Developing Container Applications with VMware vSphere Integrated Containers

vSphere Integrated Containers 1.1

vmware

Table of Contents

Developing Container Applications with vSphere Integrated Containers 1.1

- Overview for Developers 1.1.1
- Supported Docker Commands 1.1.2
  - Supported Docker Compose File Options 1.1.2.1
- Use and Limitations 1.1.3
- Obtain a VCH 1.1.4
- Configure the Docker Client 1.1.5
- Building and Pushing Images 1.1.6
- Using Volumes 1.1.7
- Container Networking 1.1.8
- Creating a Containerized App 1.1.9
  - Docker Compose Constraints 1.1.9.1
  - Example of Building an App 1.1.9.2
- Default Volume Store Error 1.1.10

Developing Container Applications with vSphere Integrated Containers

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

Product version: 1.1

Intended Audience

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

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

www.vmware.com

Overview of vSphere Integrated Containers For Container Application Developers

vSphere Integrated Containers 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 for vSphere Administrators in *vSphere Integrated Containers for vSphere Administrators*.

- Differences Between vSphere Integrated Containers and a Classic Container Environment
- What Does vSphere Integrated Containers Do?
- What Is vSphere Integrated Containers Engine?
- What Is vSphere Integrated Containers Registry?
- What Is vSphere Integrated Containers Management Portal?

Differences Between vSphere Integrated Containers and a Classic Container Environment

The main differences between vSphere Integrated Containers 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 Do?

vSphere Integrated Containers 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

With vSphere Integrated Containers:

- 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 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.

What Is vSphere Integrated Containers Engine?

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

What Is vSphere Integrated Containers Registry?

vSphere Integrated Containers Registry is an enterprise-class registry server that you can use to store and distribute container images. vSphere Integrated Containers Registry allows DevOps administrators to organize image repositories in projects, and to set up role-based access control to those projects to define which users can access which repositories. vSphere Integrated Containers Registry also provides rule-based replication of images between registries, implements Docker Content Trust, and provides detailed logging for project and user auditing.

For a more detailed overview of vSphere Integrated Containers Registry, see Managing Images, Projects, and Users with vSphere Integrated Containers Registry in *vSphere Integrated Containers for DevOps Administrators*.

What Is vSphere Integrated Containers Management Portal?

vSphere Integrated Containers Management Portal is a highly scalable and very lightweight container management platform for deploying and managing container based applications. It is designed to have a small footprint and boot extremely quickly. vSphere Integrated Containers Management Portal is intended to provide DevOps administrators with automated deployment and lifecycle management of containers.

- Rule-based resource management, allowing DevOps administrators to set deployment preferences which let vSphere Integrated Containers Management Portal manage container placement.
- Live state updates that provide a live view of the container system.
- Multi-container template management, that enables logical multi-container application deployments.

For a more information about vSphere Integrated Containers Management Portal, see View and Manage VCHs, Add Registries, and Provision Containers Through the Management Portal in *vSphere Integrated Containers for DevOps Administrators*.

Supported Docker Commands

vSphere Integrated Containers Engine 1.1 supports Docker 1.13. The supported version of the Docker API is 1.25.

- Docker Management Commands
- Image Commands
- Container Commands
- Hub and Registry Commands
- Network and Connectivity Commands
- Shared Data Volume Commands
- Docker Compose Commands
- Swarm Commands

Docker Management Commands

Command	Docker Reference	Supported
`dockerd`	Launch the Docker daemon	Not applicable. This construct does not exist in vSphere Integrated Containers
`info`	Docker system information	Yes, since 1.0. 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, since 1.0. Includes information about the container network.
`version`	Docker version information	Yes, since 1.0

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, since 1.0. 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, since 1.0
`save`	Save images	No
`tag`	Tag an image into a repository	Yes, since 1.0

Container Commands

Command	Docker Reference	Supported
`attach`	Attach to a container	Yes, since 1.0
`container list`	List Containers	Yes, since 1.0
`container resize`	Resize a container	Yes, since 1.0
`cp`	Copy files or folders between a container and the local filesystem	No
`create`	Create a container	Yes, since 1.0. `--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. Minimum value for `--memory` is 512MB, maximum unlimited. If unspecified, the 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, since 1.0. 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, since 1.0. Docker must wait for the container to shut down.
`logs`	Get container logs	Yes, since 1.0. Does not support `docker logs --timestamps` (`-t`) and `--since` options.
`pause`	Pause processes in a container	No
`port`	Obtain port data	Yes, since 1.0. 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, since 1.0. 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	Yes, since 1.1. Name resolution for renamed running containers is not supported, but if

