# Red Hat Container Development Kit 3.1

## Getting Started Guide

Quick-start guide to using and developing with Red Hat Container Development Kit

Last Updated: 2018-02-02

# Red Hat Container Development Kit 3.1 Getting Started Guide

Quick-start guide to using and developing with Red Hat Container Development Kit

Chris Negus
cnegus@redhat.com

Robert Krátký
rkratky@redhat.com

## Legal Notice



## Abstract

This guide shows how to get up to speed using Red Hat Container Development Kit. Included instructions and examples guide through first steps developing containerized applications using Docker, Kubernetes, and OpenShift Container Platform, both from your host workstation (Microsoft Windows, macOS, or Red Hat Enterprise Linux) and from within the Container Development Environment provided by Red Hat Container Development Kit.

# Table of Contents

# CHAPTER 1. GETTING STARTED WITH CONTAINER DEVELOPMENT KIT

This section contains information about setting up, installing, updating and uninstalling Container Development Kit.

## 1.1. INTRODUCING RED HAT CONTAINER DEVELOPMENT KIT

Red Hat Container Development Kit is a platform for developing containerized applications on a single, personal system. It enables developers to quickly and easily set up an environment for developing and testing containerized applications on the Red Hat Enterprise Linux platform.

Container Development Kit:

- Provides a personal Container Development Environment you can install on your own laptop, desktop, or server system. The Container Development Environment is provided in the form of a Red Hat Enterprise Linux virtual machine. The Container Development Environment itself can also be installed in a virtual machine.
- Includes the same container-development and run-time tools used to create and deploy containers for large data centers.
- CDK version 3 offers an easy installation method based on the minishift tool.
- Runs on Microsoft Windows, macOS, and Linux operating systems as a Linux virtual machine, thus allowing developers to use their favorite platform while producing applications ready to be deployed in the Red Hat Enterprise Linux ecosystem.

Container Development Kit is a part of the Red Hat Developers program, which provides tools, resources, and support for developers who wish to utilize Red Hat solutions and products to create applications, both locally and in the cloud. For additional information and to register to become a part of the program, visit developers.redhat.com.

### 1.1.1. Additional Information

Refer to the following documents for further information:

- See Red Hat Container Development Kit 3.1 Release Notes and Known Issues for information about the current release of the product as well as a list of known problems that users may encounter when using it.
- See Container Development Kit Getting Started Guide for instructions on how to install and start using the Container Development Environment to develop Red Hat Enterprise Linux-based containers using tools and services such as **OpenShift Container Platform**, **Docker**, **Eclipse**, and various command-line tools.

To report issues, refer to the following:

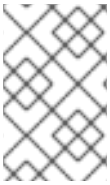
- For Red Hat Container Development Kit issues or to request new CDK features, refer to the CDK Project.
- For issues with Red Hat Container Development Kit documentation or to request new documentation, refer to CDK Project Documentation.

# 1.2. PREPARING TO INSTALL CONTAINER DEVELOPMENT KIT

## 1.2.1. Overview

Container Development Kit version 3.1 is based on the Minishift project version 1.5.0. Using a single executable file, the minishift command deploys Container Development Kit as a Red Hat Enterprise Linux virtual machine, running OpenShift (which includes Kubernetes, the docker service, and other container development and deployment software).

Container Development Kit setup procedure can, and should, be run as a regular user that has special permission to launch virtual machines. In the procedure, you will see how to assign that permission, along with ways to configure your hypervisor and command shell to start and effectively interact with Container Development Kit.



**NOTE**

Container Development Kit version 2.x was based on Vagrant. Because of the completely different deployment and management methods, there is no upgrade path from CDK 2.x to CDK 3.x versions. You need to do a fresh setup.

The following section describes how to install Container Development Kit and the required dependencies.

## 1.2.2. Prerequisites

Container Development Kit requires a hypervisor to start the virtual machine on which the OpenShift cluster is provisioned. Verify that the hypervisor of your choice is installed and enabled on your system before you set up Container Development Kit. Once the hypervisor is up and running, additional setup is required for Container Development Kit to work with that hypervisor (as described in the coming setup procedure).

Depending on your host operating system, you have the choice of the following hypervisors:

**macOS**

- xhyve (default)
- VirtualBox

**Red Hat Enterprise Linux, Fedora, CentOS or other Linux system**

- KVM (default)
- VirtualBox

**Windows**

- Hyper-V (default)
- VirtualBox

Refer to the documentation for each hypervisor to determine the hardware and operating system versions needed to run that hypervisor.

### 1.2.3. Understanding Container Development Kit Installation

These are the basic steps for setting up Container Development Kit on your personal laptop or desktop system:

1. Set up your virtualization environment
2. Download CDK software for your operating system from the Container Development Kit Download Page
3. Set up CDK
4. Start CDK
5. Configure CDK so you can use it efficiently

## 1.3. CONTAINER DEVELOPMENT KIT INSTALLATION

### 1.3.1. Installing Container Development Kit

The following steps describe how to prepare your virtualization environment (hypervisor) for CDK, download CDK software, set up CDK, and start using it.

### 1.3.2. Step 1: Set up your virtualization environment

Follow the appropriate procedure to set up virtualization for your particular operating system and hypervisor. Container Development Kit uses Docker Machine and its driver plug-in architecture to provide a consistent way to manage the OpenShift VM.

Some hypervisors require that the driver plug-in be manually installed. Container Development Kit embeds VirtualBox drivers so no additional steps are required to configure the driver. However, later you will need to run a minishift command to tell Container Development Kit to use VirtualBox.

#### 1.3.2.1. Set up hypervisor on Red Hat Enterprise Linux

Choose between KVM (default) and VirtualBox for your hypervisor. Manual driver setup is required for KVM The driver is automatically configured if you install VirtualBox. However, a minishift command would be required later to identify VirtualBox to Container Development Kit.

**On Red Hat Enterprise Linux with KVM virtualization** : Container Development Kit is currently tested against **docker-machine-driver-kvm** version 0.7.0. Follow these steps to install the KVM driver and configure your user account to use the libvirtd service.

1. As root, install the KVM binary and make it executable as follows:

```
# curl -L https://github.com/dhiltgen/docker-machine-
kvm/releases/download/v0.7.0/docker-machine-driver-kvm -o
/usr/local/bin/docker-machine-driver-kvm
# chmod +x /usr/local/bin/docker-machine-driver-kvm
```

   For more information, see the GitHub documentation of the docker machine KVM driver .

2. As root, install **libvirt** and **qemu-kvm** on your system and add yourself to the *libvirt* group:

```
# yum install libvirt qemu-kvm
# usermod -a -G libvirt <username>
```

3. Update your current user session to apply the group change:

```
$ newgrp libvirt
```

4. Start the libvirtd service as root:

```
# systemctl start libvirtd
# systemctl enable libvirtd
```

#### 1.3.2.2. Set up hypervisor on macOS

Choose between xhyve (default) and VirtualBox for your hypervisor. Manual driver setup is required for xhyve. The driver is automatically configured if you install VirtualBox. However, a minishift command would be required later to identify VirtualBox to Container Development Kit.

**On macOS with xhyve virtualization:**

Container Development Kit on macOS with xhyve virtualization is currently tested against **docker-machine-driver-xhyve** version 0.3.1. To manually install the xhyve driver, you need to download and install the **docker-machine-driver-xhyve** binary and place it in a directory which is on your **PATH** environment variable. The directory ***/usr/local/bin*** is most likely a good choice, since it is the default installation directory for Docker Machine binaries.

