Storage

Storage is certainly something you're gonna need. Stateless applications/containers are fun and all, but in the end, if you want to do something useful, you need to store some data. Even if that data is as simple as a configuration file.

Here, we are getting to the point where the Raspberry Pi is going to fail us. Simply put, there is no native CIS (Container Storage Interface) that is officially supported for arm64. And, looking at the rest that exist, how are they even used in production, since most of them are barely in beta, and only few are in stable releases?

Seriously, how is Amazon or Google handling persistent storage on their Kubernetes clusters? Maybe they just use external SAN, like 3Par, and present LUNs to each node... how then do they deal with multiple containers trying to write to the same file? I have some many questions about this.

Options

- **Rook + Ceph** – This one you can possibly get to work, but with unofficial arm64 builds thanks to https://github.com/raspbernetes/multi-arch-images. I believe this is stable on normal servers, and even production ready, but did not survive two reboots of my K8s cluster on Raspberry Pi 4. Combining that with the heavy load and not steep but vertical learning curve of Ceph, I would not recommend this for a Raspberry home cluster.
- **Longhorn** – Another native Kubernetes storage, now finally with support for arm64! https://github.com/longhorn/longhorn/issues/6 I ended up with this one as the only viable solution.
- **GlusterFS + Heketi** – GlusterFS works fine on Raspberry Pi, I tested it... Heketi is dead though,so not really native support. However, we can use GlusterFS to mount a folder on each node, and tell Kubernetes to use local FS as storage. This would make the data available on every node, and in case the pod switches to another, the persistent data will be there, waiting for it. A slight issue with GlusterFS though: It's not recommended for "live" files, a.k.a. databases, which sux... But, to be honest, I have seen MySQL running on a GlusterFS cluster in production 😅.
- **NFS** – Funnily enough, this one works just fine; you can create claims and manage it from Kubernetes (My first cluster was using NFS as persistent storage and it worked fine). However, this is not clustered, and this single point of failure turns it against the exercise we are trying to do here.

Longhorn

Fairly new Kubernetes native file-system for arm64, at the time of writing, but man it just worked! After the ordeal with Rook + Ceph, it was such a breeze to set it up! My cluster is in a 3 week stability testing phase now, and nothing has broken.

As you know from our node setup, I want to use a separate USB flash-drive 64GB on each node to be a volume for storage. Longhorn is making this simple for us; all you need to do is mount the disk under `/var/lib/longhorn`, which is a default data path. I forgot to do that beforehand, and had to change it later, it's super simple, so no worries. I ended up with my storage under `/storage`.

> 🔥 **Tip**
>
> You can leave it at `/var/lib/longhorn` if you are not using another disk or usb disk. I just changed it to /storage since this is easy to find.

There is an issue with Raspberry Pi / Ubuntu: the names for disks are assigned almost at random. Therefore, even if you have USB disks in the same slots on multiple nodes, they can be named at random, `/dev/sda` or `/dev/sdb`, and there is no easy way to enforce the naming, without messing with udev rules a lot.

Identifying disks for storage.

We are going to use Ansible again a lot, and will add new variables with disk names that will be used for storage into `/etc/ansible/hosts`.