`rename`	container	you restart the container the new name is resolved.
`restart`	Restart a container	Yes, since 1.0
`rm`	Remove a container	Yes, since 1.0. Removes associated anonymous and regular volumes. Supports the `--force` option and the `name` parameter. Does not support `docker rm -v`. 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, since 1.0. 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, since 1.0
`stats`	Get container stats based on resource usage	Yes, since 1.1. Provides statistics about CPU and memory usage. Does not yet provide statistics about network or disk usage.
`stop`	Stop a container	Yes, since 1.0. 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, since 1.0

Hub and Registry Commands

Command	Docker Reference	Supported
`login`	Log into a registry	Yes, since 1.0
`logout`	Log out from a registry	Yes, since 1.0
`pull`	Pull an image or repository from a registry	Yes, since 1.0. 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

For more information about network operations with vSphere Integrated Containers Engine, see Container Networking with vSphere Integrated Containers Engine.

Command	Docker Reference	Supported
`network connect`	Connect to a network	Yes, since 1.0. Not supported for running containers.
`network create`	Create a network	Yes, since 1.1. See the use case to connect a container to an external network in Container Networking with vSphere Integrated Containers Engine. Bridge is also supported.
`network disconnect`	Disconnect a network	No
`network inspect`	Inspect a network	Yes, since 1.0
`network ls`	List networks/	Yes, since 1.0
`network rm`	Remove a network	Yes, since 1.0. 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, since 1.0
`volume inspect`	Information about a volume	Yes, since 1.0
`volume ls`	List volumes	Yes, since 1.0
`volume rm`	Remove or delete a volume	Yes, since 1.0

Docker Compose Commands

vSphere Integrated Containers Engine 1.1 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	Yes, since 1.1
`config`	Validate and view the compose file	Yes, since 1.0
`create`	Create services	Yes, since 1.0
`down`	Stop and remove containers, networks, images, and volumes	Yes, since 1.0
`events`	Receive real time events from containers	Yes, since 1.0. Supports passive Docker events for containers and images. Does not yet support events for volumes or networks.

`exec`	Run commands in services	No. Depends on `docker exec`.
`help`	Get help on a command	Yes, since 1.0
`kill`	Kill containers	No, but `docker kill` works.
`logs`	View output from containers	Yes, since 1.0
`pause`	Pause services	No. Depends on `docker pause`.
`port`	Print the public port for a port binding	Yes, since 1.0
`ps`	List containers	Yes, since 1.0
`pull`	Pulls service images	Yes, since 1.0
`push`	Pushes images for service	No. Depends on `docker push`
`restart`	Restart services	Yes, since 1.0
`rm`	Remove stopped containers	Yes, since 1.0
`run`	Run a one-off command	Yes, since 1.0
`scale`	Set number of containers for a service	Yes, since 1.0
`start`	Start services	Yes, since 1.0
`stop`	Stop services	Yes, since 1.0
`unpause`	Unpause services	No. Depends on `docker unpause`.
`up`	Create and start containers	Yes, since 1.1
`version`	Show Docker Compose version information	Yes, since 1.0

Swarm Commands

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

Supported Docker Compose File Options

vSphere Integrated Containers Engine 1.1 supports Docker Compose file version 2 and 2.1.

This topic provides information about the Docker Compose file options that vSphere Integrated Containers Engine 1.1 supports.

- Service Configuration Options
- Volume Configuration Options
- Network Configuration Options

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	Yes
`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`	Specify isolation technology	No. Depends on `docker run --isolation .`
`labels`	Add metadata by using labels	Yes
`links`	Link to containers in another service	Yes

`logging` , `log_driver` , `log_opt`	Logging configuration	No. Depends on `docker run --log-driver` and `--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. IPv6 is not supported.
`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
`security-opt`	Override the default labeling scheme for containers	No. This option only applies to Windows containers, which are not supported.
`stop-signal`	Sets an alternative signal to stop the container.	Yes
`stop-grace-period`	Specify how long to wait stopping a container	No
`sysctls`	Kernel parameters to set in the container	No
`ulimits`	Override the default ulimits for a container	No
`userns_mode`	Disables the user namespace	No
`volumes` , `volume_driver`	Mount paths or named volumes	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

NOTE: vSphere Integrated Containers 1.1 does not support shared volumes. You can use these options for containers that do not share volumes.