The following steps explain the installation of the **docker-machine-driver-xhyve** binary to the ***/usr/local/bin/*** directory:

1. Download the **docker-machine-driver-xhyve** binary using:

```
$ sudo curl -L  https://github.com/zchee/docker-machine-driver-
xhyve/releases/download/v0.3.1/docker-machine-driver-xhyve -o
/usr/local/bin/docker-machine-driver-xhyve
```

2. Enable root access for the **docker-machine-driver-xhyve** binary and add it to the default wheel group:

```
$ sudo chown root:wheel /usr/local/bin/docker-machine-driver-xhyve
```

3. Set owner User ID (SUID) for the binary as follows:

```
$ sudo chmod u+s,+x /usr/local/bin/docker-machine-driver-xhyve
```

**NOTE**

The downloaded **docker-machine-driver-xhyve** binaries are compiled against a specific version of macOS. It is possible that the driver fails to work after an macOS version upgrade. In this case you can try to compile the driver from source:

```
$ go get -u -d github.com/zchee/docker-machine-driver-xhyve
$ cd $GOPATH/src/github.com/zchee/docker-machine-driver-xhyve

# Install docker-machine-driver-xhyve binary into
/usr/local/bin
$ make install

# docker-machine-driver-xhyve need root owner and uid
$ sudo chown root:wheel /usr/local/bin/docker-machine-driver-
xhyve
$ sudo chmod u+s /usr/local/bin/docker-machine-driver-xhyve
```

For more information, see the xhyve driver documentation on GitHub.

#### 1.3.2.3. Set up hypervisor on Windows

Choose between Hyper-V (default on Windows 10) and VirtualBox (Windows 7 or Windows 10) for your hypervisor. Manual driver setup is required for Hyper-V. The driver is automatically configured if you install VirtualBox. However, a minishift command would be required later to identify VirtualBox to Container Development Kit.

**On Windows with Hyper-V virtualization:**

1. Install Hyper-V.
2. Add an External Virtual Switch. Verify that you pair the virtual switch with a network card (wired or wireless) that is connected to the network.
3. If you have multiple virtual switches, set the environment variable **HYPERV_VIRTUAL_SWITCH** to the name of the external virtual switch you want to use for Container Development Kit. For example, on Command Prompt use:

   ```
   C:\> set HYPERV_VIRTUAL_SWITCH=External (Wireless)
   ```

   Note that using quotes in Command Prompt results in the following error:

   ```
   C:\> set HYPERV_VIRTUAL_SWITCH="External (Wireless)"
   Error creating the VM. Error with pre-create check: "vswitch
   \"\\\"External (Wireless)\\\"\" not found"
   ```

   However, on PowerShell you need to use the quotes:

   ```
   PS C:\> $env:HYPERV_VIRTUAL_SWITCH="External (Wireless)"
   ```

### 1.3.3. Step 2: Download CDK Software

Before you can download CDK software, you need to either register with the Red Hat Developer Program site or login to the Red Hat customer portal with Red Hat subscription credentials. Then go to one of the following two sites and download the software associated with your operating system:

- Red Hat Developer Program CDK Download Page
- Red Hat Customer Portal CDK Download Page

Copy the downloaded **minishift** file to a directory in your $PATH and make it executable. The downloaded minishift executable is named **cdk-3.1.0-1-minishift-darwin-amd64** (for macOS), **cdk-3.1.0-1-minishift-linux-amd64** (for Linux) or **cdk-3.1.0-1-minishift-windows-amd64.exe** (for Windows). Assuming minishift executable is in the Downloads directory, follow the procedure for your operating system:

For Red Hat Enterprise Linux:

```
$ mkdir -p ~/bin
$ cp ~/Downloads/cdk-3.1.0-1-minishift* ~/bin/minishift
$ chmod +x ~/bin/minishift
$ export PATH=$PATH:$HOME/bin
$ echo 'export PATH=$PATH:$HOME/bin' >> ~/.bashrc
```

For macOS:

```
$ mkdir -p ~/bin
$ cp ~/Downloads/cdk-3.1.0-1-minishift* ~/bin/minishift
$ chmod +x ~/bin/minishift
$ export PATH=$PATH:$HOME/bin
$ echo export PATH=$PATH:$HOME/bin >> ~/.bash_profile
```

For Windows:

Create the desired directory and copy the downloaded minishift binary to the directory, renaming the binary to **minishift**. Add the directory path to the Windows path variable. This directory MUST be on the C: drive!

If it's difficult to get **minishift** in your PATH, you can simply run it from the current directory as **./minishift** (or .\minishift in some Windows shells).

### 1.3.4. Step 3: Set up CDK

The **`minishift setup-cdk`** command gets and configures the components needed to run Container Development Kit on your system. By default, it places Container Development Kit content in your ~/.minishift directory. One thing you should know about this directory:

- To use a directory other than ~/.minishift, you must set the **`--minishift-home`** flag and the **`MINISHIFT_HOME`** environment variable, as described in Environment Variables.

Run the following command to set up Container Development Kit for Red Hat Enterprise Linux:

```
$ minishift setup-cdk
Setting up CDK 3 on host using '/home/joe/.minishift' as Minishift's home
directory
Copying minishift-rhel7.iso to '/home/joe/.minishift/cache/iso/minishift-
rhel7.iso'
```

```
Copying oc to '/home/joe/.minishift/cache/oc/v3.6.173.0.21/oc'
Creating configuration file '/home/joe/.minishift/config/config.json'
Creating marker file '/home/joe/.minishift/cdk'
Default add-ons anyuid, admin-user, xpaas installed
Default add-ons anyuid, admin-user, xpaas enabled
CDK 3 setup complete.
```

As you can see from the output, a **.minishift** directory is created in the user's home directory to hold various CDK components, and appropriate files and setting are stored there.

For Windows or macOS: Running the same **minishift setup-cdk** command on Windows and Mac results in slightly different output, based on some different components and pathnames.

### 1.3.5. Step 4: Start CDK

The **`minishift start`** command launches Container Development Kit, which consists of a Red Hat Enterprise Linux virtual machine running OpenShift. Follow these steps to launch Container Development Kit.

1. If you are using the default hypervisor for your operating system (KVM for Linux, xhyve for macOS, or Hyper-V for Windows), you can skip this step. If you have set up VirtualBox as your hypervisor, you need to configure the VirtualBox Driver. To switch the hypervisor used by Container Development Kit to VirtualBox, you have two choices.
   - Temporary: Add the **`--vm-driver virtualbox`** option to the **`minishift start`** command line to use VirtualBox immediately.
   - Persistent: To persistently change the hypervisor, run the **`minishift config set vm-driver virtualbox`** command. See the Persistent Configuration section for examples.
2. Registration. By default, **minishift start** prompts you for your Red Hat account username and password. You can enter that information or choose instead to:
   - Skip registration: Add the **`--skip-registration`** option to **minishift start** to not register the Container Development Kit VM.
   - Register permanently: You can export registration information so that minishift picks it up automatically, each time it starts as shown here:
     For Red Hat Enterprise Linux:

     ```
     $ export MINISHIFT_USERNAME=<RED_HAT_USERNAME>
     $ export MINISHIFT_PASSWORD='<RED_HAT_PASSWORD>'
     $ echo export MINISHIFT_USERNAME=$MINISHIFT_USERNAME >> ~/.bashrc
     $ echo export MINISHIFT_PASSWORD=$MINISHIFT_PASSWORD >> ~/.bashrc
     ```

     For macOS:

     ```
     export MINISHIFT_USERNAME='<RED_HAT_USERNAME>'
     export MINISHIFT_PASSWORD='<RED_HAT_PASSWORD>'
     echo export MINISHIFT_USERNAME=$MINISHIFT_USERNAME >>
     ~/.bash_profile
     echo export MINISHIFT_PASSWORD=$MINISHIFT_PASSWORD >>
     ~/.bash_profile
     ```

     For Windows:

Set these two variables: MINISHIFT_USERNAME=<RED_HAT_USERNAME> and MINISHIFT_PASSWORD=<RED_HAT_PASSWORD>.

3. Run **minishift start** to set up and start the virtual machine with the default configuration. See the description of **minishift start** in the minishift start command reference to see other options to modify **minishift start**. Here is an example of how that would look run from a Red Hat Enterprise Linux system.

```
...
-- Minishift VM will be configured with ...
   Memory:    4 GB
   vCPUs :    2
   Disk size: 20 GB
-- Starting Minishift VM ........................... OK
-- Registering machine using subscription-manager
   Registration in progress ..................... OK [42s]
-- Checking for IP address ... OK
-- Checking if external host is reachable from the Minishift VM ...
...
Extracting
Image pull complete
OpenShift server started.
The server is accessible via web console at:
    https://192.168.42.60:8443
You are logged in as:
    User:     developer
    Password: developer
To login as administrator:
    oc login -u system:admin
...
```

4. Run this command to see if the virtual machine is running:

```
$ minishift status
Running
```

For macOS and Windows system, you see similar output from **`minishift start`** when run in a macOS terminal or Windows command prompt or PowerShell, respectively.