We are going to use the `lsblk -f` command on every node and look for disk labels:

```
ubuntu@control01:~$ ansible cube -b -m shell -a "lsblk -f"
control01 | CHANGED | rc=0 >>
NAME   FSTYPE FSVER LABEL       UUID                                 FSAVAIL FSUSE% MOUNTPOINT
sda
sdb
├─sdb1 vfat   FAT32 system-boot 2EC5-A982                             100.8M    60% /boot/firmware
└─sdb2 ext4   1.0   writable    c21fdada-1423-4a06-be66-0b9c02860d1d  220.3G     2% /
cube02 | CHANGED | rc=0 >>
NAME   FSTYPE FSVER LABEL       UUID                                 FSAVAIL FSUSE% MOUNTPOINT
sda
├─sda1 vfat   FAT32 system-boot 2EC5-A982                             100.8M    60% /boot/firmware
└─sda2 ext4   1.0   writable    c21fdada-1423-4a06-be66-0b9c02860d1d   24.2G    13% /
sdb
└─sdb1 exfat  1.0   Samsung USB 64A5-F009
control02 | CHANGED | rc=0 >>
NAME   FSTYPE FSVER LABEL       UUID                                 FSAVAIL FSUSE% MOUNTPOINT
sda
├─sda1 vfat   FAT32 system-boot 2EC5-A982                             100.8M    60% /boot/firmware
└─sda2 ext4   1.0   writable    c21fdada-1423-4a06-be66-0b9c02860d1d   23.3G    16% /
sdb
cube01 | CHANGED | rc=0 >>
NAME   FSTYPE FSVER LABEL       UUID                                 FSAVAIL FSUSE% MOUNTPOINT
sda
├─sda1 vfat   FAT32 system-boot 2EC5-A982                             100.8M    60% /boot/firmware
└─sda2 ext4   1.0   writable    c21fdada-1423-4a06-be66-0b9c02860d1d   24.3G    12% /
sdb
control03 | CHANGED | rc=0 >>
NAME   FSTYPE FSVER LABEL       UUID                                 FSAVAIL FSUSE% MOUNTPOINT
sda
├─sda1 vfat   FAT32 system-boot 2EC5-A982                             100.8M    60% /boot/firmware
└─sda2 ext4   1.0   writable    c21fdada-1423-4a06-be66-0b9c02860d1d   23.1G    16% /
sdb
cube03 | CHANGED | rc=0 >>
NAME   FSTYPE FSVER LABEL       UUID                                 FSAVAIL FSUSE% MOUNTPOINT
sda
├─sda1 vfat   FAT32 system-boot 2EC5-A982                             100.8M    60% /boot/firmware
└─sda2 ext4   1.0   writable    c21fdada-1423-4a06-be66-0b9c02860d1d   24.2G    13% /
sdb
cube04 | CHANGED | rc=0 >>
NAME   FSTYPE FSVER LABEL       UUID                                 FSAVAIL FSUSE% MOUNTPOINT
sda
├─sda1 vfat   FAT32 system-boot 2EC5-A982                             100.8M    60% /boot/firmware
└─sda2 ext4   1.0   writable    c21fdada-1423-4a06-be66-0b9c02860d1d   24.1G    13% /
sdb
cube05 | CHANGED | rc=0 >>
NAME   FSTYPE FSVER LABEL       UUID                                 FSAVAIL FSUSE% MOUNTPOINT
sda
├─sda1 vfat   FAT32 system-boot 2EC5-A982                             100.8M    60% /boot/firmware
└─sda2 ext4   1.0   writable    c21fdada-1423-4a06-be66-0b9c02860d1d   24.2G    13% /
sdb
cube06 | CHANGED | rc=0 >>
NAME   FSTYPE FSVER LABEL       UUID                                 FSAVAIL FSUSE% MOUNTPOINT
sda
sdb
├─sdb1 vfat   FAT32 system-boot 2EC5-A982                             100.8M    60% /boot/firmware
└─sdb2 ext4   1.0   writable    c21fdada-1423-4a06-be66-0b9c02860d1d   24.2G    13% /
```

As you can see, each node has two disks, `sda` and `sdb`, but assigned at boot. Every disk that splits to `<name>1` and `<name>2`, and has `/boot/firmware` and `/` as mount points, which are our OS disks. The other one is our "storage" 😊. Not all my disks are wiped though, so let's take care of that. You can see disks with: `sdb1 exfat 1.0 Samsung USB 64A5-F009` which is the default FAT partition for a USB disk for windows, and your USB might be formated to that from factory...

