



English ▾

Multi-page HTML ▾

[GETTING STARTED WITH KUBERNETES \(/DOCUMENTATION/EN-US/RED\\_HAT\\_ENTERPRISE\\_LINUX\\_ATOMIC\\_HOST/7/HTML/GETTING\\_STARTED\\_WITH\\_KUBERNETES/\)](#)

[1. GET STARTED ORCHESTRATING CONTAINERS WITH KUBERNETES \(/DOCUMENTATION/EN-US/RED\\_HAT\\_ENTERPRISE\\_LINUX\\_ATOMIC\\_HOST/7/HTML/GETTING\\_STARTED\\_WITH\\_KUBERNETES/GET\\_STARTED\\_ORCHESTRATING\\_CONTAINERS\\_WITH\\_KUBERNETES/\)](#)

[1.1. Overview \(/documentation/en-us/red\\_hat\\_enterprise\\_linux\\_atomic\\_host/7/html/getting\\_started\\_with\\_kubernetes/get\\_started\\_orchestrating\\_containers\\_with\\_kubernetes#get\\_started\\_with\\_kubernetes\)](#)

[1.2. Understanding Kubernetes \(/documentation/en-us/red\\_hat\\_enterprise\\_linux\\_atomic\\_host/7/html/getting\\_started\\_with\\_kubernetes/get\\_started\\_orchestrating\\_containers\\_with\\_kubernetes#understanding\\_kubernetes\)](#)

[1.3. Running Containers from Kubernetes Pods \(/documentation/en-us/red\\_hat\\_enterprise\\_linux\\_atomic\\_host/7/html/getting\\_started\\_with\\_kubernetes/get\\_started\\_orchestrating\\_containers\\_with\\_kubernetes#running\\_containers\\_from\\_kubernetes\\_pods\)](#)

[1.3.1. Setting up to Deploy Docker Containers with Kubernetes \(/documentation/en-us/red\\_hat\\_enterprise\\_linux\\_atomic\\_host/7/html/getting\\_started\\_with\\_kubernetes/get\\_started\\_orchestrating\\_containers\\_with\\_kubernetes#setting\\_up\\_to\\_deploy\\_docker\\_containers\\_with\\_kubernetes\)](#)

[1.3.2. Starting Kubernetes \(/documentation/en-us/red\\_hat\\_enterprise\\_linux\\_atomic\\_host/7/html/getting\\_started\\_with\\_kubernetes/get\\_started\\_orchestrating\\_containers\\_with\\_kubernetes#starting\\_kubernetes\)](#)

[1.3.3. Launching container pods with Kubernetes \(/documentation/en-us/red\\_hat\\_enterprise\\_linux\\_atomic\\_host/7/html/getting\\_started\\_with\\_kubernetes/get\\_started\\_orchestrating\\_containers\\_with\\_kubernetes#launching\\_container\\_pods\\_with\\_kubernetes\)](#)

[1.4. Exploring Kubernetes pods \(/documentation/en-us/red\\_hat\\_enterprise\\_linux\\_atomic\\_host/7/html/getting\\_started\\_with\\_kubernetes/get\\_started\\_orchestrating\\_containers\\_with\\_kubernetes#exploring\\_kubernetes\\_pods\)](#)

[2. GET STARTED PROVISIONING STORAGE IN KUBERNETES \(/DOCUMENTATION/EN-US/RED\\_HAT\\_ENTERPRISE\\_LINUX\\_ATOMIC\\_HOST/7/HTML/GETTING\\_STARTED\\_WITH\\_KUBERNETES/GET\\_STARTED\\_PROVISIONING\\_STORAGE\\_IN\\_KUBERNETES/\)](#)

[2.1. Overview \(/documentation/en-us/red\\_hat\\_enterprise\\_linux\\_atomic\\_host/7/html/getting\\_started\\_with\\_kubernetes/get\\_started\\_provisioning\\_storage\\_in\\_kubernetes#overview-1\)](#)

[2.2. Kubernetes Persistent Volumes \(/documentation/en-us/red\\_hat\\_enterprise\\_linux\\_atomic\\_host/7/html/getting\\_started\\_with\\_kubernetes/get\\_started\\_provisioning\\_storage\\_in\\_kubernetes#persistent-volumes\)](#)

## CHAPTER 2. GET STARTED PROVISIONING STORAGE IN KUBERNETES

### 2.1. OVERVIEW


This section explains how to provision storage in Kubernetes.