### 1.3.6. Step 5: Configure CDK

With CDK virtual machine running, you can configure and start using CDK through two primary interfaces: **oc** (OpenShift client binary command) or the **OpenShift web console**.

- **oc**: Use **`minishift oc-env`** to display the command you need to type into your shell in order to add the **oc** binary to your **PATH** environment variable. The output of **`oc-env`** will differ depending on your user directory, operating system and shell type.
  For Red Hat Enterprise Linux:

```
$ minishift oc-env
export PATH="/home/joe/.minishift/cache/oc/v3.6.173.0.21:$PATH"
# Run this command to configure your shell:
# eval $(minishift oc-env)
```

To use that information to add the PATH to the oc command permanently to the .bashrc file, type the following:

```
$ echo export
PATH=\"/home/joe/.minishift/cache/oc/v3.6.173.0.21:\$PATH\" >>
~/.bashrc
$ source ~/.bashrc
```

For macOS:

```
$ minishift oc-env
export PATH="/home/joe/.minishift/cache/oc/v3.6.173.0.21:$PATH"
# Run this command to configure your shell:
# eval $(minishift oc-env)
$ echo export
PATH=/Users/joe/.minishift/cache/oc/v3.6.173.0.21:\$PATH >>
~/.bash_profile
$ source ~/.bash_profile
```

For Windows:

Set the oc binary path in the environment path variable.

For more information about interacting with OpenShift from the command-line interface, see the OpenShift Client Binary section.

- **OpenShift web console**: To access OpenShift from the web console, you can either:
  - Run **minishift console** to open the OpenShift web console from your default browser or
  - Open a browser to the URL output from the **minishift start** command (for example, https://192.168.42.60:8443).

  From the web console screen, enter the user name (developer) and password (developer) displayed from the output.

You are now ready to start using OpenShift using either of those two interfaces.

### 1.3.7. Deploying a Sample Application

OpenShift provides various sample applications, such as templates, builder applications, and quickstarts. To begin creating an application, see :

- **Creating an Application Using the CLI** to use the oc command to create a new application.
- **Creating an Application Using the Web Console** to use the OpenShift web user interface to create applications.

The following steps describe how to deploy a sample Node.js application from the command-line.

1. Create a Node.js example app.

   ```
   $ oc new-app https://github.com/openshift/nodejs-ex -l name=myapp
   ```

2. Track the build log until the app is built and deployed.

```
$ oc logs -f bc/nodejs-ex
```

3. Expose a route to the service.

```
$ oc expose svc/nodejs-ex
```

4. Access the application.

```
$ minishift openshift service nodejs-ex --in-browser
```

5. If you are done with the VM, use **`minishift stop`** to stop the VM temporarily, so you can return to the same state later (with **`minishift start`**).

```
$ minishift stop
Stopping local OpenShift cluster...
Stopping "minishift"...
```

See minishift stop for details on managing subscriptions as you stop and start the VM.

For more information about creating applications in OpenShift, see Creating New Applications in the OpenShift documentation.

## 1.4. UNINSTALLING CONTAINER DEVELOPMENT KIT

### 1.4.1. Overview

This section describes how you can uninstall the minishift binary and delete associated files.

### 1.4.2. Uninstalling Container Development Kit

1. Delete CDK VM and any VM-specific files.

```
$ minishift delete
```

This command deletes everything in the ***CDK_HOME/.minishift/machines/minishift*** directory. Other cached data and the persistent configuration are not removed.

2. To completely uninstall Container Development Kit, delete everything in the ***MINISHIFT_HOME*** directory (default ***~/.minishift***) and ***~/.kube***, run the following commands:
**For Red Hat Enterprise Linux and macOS:**

```
$ rm -rf ~/.minishift
$ rm -rf ~/.kube
```

**For Windows powershell:**

Replace <MINISHIFT_HOME> with the location of your home directory.

```
PS C:\> Remove-Item -Recurse -Force C:\<MINISHIFT_HOME>\.minishift\
PS C:\> Remove-Item -Recurse -Force C:\<MINISHIFT_HOME>\.kube\
```

**For Windows command prompt:**

Replace <MINISHIFT_HOME> with the location of your home directory. (You may need to use the **`del /s`** command instead.)

```
c:\> rm -r c:\<MINISHIFT_HOME>\.minishift
c:\> rm -r c:\<MINISHIFT_HOME>\.kube
```

3. With your hypervisor management tool, confirm that there are no remaining artifacts related to CDK VM. For example, if you use KVM, you need to run the **`virsh`** command.

# CHAPTER 2. INTERACTING WITH OPENSHIFT

Container Development Kit creates a virtual machine and provisions a local, single-node OpenShift cluster in this VM. The following sections describe how Container Development Kit can assist you in interacting and configuring your local OpenShift cluster.

For details about managing Container Development Kit VM, see the Managing Container Development Kit section.

## 2.1. USING THE OPENSHIFT CLIENT BINARY (OC)

### 2.1.1. Overview

The **`minishift start`** command creates an OpenShift cluster using the cluster up approach. For this purpose it copies the **oc** binary onto your host.

The **oc** binary is located in the ***~/.minishift/cache/oc/v1.5.1*** directory, assuming that you use CDK's default version of OpenShift. You can add this binary to your **PATH** using **`minishift oc-env`**, which displays the command you need to type into your shell.

The output of **`minishift oc-env`** differs depending on the operating system and the shell type.

```
$ minishift oc-env
export PATH="/Users/john/.minishift/cache/oc/v3.6.173.0.21:$PATH"
# Run this command to configure your shell:
# eval $(minishift oc-env)
```

### 2.1.2. Container Development Kit CLI Profile

As a part of the **`minishift start`** command, a Container Development Kit CLI profile is also created. This profile, also known as a *context*, contains the configuration to communicate with your OpenShift cluster.

Container Development Kit activates this context automatically, but if you need to switch back to it after, for example, logging into another OpenShift instance, you can run:

```
$ oc config use-context minishift
```

For an introduction to **oc** usage, see the Get Started with the CLI topic in the OpenShift documentation.

### 2.1.3. Logging Into the Cluster

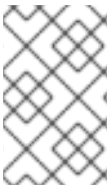
By default, **`cluster up`** uses AllowAllPasswordIdentityProvider to authenticate against the local cluster. This means any non-empty user name and password can be used to login to the local cluster.

The recommended user name and password is **`developer`** and **`developer`**, respectively. This is because they are already assigned to the default project **myproject** and also can impersonate the administrator. This allows you to run administrator commands using the **`--as system:admin`** parameter.

To login as administrator, use the system account:

```
$ oc login -u system:admin
```

In this case, client certificates are used. The certificates are stored in *~/.kube/config*. The **cluster up** command installs the appropriate certificates as a part of the bootstrap.



**NOTE**

If you run the command **oc login -u system -p admin**, you will log in but not as an administrator. Instead, you will be logged in as an unprivileged user with no particular rights.

To view the available login contexts, run:

```
$ oc config view
```

### 2.1.4. Accessing the Web Console

To access the OpenShift Web console, you can run this command in a shell after running the minishift command to get the URL of the Web console:

```
$ minishift console --url
```

Alternatively, after starting Container Development Kit, you can use the following command to directly open the console in a browser:

```
$ minishift console
```

### 2.1.5. Accessing OpenShift Services

To access a service that is exposed with a route, run this command in a shell:

```
$ minishift openshift service [-n NAMESPACE] [--url] NAME
```

To list all the services in the OpenShift instance, run this command in a shell:

```
$ minishift openshift service list
```

For more information on exposing services refer also to Exposing Services.

### 2.1.6. Viewing OpenShift Logs

To access OpenShift logs, run the **logs** command after starting Container Development Kit:

```
$ minishift logs
```

### 2.1.7. Updating OpenShift Configuration

While OpenShift is running, you can view and change the master or the node configuration of your cluster.

To view the OpenShift master configuration file ***master-config.yaml***, run the following command:

```
$ minishift openshift config view
```

To show the node configuration instead of the master configuration, specify the **`target`** flag.

**NOTE**

After you update the OpenShift configuration, OpenShift will transparently restart.