Edit `/etc/ansible/hosts`, and add a new variable (I have chosen name var_disk) with the disk to wipe. **TAKE YOUR TIME AND LOOK TWICE !** the wipefs command we're gonna use will not wipe your OS disk but it might wipe any other that is not mounted.

```
[control]
control01  ansible_connection=local var_hostname=control01 var_disk=sda
control02  ansible_connection=ssh   var_hostname=control02 var_disk=sdb
control03  ansible_connection=ssh   var_hostname=control03 var_disk=sdb

[workers]
cube01  ansible_connection=ssh  var_hostname=cube01 var_disk=sdb
cube02  ansible_connection=ssh  var_hostname=cube02 var_disk=sdb
cube03  ansible_connection=ssh  var_hostname=cube03 var_disk=sdb
cube04  ansible_connection=ssh  var_hostname=cube04 var_disk=sdb
cube05  ansible_connection=ssh  var_hostname=cube05 var_disk=sdb
cube06  ansible_connection=ssh  var_hostname=cube06 var_disk=sda

[cube:children]
control
workers
```

Wipe

Wipe them! Wipe them all with `wipefs -a`, it's the only way to be sure!

```
ansible cube -b -m shell -a "wipefs -a /dev/{{ var_disk }}"
```

The FS was wiped from `/dev/sdb` on the `cube2` node. You can check with the same command as before. All disks are ready; remove the variable from `/etc/ansible/hosts`, because we can't guarantee that the names will be the same at the next boot (they should, but who knows...).

Filesystem and mount

We need to mount the storage disks, but before that, we need some file-systems on them. Generally, ext4 is recommended.

```
ansible cube -b -m shell -a "mkfs.ext4 /dev/{{ var_disk }}"
ansible cube -m shell -a "mkdir /storage" -b
ansible cube -b -m shell -a "mount /dev/{{ var_disk }} /storage"
```

We also need to add these disks into `/etc/fstab`, so that after reboot they mount automagically. For that, we need the UUID of the disks.

```
ubuntu@control01:~$ ansible cube -b -m shell -a "blkid -s UUID -o value /dev/{{ var_disk }}"
control03 | CHANGED | rc=0 >>
a0aa3b5c-743a-4948-a490-d38ee1ab2211
control01 | CHANGED | rc=0 >>
4698b93a-0466-401a-9900-beff65cca358
cube02 | CHANGED | rc=0 >>
5fc0389b-c8bf-4229-94db-99d7362496b6
cube01 | CHANGED | rc=0 >>
9792876f-ce6b-4a40-83da-d92aae49a51d
control02 | CHANGED | rc=0 >>
accdfe09-a0f8-43e3-9663-c095e9929fc9
cube03 | CHANGED | rc=0 >>
87296139-c7f8-4e5b-8ffa-f651fd445d76
cube04 | CHANGED | rc=0 >>
7ad7354b-8846-45f6-b0e1-ef0dc71d99fe
cube05 | CHANGED | rc=0 >>
40ee7dc2-6cc2-4520-bd84-367d0e2565be
```

```
cube06 | CHANGED | rc=0 >>
c416b477-7766-47a8-b9af-e8ae2768c55f
```

Add these to `/etc/ansible/hosts`, with another custom variable, for example:

```
[control]
control01  ansible_connection=local var_hostname=control01 var_disk=sda var_uuid=4690b93a-0466-401a-9900-beff65cca358
control02  ansible_connection=ssh   var_hostname=control02 var_disk=sda var_uuid=accdfe09-a0f8-43e3-9663-c095e9929fc9
control03  ansible_connection=ssh   var_hostname=control03 var_disk=sda var_uuid=a0aa3b5c-743a-4948-a490-d38ee1ab2211

[workers]
cube01  ansible_connection=ssh  var_hostname=cube01 var_disk=sdb var_uuid=9792876f-ce6b-4a40-83da-d92aae49a51d
cube02  ansible_connection=ssh  var_hostname=cube02 var_disk=sda var_uuid=5fc03899-c8bf-4229-94db-99d7362496b6
cube03  ansible_connection=ssh  var_hostname=cube03 var_disk=sda var_uuid=87296139-c7f8-4e5b-8ffa-f651fd445d76
cube04  ansible_connection=ssh  var_hostname=cube04 var_disk=sdb var_uuid=7ad7354b-8846-45f6-b0e1-ef0dc71d99fe
cube05  ansible_connection=ssh  var_hostname=cube05 var_disk=sdb var_uuid=40ee7dc2-6cc2-4520-bd84-367d0e2565be
cube06  ansible_connection=ssh  var_hostname=cube06 var_disk=sdb var_uuid=c416b477-7766-47a8-b9af-e8ae2768c55f

[cube:children]
control
workers
```

