Local Storage (any why you are getting the error 0/1 deployed)

So you have finally deployed your new and sparkly KubeAdm K8s cluster! Hurray! Now let't get deeper and let's explain What are Storage Classes, What we are using and Why and most importantly Why you are getting that pesky 0/1 Pods deployed error when you are deploying something that needs storage (tl;dr RTFM). Let's go!

The magical world of Storage Classes

In order for you do understand the usability of Storage Classes, you have to understand what Kubernetes is. Kubernetes is a mainly a **Cloud-Centric** Container Orchestration tool. It hasn't made particularly for Local Usage but by large Cloud Provider that certainly have more than one storage class (that means storage "rank" that they offer). Let's say that a provider has a slow HDD storage rank, a fast SSD storage rank and an even faster Ceph storage rank that even utilizes caching. Wow! How that dearly cloud provider can distribute its containers according to its storage type in k8s? Storage Classes of course! So, we can simplify storage classes by telling you that it is something like describing what kind of storage you have in the host machines of K8s and what storage type every container that is deployed in k8s is going to use.

You got all of that? No? OK, so for this use case we are only going to use Local Storage. Because yes, we are plebeians and we don't use a Dynamic Storage provider (like Ceph or S3) for our Dev K8s cluster, thank you very much.

Let's deploy our first Local Storage Provider

So this is the important bit. We need to deploy a Storage Class that uses local paths (etc /home/foo/bar) for its storage needs.

Before we do that though, where are we going to save our data?

For our dev k8s cluster, I, very democratically, have chosen to use the folder **/home/fint/storage** as our K8s storage folder. Inside this folder there are subfolders, usually named after namespaces or specialized containers.

Got all that? Understood the folder structure? Lets start then!

First we need to deploy the Storage Class Provider that tells our cluster that we are going to locally save the files.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: standard
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
```

copy-paste, save to file and **kubectl apply -f**

What we did there? We made a StorageClass named **standard**, that does not use a provisioning at all and "connects" with the folder only if a Container asks to save something.

The "standard" is only a name. Some people use "local-storage" as a name, you can use foobar. We actually dont really care about the name. It is there to only help you distinguish the storage classes you have.

So OK, we made our first and only storage class. That does not mean however that our k8s cluster is going to automagically save to it, we need to make the class our **default class**

First of all, let's see our available storage classes by using `kubectl get storageclass`.

```
NAME                 PROVISIONER               AGE
standard             kubernetes.io/gce-pd      1d
```

This should be your output. If not WHY YOU ARE NOT USING KUBEADM? MANOS ANGRYYYY!

Next, by using the command `kubectl patch storageclass standard -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'` we are making "standard" as our default storage class. Let's make sure though by using `kubectl get storageclass`

```
NAME                 PROVISIONER               AGE
standard (default)   kubernetes.io/gce-pd      1d
```

There why go! Sweet! Now you have a StorageClass and the fun part begins!

PersistentVolume and PersistentVolumeClaim

PV? PVC? Are we using plastic tubes? Got confused? OK, let's explain what those are!

PersistentVolume is like a declaration. With a PersisentVolume component you are actually declaring to k8s the piece of the storage you are providing (storage type, storage size and other configs according to storage type). There are times that with K8s Managers (like HELM) and Dynamic Storage Classes (with providers) that you wont actually need to create a Persistent Volume component. That, however, does not apply in our use case. As previously said, we are using a Local type Storage Class that does not and cannot have a provider, thus Dynamic Allocation.

TL;DR, you will need to manually allocate the space needed by the K8s application.

So, how our K8s application will see the available Persistent Volume that we will create? With a PersistentVolumeClaim. PVCs are like a written application. PVCs request a space with certain requirements from K8s. Usually, if we use Helm, we dont actually have to create or manage PersistentVolumeClaims, but if we do not use a chart for our deployment, a PVC is very much needed. In the example bellow, we will assume that the deployment is manual (the best kind of deployment).

Technical mumbo jumbo aside, the procedure for providing space to a K8s application/service in our use case is this: create a PersistentVolumeClaim describing the specifications of the space you want, create a folder inside /home/fint/storage that will be used for saving (you'll need SSH for that) and finally create a PersistentVolume. If all these are correct, the volume and claim will bind, allowing your application to save thingz.

We assume that we have already created the folder /home/fint/storage/foo and we have created the namespace test in our k8s environment

Let's create the PersistentVolumeClaim

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: foo
  namespace: test
  finalizers:
    - kubernetes.io/pvc-protection
spec:
  accessModes:
    - ReadWriteOnce
  selector:
    matchLabels:
      app: foo
      type: local
  resources:
    requests:
      storage: 1Gi
  volumeName: foo
  storageClassName: standard
  volumeMode: Filesystem
```

So, let's describe what we did there. We have created a PersistentVolumeClaim with the name foo in the namespace test that will **need to** Read and Write sequentially, have total capacity of 1GB, use the StorageClass "standard" and have a mode of Filesystem. To help our

PersistentVolumeClaim bond easier with the PersistentVolume we'll create in a bit, we are using MatchLabels. MatchLabels are some specs that help PersistentVolumeClaims to bond easier and faster with the PersistentVolume that has the exact same labels. We'll see that in a bit. Lets apply our PersistentVolumeClaim by using **kubectl apply -f**

Having created the PersistentVolumeClaim, lets now create the PersistentVolume

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: foo
  labels:
    app: foo
    type: local
  finalizers:
    - kubernetes.io/pv-protection
spec:
  capacity:
    storage: 1Gi
  hostPath:
    path: /home/fint/storage/foo
    type: ''
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: standard
  volumeMode: Filesystem
```

With the above, we are creating a PersistentVolume named foo that has 1Gb storage capacity in the path /home/fint/storage/foo, with no specific type, that Reads and Writes sequentially, that will not be deleted if the volume is reclaimed by another PVC and uses the Storage Class "standard". Please pay close attention to the labels metadata. Those are the exact save as the MatchLabels that we have in our PVC. Now let's apply our PersistentVolume by using **kubectl apply -f**

Give it a minute after applying and the PersistentVolumeClaim should now be bound to our PersistentVolume.

Lets test that by using the commands
kubectl get pv --namespace test

```
NAME           CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM                                                   STORAGECLASS
REASON   AGE
foo            1Gi        RWO            Retain           Bound
default/foo
standard                2m23s
```

kubectl get pvc --namespace test

```
NAME   STATUS   VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   AGE
foo    Bound    foo      1Gi        RWO            standard       3m51s
```

Job well done my friend!🎉

⚠️ Troubleshooting⚠️

0/1 nodes are available: 1 node(s) didn't find available persistent volumes to bind. preemption: 0/1 nodes are available: 1 Preemption is not helpful for scheduling.

TL;DR your storage class is either incorrect or not default and you might not have created a PersistentVolume.

Let me guess, this happened right after using a helm chart. And the pod is failing due to the message above. The PVC is showing “Pending” and your are “pending” to start a convoluted headbanging procedure to your desk. Calm down, let’s see some things.

1. Do you have a default storage class? Helm, if it cannot find a default or clearly stated Storage Class, uses *nil.* Please check with `kubectl get storageclass` if you have a default StorageClass. If not please use the guide above to make a StorageClass default.
2. We use a Local StorageClass. That means that we actually dont have Dynamic Allocation of space in our cluster (no provisioning). For each PVC we have to **manually** create a PV with the specs and requirement of the PVC using the folder /home/fint/storage
3. If you did all that and nothing worked, please use MatchLabels. See the guide above on how to use MatchLabels in PVC and Labels in PV.