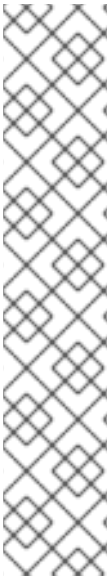
#### 2.1.7.1. Example: Configuring cross-origin resource sharing

In this example, you configure cross-origin resource sharing (CORS) by updating the OpenShift master configuration to allow additional IP addresses to request resources.

By default, OpenShift only allows cross-origin resource requests from the IP address of the cluster or from localhost. This setting is stored in the **`corsAllowedOrigins`** property of the master configuration file ***master-config.yaml***.

To change the property value and allow cross-origin requests from all domains, run the following command:

```
$  minishift openshift config set --patch '{"corsAllowedOrigins": [".*"]}'
```



**NOTE**

If you get the error *The specified patch need to be a valid JSON.*when you run the above command, you need to modify the above command depending on your Operating System, your shell environment and its interpolation behavior.

For example, if you use PowerShell on Windows 7 or 10, modify the above command to:

```
$ minishift openshift config set --patch
'{\"corsAllowedOrigins\": [\".*\"]}'
```

If you use Command Prompt you might need to use:

```
$ minishift openshift config set --patch "
{\"corsAllowedOrigins\": [\".*\"]}"
```

#### 2.1.7.2. Example: Changing the OpenShift routing suffix

In this example, you change the OpenShift routing suffix in the master configuration.

If you use a static routing suffix, you can set the **`routing-suffix`** flag as part of the **`minishift start`** command. By default, The minishift command uses a dynamic routing prefix based on nip.io, in which the IP address of the VM is a part of the routing suffix, for example **192.168.99.103.nip.io**.

If you experience issues with **nip.io**, you can use xip.io, which is based on the same principles.

To set the routing suffix to **xip.io**, run the following command:

```
$ minishift openshift config set --patch '{"routingConfig": {"subdomain":
"<IP-ADDRESS>.xip.io"}}'
```

Make sure to replace **IP-ADDRESS** in the above example with the IP address of your Container Development Kit VM. You can retrieve the IP address by running the **minishift ip** command.

## 2.2. EXPOSING SERVICES

### 2.2.1. Overview

There are several ways you can expose your service after you deploy it on OpenShift. The following sections describes the various methods and when to use them.

### 2.2.2. Routes

If you are deploying a Web application, the most common way to expose it is by a route. A route exposes the service as a host name. You can create a route using the Web console or the CLI:

```
oc expose svc/frontend --hostname=www.example.com
```

To see a full example of creating an application and exposing it with a route, see the Deploying a Sample Application section.

### 2.2.3. NodePort Services

In case the service you want to expose is not HTTP based, you can create a **NodePort** service. In this case, each OpenShift node will proxy that port into your service. To access this port on your Container Development Kit VM, you need to configure an Ingress IP Self-Service using **oc expose** with the parameter **type=LoadBalancer**.

A common use-case for Ingress IP Self-Service is the ability to expose a database service. The following example shows the complete workflow to create and expose a MariaDB instance using the minishift and oc commands:

```
$ minishift start
$ eval $(minishift oc-env)
$ oc new-app -e MYSQL_ROOT_PASSWORD=admin
https://raw.githubusercontent.com/openshift/origin/master/examples/db-
templates/mariadb-persistent-template.json
$ oc rollout status -w dc/mariadb
$ oc expose dc mariadb --type=LoadBalancer --name=mariadb-ingress
$ oc export svc mariadb-ingress
 ....
ports:
    - nodePort: 30907
 ....
```

After the service is exposed, you can access MariaDB with the **mysql** CLI using Container Development Kit VM IP and the exposed NodePort service.

```
$ mysql --user=root --password=foo --host=$(minishift ip) --port=30907
```

### 2.2.4. Port Forwarding

If you want to quickly access a port of a specific pod of your cluster, you can also use the **`oc port-forward`** command of the OpenShift CLI.

```
$ oc port-forward POD [LOCAL_PORT:]REMOTE_PORT
```

## 2.3. ACCESSING THE OPENSHIFT DOCKER REGISTRY

### 2.3.1. Overview

OpenShift provides an integrated Docker registry which can be used for development as well. Images present in the registry can directly be used for applications, speeding up the local development workflow.

### 2.3.2. Logging Into the Registry

1. Run the **minishift start** command and add the **oc** binary to the PATH (based on output from the **minishift oc-env** command)
2. Make sure your shell is configured to reuse Container Development Kit docker daemon .
3. Log into the OpenShift Docker registry.

```
$ docker login -u developer -p $(oc whoami -t) $(minishift
openshift registry)
```

### 2.3.3. Deploying Applications

The following example shows how to deploy an OpenShift application directly from a locally-built docker image. This example uses the OpenShift project **myproject**. This project is automatically created by **`minishift start`**.

1. Make sure your shell is configured to reuse Container Development Kit docker daemon .
2. Build the docker image as usual.
3. Tag the image against the OpenShift registry.

```
$ docker tag my-app $(minishift openshift registry)/myproject/my-
app
```

4. Push the image to the registry to create an image stream with the same name as the application.

```
$ docker push $(minishift openshift registry)/myproject/my-app
```

5. Create an application from the image stream and expose the service.

```
$ oc new-app --image-stream=my-app --name=my-app
$ oc expose service my-app
```

# CHAPTER 3. USING CONTAINER DEVELOPMENT KIT

This section describes how to use the minishift CLI and manage your Container Development Kit VM. It also helps troubleshooting common issues.

For details about using the minishift and oc commands to manage your local OpenShift cluster, see the Interacting with OpenShift section.

## 3.1. MANAGING CONTAINER DEVELOPMENT KIT

### 3.1.1. Overview

When you use the minishift command, you interact with two components:

- a virtual machine (VM) created by the minishift command
- the OpenShift cluster provisioned by the minishift command within the VM

The following sections contain information about managing Container Development Kit VM. For details about using Container Development Kit to manage your local OpenShift cluster, see the Interacting with OpenShift section.

### 3.1.2. Container Development Kit Life-cycle

Use different options to the minishift command to manage Container Development Kit throughout its life-cycle.

#### 3.1.2.1. The minishift setup-cdk Command

The **`minishift setup-cdk`** command creates and configures the Container Development Kit VM and provisions a local, single-node OpenShift cluster within the Container Development Kit VM.

The command also copies the **oc** binary to your host so that you can interact with OpenShift through the **oc** command-line tool or through the Web console, which can be accessed through the URL provided in the output of the **`minishift start`** command.

#### 3.1.2.2. The minishift start Command

The **`minishift start`** command actually starts up the Container Development Kit VM, after it is has been provisioned by the **`minishift setup-cdk`** command.

#### 3.1.2.3. The minishift stop Command

The **`minishift stop`** command stops your OpenShift cluster and shuts down the Container Development Kit VM, but preserves the OpenShift cluster state.

Running the **`minishift start`** command again will restore the OpenShift cluster, allowing you to continue working from the last session. However, you must enter the same parameters that you used in the original start command.

So not to consume RHEL entitlements when the VM is not running, the VM is unregistered by default with **`minishift stop`**. To retain entitlement credentials (for example, if you want to work offline temporarily), follow these steps:

1. Stop the VM and skip unregistration:

```
$ minishift stop --skip-unregistration
```

2. Start the VM while offline (skipping registration):

```
$ minishift start --skip-registration
```

Repeat the **`--skip-unregistration`** when you stop the VM. The next time you start the VM with the system online, simply type **`minishift start`** to have the VM use the existing credentials again.

Efforts to further refine the minishift start and stop experience are in progress. For details, see the GitHub issue #179.

#### 3.1.2.4. The minishift delete Command

The **`minishift delete`** command deletes the OpenShift cluster, and also shuts down and deletes the Container Development Kit VM. No data or state are preserved. However, some files may be left behind in the .minishift and .kde directories in your home directory. So you can remove those directories to completely clean up your system.

### 3.1.3. Runtime Options

The runtime behavior of the minishift command can be controlled through flags, environment variables, and persistent configuration options.

The following precedence order is applied to control the behavior of the minishift command. Each action in the following list takes precedence over the action below it:

1. Use command-line flags as specified in the Flags section.
2. Set environment variables as described in the Environment Variables section.
3. Use persistent configuration options as described in the Persistent Configuration section.
4. Accept the default value as defined by the minishift command.

