OpenEBS Documentation

Release 0.3

[author]

Oct 09, 2017

Getting Started

Contents:

CHAPTER 1

OpenEBS Introduction

OpenEBS is a cloud native storage solution built with the goal of providing containerized storage for containers. Using OpenEBS, a developer can seamlessly get the persistent storage for stateful applications with ease, much of which is automated, while using the popular orchestration platforms such as Kubernetes.

Persistent storage presents significant challenges to the developer interfacing with stateful applications such as databases. While a developer can get the initial needs of persistent storage using Docker volume plug-in, Kubernetes stateful sets and so on, there is lots more to the storage needs of an application than just the connectivity.

Note:

OpenEBS integration support is provided only for Kubernetes.

A DevOps developer gets the following from the OpenEBS solution.

- OpenEBS operator yaml file that installs the OpenEBS components onto a k8s cluster
- A set of yaml files containing configuration examples of how to use OpenEBS storage classes
- A CLI for monitoring the persistent volume and its replicas

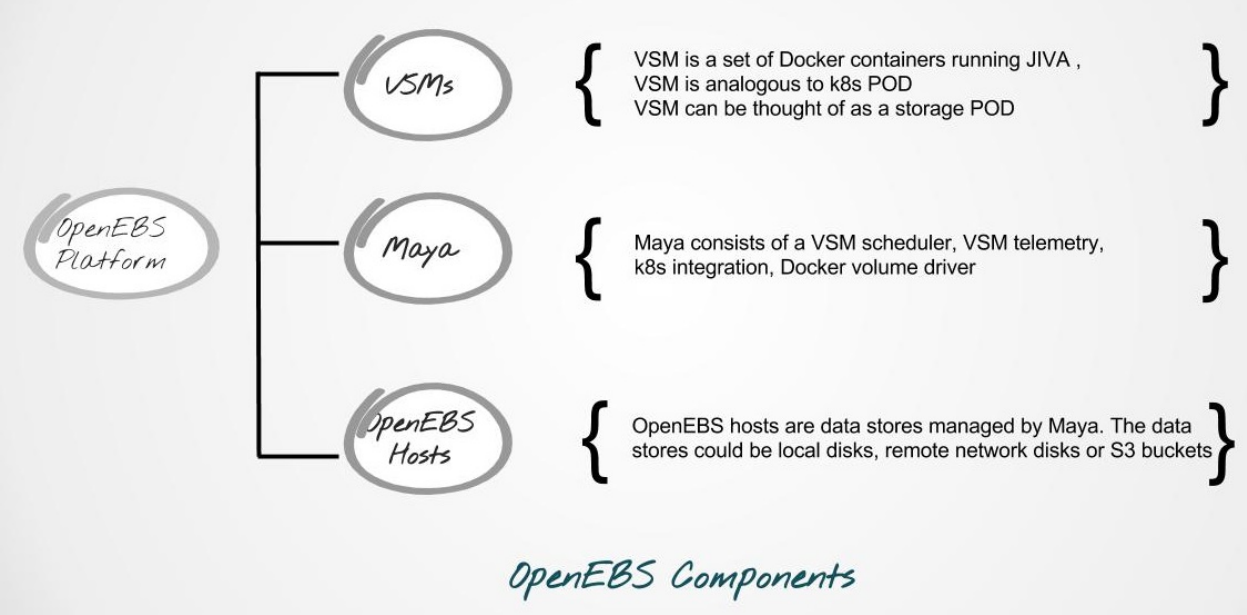
Using the above tools, a developer can easily provision the persistent storage from the hostdir of the minion node. Much of the tasks for the developer are automated by the OpenEBS storage class, including, scheduling the volume and replicas on k8s minions, connectivity to the container via a mount point.

1.1 Components

This section includes OpenEBS components.

OpenEBS platform contains storage containers:

- Storage PODs
- An orchestration engine or VSM Scheduler called Maya
- The OpenEBS hosts that provide the data store from either local disks or remote disks



See Also:

Changelog

CHAPTER 2

Quick Start

This section allows you to setup OpenEBS instantly.

2.1 Prerequisites

You must have the following:

- Kubernetes installed.
- iSCSI initiator enabled on your minion nodes. The procedure to setup iSCSI initiator will differ based on your host Operating System (OS).

The following example allows you set up Open-iSCSI on **Ubuntu**. Use the following commands at the command prompt:

- sudo apt-get update
- sudo apt-get install open-iscsi

CHAPTER 3

Setting up OpenEBS

Perform this procedure to run OpenEBS operator. Use the following commands at the command prompt:

1. kubectl create/apply -f openebs-operator.yaml
2. kubectl create/apply -f openebs-storageclasses.yaml

See Also:

Amazon Cloud

CHAPTER 4

How does Storage High Availability Work?

High Availability storage (HA storage) is a storage system that is continuously operational. Redundancy is a key feature of HA storage, as it allows data to be kept in more than one place and protects it.

OpenEBS Jiva Volume is a controller with one or more replicas. The controller is an iSCSI target whereas the replica plays the role of disk. The controller exposes the iSCSI target while the actual data is written. The controller and each individual replica run inside a dedicated container.

OpenEBS Jiva Volume controller exists as a single instance but there can be multiple instances of OpenEBS Jiva volume replicas. Peristent data is synchronized between replicas.

OpenEBS Jiva Volume HA is based on various scenarios as explained in the following sections.

4.1 OpenEBS Jiva Volume with two or more replicas and a single controller

NOTE: In this deployment, each of the replicas get scheduled in a unique K8s node, that is, a K8s node will never have two replicas of an OpenEBS volume.

Scenario 1 - When an OpenEBS volume controller POD crashes, the following occurs.