Before undertaking the exercises in this topic, you must have a working Kubernetes configuration in place. Follow the instructions in [Get Started Orchestrating Containers with Kubernetes \(get\\_started\\_orchestrating\\_containers\\_with\\_kubernetes#get\\_started\\_with\\_kube\)](#) to manually configure Kubernetes.

#### NOTE

While you can use the procedure for orchestrating Kubernetes to test a manual configuration of Kubernetes, you should not use that configuration for production purposes. For a Kubernetes configuration that is supported by Red Hat, you must use OpenShift (which is available in various online and installable forms).

### 2.2. KUBERNETES PERSISTENT VOLUMES

This section provides an overview of Kubernetes Persistent Volumes. The example below explains how to use the `nginx` web server to serve content from a persistent volume. This section assumes that you understand the basics of Kubernetes and that you have a Kubernetes cluster up and running. 

A Persistent Volume (PV) in Kubernetes represents a real piece of underlying storage capacity in the infrastructure. Before using Kubernetes to mount anything, you must first create whatever storage that you plan to mount. Cluster administrators must create their GCE disks and export their NFS shares in order for Kubernetes to mount them.

Persistent volumes are intended for "network volumes" like GCE Persistent Disks, NFS shares, and AWS ElasticBlockStore volumes. HostPath was included for ease of development and testing. You'll create a local HostPath for this example.

### IMPORTANT

In order for HostPath to work, you will need to run a single node cluster. Kubernetes does not support local storage on the host at this time. There is no guarantee that your pod will end up on the correct node where the HostPath resides.

```
// this will be nginx's webroot $ mkdir /tmp/data01 $ echo 'I love Kubernetes storage!' > /tmp/data01/index.html
```

Define physical volumes in a YAML file.

```
$ mkdir -p ~/examples/persistent-volumes/volumes/ $ vi ~/examples/persistent-volumes/volumes/local-01.yaml
```

Create the following content in the `local-01.yaml` file:

```
kind: PersistentVolume apiVersion: v1 metadata: name: pv0001 labels: type: local spec: capacity: storage: 10Gi accessModes: - ReadWriteOnce hostPath: path: "/tmp/data01"
```

Create physical volumes by posting them to the API server.

```
$ kubectl create -f ~/examples/persistent-volumes/volumes/local-01.yaml persistentvolume "pv0001" created $ kubectl get pv NAME CAPACITY ACCESSMODES STATUS CLAIM REASON AGE pv0001 10Gi RW0 Available 14s
```

## 2.2.1. Requesting storage

Users of Kubernetes request persistent storage for their pods. The nature of the underlying provisioning need not be known by users. Users must know that they can rely on their claims to storage and that they can manage that storage's lifecycle independently of the many pods that may use it.

Claims must be created in the same namespace as the pods that use them.

Create a YAML file defining the storage claim.

```
$ mkdir -p ~/examples/persistent-volumes/claims/ $ vi ~/examples/persistent-volumes/claims/claim-01.yaml
```

Add the following content to the `claim-01.yaml` file:

```
kind: PersistentVolumeClaim apiVersion: v1 metadata: name: myclaim-1 spec: accessModes: - ReadWriteOnce resources: requests: storage: 3Gi
```

Create the claim.

```
$ kubectl create -f ~/examples/persistent-volumes/claims/claim-01.yaml persistentvolumeclaim "myclaim-1" created
```

A background process will attempt to match this claim to a volume. The state of your claim will eventually look something like this:

```
$ kubectl get pvc NAME STATUS VOLUME CAPACITY ACCESSMODES AGE myclaim-1 Bound pv0001 10Gi RW0 7s $ kubectl get pv NAME CAPACITY ACCESSMODES STATUS CLAIM REASON AGE pv0001 10Gi RW0 Bound default/myclaim-1
```



## 2.2.2. Using your claim as a volume

Claims are used as volumes in pods. Kubernetes uses the claim to look up its bound PV. The PV is then exposed to the pod.