#### 3.1.3.1. Flags

You can use command line flags with the minishift command to specify options and direct its behavior. This has the highest precedence. Almost all commands have flags, although different commands might have different flags. Some of the commonly-used command line flags of the **`minishift start`** command are **`cpus`**, **`memory`** or **`vm-driver`**.

#### 3.1.3.2. Environment Variables

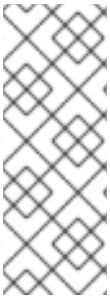
The minishift command allows you to specify command-line flags you commonly use through environment variables. To do so, apply the following rules to the flag you want to set as an environment variable.

1. Apply **`MINISHIFT_`** as a prefix to the flag you want to set as an environment variable. For example, the **`vm-driver`** flag of the **`minishift start`** command becomes **`MINISHIFT_vm-driver`**.

2. Use uppercase characters for the flag, so **`MINISHIFT_vm-driver`** in the above example becomes **`MINISHIFT_VM-DRIVER`**.

3. Replace **`-`** with **`_`**, so **`MINISHIFT_VM-DRIVER`** becomes **`MINISHIFT_VM_DRIVER`**.

Environment variables can be used to replace any option of any minishift command.



**NOTE**

You can use the **`MINISHIFT_HOME`** environment variable, to choose a different home directory for Container Development Kit. By default, the minishift command places all runtime state into ***~/.minishift***. This environment variable is currently experimental and semantics might change in future releases. To use **`MINISHIFT_HOME`**, you should set the new home directory when you first set up Container Development Kit. For example, this sets the minishift home directory to ~/.mynewdir on a Linux system:

```
$ minishift setup-cdk --minishift-home ~/.mynewdir
$ export MINISHIFT_HOME=~/.mynewdir
$ echo "export MINISHIFT_HOME=~/.mynewdir" >> ~/.bashrc
```

### 3.1.3.3. Persistent Configuration

Using persistent configuration allows you to control the Container Development Kit behavior without specifying actual command line flags, similar to the way you use environment variables.

The minishift command maintains a configuration file in ***$MINISHIFT_HOME/config/config.json***. This file can be used to set commonly-used command-line flags persistently.

**NOTE**

Persistent configuration can only be applied to the set of supported configuration options that are listed in the synopsis of the **`minishift config`** sub-command. This is different from environment variables, which can be used to replace any option of any command.

#### 3.1.3.3.1. Setting Persistent Configuration Values

The easiest way to change a persistent configuration option is with the **`minishift config set`** sub-command. For example:

```
# Set default memory 4096 MB
$ minishift config set memory 4096
```

Flags which can be used multiple times per command invocation, like **`docker-env`** or **`insecure-registry`**, need to be comma-separated when used with the **`config set`** command. For example, from the CLI, you can use **`insecure-registry`** like this:

```
$ minishift start --insecure-registry hub.foo.com --insecure-registry
hub.bar.com
```

If you want to configure the same registries in the persistent configuration, you would run:

```
$ minishift config set insecure-registry hub.foo.com,hub.bar.com
```

In another example, to identify the hypervisor as VirtualBox at Container Development Kit start time, use the **`--vm-driver virtualbox`** option as follows:

```
$ minishift start --vm-driver virtualbox
```

To switch from the default hypervisor to have Container Development Kit persistently use VirtualBox as the hypervisor, type:

```
$ minishift config set vm-driver virtualbox
```

To view all persistent configuration values, you can use the **`minishift config view`** sub-command:

```
$ minishift config view
- memory: 4096
```

Alternatively, you can display a single value with the **`minishift config get`** sub-command:

```
$ minishift config get memory
4096
```

#### 3.1.3.3.2. Unsetting Persistent Configuration Values

To remove a persistent configuration option, you can use the **`minishift config unset`** sub-command. For example:

```
$ minishift config unset memory
```

### 3.1.3.4. Driver-Specific Environment Variables

You can also set driver-specific environment variables. Each docker-machine driver supports its own set of options and variables. A good starting point is the official docker-machine driver documentation.

xhyve and KVM documentation is available under their respective GitHub repository docker-machine-driver-xhyve and docker-machine-kvm.

To use driver-specific options, make sure to export the variable as defined in its driver documentation before running **`minishift start`**. For example, the xhyve experimental NFS sharing can be enabled by executing:

```
$ export XHYVE_EXPERIMENTAL_NFS_SHARE=true
$ minishift start --vm-driver xhyve
```

#### CAUTION

Driver-specific options might overlap with values specified using minishift-specific flags and environment variables. Examples are memory size, cpu count, and so on. In this case, driver-specific environment variables will override minishift-specific settings.

## 3.1.4. Caching OpenShift images (experimental)

To speed up provisioning of the OpenShift cluster and to minimize network traffic, the core OpenShift images can be cached on the host. This feature is considered experimental and needs to be explicitly enabled using the **`minishift config set`** command:

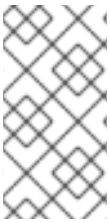
```
$ minishift config set image-caching true
```

Once enabled, caching occurs transparently, in a background process, the first time you use the **`minishift start`** command. Once the images are cached under *$MINISHIFT_HOME/cache/images*, successive Container Development Kit VM creations will use these cached images.

Each time an image exporting background process runs, a log file is generated under *$MINISHIFT_HOME/logs* which can be used to verify the progress of the export.

You can disable the caching of the OpenShift images by setting **`image-caching`** to **`false`** or removing the setting altogether using **`minishift config unset`**:

```
$ minishift config unset image-caching
```



**NOTE**

Image caching is considered experimental and its semantics and API is subject to change. The aim is to allow caching of arbitrary images, as well as using a better format for storing the images on the host. You can track the progress on this feature on the GitHub issue #952.

### 3.1.5. Persistent Volumes

As part of the OpenShift cluster provisioning, 100 persistent volumes are created for your OpenShift cluster. This allows applications to make persistent volumes claims. The location of the persistent data is determined in the **`host-pv-dir`** flag of the **`minishift start`** command and defaults to ***/var/lib/minishift/openshift.local.pv*** on Container Development Kit VM.

### 3.1.6. HTTP/HTTPS Proxies

If you are behind an HTTP/HTTPS proxy, you need to supply proxy options to allow Docker and OpenShift to work properly. To do this, pass the required flags during **`minishift start`**.

For example:

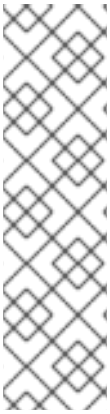
```
$ minishift start --http-proxy http://YOURPROXY:PORT --https-proxy
https://YOURPROXY:PORT
```

In an authenticated proxy environment, the **`proxy_user`** and **`proxy_password`** must be a part of proxy URI.

```
 $ minishift start --http-proxy http://<proxy_username>:
<proxy_password>@YOURPROXY:PORT \
                   --https-proxy https://<proxy_username>:
<proxy_password>@YOURPROXY:PORT
```

You can also use the **`--no-proxy`** flag to specify a comma-separated list of hosts that should not be proxied.

Using the proxy options will transparently configure the docker daemon as well as OpenShift to use the specified proxies.



**NOTE**

- **`minishift start`** honors the environment variables **`HTTP_PROXY`**, **`HTTPS_PROXY`** and **`NO_PROXY`**. If these variables are set, they are implicitly used during **`minishift start`** unless explicitly overridden by the corresponding command line flags.
- Use the **`minishift start --ocp-tag`** flag to request a specific version of OpenShift Container Platform. You can list all Container Development Kit-compatible OpenShift Container Platform versions with the **`minishift openshift version list`** command.

### 3.1.7. Networking

The Container Development Kit VM is exposed to the host system with a host-only IP address that can be obtained with the **`minishift ip`** command.

### 3.1.8. Connecting to Container Development Kit VM with SSH

You can use the **`minishift ssh`** command to interact with Container Development Kit VM.

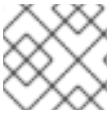
You can run **`minishift ssh`** without a sub-command to open an interactive shell and run commands on Container Development Kit VM in the same way that you run commands interactively on any remote machine using SSH.