Option	Compose File Reference	Supported
`driver`	Specify driver to use for this volume	Yes
`driver_opts`	Specify options to pass to the driver for this volume	Yes
`labels`	Add metadata to containers	Yes
`external`	Specify that volume has been created outside of Compose	Yes

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. IPv6 is not supported.
`ipam`	Specify custom IPAM configuration	Yes
`internal`	Create an externally isolated overlay network	Yes
`labels`	Add metadata to containers	Yes
`external`	Specify that network has been created outside of Compose	Yes

Use and Limitations of 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`
- Pulling images from Docker hub and private registries
- Named data volumes
- Anonymous data volumes
- Bridged networks
- External networks
- Port mapping
- Network links/aliases

Unsupported Docker Features

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

- Pulling images via image digest
- Sharing concurrent data volume between containers
- Mapping a local host folder to a container volume
- Mapping a local host file to a container
- `docker push`
- `docker build`
- `docker cp`

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

Limitations of vSphere Integrated Containers Engine

vSphere Integrated Containers Engine includes these limitations:

- If you do not configure a `PATH` environment variable, or if you 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.

Using `docker-compose` with TLS

vSphere Integrated Containers supports TLS v1.2, so you must configure `docker-compose` to use TLS 1.2. However, `docker-compose` does not allow you to specify the TLS version on the command line. You must use environment variables to set the TLS version for `docker-compose`. For more information, see `docker-compose` issue 4651. Furthermore, `docker-compose` has a limitation that requires you to set TLS options either by using command line options or by using environment variables. You cannot use a mixture of both command line options and environment variables.

To use `docker-compose` with vSphere Integrated Containers and TLS, set the following environment variables:

```
COMPOSE_TLS_VERSION=TLSv1_2
DOCKER_TLS_VERIFY=1
DOCKER_CERT_PATH="path to your cert files"
```

The certificate file path must lead to `CA.pem`, `client_key.pem`, and `client cert.pem`. You can run `docker-compose` with the following command:

```
docker-compose -H vch_address up
```

Obtain a VCH

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

When the vSphere administrator uses `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.

If the vSphere administrator deploys VCHs with TLS authentication, `vic-machine create` generates a file named `vch_name.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. The vSphere administrator or an automated provisioning service for VCHs could potentially provide the `env` file to you when you request a VCH.

Configure the Docker Client for Use with vSphere Integrated Containers

If your container development environment uses vSphere Integrated Containers, you must run Docker commands with the appropriate options, and configure your Docker client accordingly.

- Connecting to the VCH
- Using Docker Environment Variables
- Using vSphere Integrated Containers Registry
- Using vSphere Integrated Containers Registry with Notary

Connecting to the VCH

How you connect to your virtual container host (VCH) depends on the security options with which the vSphere administrator deployed the VCH.

- If the VCH implements any level of TLS authentication, you connect to the VCH at *vch_address*:2376 when you run Docker commands.
- If the VCH implements mutual authentication between the Docker client and the VCH by using both client and server certificates, you must provide a client certificate to the Docker client so that the VCH can verify the client's identity. This configuration is commonly referred to as `tlsverify` in documentation about containers and Docker. You must obtain a copy of the client certificate that was either used or generated when the vSphere administrator deployed the VCH. You can provide the client certificate to the Docker client in either of the following ways:
  - By using the `--tlsverify`, `--tlscert`, and `--tlskey` options when you run Docker commands. You must also add `--tlscacert` if the server certificate is signed by a custom Certificate Authority (CA). For example:

    ```
    docker -H vch_address:2376
    --tlsverify
    --tlscert=path_to_client_cert/cert.pem
    --tlskey=path_to_client_key/key.pem
    --tlscacert=path/ca.pem
    info
    ```

  - By setting Docker environment variables:

    ```
    DOCKER_CERT_PATH=client_certificate_path/cert.pem
    DOCKER_TLS_VERIFY=1
    ```

- If the VCH uses server certificates but does not authenticate the Docker client, no client certificate is required and any client can connect to the VCH. This configuration is commonly referred to as `no-tlsverify` in documentation about containers and Docker. In this configuration, the VCH has a server certificate and connections are encrypted, requiring you to run Docker commands with the `--tls` option. For example:

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

  In this case, do not set the `DOCKER_TLS_VERIFY` environment variable. Setting `DOCKER_TLS_VERIFY` to 0 or to `false` has no effect.
- If TLS is completely disabled on the VCH, you connect to the VCH at *vch_address*:2375. Any Docker client can connect to the VCH and communications are not encrypted. As a consequence, you do not need to specify any additional TLS options in Docker commands or set any environment variables. This configuration is not recommended in production environments. For example:

  ```
  docker -H vch_address:2375 info
  ```

Using Docker Environment Variables

If the vSphere administrator deploys the VCHs with TLS authentication, `vic-machine create` generates a file named `vch_name.env`. The `env` file contains Docker environment variables that are specific to the VCH. You can use the `env` file to set environment variables in your Docker client.