- The controller gets re-scheduled as a new Kubernetes POD.
- Policies are in place that ensures faster rescheduling.

Scenario 2 - When a K8s node that hosts OpenEBS volume controller goes offline, the following occurs.

- The controller gets re-scheduled as a new Kubernetes POD.
- Policies are in place that ensures faster rescheduling.
- If Kubernetes node is unavailable, the controller gets scheduled on one of the available nodes.

Scenario 3 - When an OpenEBS volume replica POD crashes for reasons other than node not-ready and node unreachable, the following occurs.

- The replica is re-scheduled as a new Kubernetes POD.

- The replica may or may not be re-scheduled on the same K8s node.
- There is data loss with this newly scheduled replica if it gets re-scheduled on a different K8s node.

Scenario 4 - When a K8s node that hosts OpenEBS volume replica goes offline, the following occurs.

- There is no storage downtime as the other available replica displays inputs/outputs.
- Policies are in place that does not allow re-scheduling of crashed replica (as replica is tied to a node's resources) to any other node.

CHAPTER 5

Setup

5.1 Setting Up OpenEBS - Overview

OpenEBS can run on various platforms: from your laptop, to VMs on a cloud provider. OpenEBS is also aimed at providing the option of using hybrid deployments where data is distributed between cloud and on-premise environments.

If you are beginner to Kubernetes and OpenEBS, OpenEBS recommends you to get started with either of the following:

- running an OpenEBS cluster on your laptop using Vagrant OR
- if you have access to the Cloud, you can use our custom solutions for Google and Amazon Cloud providers.

The Kubernetes documentation provides you various Kubernetes installation options to choose from. See Setup.

If you are already an experienced Kubernetes user, you can easily setup OpenEBS on your existing Kubernetes cluster with a few simple kubectl commands. See, *Quick Start*.

We are looking for help from the community in including additional platforms where OpenEBS has been successfully deployed. Please share your story through GitHub Issues or Pull requests.

CHAPTER 6

Cloud Solutions

6.1 Amazon Cloud

6.1.1 Setting up OpenEBS with Kubernetes on Amazon Web Services

This section provides instructions to set up a Kubernetes cluster on Amazon Web Services (AWS) and to have OpenEBS running in hyper converged mode.

Prerequisites

Perform the following procedure to setup the prerequisites for AWS.

1. **Signup for AWS here.** If you already have an AWS account, skip the above step.
2. Start your browser.
3. Open **AWS Management Console**.
4. Select **IAM** under **Security, Identity & Compliance**.
5. Under **Dashboard** in the left pane, click **Users**.
6. Click **Add user**.
7. In the **User name** field, enter the name of the user you want to create. For example, *openebsuser*.
8. Select **Access type** as **Programmatic access**.
9. Click **Next: Permissions**.
10. Select **Attach existing policies directly**.
11. In the **Search Box**, enter *IAMFullAccess* and select the listed permission.
12. Click **Next: Review**.
13. Click **Create user**.

A *openebsuser* user will be created and an Access key ID and a Secret access key will be assigned as in the following example.

```
User              Access key ID              Secret access key
openebsuser     AKIAI3MRLHNGUEXAMPLE      udxZi33tvSptXCky31kEt4KLRS6LSMMsmEXAMPLE
```

Note:

Note down the *Access key ID* and the *Secret access key* as AWS will not display it again.

6.1.2 kops, terraform and awscli

OpenEBS has created a script that does most of the work for you. Download the *oebs-cloud.sh* script file using the following commands.

```
$ mkdir -p openebs
$ cd openebs
$ wget https://raw.githubusercontent.com/openebs/openebs/master/e2e/terraform/oebs-
↪cloud.sh
$ chmod +x oebs-cloud.sh
```

The list of operations performed by the *oebs-cloud.sh* script are as follows:

```
 $ ./oebs-cloud.sh
 Usage :
     oebs-cloud.sh --setup-local-env
     oebs-cloud.sh --create-cluster-config [--ami-vm-os=[ubuntu|coreos]]
     oebs-cloud.sh --list-aws-instances
     oebs-cloud.sh --ssh-aws-ec2 [ ipaddress|=ipaddress]
     oebs-cloud.sh --help

$ ./oebs-cloud.sh
Usage :
   oebs-cloud.sh --setup-local-env
   oebs-cloud.sh --create-cluster-config
   oebs-cloud.sh --ssh-aws-ec2

Sets Up OpenEBS On AWS

 -h|--help                       Displays this help and exits.
 --setup-local-env               Sets up, AWSCLI, Terraform and KOPS.
 --create-cluster-config         Generates a terraform file (.tf) and Passwordless SSH
 --ami-vm-os                     The OS to be used for the Amazon Machine Image.
                                 Defaults to Ubuntu.
 --list-aws-instances            Outputs the list of AWS instances in the cluster.
 --ssh-aws-ec2                   SSH to Amazon EC2 instance with Public IP Address.
```

Running the following command allows you to install the required tools on your workstation.

```
$ ./oebs-cloud.sh --setup-local-env
```

The following tools are installed.

- awscli
- kops >= 1.6.2
- terraform >= 0.9.11

Updating the .profile File

The tools **awscli** and **kops** require the AWS credentials to access AWS services.