Start by creating a `pod.yaml` file.

```
$ mkdir -p ~/examples/persistent-volumes/simpletest/ $ vi ~/examples/persistent-volumes/simpletest/pod.yaml
```

Add the following content to the `pod.yaml` file:

```
kind: Pod apiVersion: v1 metadata: name: mypod labels: name: frontendhttp spec: containers: - name: myfrontend image: nginx ports: - containerPort: 80 name: "http-server" volumeMounts: - mountPath: "/usr/share/nginx/html" name: mypd volumes: - name: mypd persistentVolumeClaim: claimName: myclaim-1
```

Use `pod.yaml` to create the pod and the claim, then check that it was all done properly.

```
$ kubectl create -f ~/examples/persistent-volumes/simpletest/pod.yaml $ kubectl describe pods mypod | less Name: mypod Namespace: default Node: 127.0.0.1/127.0.0.1 Start Time: Tue, 16 Aug 2016 09:42:03 -0400 Labels: name=frontendhttp Status: Running IP: 172.17.0.2
```

Page through the **kubectl describe** content until you see the IP address for the pod. Use that IP address in the next steps.

## 2.2.3. Check the service

Query the service using the `curl` command, with the IP address and port number, to make sure the service is running. In this example, the address is 172.17.0.2. If you get a "forbidden" error, disable SELinux using the `setenforce 0` command.

```
# curl 172.17.0.2:80 I love Kubernetes storage!
```

If you see the output shown above, you have a successfully created a working persistent volume, claim and pod that is using that claim.

## 2.3. VOLUMES

Kubernetes abstracts various storage facilities as "volumes".

Volumes are defined in the **volumes** section of a pod's definition. The source of the data in the volumes is either:

- a remote NFS share,
- an iSCSI target,
- an empty directory, or
- a local directory on the host.

It is possible to define multiple volumes in the **volumes** section of the pod's definition. Each volume must have a unique name (within the context of the pod) that is used during the mounting procedure as a unique identifier within the pod.

These volumes, once defined, can be mounted into containers that are defined in the **containers** section of the pod's definition. Each container can mount several volumes; on the other hand, a single volume can be mounted into several containers. The **volumeMounts** section of the container definition specifies where the volume should be mounted.

### 2.3.1. Example

```
apiVersion: v1 kind: Pod metadata: name: nfs-web spec: volumes: # List of volumes to use, i.e. *what* to mount - name: myvolume <
volume details, see below > - name: mysecondvolume < volume details, see below > containers: - name: mycontainer volumeMounts: #
List of mount directories, i.e. *where* to mount # We want to mount 'myvolume' into /usr/share/nginx/html - name: myvolume
mountPath: /usr/share/nginx/html/ # We want to mount 'mysecondvolume' into /var/log - name: mysecondvolume mountPath: /var/log/
```

## 2.4. KUBERNETES AND SELINUX PERMISSIONS

Kubernetes, in order to function properly, must have access to a directory that is shared between the host and the container. SELinux, by default, blocks Kubernetes from having access to that shared directory. Usually this is a good idea: no one wants a compromised container to access the host and cause damage. In this situation, though, we want the directory to be shared between the host and the pod without SELinux intervening to prevent the share.

Here's an example. If we want to share the directory `/srv/my-data` from the Atomic Host to a pod, we must explicitly relabel `/srv/my-data` with the SELinux label `svirt_sandbox_file_t`. The presence of this label on this directory (which is on the host) causes SELinux to permit the container to read and write to the directory. Here's the command that attaches the `svirt_sandbox_file_t` label to the `/srv/my-data` directory:

```
$ chcon -R -t svirt_sandbox_file_t /srv/my-data
```

The following example steps you through the procedure:

1. Define this container, which uses `/srv/my-data` from the host as the HTML root:

```
{ "apiVersion": "v1", "kind": "Pod", "metadata": { "name": "host-test" }, "spec": { "containers": [ { "name": "host-test",
"image": "nginx", "privileged": false, "volumeMounts": [ { "name": "srv", "mountPath": "/usr/share/nginx/html", "readOnly":
false } ] }, "volumes": [ { "name": "srv", "hostPath": { "path": "/srv/my-data" } } ] }
```

2. Run the following commands on the container host to confirm that SELinux denies the nginx container read access to `/srv/my-data` (note the failed `curl` command):

```
$ mkdir /srv/my-data $ echo "Hello world" > /srv/my-data/index.html $ curl <IP address of the container> <html> <head>
<title>403 Forbidden</title></head> ...
```

3. Apply the label `svirt_sandbox_file_t` to the directory `/srv/my-data`:

```
$ chcon -R -t svirt_sandbox_file_t /srv/my-data
```

4. Use `curl` to access the container and to confirm that the label has taken effect:

```
$ curl <IP address of the container> Hello world
```

If the `curl` command returned "Hello world", the SELinux label has been properly applied.