Using Ansible, we add this line into `/etc/fstab` (yeah, I know, we should have used the lineinfile module...).

```
ansible cube -b -m shell -a "echo 'UUID={{ var_uuid }}  /storage       ext4    defaults        0       2' >> /etc/fstab"
#Check
ansible cube -b -m shell -a "grep UUID /etc/fstab"
#Make sure mount have no issues.
ansible cube -b -m shell -a "mount -a"
```

Longhorn requirements

Install open-iscsi on each node:

```
ansible cube -b -m apt -a "name=open-iscsi state=present"
```

Install Longhorn

```
#Switch to home dir
cd
#Clone Longhorn, I did specified v1.1.0 since this was the first not yet released
#arm64 supporting version. I think its out now and you don't need -b v1.1.0
git clone -b v1.1.0 https://github.com/longhorn/longhorn.git
#Ah look Helm we installed before, use it !
kubectl create namespace longhorn-system
helm install longhorn ./longhorn/chart/ --namespace longhorn-system --kubeconfig /etc/rancher/k3s/k3s.yaml --set defaultSettings.defaultDataPath="/storage"
```

Give it some time; it should deploy, maybe some pods will restart, but in the end, it should look something like this.

More or less, this already has storage, created and attached to the docker-registry. Everything under namespace `longhorn-system` should be `1/1 Running`.

```
root@control01:/home/ubuntu# kubectl -n longhorn-system get pod
NAME                                        READY   STATUS    RESTARTS   AGE
csi-attacher-5dcdcd5984-6txrq               1/1     Running   0          6d
csi-attacher-5dcdcd5984-n9c5b               1/1     Running   0          6d
csi-attacher-5dcdcd5984-pm5pz               1/1     Running   0          6d
csi-provisioner-5c9dfb6446-6cpn4            1/1     Running   0          6d
csi-provisioner-5c9dfb6446-wcwsl            1/1     Running   0          6d
csi-provisioner-5c9dfb6446-xsnhx            1/1     Running   0          6d
csi-resizer-54d484bf8-2f8mh                 1/1     Running   0          6d
csi-resizer-54d484bf8-fvrr7                 1/1     Running   0          6d
csi-resizer-54d484bf8-pv7pp                 1/1     Running   0          6d
csi-snapshotter-96bfff7c9-cj7rt             1/1     Running   0          6d
csi-snapshotter-96bfff7c9-hv4gk             1/1     Running   0          6d
csi-snapshotter-96bfff7c9-jwbjc             1/1     Running   0          6d
engine-image-ei-d73fbea2-2kvjx              1/1     Running   0          6d
engine-image-ei-d73fbea2-6s268              1/1     Running   0          6d
engine-image-ei-d73fbea2-97gkp              1/1     Running   0          6d
engine-image-ei-d73fbea2-dhlg2              1/1     Running   0          6d
engine-image-ei-d73fbea2-hbczq              1/1     Running   0          6d
engine-image-ei-d73fbea2-jd1dc              1/1     Running   1          6d
engine-image-ei-d73fbea2-m5c7r              1/1     Running   0          6d
engine-image-ei-d73fbea2-mtgd9              1/1     Running   0          6d
engine-image-ei-d73fbea2-vhjb5              1/1     Running   0          6d