- Use the credentials that were generated earlier for the user *openebsuser*.
- Add path */usr/local/bin* to the PATH environment variable.

```
$ vim ~/.profile

# Add the AWS credentials as environment variables in .profile
export AWS_ACCESS_KEY_ID=<access key>
export AWS_SECRET_ACCESS_KEY=<secret key>

# Add /usr/local/bin to PATH
PATH="$HOME/bin:$HOME/.local/bin:/usr/local/bin:$PATH"

$ source ~/.profile
```

Creating the Cluster Configuration

- You must generate a terraform file (.tf) that will later spawn -
  - One Master
  - Two Nodes
- Run the following command in a terminal.

```
$ ./oebs-cloud.sh --create-cluster-config
```

Running *–create-cluster-config* command without any arguments defaults to **Ubuntu**. You can also run *–create-cluster-config* command with *–ami-vm-os=ubuntu* or *–ami-vm-os=coreos* commands and the following occurs.

- A *kubernetes.tf* terraform file is generated in the same directory.
- Passwordless SSH connection between the local workstation and the remote EC2 instances is established.

Note:

- The script uses *t2.micro* instance for the worker nodes, which must be well within the **Amazon Free Tier** limits.
- For process intensive containers you may have to modify the script to use *m3.large* instances, which could be charged.

Creating a Cluster on AWS using Terraform

- Run the following command to verify successful installation of terraform.

```
$ terraform
Usage: terraform [--version] [--help] <command> [args]

The available commands for execution are listed below. The most common and useful
commands are shown first,followed by less common or more advanced commands. If you
are just getting started with Terraform, use the common commands. For other␣
↪commands,
read the help and documentation before using them.
```

```
Common commands:

  apply              Builds or changes infrastructure
  console            Interactive console for Terraform interpolations
# ...
```

- Run the *terraform init* command to initialize terraform.
- Run the *terraform plan* command from the directory where the generated terraform file (.tf) is placed.
  - Terraform outputs a chunk of JSON data containing changes that would be applied on AWS.
  - *terraform plan* command verifies your terraform files (.tf) and displays errors that it encountered.
  - Fix these errors and verify again with the *terraform plan* command before running the terraform *apply* command.
- Run the command *terraform apply* to initiate infrastructure creation.

List AWS EC2 Instances

From your workstation, run the following command to list the AWS EC2 instances created.

```
$ ./oebs-cloud.sh --list-aws-instances

Node                            Private IP Address   Public IP Address
nodes.openebs.k8s.local         172.20.36.126        54.90.239.23
nodes.openebs.k8s.local         172.20.37.115        34.24.169.116
masters.openebs.k8s.local       172.20.53.140        34.202.205.27
```

SSH to the Kubernetes Node

From your workstation, run the following commands to connect to the EC2 instance running the Kubernetes Master.

For Ubuntu

```
$ ./oebs-cloud.sh --ssh-aws-ec2
Welcome to Ubuntu 16.04 LTS (GNU/Linux 4.4.0-93-generic x86_64)
ubuntu@ip-172-20-53-140 ~ $
```

For CoreOS

```
$ ./oebs-cloud.sh --ssh-aws-ec2
Container Linux by CoreOS stable (1465.6.0)
core@ip-172-20-53-140 ~ $
```

Running *–ssh-aws-ec2* command without any arguments, by default, connects you to the Kubernetes Master.

You can also run *–ssh-aws-ec2* command as *–ssh-aws-ec2=ipaddress*, where *ipaddress* is the Public IP Address of the AWS EC2 instance.

You should now be running inside the AWS EC2 instance.

Deploying OpenEBS on AWS

Kubernetes must be running on the EC2 instances while deploying OpenEBS. Verify if a Kubernetes cluster is created.

For Ubuntu

```
ubuntu@ip-172-20-53-140:~$ kubectl get nodes
NAME                              STATUS    AGE       VERSION
ip-172-20-36-126.ec2.internal   Ready     1m        v1.7.2
ip-172-20-37-115.ec2.internal   Ready     1m        v1.7.2
ip-172-20-53-140.ec2.internal   Ready     3m        v1.7.2
```

OpenEBS is deployed by the time you log in to Amazon Web Services (AWS).

```
ubuntu@ip-172-20-53-140:~$ kubectl get pods
NAME                        READY     STATUS    RESTARTS   AGE
maya-apiserver-h714w        1/1       Running   0          12m
openebs-provisioner-5e6ij 1/1       Running   0          9m
```

For CoreOS

```
core@ip-172-20-53-140:~$ kubectl get nodes
NAME                              STATUS    AGE       VERSION
ip-172-20-36-126.ec2.internal   Ready     1m        v1.7.2
ip-172-20-37-115.ec2.internal   Ready     1m        v1.7.2
ip-172-20-53-140.ec2.internal   Ready     3m        v1.7.2
```

OpenEBS is deployed by the time you log in to Amazon Web Services (AWS).

```
core@ip-172-20-53-140:~$ kubectl get pods
NAME                        READY     STATUS    RESTARTS   AGE
maya-apiserver-h714w        1/1       Running   0          12m
openebs-provisioner-5e6ij 1/1       Running   0          9m
```

6.2 Google Cloud

6.2.1 Setting up OpenEBS with Kubernetes on Google Container Engine

This section, provides detailed instructions on how to setup and use OpenEBS in Google Container Enginer (GKE). This section uses a three node container cluster.

1. Preparing your Container Cluster

You can either use an existing container cluster or create a new one. To create a new cluster, go to **Google Cloud Platform -> Container Engine -> Create Container Cluster**.

Minimum requirements for container cluster are as follows:

- Machine Type - (Minimum 2 vCPUs)
- Node Image - (container-vm)
- Size - (Minimum 3)
- Cluster Version - (1.6.4+)

Note:

The example commands below were run on a container cluster *demo-openebs03* in zone *us-central1-a* with project unique ID *strong-eon-153112*. When you copy paste the command, ensure that you use the details from your project.