## 2.5. NFS

In order to test this scenario, you must already have prepared NFS shares. In this example, you will mount the NFS shares into a pod.

The following example mounts the NFS share into `/usr/share/nginx/html/` and runs the `nginx` webserver.

1. Create a file named `nfs-web.yaml`:

```
apiVersion: v1 kind: Pod metadata: name: nfs-web spec: volumes: - name: www nfs: # Use real NFS server address here. server:
192.168.100.1 # Use real NFS server export directory. path: "/www" readOnly: true containers: - name: web image: nginx
ports: - name: web containerPort: 80 protocol: tcp volumeMounts: # 'name' must match the volume name below. - name: www #
Where to mount the volume. mountPath: "/usr/share/nginx/html/"
```

2. Start the pod: The following command tells Kubernetes to mount `192.168.100.1:/www` into `/usr/share/nginx/html/`` inside the nginx container and run it.

```
$ kubectl create -f nfs-web.yaml
```

3. Confirm that the webserver receives data from the NFS share:

```
$ curl 172.17.0.6 Hello from NFS
```

## Mount options in Kubernetes

Kubernetes 1.6 includes the ability to add mount options to certain volume types. These include: GCEPersistentDisk, AWSElasticBlockStore, AzureFile, AzureDisk, NFS, iSCSI, RBD (Ceph Block Device), CephFS, Cinder (OpenStack block storage), Glusterfs, VsphereVolume, Quobyte Volumes, VMware, and Photon. You can add mount options by setting annotations to PersistentVolume objects. For example:

```
apiVersion: "v1" kind: "PersistentVolume" metadata: name: my-disk annotations: volume.beta.kubernetes.io/mount-options: "discard"
```

Prior to Kubernetes 1.6, the ability to add mount options was not supported. For details, see [Kubernetes Persistent Volumes \(https://kubernetes.io/docs/concepts/storage/persistent-volumes/#mount-options\)](https://kubernetes.io/docs/concepts/storage/persistent-volumes/#mount-options).

## Troubleshooting

**403 Forbidden error:** if you receive a "403 Forbidden" response from the webserver, make sure that SELinux allows Docker containers to read data over NFS by running the following command:

```
$ setsebool -P virt_use_nfs 1
```

## 2.6. iSCSI

To use iSCSI storage, make sure that the iSCSI target is properly configured. Then, make sure that all Kubernetes nodes have sufficient privileges to attach a LUN from the iSCSI target.

1. Create a file named `iscsi-web.yaml`, containing the following pod definition:

```
apiVersion: v1 kind: Pod metadata: name: iscsi-web spec: volumes: - name: www iscsi: # Address of the iSCSI target portal targetPortal: "192.168.100.98:3260" # IQN of the portal iqn: "iqn.2003-01.org.linux-iscsi.iscsi.x8664:sn.63b56adc495d" # LUN we want to mount lun: 0 # Filesystem on the LUN fsType: ext4 readOnly: false containers: - name: web image: nginx ports: - name: web containerPort: 80 protocol: tcp volumeMounts: # 'name' must match the volume name below. - name: www # Where to mount the volume. mountPath: "/usr/share/nginx/html/"
```

2. Create the pod: From the following command, Kubernetes logs into the iSCSI target, attaches LUN 0 (typically as `/dev/sdXYZ`), mounts the filesystem specified (in our example, it's ext4) to `/usr/share/nginx/html/` inside the nginx container, and runs it.

```
$ kubectl create -f iscsi-web.yaml
```

3. Check that the web server uses data from the iSCSI volume:

```
$ curl 172.17.0.6 Hello from iSCSI
```

## 2.7. GOOGLE COMPUTE ENGINE

If you are running your cluster on Google Compute Engine, you can use a Google Compute Engine Persistent Disk (GCE PD) as your persistent storage source. In the following example, you will create a pod which serves html content from a GCE PD.