instance-manager-e-222f0b35                 1/1     Running   0          6d
instance-manager-e-4c86d798                 1/1     Running   0          6d
instance-manager-e-597117S3                 1/1     Running   0          6d
instance-manager-e-787819f8                 1/1     Running   0          6d
instance-manager-e-ada009c8                 1/1     Running   0          6d
instance-manager-e-b1661214                 1/1     Running   0          6d
instance-manager-e-d52fe797                 1/1     Running   0          6d
instance-manager-e-e1e5db80                 1/1     Running   0          6d
instance-manager-e-eef19185                 1/1     Running   0          6d
instance-manager-r-169dc3b5                 1/1     Running   0          6d
instance-manager-r-1b1bee2d                 1/1     Running   0          6d
instance-manager-r-2ea20bcf                 1/1     Running   0          6d
instance-manager-r-36d070bf                 1/1     Running   0          6d
instance-manager-r-527ae8a4                 1/1     Running   0          6d
instance-manager-r-9e702e8d                 1/1     Running   0          6d
instance-manager-r-bfeba198                 1/1     Running   0          6d
instance-manager-r-d0d7ca7b                 1/1     Running   0          6d
instance-manager-r-e0405b80                 1/1     Running   0          6d
longhorn-csi-plugin-2727k                   2/2     Running   0          6d
longhorn-csi-plugin-44hwr                   2/2     Running   0          6d
longhorn-csi-plugin-9fvsz                   2/2     Running   0          6d
longhorn-csi-plugin-b6kv2                   2/2     Running   0          6d
longhorn-csi-plugin-k8996                   2/2     Running   0          6d
longhorn-csi-plugin-m4c98                   2/2     Running   0          6d
longhorn-csi-plugin-qvdnl                   2/2     Running   0          6d
longhorn-csi-plugin-th6kh                   2/2     Running   0          6d
longhorn-csi-plugin-vlpwg                   2/2     Running   0          6d
longhorn-driver-deployer-75fcf87c46-8nsh7   1/1     Running   0          6d
longhorn-manager-56cx4                      1/1     Running   3          6d
longhorn-manager-84csp                      1/1     Running   3          6d
longhorn-manager-8mcq9                      1/1     Running   3          6d
longhorn-manager-c8fc4                      1/1     Running   4          6d
longhorn-manager-gdf6b                      1/1     Running   3          6d
longhorn-manager-mfzwf                      1/1     Running   4          6d
longhorn-manager-n8c95                      1/1     Running   3          6d
longhorn-manager-pzsvl                      1/1     Running   3          6d
longhorn-manager-zp2bj                      1/1     Running   4          6d
longhorn-ui-758c86dfbc-hlfdm                1/1     Running   0          6d
```

Look also at services. Longhorn-frontend is a management UI for storage, similar to what Rook + Ceph have; very useful!

Later on, we will assign it its own LoadBalancer IP.