6.2.2 Add iSCSI Support

SSH into the nodes of the cluster (**Google Cloud Platform -> Compute Engine -> VM instances**) to install open-iscsi package. OpenEBS uses iSCSI to connect to the block volumes.

```
sudo apt-get update
sudo apt-get install open-iscsi
sudo service open-iscsi restart
```

Verify that iSCSI is configured

Check that initiator name is configured and iSCSI service is running using the following commands.

```
sudo cat /etc/iscsi/initiatorname.iscsi
sudo service open-iscsi status
```

2. Run OpenEBS Operator through Google Cloud Shell

Download the latest OpenEBS Operator files using the following commands.

```
git clone https://github.com/openebs/openebs.git
cd openebs/k8s
```

Setup the kubectl to run in admin context. See *Appendix* below for creating an administration context in Google Cloud Platform (GCP. The following commands will prompt you for username and password. Provide username as *admin*. Password for the admin can be obtained from **Google Cloud Platform -> Container Engine -> (cluster) -> Show Credentials**

```
kubectl config use-context demo-openebs03
kubectl apply -f openebs-operator.yaml
kubectl config use-context gke_strong-eon-153112_us-central1-a_demo-openebs03
```

Add OpenEBS related storage classes, that can then be used by developers and applications using the following command.

```
kubectl apply -f openebs-storageclasses.yaml
```

Note:

The persistent storage is carved out from the space available on the nodes (default host directory : */var/openebs*). Development is in progress to provide administrator with additional options of consuming the storage (as outlined in *openebs-config.yaml*). These are slated to work hand-in-hand with the local storage manager of Kubernetes that is due in Kubernetes 1.7/1.8.

3. Running Stateful Workloads with OpenEBS Storage

To use OpenEBS as persistent storage for your stateful workloads, set the storage class in the Persistent Volume Claim (PVC) to the OpenEBS storage class.

Get the list of storage classes using the following command. Choose the storage class that best suits your application.

```
kubectl get sc
```

Some sample YAML files for stateful workloads using OpenEBS are provided in the openebs/k8s/demo

The *kubectl apply -f demo/jupyter/demo-jupyter-openebs.yaml* command creates the following, which can be verified using the corresponding kubectl commands.

- Launch a Jupyter Server, with the specified notebook file from github (kubectl get deployments)
- Create an OpenEBS Volume and mounts to the Jupyter Server Pod (/mnt/data) (kubectl get pvc) (kubectl get pv) (kubectl get pods)
- Expose the Jupyter Server to external world via the http://NodeIP:32424 (NodeIP is any of the nodes external IP) (kubectl get pods)

Note: To access the Jupyter Server over the internet, set the firewall rules to allow traffic on port 32424 in you GCP / Networking / Firewalls.

6.2.3 Appendix

Setting Kubernetes Cluster Administration Context

To create or modify service accounts and grant privileges, kubectl must be run with Administration privileges. The following procedure helps you setup and use the administration context for Google Container Engine through the Google Cloud Shell.

1. Initialize credentials to allow kubectl to execute commands on the container cluster.

```
gcloud container clusters list
gcloud container clusters get-credentials demo-openebs03 --zone us-central1-a
```

2. Setup the administration context.

- Access the credentails from **Google Cloud Platform -> Container Engine -> (cluster) -> Show Credentials**.
- Save the *Cluster CA Certificate* to *~/.kube/admin.key*.
- Create a administration configuration context from the configuration shell using the following commands.

```
gcloud container clusters list
kubectl config set-context demo-openebs03 --cluster=gke_strong-eon-153112_us-central1-
↪a_demo-openebs03 --user=cluster-a
```

CHAPTER 7

On-Premise Solutions

7.1 Using Vagrant

7.1.1 Setting Up OpenEBS with Kubernetes on Local Machine

The following procedure helps you setup and use OpenEBS on a local machine:

1. Install Vagrant Box

   To run the kubernetes cluster on local machine, you need a vagrant box. If you do not have vagrant box follow the steps gi

2. Download OpenEBS Vagrant file using the following command.

```
$ wget https://raw.githubusercontent.com/openebs/openebs/master/k8s/lib/vagrant/test/
↪k8s/1.6/Vagrantfile
```

3. Bring up k8s Cluster.

```
openebs@openebs:~$ cd openebs/k8s/lib/vagrant/test/k8s/1.6
openebs@openebs:~/openebs/k8s/lib/vagrant/test/k8s/1.6$ vagrant up
```

It will bring up one kubemaster and two kubeminions.

4. SSH to kubemaster using the following command.

```
openebs@openebs:~/openebs/k8s/lib/vagrant/test/k8s/1.6$ vagrant ssh kubemaster-01
```

5. Run OpenEBS Operator.

   - Download the latest OpenEBS Operator Files inside kubemaster-01 using the following commands.

```
ubuntu@kubemaster-01:~$ git clone https://github.com/openebs/openebs
ubuntu@kubemaster-01:~$ cd openebs/k8s
```

   - Run OpenEBS Operator using the following command.

```
ubuntu@kubemaster-01:~/openebs/k8s$ kubectl apply -f openebs-operator.yaml
```

- Add OpenEBS related storage classes using the following command, that can then be used by developers or applications.

```
ubuntu@kubemaster-01:~/openebs/k8s$ kubectl apply -f openebs-storageclasses.
↪yaml
```

6. Run stateful workloads with OpenEBS storage.

To use OpenEBS as persistent storage for your stateful workloads, set the storage class in the PVC to OpenEBS storage class.

Get the list of storage classes using the following command. Choose the storage class that best suits your application.

```
ubuntu@kubemaster-01:~$ kubectl get sc
```

Some sample yaml files for stateful workloads using OpenEBS are provided in the openebs/k8s/demo.