You can also run **`minishift ssh`** with a sub-command to send the sub-command directly to Container Development Kit VM and return the result to your local shell. For example:

```
$ minishift ssh -- docker ps
CONTAINER ID  IMAGE                     COMMAND        CREATED
STATUS        NAMES
71fe8ff16548  .../ose-pod:v3.6.173.0.21 "/usr/bin/pod" 4 minutes ago  Up 4
minutes  k8s_POD...
```

## 3.2. ADD-ONS

### 3.2.1. Overview



**NOTE**

This feature is still considered experimental and might change in future releases.

Container Development Kit allows you to extend the vanilla OpenShift setup provided by **cluster up** with an add-on mechanism.

Add-ons are directories that contain a text file with the ***.addon*** extension. The directory can also contain other resource files such as JSON template files. However, only one ***.addon*** file is allowed per add-on.

The following example shows the contents of an add-on file, including the name and description of the add-on, additional metadata, and the actual add-on commands to apply.

**Example: anyuid add-on definition file**

```
# Name: anyuid
(1)
# Description: Allows authenticated users to run images under a non pre-
allocated UID    (2)
# Required-vars: ACME_TOKEN
(3)

oc adm policy add-scc-to-group anyuid system:authenticated
(4)
```

(1) (Required) Name of the add-on.

(2) (Required) Description of the add-on.

(3) (Optional) Comma separated list of required interpolation variables. See Variable Interpolation

(4) Actual add-on command. In this case, the command executes the **oc** binary.

**NOTE**

Comment lines, starting with the '#' character, can be inserted at anywhere in the file.

Enabled add-ons are applied during **`minishift start`**, immediately after the initial cluster provisioning is successfully completed.

### 3.2.2. Add-on Commands

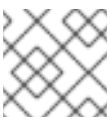
This section describes the commands that an add-on file can contain. They are forming a mini-DSL for Container Development Kit add-ons:

**ssh**

If the add-on command starts with **ssh**, you can run any command within the minishift-managed VM. This is similar to running **`minishift ssh`** and then executing any command on the VM. For more information about the **`minishift ssh`** command usage, see Connecting to Container Development Kit VM with SSH.

**oc**

If the add-on command starts with **oc**, it uses the **oc** binary that is cached on your host to execute the specified **oc** command. This is similar to running **`oc --as system:admin ...`** from the command-line.



**NOTE**

The **oc** command is executed as **system:admin**.

**openshift**

If the add-on command starts with **openshift**, you can run the **openshift** binary within the container that runs OpenShift. This means that any file parameters or other system-specific parameters must match the environment of the container instead of your host.

**docker**

If the add-on command starts with **docker**, it executes a **docker** command against the docker daemon within Container Development Kit VM. This is the same daemon on which the single-node OpenShift cluster is running as well. This is similar to running **eval $(minishift docker-env)** on your host and then executing any **docker** command. See also **minishift docker-env**.

**echo**

If the add-on command starts with **echo**, the arguments following the **echo** command are printed to the console. This can be used to provide additional feedback during add-on execution.

**sleep**

If the add-on command starts with **sleep**, it waits for the specified number of seconds. This can be useful in cases where you know that a command such as **oc** might take a few seconds before a certain resource can be queried.

**NOTE**

Trying to use an undefined command will cause an error when the add-on gets parsed.

### 3.2.3. Variable Interpolation

Container Development Kit allows the use of variables within the add-on commands. Variables have the format **#{<variable-name>}**. The following example shows how the OpenShift routing suffix can be interpolated into an **openshift** command to create a new certificate as part of securing the OpenShift registry. The used variable **#{routing-suffix}** is part of the built-in add-on variables.

**Example: Usage of the routing-suffix variable**

```
$ openshift admin ca create-server-cert \
  --signer-cert=/var/lib/origin/openshift.local.config/master/ca.crt \
  --signer-key=/var/lib/origin/openshift.local.config/master/ca.key \
  --signer-
serial=/var/lib/origin/openshift.local.config/master/ca.serial.txt \
  --hostnames='docker-registry-default.#{routing-suffix},docker-
registry.default.svc.cluster.local,172.30.1.1' \
  --cert=/etc/secrets/registry.crt \
  --key=/etc/secrets/registry.key
```

#### 3.2.3.1. Built-in Variables

There exist several built-in variables which are available for interpolation at all times. The following table shows these variables.

**Table 3.1. Supported built-in add-on variables**

| Variable | Description |
|---|---|
| ip | IP of Container Development Kit VM. |

| Variable | Description |
|---|---|
| routing-suffix | OpenShift routing suffix for the application. |
| addon-name | Name of the current add-on. |

#### 3.2.3.2. Dynamic Variables

The commands **`minishift start`** as well as **`minishift addons apply`** also provide an **`--addon-env`** flag which allows to dynamically pass variables for interpolation, for example:

```
$ minishift addons apply --addon-env PROJECT_USER=john acme
```

The **`--addon-env`** flag can be specified multiple times to define more than one variables for interpolation.

Specifying dynamic variables also works in conjunction with setting persistent configuration values.

```
$ minishift config set addon-env PROJECT_USER=john
$ minishift addons apply acme
```

**TIP**

Multiple variables need to be comma separated when the **`minishift config set`** is used.

There is also the possibility to dynamically interpolate a variable with the value of an environment variable at the time the add-on gets applied. For this the variable value needs to be prefixed with *env*.

```
$ minishift config set addon-env PROJECT_USER=env.USER          (1)
$ minishift addons apply acme                                   (2)
```

(1) Using the *env* prefix ensures that instead of literally replacing '#{PROJECT_USER}' with 'env.USER', the value of the environment variable **USER** is used. If the environment variable is not set not, interpolation does not occur.

(2) When the add-on is applied, each occurrence of **#{PROJECT_USER}** within an add-on command gets replaced with the value of the environment variable **USER**.

As an add-on developer, you can enforce that a variable value is provided when the add-on gets applied by adding the varaible name to the *Required-Vars* metadata header. Multiple variables need to be comma separated.

```
# Name: acme
# Description: ACME add-on
# Required-Vars: PROJECT_USER
```

### 3.2.4. Default Add-ons

Container Development Kit provides a set of built-in add-ons that offer some common OpenShift customization to assist with development. During **minishift setup-cdk**, minishift automatically installs and enables xpaas, anyuid, and admin-user add-ons. To install the default add-ons, run:

```
$ minishift addons install --defaults
```

This command extracts the default add-ons to the add-on installation directory ***$MINISHIFT_HOME/addons***. To view the list of installed add-ons, you can then run:

```
$ minishift addons list --verbose=true
```

This command prints a list of installed add-ons. You should at least see the **anyuid** add-on listed. This is an important add-on that allows you to run images that do not use a pre-allocated UID. By default, this is not allowed in OpenShift.

### 3.2.5. Enabling and Disabling Add-ons

Add-ons are enabled with the **`minishift addons enable`** command and disabled with the **`minishift addons disable`** command. Enabled add-ons automatically get executed during **`minishift start`**.

The following examples show how to enable and disable the **anyuid** add-on.

**Example: Enabling the anyuid add-on**

```
$ minishift addons enable anyuid
```

**Example: Disabling the anyuid add-on**

```
$ minishift addons disable anyuid
```

#### 3.2.5.1. Add-on Priorities

When you enable an add-on, you can also specify a priority, which determines the order that the add-ons are applied.

The following example shows how to enable the **registry** add-on with a higher priority value.

**Example: Enabling the registry add-on with priority**

```
$ minishift addons enable registry --priority=5
```

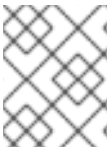
The add-on priority attribute determines the order in which add-ons are applied. By default, an add-on has the priority 0. Add-ons with a lower priority value are applied first.

In the following example, the **anyuid**, **registry**, and **eap** add-ons are enabled with the respective priorities of 0, 5 and 10. This means that **anyuid** is applied first, followed by **registry**, and lastly the **eap** add-on.

**Example: List command output with explicit priorities**

```
$ minishift addons list
- anyuid         : enabled    P(0)
```

```
- registry       : enabled    P(5)
- eap            : enabled    P(10)
```



**NOTE**

If two add-ons have the same priority the order in which they are getting applied is not determined.