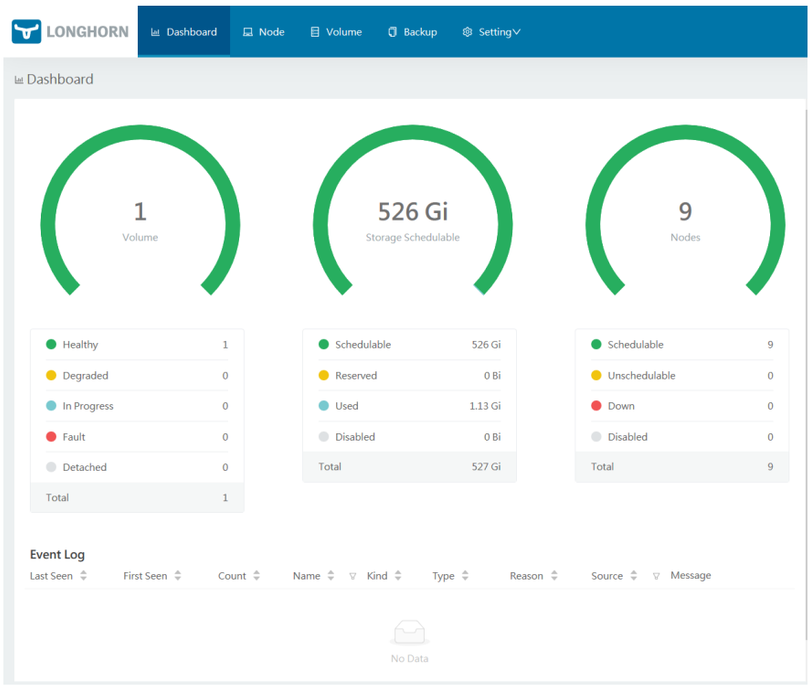
```
root@control01:/home/ubuntu# kubectl -n longhorn-system get svc
NAME                TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)     AGE
csi-attacher        ClusterIP   10.43.125.73    <none>        12345/TCP   6d
csi-provisioner     ClusterIP   10.43.118.73    <none>        12345/TCP   6d
csi-resizer         ClusterIP   10.43.245.224   <none>        12345/TCP   6d
csi-snapshotter     ClusterIP   10.43.230.3     <none>        12345/TCP   6d
longhorn-backend    ClusterIP   10.43.118.82    <none>        9500/TCP    6d
longhorn-frontend   ClusterIP   10.43.204.227   <none>        80/TCP      6d
```

UI

> **Important**
>
> You don't need to do this step; we added /storage during installation already. However, I keep this here for future reference. Continue on, **"Make Longhorn the default storageclass"**.

Either directly from a node, or from ssh tunneling the fronted port to your workstation, you can access the UI. There is no login, and we are not going to use one; Because my cluster is not accessible from the Internet, it can be as it is. I'm sure there is a setup to protect it.

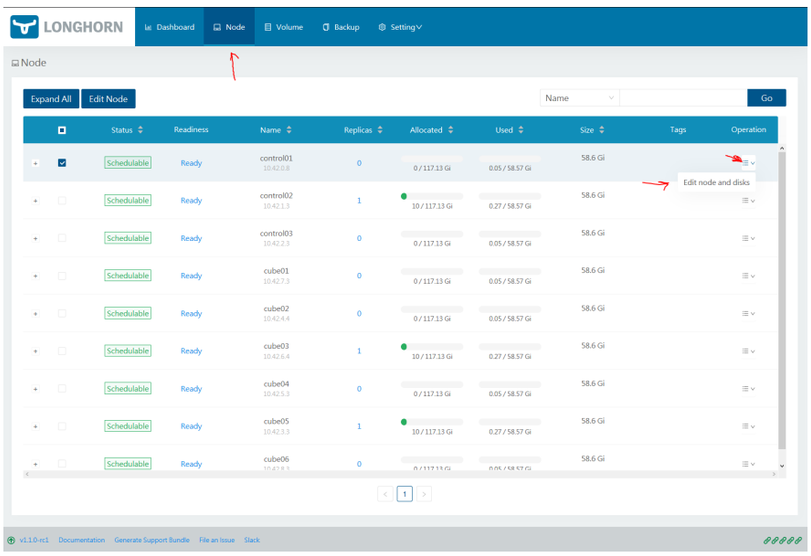
This is how it looks with one volume already claimed (You should have 0 in the left dial):