The contents of the `env` files are different depending on the level of authentication with which the VCH was deployed.

- Mutual TLS authentication with client and server certificates:

  ```
  DOCKER_TLS_VERIFY=1
  DOCKER_CERT_PATH=client_certificate_path\vch_name
  DOCKER_HOST=vch_address:2376
  ```

- TLS authentication with server certificates without client authentication:

  ```
  DOCKER_HOST=vch_address:2376
  ```

- No `env` file is generated if the VCH does not implement TLS authentication.

For information about how to obtain the `env` file, see Obtain a VCH.

Using vSphere Integrated Containers Registry

If your development environment uses vSphere Integrated Containers Registry or another private registry server that uses CA server certificates, you must pass the registry's CAcertificate to the Docker client. The vSphere administrator must also have configured the VCH to access the registry. For information about how to obtain the CAcertificate from vSphere Integrated Containers Registry and how to deploy a VCH so that it can access a private registry, see Deploy a VCH for Use with vSphere Integrated Containers Registry.

NOTE: The level of security of the connection between the Docker client and the VCH is independent from the level of security of the connection between the Docker client and the registry. Connections between the Docker client and the registry can be secure while connections between the Docker client and the VCH are insecure, and the reverse.

Docker on Linux

This example configures a Linux Docker client so that you can log into vSphere Integrated Containers Registry by using both its fully-qualified domain name (FQDN) and by using its IP address.

1. Copy the certificate file to the Linux machine on which you run the Docker client.
2. Switch to `sudo` user.

   ```
   $ sudo su
   ```

3. Create two subfolders in the Docker certificates folder, naming one with the registry's FQDN and one with the registry's IP address.

   ```
   $ mkdir -p /etc/docker/certs.d/registry_fqdn
   ```

   ```
   $ mkdir -p /etc/docker/certs.d/registry_ip
   ```

4. Copy the registry's CAcertificate into both folders.

   ```
   $ cp ca.crt /etc/docker/certs.d/registry_fqdn/
   ```

   ```
   $ cp ca.crt /etc/docker/certs.d/registry_ip/
   ```

5. Open a new terminal and attempt to log in to the registry server by using both the FQDN and the IP address of the registry server.

```
$ docker login registry_fqdn
```

```
$ docker login registry_ip
```

6. If the login fails with a certificate error, restart the Docker daemon.

```
$ sudo systemctl daemon-reload
```

```
$ sudo systemctl restart docker
```

Docker on Windows

To pass the registry's CAcertificate to a Docker client that is running on Windows 10, use the Windows Certificate Import Wizard.

1. Copy the `ca.crt` file to the Windows 10 machine on which you run the Docker client.
2. Right-click the `ca.crt` file and select **Install Certificate**.
3. Follow the prompts of the wizard to install the certificate.
4. Restart the Docker daemon:
   - Click the up arrow in the task bar to show running tasks.
   - Right-click the Docker icon and select **Settings**.
   - Select **Reset** and click **Restart Docker**.
5. Log in to the registry server.

```
docker login registry_address
```

Using vSphere Integrated Containers Registry with Notary

vSphere Integrated Containers Registry provides a Docker Notary server that allows you to implement content trust by signing and verifying the images in the registry. For information about Docker Notary, see Content trust in Docker in the Docker documentation.

To use the Docker Notary server from vSphere Integrated Containers Registry, you must pass the registry's CAcertificate to your Docker client and set up Docker Content Trust. By default, the vSphere Integrated Containers Registry Notary server runs on port 4443 on the vSphere Integrated Containers appliance.

1. If you are using a self-signed certificate, copy the CAroot certificate to the Docker certificates folder.

   To pass the certificate to the Docker client, follow the procedure in Using vSphere Integrated Containers Registry above.

2. If you are using a self-signed certificate, copy the CAcertificate to the Docker TLS service.

```
$ cp ca.crt ~/.docker/tls/registry_ip:4443/
```

3. Enable Docker Content Trust by setting environment variables.

```
export DOCKER_CONTENT_TRUST=1
export DOCKER_CONTENT_TRUST_SERVER=https://registry_ip:4443
```

4. (Optional) Set an alias for Notary.