The *ubuntu@kubemaster-01:~$ kubectl apply -f demo/jupyter/demo-jupyter-openebs.yaml* command creates the following, which you can verify using the corresponding kubectl commands.

- Launch a Jupyter Server, with the specified notebook file from github (kubectl get deployments)
- Create an OpenEBS Volume and mount to the Jupyter Server Pod (/mnt/data) (kubectl get pvc) (kubectl get pv) (kubectl get pods)
- Expose the Jupyter Server to external world through http://NodeIP:8888 (NodeIP is any of the minion nodes' external IP) (kubectl get pods)

7.2 Using Ansible

7.2.1 Setting Up OpenEBS on Ubuntu Hosts or Virtual Machines

This section provides detailed instructions on how to perform the OpenEBS on-premise deployment. The objective of this procedure is to have the following functional.

- Kubernetes cluster (K8s master & K8s minions/host) configured with the OpenEBS iSCSI flexvol driver,
- OpenEBS Maya Master
- OpenEBS Storage Hosts

Depending on your need, you can either setup only the Kubernetes cluster or the OpenEBS cluster or both. The number of nodes in each category is configurable.

The Kubernetes cluster is setup, in this framework using *kubeadm*.

7.2.2 Running the Setup on Ubuntu 16.04

The following instructions have been verified on -

- Baremetal and VMware virtual machines installed with Ubuntu 16.04 64 bit
- Ubuntu 16.04 64 bit Vagrant VMs running on Windows 10 (Vagrant (>=1.9.1), VirtualBox 5.1)

7.2.3 Prerequisites:

- At least three Linux machines of either VMs or bare-metal, if deploying the setup in a hyperconverged mode (with K8s as well as OpenEBS residing on the same machines) or five Linux machines (with K8s and OpenEBS running on separate machines)
- The above instruction assumes a minimal setup with a test-harness, K8s/OpenEBS master and a single K8s minion/OpenEBS node. The masters and nodes can be scaled if the user so desires
- All Linux machines are required to have :
  - Basic development packages (dpkg-dev,gcc,g++,libc6-dev,make,libssl-dev,sshpass)
  - Python2.7-minimal
  - SSH services enabled
- The machine used as test-harness must also have the following:
  - Git
  - Ansible (version >= 2.3)
- The deployment can be performed by both root as well as non-root users. In case of the latter, ensure that the users are part of the sudo group. This is required to run certain operations which require root privileges.

7.2.4 Download

Setup the local working directory where the ansible code will be downloaded. Perform a git clone of the OpenEBS repository, and navigate to e2e/ansible.

```
testuser@OpenEBSClient:~$ git clone https://github.com/openebs/openebs.git
testuser@OpenEBSClient:~$ ls
openebs
testuser@OpenEBSClient:~$ cd openebs/e2e/ansible/
testuser@OpenEBSClient:~/openebs/e2e/ansible$ ls -l
total 68
-rw-rw-r--  1 testuser testuser 14441 Jun  5 09:29 ansible.cfg
-rw-rw-r--  1 testuser testuser   470 Jun  5 09:29 ci.yml
drwxrwxr-x  2 testuser testuser  4096 Jun  5 09:29 files
drwxrwxr-x  3 testuser testuser  4096 Jun  5 10:00 inventory
drwxrwxr-x  4 testuser testuser  4096 Jun  5 09:29 playbooks
drwxrwxr-x  3 testuser testuser  4096 Jun  5 09:29 plugins
-rw-rw-r--  1 testuser testuser    57 Jun  5 09:29 pre-requisites.yml
-rw-rw-r--  1 testuser testuser  7058 Jun  5 09:29 README.md
drwxrwxr-x 17 testuser testuser  4096 Jun  5 09:29 roles
-rw-rw-r--  1 testuser testuser  1864 Jun  5 09:29 run-tests.yml
-rw-rw-r--  1 testuser testuser   379 Jun  5 09:29 setup-openebs.yml
-rw-rw-r--  1 testuser testuser  4221 Jun  5 09:29 Vagrantfile
```

7.2.5 Setup Environment for OpenEBS Installation

- Setup environment variables for the usernames and passwords of all the machines which have been brought up in the previous steps on the test-harness (this machine will be interchangeably used with the term 'localhost'). Ensure that these are setup in the .profile of the localhost user which will be running the ansible code or playbooks, that is the ansible_user.

- Ensure that the env variables setup in the previous step are available in the current user session. Perform source ~/.profile to achieve the same and verify through echo $VARIABLE.
- Edit the *inventory/machines.in* file to place the latest HostCode, IP, username variable, password variable for all the machines setup. For more details on editing *machines.in*, see the Inventory README.
- Edit the global variables file *inventory/group_vars/all.yml* to reflect the desired storage volume properties and network CIDR that will be used by the maya api server to allot the IP for the volume containers. Also update the ansible run-time properties to reflect the machine type (is_vagrant), whether the playbook execution needs to be recorded using the Ansible Run Analysis framework (setup_ara), whether slack notifications are needed (in case they are required, a $SLACK_TOKEN env variable needs to be setup. The token is usually the last part of the slack webhook URL which is user generated) and so on.
- (Optional) Execute the setup_ara playbook to install the ARA notification plugins and custom modules. This step will cause changes to the ansible configuration file *ansible.cfg* (though a backup will be taken at the time of execution in case you need to revert). A web URL is provided as a playbook run message at the end of the ara setup procedure, which can be used to track all the playbook run details after this point.

```
testuser@OpenEBSClient:~/openebs/e2e/ansible$ ansible-playbook setup_ara.yml
```

- Note that the above playbook must be run separately and not as part of any the *master* playbook run as the changes to ansible default configuration may fail to take effect dynamically
- Execute the prerequisites ansible playbook to generate the ansible inventory, that is, *hosts* file from the data provided in the *machines.in* file.

```
testuser@OpenEBSClient:~/openebs/e2e/ansible$ ansible-playbook pre-requisites.yml
```

- Verify generation of the hosts file in the *openebs/e2e/ansible/inventory* directory. Check the *host-status.log* in the same location for details on the inventory file generation in case of any issues.

```
testuser@OpenEBSClient:~/openebs/e2e/ansible/inventory$ ls -ltr hosts
-rw-rw-r-- 1 testuser testuser 1482 Jun  5 10:00 hosts
```

- OpenEBS installation can be performed:
    1. in hyperconverged mode, where the OpenEBS storage services run as pods on the Kubernetes cluster itself.