Add /storage

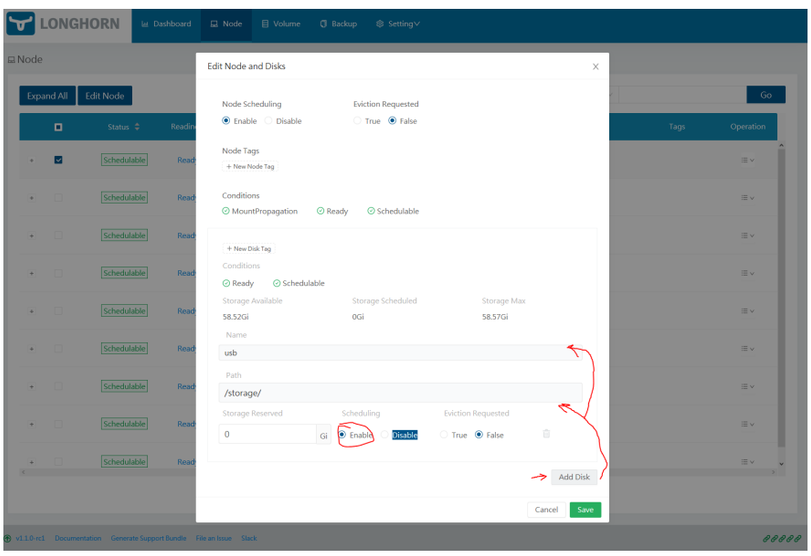
> **Important**
>
> You don't need to do this step; we added /storage during installation already. However, I keep this here for future reference. Continue on, **"Make Longhorn the default storageclass"**.

We need to add our mounted storage as a disk for Longhorn. Navigate to `Node` via the web UI. You need to do this for each node. Click on `Operation -> Edit node and disks`:



You will already have a node populated with default storage `/var/lib/longhorn`, and some random name.

First, click on `Add Disk`, and fill in a new disk location, making sure you switch `Scheduling` to `Enable`.



Second, on the disk with path `/var/lib/longhorn`, switch `Scheduling` to `Disable`. Then, click on the trash can to remove it.

Do the same for each node.

Make Longhorn the default StorageClass

Almost done! I'd like to make Longhorn the default storage provider, so that when using Helm (which already has a pre-set chart to use default storage provider) it will choose Longhorn.

By default, it would look like this after fresh k3s install.

```
root@control01:/home/ubuntu# kubectl get storageclass
NAME                 PROVISIONER             RECLAIMPOLICY   VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION   AGE
local-path (default)         rancher.io/local-path   Delete          WaitForFirstConsumer   false                  6d1h
longhorn (default)   driver.longhorn.io      Delete          Immediate              true                   6d1h
```

Execute:

```
kubectl patch storageclass local-path -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-class":"false"}}}'
```

Result:

```
root@control01:/home/ubuntu# kubectl get storageclass
NAME                 PROVISIONER             RECLAIMPOLICY   VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION   AGE
local-path           rancher.io/local-path   Delete          WaitForFirstConsumer   false                  6d1h
longhorn (default)   driver.longhorn.io      Delete          Immediate              true                   6d1h
```

Now, Longhorn is the default storage class.

That's all for now; I'll get into how to create PV and PVC for deployments when we are going to install a docker-registry.

Last update: May 26, 2022

Comments

What do you think?
10 Responses

Upvote Funny Love Surprised Angry Sad

9 Comments https://rpi4cluster.com Disqus' Privacy Policy Login

Favorite Tweet Share Sort by Best

Join the discussion…

LOG IN WITH OR SIGN UP WITH DISQUS ?

Name

Dimitrios Kordas • 3 months ago
I am experiencing the issue below after rebooting the cluster: The local-path storage class is once again marked as default. This has a side-effect that new workloads using helm, for example, fail since there are 2 default storage classes.

It seems that patching the local-path storage class to not be default is lost after a reboot. Any clue as to why that would happen?

Thanks!

Reply • Share ›

Rich Durso ➔ Dimitrios Kordas • 3 months ago • edited
If you are already up and running, do the disable one more time:

```
$ kubectl patch storageclass local-path -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-class":"false"}}}'
```

Then edit the k3s service:

```
sudo systemctl edit k3s
```

At the end of the "ExecStart=" add "--disable local-storage", save and exit. Reboot.

