Docker Compose 1.8.1.

In the procedure below, run the commands from the `example-voting-app` folder that contains the modified `docker-compose.yml` file.

1. Run the `docker-compose` command.

   ```
   docker-compose -H vch_address:2375 up -d
   ```

2. In a browser, go to http://*vch_address*:5000 and http://*vch_address*:5001 to verify that the Docker voting application is running.

   You can vote on port 5000 and see the results on port 5001.

Docker Commands Fail with a Docker API Version Error

After a successful deployment of a vSphere Integrated Containers Engine virtual container host (VCH), 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 VCH, the command fails with the following error:

```
Error response from daemon: client is newer than server
(client API version: x.xx, server API version: y.yy)
```

Cause

This version of vSphere Integrated Containers Engine supports Docker 1.13, that includes version 1.25 of the Docker API. You are using a more recent version of the Docker client, that includes a version of the Docker API that is incompatible.

Solution

1. Open a terminal on the system on which you run the Docker client.
2. Set the Docker client API to the same version as the one that is used by vSphere Integrated Containers Engine.

   ```
   export DOCKER_API_VERSION=1.25
   ```

3. Check that your Docker client can now connect to the VCH by running a Docker command.
   - With TLS authentication:

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

   - Without TLS authentication:

     ```
     docker -H virtual_container_host_address:2375 info
     ```

   The `docker info` command should succeed and you should see information about the VCH.

Default Volume Store Error

When you create or run a container, the Docker operation fails with an error about a missing volume store.

Problem

Running the container fails with error:

```
docker: Error response from daemon: No volume store named (default) exists.
```

Cause

By default, vic-machine create does not create a volume store when you or the vSphere administrator deploy a VCH. To run containers from images that use volumes, you or the vSphere administrator must specify a volume store named default when deploying the VCH.

Solution

Deploy a VCH by using the `vic-machine create --volume-store` option to create a VCH with a volume store named `default`. See `--volume-store` in VCH Deployment Options and Specify One or More Volume Stores in Advanced Examples of Deploying a VCH in *vSphere Integrated Containers Installation*.

Use `docker volume inspect` to get information about the volume.

Bridge Pool Mask Error

When you create new bridge networks and specify an IP range, you might encounter an error in the log.

Problem

The error in the log states:

```
could not initialize port layer: bridge mask is not compatible with bridge pool mask
```

Cause

You specified a `--bridge-network-range` that cannot accommodate a /16 network. By default, the range is 172.16.0.0/12, which can accept 16 /16 networks.

Solution

Use a bridge network of at least /16 or larger. See Other Advanced Options in the VCH Deployment Options section of *vSphere Integrated Containers Installation*.

Send Documentation Feedback

Help us to improve the vSphere Integrated Containers documentation.

- Send doc feedback to VMware by email
- Submit an issue in Github
- Send us a message on https://vmwarecode.slack.com/messages/vic-doc