The subsequent section explains the installation procedure for hyperconverged mode.

7.2.6 OpenEBS Installation - Hyperconverged Mode

- Update the *inventory/group_vars/all.yml* with the appropriate value *hyperconverged* for the key *deployment_mode*.
- In this mode, the OpenEBS maya-apiserver and openebs-storage provisioner are run as deployments on the Kubernetes cluster with associated pods, and the Kubernetes hosts act as the OpenEBS storage hosts as well. These are setup using an openebs-operator on the Kubernetes cluster. The setup also involves integration of OpenEBS storage-classes into the Kubernetes cluster. These essentially define the storage profile such as size, number of replicas, type of pool atec, and the provisioner associated with it.

  Applications can consume storage by specifying a persistent volume claim in which the storage class is an openebs-storage class.
- Setup the Kubernetes cluster using the setup-kubernetes playbook, followed by the setup-openebs playbook to deploy the OpenEBS pods. Internally, this runs the hyperconverged ansible role which executes the openebs-operator and integrates openebs-storage classes into the Kubernetes cluster.

  - Execute the setup-kubernetes ansible playbook to create the Kubernetes cluster followed by the setup-openebs playbook. These playbooks install the requisite dependencies on the machines, update the configuration files on the boxes and sets up Kubernetes cluster.

```
testuser@OpenEBSClient:~/openebs/e2e/ansible$ ansible-playbook setup-
↪kubernetes.yml
testuser@OpenEBSClient:~/openebs/e2e/ansible$ ansible-playbook setup-
↪kubernetes.yml
```

  - Check status of the Kubernetes cluster

```
name@KubeMaster:~$ kubectl get nodes
NAME          STATUS    AGE       VERSION
kubehost01    Ready     2d        v1.6.3
kubehost02    Ready     2d        v1.6.3
kubemaster    Ready     2d        v1.6.3
```

- Verify that the Kubernetes cluster is running using the kubectl get nodes command.
- Verify that the maya-apiserver and openebs-provisioner are deployed successfully on the Kubernetes cluster.

```
name@MayaMaster:~$ kubectl get deployments
NAME                  DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
maya-apiserver        1         1         1            1           4h
openebs-provisioner   1         1         1            1           4h
name@MayaMaster:~$ kubectl get pods
NAME                                     READY     STATUS    RESTARTS   AGE
maya-apiserver-1633167387-v4sf1          1/1       Running   0          4h
openebs-provisioner-1174174075-n989p     1/1       Running   0          4h
```

- Verify that the OpenEBS storage classes are applied successfully.

```
name@MayaMaster:~$ kubectl get sc
NAME              TYPE
openebs-basic     openebs.io/provisioner-iscsi
openebs-jupyter   openebs.io/provisioner-iscsi
openebs-percona   openebs.io/provisioner-iscsi
```

7.2.7 Run Sample Applications on the OpenEBS Setup

- Test the OpenEBS setup installed using the above procedure by deploying a sample application pod.
- *run-hyperconverged-tests.yml* can be used to run tests on the hyperconverged installation.
- By default, all tests are commented in the above playbooks. Uncomment the desired test and execute the playbook. In the example below, a percona mysql DB is deployed on a hyperconverged installation.

```
ciuser@OpenEBSClient:~/openebs/e2e/ansible$ ansible-playbook run-hyperconverged-
↪tests.yml
```

- Verify that the pod is deployed on the Kubernetes minion along with the OpenEBS storage pods created as per the storage-class in the persistent volume claim, by executing the following command on the Kubernetes master.

```
name@MayaMaster:~$ kubectl get pods
NAME                                                              READY     STATUS␣
↪  RESTARTS   AGE
maya-apiserver-1633167387-v4sf1                                   1/1       Running␣
↪  0          4h
```

```
openebs-provisioner-1174174075-n989p                              1/1       Running␣
↪  0          4h
percona                                                           1/1       Running␣
↪  0          2m
pvc-4644787a-5b1f-11e7-bf1c-000c298ff5fc-ctrl-693727538-dph14     1/1       Running␣
↪  0          2m
pvc-4644787a-5b1f-11e7-bf1c-000c298ff5fc-rep-871457607-l392p      1/1       Running␣
↪  0          2m
pvc-4644787a-5b1f-11e7-bf1c-000c298ff5fc-rep-871457607-n9m73      1/1       Running␣
↪  0          2m
```

- For more details about the pod, execute the following command.

```
kubectl describe pod <pod name>
```

- The storage volume that is the persistent volume associated with the persistent volume claim, can be viewed using the *vsm-list* command in the maya-apiserver pod.

```
name@MayaMaster:~$ kubectl exec maya-apiserver-1633167387-v4sf1 -c maya-apiserver␣
↪-- maya vsm-list
Name                                        Status
pvc-a2a6d71f-5b21-11e7-bf1c-000c298ff5fc  Running
```