   By default, the local directory for storing meta files for the Notary client is different from the folder for the Docker client. Set an alias to make it easier to use the Notary client to manipulate the keys and meta files that Docker Content Trust generates.

```
alias notary="notary -s https//registry_ip:4443 -d ~/.docker/trust --tlscacert
/etc/docker/certs.d/registry_ip/ca.crt"
```

Building and Pushing Images with vSphere Integrated Containers

The current version of vSphere Integrated Containers Engine does not support `docker build` or `docker push`. As a consequence, the workflow for developing container images and pushing them to a registry server is slightly different to the workflow in a regular Docker environment.

- You use standard Docker to build, tag, and push a container image to a registry.
- You pull the image from the registry to a vSphere Integrated Containers virtual container host (VCH) to use it.

This topic provides an example of pushing and pulling an image to and from vSphere Integrated Containers Registry. You can use a different private registry server. For simplicity, the example uses the `busybox` container image instead of building a new image.

Prerequisites

- You have access to an image repository. For example, a project repository must exist in vSphere Integrated Containers Registry and you must have a user account that can access that project repository.
- Configure your Docker client to use the vSphere Integrated Containers Registry certificate. For information about how to pass the registry certificate to the Docker client, see Using vSphere Integrated Containers Registry in Configure the Docker Client for Use with vSphere Integrated Containers.
- You have access to a VCH that the vSphere administrator configured so that it can connect to the registry. For information about how deploy a VCH so that it can access a private registry, see the Private Registry Options section of VCH Deployment Options and Deploy a VCH for Use with vSphere Integrated Containers Registry in *vSphere Integrated Containers for vSphere Administrators*
- In the example, connections to the registry are secured by TLS, but for simplicity the connection between the Docker client and the VCH is not. As a consequence, the Docker commands to run in the VCH do not include any TLS options. If your VCH uses TLS authentication, adapt the Docker commands accordingly, and use port 2376 instead of 2375 when connecting to the VCH. For information about how to connect a Docker client to a VCH that uses TLS authentication, see Connecting to the VCH in Configure the Docker Client for Use with vSphere Integrated Containers.

Procedure

1. Pull the `busybox` container image from Docker Hub.

   ```
   docker pull busybox
   ```

   In a real-world scenario you would build a new container image rather than pulling the `busybox` image.
2. Tag the image for uploading to the appropriate project repository in vSphere Integrated Containers Registry.

   ```
   docker tag busybox:1.26 registry_address/project_name/busybox:1.26
   ```

   If vSphere Integrated Containers Registry listens for connections on a non-default port, include the port number in the registry address.
3. Log in to vSphere Integrated Containers Registry.

   ```
   docker login registry_address
   ```

4. Push the image from the standard Docker host to vSphere Integrated Containers Registry.

   ```
   docker push registry_address/project_name/busybox:1.26
   ```

5. Pull the image from vSphere Integrated Containers Registry into the VCH.

   ```
   docker -H vch_address:2375 pull registry_address/project_name/busybox:1.26
   ```

6. List the images that are running in your VCH.

```
docker -H vch_address:2375 images
```

Result

The image that you pulled from vSphere Integrated Containers Registry appears in the list of images that are available in this VCH.

```
REPOSITORY                                      TAG          IMAGE ID
registry_address/project_name/busybox    1.26         7e156d496c9f
```

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, the vSphere administrator must configure the virtual container host (VCH) appropriately at the moment of deployment of the VCH. When the vSphere administrator creates a VCH, 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 for vSphere Administrators*.

- 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 with a New Anonymous or Named Volume
- Mount an Existing Volume on 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 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 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 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.

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 for vSphere Administrators*.

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' --comp
ute-resource /ha-datacenter/host/*/Resources --container-network pg1:pg1 --container-network pg2:pg2 --container-network pg3:pg3
```

`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 with Docker Compose
- Example of Building an Application with vSphere Integrated Containers Engine

Constraints of Using vSphere Integrated Containers Engine with Docker Compose

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 supports and does not support, see Use and Limitations of Containers in vSphere Integrated Containers Engine.

Building Container Images

This release does not support the `docker build` or `push` commands. As a consequence, you must use regular Docker to build a container image and to push it to the global hub or to your private registry server.

Sharing Configuration

This release does not support data volume sharing or `docker cp`. 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, 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 container application.

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

Since this release 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 with Docker Compose.

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` or `push` commands. Use regular Docker 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 an excerpt of 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.

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 the vSphere administrator deploys a VCH. To run containers from images that use volumes, the vSphere administrator must specify a volume store named default when they deploy 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 Volume Stores in Advanced Examples of Deploying a VCH in *vSphere Integrated Containers Installation*.

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