### 3.2.6. Applying Add-ons

Add-ons can be explicitly executed with the **`minishift addons apply`** command. You can use the **`apply`** command for both enabled and disabled add-ons. To apply multiple add-ons with a single command, specify add-on names separated by space.

The following example shows how to explicitly apply the **anyuid** and the **admin-user** add-ons.

**Example: Applying anyuid and admin-user add-ons**

```
$ minishift addons apply anyuid admin-user
```

### 3.2.7. Writing Custom Add-ons

To write a custom add-on, you should create a directory and in it create one (and only one) text file with the extension ***.addon***, for example ***admin-role.addon***.

This file needs to contain the **Name** and **Description** metadata fields, as well as the commands that you want to execute as a part of the add-on.

The following example shows the definition of an add-on that gives the developer user cluster-admin privileges.

**Example: Add-on definition for admin-role**

```
# Name: admin-role
# Description: Gives the developer user cluster-admin privileges

oc adm policy add-role-to-user cluster-admin developer
```

After you define the add-on, you can install it by running:

```
$ minishift addons install <ADDON_DIR_PATH>
```

**NOTE**

You can also edit your add-on directly in the minishift add-on install directory ***$MINISHIFT_HOME/addons***. Be aware that if there is an error in the add-on, it will not show when you run any **`addons`** commands, and it will not be applied during the **`minishift start`** process.

## 3.3. HOST FOLDERS

### 3.3.1. Overview

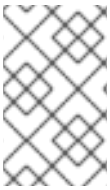
Host folders are directories on the host which are shared between the host and Container Development Kit VM. They allow for a two way file synchronization between host and VM. The following sections discuss the various types of host folders, driver provided host folders, as well as the `minishift hostfolder` command.

### 3.3.2. Driver-Provided Host Folders

Some drivers mount a default host folder into the VM in order to share files between the VM and the host. These folders are currently not configurable and differ for each driver and OS.

**Table 3.2. Driver-provided host folders**

| Driver | OS | HostFolder | VM |
|---|---|---|---|
| Xhyve | OSX | /Users | /Users |

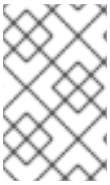


**NOTE**

- Host folder sharing is not implemented in the KVM and Hyper-V drivers. If you use one of these drivers, you need to use the `minishift hostfolder` command to set up and configure host folders.

### 3.3.3. The minishift hostfolder Command

Container Development Kit provides the `minishift hostfolder` command to list, add, mount, unmount and remove host folders. In contrast to the driver-provided host folders, you can use the `hostfolder` command to mount multiple shared folders onto custom specified mount points.



**NOTE**

Currently only CIFS is supported as a host folder type. Support for SSHFS-based host folders is in progress, as described in GitHub issue #317. If you want to manually set up SSHFS, see SSHFS Host Folders.

#### 3.3.3.1. Prerequisites

To use the `minishift hostfolder` command, you need to be able to share directories using CIFS. On Windows, CIFS is the default technology for sharing directories. For example, on Windows 10 the ***C:\Users*** folder is shared by default and can be accessed by locally-authenticated users.

It is also possible to use CIFS on macOS and Linux. On macOS you can enable CIFS-based shares under **System Preferences > Sharing**. See How to connect with File Sharing on your Mac for detailed setup instructions.

On Linux, follow your distribution-specific instructions to install Samba. Refer to Samba File and Print Server in RHEL to learn how to configure the Samba implementation of CIFS in Red Hat Enterprise Linux.

#### 3.3.3.2. Displaying Host Folders

The **`minishift hostfolder list`** command gives you an overview of the defined host folders, their names, mount points, remote paths and whether they are currently mounted.

An example output could look like:

```
$ minishift hostfolder list
Name        Mountpoint            Remote path              Mounted
myshare     /mnt/sda1/myshare     //192.168.1.82/MYSHARE   N
```

In this example, there is a host folder with the name ***myshare*** which mounts ***//192.168.1.82/MYSHARE*** onto ***/mnt/sda1/myshare*** in Container Development Kit VM. The share is currently not mounted.

**NOTE**

The remote path must be reachable from within the VM. In the example above, **192.168.1.82** is the IP of host within the LAN, which is one option you can use. You can use **`ifconfig`** (or **`Get-NetIPAddress | Format-Table`** on Windows) to determine a routable IP address.

#### 3.3.3.3. Adding Host Folders

The **`minishift hostfolder add`** command allows you to define a new host folder. This in an interactive process that queries the relevant details for a host folder based on CIFS.

**Adding a CIFS based hostfolder**

```
$ minishift hostfolder add myshare   (1)
UNC path: //192.168.99.1/MYSHARE     (2)
Mountpoint [/mnt/sda1/myshare]:      (3)
Username: john                       (4)
Password: [HIDDEN]                   (5)
Domain:                              (6)
Added: myshare
```

1. (Required) Actual **`minishift hostfolder add`** command that specifie a host folder with a name of ***myshare***.
2. (Required) The UNC path for the share.
3. The mount point within the VM. The default is ***/mnt/sda1/<host folder name>***.
4. (Required) The user name for the CIFS share.
5. (Required) The password for the CIFS share.
6. The domain of the share. Often this can be left blank, but for example on Windows, when your account is linked to a Microsoft account, you must use the Microsoft account email address as user name as well as your machine name as displayed by **`$env:COMPUTERNAME`** as a domain.

**TIP**

On Windows hosts, the **`minishift hostfolder add`** command also provides a **`users-share`** option. When this option is specified, no UNC path needs to be specified and the ***C:\Users*** is assumed.

#### 3.3.3.3.1. Instance-Specific Host Folders

By default, host folder definitions are persistent, similar to other persistent configuration options. This means that these host folder definitions will survive the deletion and subsequent re-creation of a Container Development Kit VM.

In some cases you might want to define a host folder just for a specific Container Development Kit instance. To do so, you can use the **`instance-only`** flag of the **`minishift hostfolder add`** command. Host folder definition that are created with the **`instance-only`** flag will be removed together with any other instance-specific state during **`minishift delete`**.

### 3.3.3.4. Mounting Host Folders

After you add host folders, you use the **`minishift hostfolder mount`** command to mount a host folder by its name:

```
$ minishift hostfolder mount myshare
Mounting 'myshare': '//192.168.99.1/MYSHARE' as '/mnt/sda1/myshare' ... OK
```

You can verify that the host folder is mounted by running:

```
$ minishift hostfolder list
Name       Mountpoint          Remote path              Mounted
myshare    /mnt/sda1/myshare   //192.168.99.1/MYSHARE   Y
```

Alternatively, you can list the actual content of the mounted host folder:

```
$ minishift ssh "ls -al /mnt/sda1/myshare"
```

#### 3.3.3.4.1. Auto-Mounting Host Folders

Host folders can also be mounted automatically each time you run **`minishift start`**. To set auto-mounting, you need to set the **`hostfolder-automount`** option in the minishift configuration file.

```
$ minishift config set hostfolders-automount true
```

After the **`hostfolders-automount`** option is set, Container Development Kit will attempt to mount all defined host folders during **`minishift start`**.

### 3.3.3.5. Unmounting Host Folders

You use the **`minishift hostfolder umount`** command to unmount a host folder.

```
$ minishift hostfolder umount myshare
Unmounting 'myshare' ... OK

$ minishift hostfolder list
Name       Mountpoint          Remote path              Mounted
```

```
myshare    /mnt/sda1/myshare    //192.168.99.1/MYSHARE    N
```

#### 3.3.3.6. Deleting Host Folders

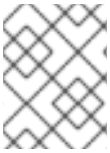
You use the **`minishift hostfolder remove`** command to remove a host folder definition.

```
$ minishift hostfolder list
Name        Mountpoint            Remote path              Mounted
myshare     /mnt/sda1/myshare     //192.168.1.82/MYSHARE   N

$ minishift hostfolder remove myshare
Removed: myshare

$ minishift hostfolder list
No host folders defined
```

#### 3.3.3.7. SSHFS Host Folders



**NOTE**

This host folder type is not supported by the **`minishift hostfolder`** command and requires manual configuration.

You can use SSHFS-based host folders if you have an SSH daemon running on your host. Normally, this prerequisite is met by default on Linux and macOS.