- Verify that the storage volume is receiving input/output by checking the increments to *DataUpdateIndex* in the output of the vsm-stats command issued in the maya-apiserver pod. Some additional performance statistics are also available in the command output.

```
 name@MayaMaster:~$ kubectl exec maya-apiserver-1633167387-v4sf1 -c maya-
↪apiserver -- maya vsm-stats pvc-a2a6d71f-5b21-11e7-bf1c-000c298ff5fc
------------------------------------
  IQN: iqn.2016-09.com.openebs.jiva:pvc-a2a6d71f-5b21-11e7-bf1c-000c298ff5fc
Volume: pvc-a2a6d71f-5b21-11e7-bf1c-000c298ff5fc
Portal: 10.104.223.35:3260
  Size: 5G

Replica         Status      DataUpdateIndex
10.36.0.2       Online      2857
10.44.0.3       Online      2857
------------------------------------
 r/s|    w/s|    r(MB/s)|    w(MB/s)|    rLat(ms)|    wLat(ms)|    rBlk(KB)|    wBlk(KB)|
   0|      3|      0.000|      1.109|       0.000|      10.602|           0|         378|
name@MayaMaster:~$
```

7.2.8 Tips and Gotchas

- Use the -v flag while running the playbooks to enable verbose output and logging. Increase the number of 'v's to increase the verbosity.
- Sometimes, the minions take time to join the Kubernetes master. This could be caused due to slow internet or less resources on the box. The time could range between a few seconds to a few minutes.
- As with minions above, the OpenEBS volume containers (Jiva containers) may take some time to get initialized (involves a docker pull) before they are ready to input/output. Any pod deployment (which uses the openEBS iSCSI flexvol driver) while in progress, gets queued and resumes once the storage is ready.

CHAPTER 8

Developer Solutions

8.1 Minikube

8.1.1 Setting up OpenEBS with Kubernetes using Minikube

Minikube helps developers to quickly setup a single-node Kubernetes cluster for their development environment. There are several options available for developers to install Minikube. See, Minikube Setup.

If you are already an experienced Minikube user, you can easily setup OpenEBS on your existing Kubernetes cluster with a few simple kubectl commands. See, *Quick Start*.

This section provides instructions to set up Kubernetes using Minikube directly on Ubuntu 16.06 (without using any VM drivers) and to have OpenEBS running in hyperconverged mode.

8.1.2 Prerequisites

Minimum requirements for using Minikube

- Machine Type - (Minimum 4 vCPUs)
- RAM - (Minimum 4 GB)

Make sure *docker* is installed on your Ubuntu host.

8.1.3 Add iSCSI Support

On your Ubuntu host, install open-iscsi package. OpenEBS uses iSCSI to connect to the block volumes.

```
sudo apt-get update
sudo apt-get install open-iscsi
sudo service open-iscsi restart
```

Verify that iSCSI is configured

Check that initiator name is configured and iSCSI service is running using the following commands.

```
sudo cat /etc/iscsi/initiatorname.iscsi
sudo service open-iscsi status
```

8.1.4 Download and setup Minikube and kubectl

On your Ubuntu host, install minikube.

```
curl -Lo minikube https://storage.googleapis.com/minikube/releases/latest/minikube-
↪linux-amd64
chmod +x minikube
sudo mv minikube /usr/local/bin/
```

On your Ubuntu host, install kubectl.

```
curl -Lo kubectl https://storage.googleapis.com/kubernetes-release/release/$(curl -s␣
↪https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/
↪amd64/kubectl
chmod +x kubectl
sudo mv kubectl /usr/local/bin/
```

On your Ubuntu host, setup directories for storing minkube and kubectl configuration.

```
mkdir $HOME/.kube || true
touch $HOME/.kube/config
```

On your Ubuntu host, setup env for minikube. Copy the following to ~/.profile.

```
export MINIKUBE_WANTUPDATENOTIFICATION=false
export MINIKUBE_WANTREPORTERRORPROMPT=false
export MINIKUBE_HOME=$HOME
export CHANGE_MINIKUBE_NONE_USER=true
export KUBECONFIG=$HOME/.kube/config
```

On your Ubuntu host, start minikube.

```
sudo -E minikube start --vm-driver=none
```

Verify that minikube is configured

Check that minikube is configured and it has started using the following commands.

```
minikube status
```

When minikube is configured properly, *minikube status* will display the following output:

```
minikube: Running
cluster: Running
kubectl: Correctly Configured: pointing to minikube-vm at 127.0.0.1
```

Note

- If minikube status displays *Stopped*, add sudo minikube start command.
- If minikube displays errors indicating permission denied to configuration files, fix the permissions by running the following commands.

```
sudo chown -R $USER $HOME/.kube
sudo chgrp -R $USER $HOME/.kube
sudo chown -R $USER $HOME/.minikube
sudo chgrp -R $USER $HOME/.minikube
```

Verify that Kubernetes is configured

Check that kubectl is configured and services are running using the following commands.

```
kubectl get pods
kubectl get nodes
```

When configured properly, the above kubectl commands will display output similar to following:

```
vagrant@minikube-dev:~$ kubectl get nodes
NAME           STATUS    AGE       VERSION
minikube-dev   Ready     8m        v1.7.5
vagrant@minikube-dev:~$ kubectl get pods --all-namespaces
NAMESPACE     NAME                                   READY     STATUS    RESTARTS   AGE
kube-system   kube-addon-manager-minikube-dev        1/1       Running   1          8m
kube-system   kube-dns-910330662-4q4bm               3/3       Running   3          8m
kube-system   kubernetes-dashboard-txn8f             1/1       Running   1          8m
vagrant@minikube-dev:~$
```

8.1.5 Setup OpenEBS

Download the latest OpenEBS Operator files using the following commands.

```
git clone https://github.com/openebs/openebs.git
cd openebs/k8s
kubectl apply -f openebs-operator.yaml
```

Note By default, OpenEBS launches OpenEBS Volumes with two replicas. To set one replica, as is the case with single-node Kubernetes cluster, specify the env variable *DEFAULT_REPLICA_COUNT=1*. This is supported in OpenEBS version 0.4 onwards.