1. If you have the GCE SDK set up, create a persistent disk using the following command. (Otherwise you can create the disk through the GCE web interface. If you want to set up the GCE SDK follow the instructions [here \(https://cloud.google.com/sdk/\)](https://cloud.google.com/sdk/).)

```
$ gcloud compute disks create --size=250GB {Persistent Disk Name}
```

2. Create a file named `gce-pd-web.yaml`:

```
apiVersion: v1 kind: Pod metadata: name: gce-web spec: containers: - name: web image: nginx ports: - name: web
containerPort: 80 protocol: tcp volumeMounts: - name: html-pd mountPath: "/usr/share/nginx/html" volumes: - name: html-pd
gcePersistentDisk: # Add the name of your persistent disk below pdName: {Persistent Disk Name} fsType: ext4
```

3. Create the pod. Kubernetes will create the pod and attach the disk but it will not format and mount it. (This is due to a bug which will be fixed in future versions of Kubernetes. The step that follows works around this issue.)

```
$ kubectl create -f gce-pd-web.yaml
```

4. Format and mount the persistent disk.

The disk will be attached to the virtual machine and a device will appear under `/dev/disk/by-id/`` with the name `scsi-0Google_PersistentDisk_{Persistent Disk Name}``. If this disk is already formatted and contains data proceed to the next step. Otherwise run the following command as root to format it:

```
$ mkfs.ext4 /dev/disk/by-id/scsi-0Google_PersistentDisk_{Persistent Disk Name}
```

5. When the disk is formatted, mount it in the location expected by Kubernetes. Run the following commands as root:

```
# mkdir -p /var/lib/kubelet/plugins/kubernetes.io/gce-pd/mounts/{Persistent Disk Name} && mount /dev/disk/by-id/scsi-
0Google_PersistentDisk_{Persistent Disk Name} /var/lib/kubelet/plugins/kubernetes.io/gce-pd/mounts/{Persistent Disk Name}
```

[NOTE] The `mkdir` command and the `mount` command must be run in quick succession as above because Kubernetes clean up will remove the directory if it sees nothing mounted there.

6. Now that the disk is mounted it must be given the correct SELinux context. As root run the following:

```
$ sudo chcon -R -t svirt_sandbox_file_t /var/lib/kubelet/plugins/kubernetes.io/gce-pd/mounts/{Persistent Disk Name}
```

7. Create some data for your web server to serve:

```
$ echo "Hello world" > /var/lib/kubelet/plugins/kubernetes.io/gce-pd/mounts/{Persistent Disk Name}/index.html
```

8. You should now be able to get HTML content from the pod:

```
$ curl {IP address of the container} Hello World!
```



[1.4. Exploring Kubernetes pods \(/documentation/en-us/red\\_hat\\_enterprise\\_linux\\_atomic\\_host/7/html/getting\\_started\\_with\\_kubernetes/get\\_started\\_orchestrating\\_containers\\_with\\_kubernetes#exploring\\_kubernetes\)](https://www.redhat.com/en/documentation/enterprise/linux/atomic/host/7/html/getting_started_with_kubernetes/get_started_orchestrating_containers_with_kubernetes#exploring_kubernetes)

## WHERE DID THE COMMENT SECTION GO?

Red Hat's documentation publication system recently went through an upgrade to enable speedier, more mobile-friendly content. We decided to re-evaluate our commenting platform to ensure that it meets your expectations and serves as an optimal feedback mechanism. During this redesign, we invite your input on providing feedback on Red Hat documentation via the [discussion platform \(/node/add/discussion?field\\_tags\[\]=docs-feedback&field\\_product\[\]=red\\_hat\\_enterprise\\_linux\\_atomic\\_host\)](https://discussion.platform (/node/add/discussion?field_tags[]=docs-feedback&field_product[]=red_hat_enterprise_linux_atomic_host)).

[Privacy Policy \(http://www.redhat.com/en/about/privacy-policy\)](http://www.redhat.com/en/about/privacy-policy)

[Customer Portal Terms of Use \(https://access.redhat.com/help/terms/\)](https://access.redhat.com/help/terms/)

[All Policies and Guidelines \(http://www.redhat.com/en/about/all-policies-guidelines\)](http://www.redhat.com/en/about/all-policies-guidelines)

Copyright © 2017 Red Hat, Inc.