Most Linux distributions have an SSH daemon installed. If not, follow the instructions for your specific distribution to install an SSH daemon.

macOS also has a built-in SSH server. To use it, make sure that **Remote Login** is enabled in **System Preferences > Sharing**.

On Windows, you can install OpenSSH for Windows.

The following steps demonstrate how to mount host folders with SSHFS.

1. Run **`ifconfig`** (or **`Get-NetIPAddress`** on Windows) to determine the local IP address from the same network segment as your Container Development Kit instance.
2. Create a mountpoint and mount the shared folder.

   ```
   $ minishift ssh "sudo mkdir -p /Users/<username>"
   $ minishift ssh "sudo chown -R docker /Users"
   $ minishift ssh
   $ sshfs <username>@<IP>:/Users/<username>/ /Users
   ```

3. Verify the share mount.

   ```
   $ minishift ssh "ls -al /Users/<username>"
   ```

## 3.4. CONTAINER DEVELOPMENT KIT DOCKER DAEMON

### 3.4.1. Reusing the docker Daemon

When running OpenShift in a single VM, you can reuse the docker daemon managed by Container Development Kit for other docker use-cases as well. By using the same docker daemon as the one used in Container Development Kit, you can speed up your local development.

In order to configure your console to reuse Container Development Kit docker daemon, follow these steps:

1. Make sure that you have the docker client binary installed on your machine. For information about specific binary installations for your operating system, see the docker installation site. For a Red Hat Enterprise Linux system, install the docker package as described in Get Started with Docker-Formatted Container Images, but don't start the docker service on the host.

2. Start Container Development Kit with the **`minishift start`** command.

3. Run the **`minishift docker-env`** command to display the command you need to type into your shell in order to configure your Docker client. The command output will differ depending on OS and shell type.

```
$ minishift docker-env
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.99.101:2376"
export DOCKER_CERT_PATH="/Users/john/.minishift/certs"
export DOCKER_API_VERSION="1.24"
# Run this command to configure your shell:
# eval $(minishift docker-env)
```

4. Test the connection by running the following command:

```
$ docker ps
```

   If successful, the shell will print a list of running containers.

## 3.5. TROUBLESHOOTING CONTAINER DEVELOPMENT KIT

### 3.5.1. Overview

This section contains solutions to common problems that you might encounter while using Container Development Kit.

### 3.5.2. Special characters cause passwords to fail

Depending on your operating system and shell environment, certain special characters can trigger variable interpolation and therefore cause passwords to fail.

Workaround: When creating and entering passwords, wrap the string with single quotes in the following format: **`'<password>'`**

### 3.5.3. Undefining virsh snapshots fail

If you use **`virsh`** on KVM/libvirt to create snapshots in your development workflow, and then use **`minishift delete`** to delete the snapshots along with the VM, you might encounter the following error:

```
$ minishift delete
Deleting the Minishift VM...
Error deleting the VM:  [Code-55] [Domain-10] Requested operation is not
valid: cannot delete inactive domain with 4 snapshots
```

Cause: The snapshots are stored in ***~/.minishift/machines***, but the definitions are stored in ***/var/lib/libvirt/qemu/snapshot/minishift***.

Workaround: To delete the snapshots, you need to perform the following steps as root.

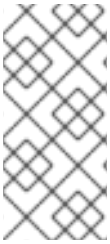
1. Delete the definitions.

```
# virsh snapshot-delete --metadata minishift <snapshot-name>
```

2. Undefine the minishift domain.

```
# virsh undefine minishift
```

You can now run **`minishft delete`** to delete the VM and restart Container Development Kit.



**NOTE**

If these steps do not resolve the issue, you can also use the following command to delete the snapshots:

```
$ rm -rf ~/.minishift/machines
```

It is recommended to avoid using metadata when you create snapshots. To ensure this, you can specify the **`--no-metadata`** flag. For example:

```
# virsh snapshot-create-as --domain vm1 overlay1 --diskspec
vda,file=/export/overlay1.qcow2 --disk-only --atomic --no-metadata
```

### 3.5.4. KVM: Error creating new host: dial tcp: missing address

The problem is likely that the **libvirtd** service is not running. You can check this with the following command:

```
$ systemctl status libvirtd
```

If **libvirtd** is not running, start it and enable it to start on boot:

```
$ systemctl start libvirtd
$ systemctl enable libvirtd
```

### 3.5.5. KVM: Failed to connect socket to '/var/run/libvirt/virtlogd-sock'

The problem is likely that the **virtlogd** service is not running. You can check this with the following command:

```
$ systemctl status virtlogd
```

If **virtlogd** is not running, start it and enable it to start on boot:

```
$ systemctl start virtlogd
$ systemctl enable virtlogd
```

### 3.5.6. KVM: Domain 'minishift' already exists...

If you try **`minishift start`** and the this error appears, ensure that you use **`minishift delete`** to delete the VMs that you created earlier. However, if this fails and you want to completely clean up Container Development Kit and start fresh, do the following:

1. As root, check if any existing {project} VMs are running:

   ```
   # virsh list --all
   ```

2. If any VM named minishift is running, stop it:

   ```
   # virsh destroy minishift
   ```

3. Delete the VM:

   ```
   # virsh undefine minishift
   ```

4. As your regular user, delete the ***~/.minishift/machines*** directory:

   ```
   $ rm -rf ~/.minishift/machines
   ```

In case all of this fails, you might want to uninstall Container Development Kit and do a fresh install of Container Development Kit.

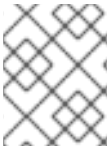
### 3.5.7. xhyve: Could not create vmnet interface

The problem is likely that the xhyve driver is not able to clean up **vmnet** when a VM is removed. **vmnet.framework** determines the IP address based on the following files:

- ***/var/db/dhcpd_leases***
- ***/Library/Preferences/SystemConfiguration/com.apple.vmnet.plist***

Reset Container Development Kit-specific IP database, ensure that you remove the **`minishift`** entry section from the **`dhcpd_leases`** file, and reboot your system.

```
{
  ip_address=192.168.64.2
  hw_address=1,2:51:8:22:87:a6
  identifier=1,2:51:8:22:87:a6
  lease=0x585e6e70
  name=minishift
}
```



**NOTE**

You can completely reset the IP database by removing the files manually but this is very risky.

### 3.5.8. VirtualBox: Error machine does not exist

If you use Windows, ensure that you set the `--vm-driver virtualbox` flag in the `minishift start` command. Alternatively, the problem might be an outdated version of VirtualBox.

To avoid this issue, it is recommended to use VirtualBox 5.1.12 or later.

### 3.5.9. Hyper-V: Hyper-V commands must be run as an Administrator

If you run Container Development Kit with Hyper-V on Windows as a normal user or as a user with Administrator privileges, you might encounter the following error:

```
Error starting the VM: Error creating the VM. Error with pre-create check:
"Hyper-V commands must be run as an Administrator".
```

Workaround: You can either add yourself to the Hyper-V Administrators group, which is recommended, or run the shell in an elevated mode.

If you are using PowerShell, you can add yourself to the Hyper-V Administrators group as follows:

1. As an administrator, run the following command:

```
([adsi]"WinNT://./Hyper-V
Administrators,group").Add("WinNT://$env:UserDomain/$env:Username,us
er")
```

2. Log out and log back in for the change to take effect.

You can also use the GUI to add yourself to the Hyper-V Administrators group as follows:

1. Click the **Start** button and choose **Computer Management**.
2. In the **Computer Management** window, select **Local Users And Groups** and then double click on **Groups**.
3. Double click on the **Hyper-V Administrators** group, the **Hyper-V Administrators Properties** dialog box is displayed.
4. Add your account to the Hyper-V Administrators group and log off and log in for the change to take effect.

Now you can run the Hyper-V commands as a normal user.

For more options for Hyper-V see creating Hyper-V administrators local group .

### 3.5.10. Hyper-V: Container Development Kit running with Hyper-V fails when connected to OpenVPN

If you try to use Container Development Kit with Hyper-V using an external virtual switch while you are connected to a VPN such as OpenVPN, Container Development Kit might fail to provision the VM.

Cause: Hyper-V networking might not route the network traffic in both directions properly when connected to a VPN.

Workaround: Disconnect from the VPN and try again after stopping the VM from the Hyper-V manager.