The local-path storage will remain non-default:

```
$ kubectl get storageclass
NAME PROVISIONER RECLAIMPOLICY VOLUMEBINDINGMODE ALLOWVOLUMEEXPANSION AGE
freenas-iscsi-csi org.democratic-csi.iscsi Delete Immediate true 10d
freenas-nfs-csi org.democratic-csi.nfs Delete Immediate true 9d
longhorn (default) driver.longhorn.io Delete Immediate true 11h
local-path rancher.io/local-path Delete WaitForFirstConsumer false 15d
```

1 Reply • Share ›

vladoportos Mod ➔ Rich Durso • 3 months ago
Interesting, thanks for pointing that out.

Reply • Share ›

Rich Durso ➔ Dimitrios Kordas • 3 months ago • edited
I was able to reproduce this as well. The instructions look correct. It seems to be a k3s specific issue. Look in this file: /var/lib/rancher/k3s/server/manifests/local-storage.yaml

```
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
name: local-path
annotations:
storageclass.kubernetes.io/is-default-class: "true"
provisioner: rancher.io/local-path
volumeBindingMode: WaitForFirstConsumer
reclaimPolicy: Delete
```

It appears this is processed after every reboot. I found some GitHub references to installing k3s with the "--disable local-storage" flag. Editing the file to be "false" might be a shot-term thing, next K3s update delivers a file with that set to true again.

1 Reply • Share ›

Ben Karmay • 10 months ago
Great tutorial, really appreciating the detailed and contextualized instructions you're achieving! I'm a real newbie to kube reality so its a steep learning curve. I've made it to the Central Logging stage, and looking ahead I'm a bit unclear on the distributed/persistent storage design and purpose.

- What exactly is the dedicated persistent storage needed for, what data element require/benefit from persistent storage?
- What is the distributed design achieving?
- Can the disk used for the OS be used/shared to achieve the same/similar result?

Geez I hope this make sense...

Reply • Share ›

vladoportos Mod ➔ Ben Karmay • 10 months ago
Hi,

Makes sense, don't worry :)

To explain better, lets take an example of running database container in your Kubernetes cluster. This container consist of the database binary files and all the little files that the database need to function. But it also have to store the data you create, like you database tables, configuration, logs etc... Normally if you had installed database on one server it would store this data in a file on the disk same as where the binary files are, but for containers its slightly different, containers are made to be "static". That mean that inside of the container does not change (it can, but when its run again on other worker node its run from container image and the changes you store before will disappear). To solve this issue where application needs to store some data, persistent storage come to play.

You need to tell your container when its creating to mount (attach) external folder into container and anything that is written there will stay there even if the image is destroyed/recreated/updated . For example in your container you have folder called /database and this folder store database configuration files and database tables, and your containerized database will be setup to load the files from that folder all the time. This /database folder inside container will be linked for example to /some_folder/database on the worker node where the container is running. Main idea is to be able to update the binary files (container image) of application (or downgrade, basically be able to switch versions of application) without being tight to data that are loading into it.

And now we are getting to more advance stuff :) Until now that worked just fine if you had one worker where the data is stored in the same place and container can find them there all the time... now what happen when the container is switched to another worker node. It will look locally for /some_folder/database to mount it to container but the data are on the old worker node and container fails to load (or loads with empty database). So we need to somehow make the data available on

see more

Reply • Share ›

Ben Karmay ➔ vladoportos • 10 months ago
Thanks, yes that's super helpful! I also had a look over etcd and rancher documentation. Back into the project later today hopefully! Cheers :)

Reply • Share ›

bigbrovar • a year ago
Hey, thanks for this write up. I am a bit confused though. In my case I intend to use 3 nodes for storing longhorn volumes. I have these nodes mount the partition I intend to use for the storage to /var/lib/longhorn because this is the default path (from what I read) that long expect the volumes to be mounted. What confuses me is the need for /storage when you can just mount them directly to /var/lib/longhorn.

Apologies if this sound stupid. I am still a newbie at this :(

Reply • Share ›

vladoportos Mod ➔ bigbrovar • a year ago
Hello, you can safely ignore the /storage and use /var/lib/longhorn, honestly I moved it to /storage because I keep forgetting where the default was :D and also its good to know how to use different path for the storage, just in case.

Reply • Share ›