The following snippet of the openebs-operator.yaml -> maya-apiserver section shows the addition of DEFAULT_REPLICA_COUNT:

```
---
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: maya-apiserver
  namespace: default
spec:
  replicas: 1
  template:
    metadata:
      labels:
        name: maya-apiserver
```

```
    spec:
      serviceAccountName: openebs-maya-operator
      containers:
      - name: maya-apiserver
        imagePullPolicy: Always
        image: openebs/m-apiserver:0.3-RC4
        ports:
        - containerPort: 5656
        env:
        - name: DEFAULT_REPLICA_COUNT
          value: "1"
---
```

Add OpenEBS related storage classes, that can then be used by developers and applications using the following command.

```
kubectl apply -f openebs-storageclasses.yaml
```

8.1.6 Running Stateful Applications with OpenEBS Storage

To use OpenEBS as persistent storage for your stateful workloads, set the storage class in the Persistent Volume Claim (PVC) of your application to one of the OpenEBS storage class.

Get the list of storage classes using the following command. Choose the storage class that best suits your application.

```
kubectl get sc
```

Some sample YAML files for stateful workloads using OpenEBS are provided in the openebs/k8s/demo

CHAPTER 9

Prerequisites

Prerequisites include the following:

- A fully configured Kubernetes cluster (versions 1.6.3/4/6 and 1.7.0 have been tested) with kube master and at least one kube node. This maybe created on cloud platforms like GKE, on-premise virtual machines (vagrant/VMware/Hyper-V) or bare-metal boxes.

Note:

- OpenEBS recommends using a 3-node cluster, with one master and two nodes. This aids in creating storage replicas on separate nodes and is helpful in maintaining redundancy and data availability.
- If you are using gcp, view the appendix in this section for additional steps to set up cluster administration context and use it.

Verify that the Kubernetes cluster is in optimal state by using the following commands.

```
name@Master:~$ kubectl get nodes
NAME         STATUS    AGE       VERSION
host01   Ready     5d        v1.6.3
host02   Ready     5d        v1.6.3
master   Ready     5d        v1.6.3
```

- Sufficient resources on the nodes to host the OpenEBS storage pods and Percona application pods - This includes sufficient disk space, in this example, physical storage for the pvolume containers will be carved out from the local storage.
- iSCSI support on the nodes - This is required for being able to consume the iSCSI target exposed by the OpenEBS volume container (that is, VSM). In ubuntu, you can install the iSCSI initiator using the following procedure.

```
sudo apt-get update
sudo apt-get install open-iscsi
sudo service open-iscsi restart
```

Verify that iSCSI is configured using the following commands.

```
sudo cat /etc/iscsi/initiatorname.iscsi
sudo service open-iscsi status
```

CHAPTER 10

Run OpenEBS Operator

Download the latest OpenEBS operator files and sample *percona-mysql* application pod yaml on the kubemaster from the OpenEBS git repository.

```
git clone https://github.com/openebs/openebs.git
cd openebs/k8s
```

Apply openebs-operator on the Kubernetes cluster. This creates the maya api-server and openebs provisioner deployments.

```
kubectl apply -f openebs-operator.yaml
```

Add the OpenEBS storage classes using the following command, that can then be used by developers to map a suitable storage profile for their applications in their respective persistent volume claims.

```
kubectl apply -f openebs-storageclasses.yaml
```

Check whether the deployments are running successfully using the following commands.

```
name@Master:~$ kubectl get deployments
NAME                                    DESIRED   CURRENT   UP-TO-DATE   ␣
↪AVAILABLE   AGE
maya-apiserver                          1         1         1            1    ␣
↪       2m
openebs-provisioner                     1         1         1            1    ␣
↪       2m
```

```
name@Master:~$ kubectl get pod
NAME                                    READY     STATUS    RESTARTS   AGE
maya-apiserver-1633167387-5ss2w         1/1       Running   0          24s
openebs-provisioner-1174174075-f2ss6    1/1       Running   0          23s
```

Check whether the storage classes are applied successfully using the following commands.

```
name@Master:~$ kubectl get sc
NAME              TYPE
openebs-basic     openebs.io/provisioner-iscsi
openebs-jupyter   openebs.io/provisioner-iscsi
openebs-percona   openebs.io/provisioner-iscsi
```

CHAPTER 11

Percona DB

11.1 Running Percona Pod on OpenEBS

This section provides detailed instructions on how to run a *percona-mysql* application pod on OpenEBS storage in a Kubernetes cluster and uses a *mysql-client* container to generate load (in the form of insert and select DB queries) in order to illustrate input/output traffic on the storage.

11.2 Run percona-mysql Pod with OpenEBS Storage

Use OpenEBS as persistent storage for the percona pod by selecting an OpenEBS storage class in the persistent volume claim. A sample percona pod yaml (with container attributes and pvc details) is available in the OpenEBS git repository (which was cloned in the previous steps).

Apply the percona pod yaml using the following commands.

```
cd demo/percona
kubectl apply -f demo-percona-mysql-pvc.yaml
```

Verify that the OpenEBS storage pods, that is, the jiva controller and jiva replicas are created and the percona pod is running succesfully using the following commands.

```
name@Master:~$ kubectl get pods
NAME                                                              READY     STATUS    ␣
↪RESTARTS   AGE
maya-apiserver-1633167387-5ss2w                                   1/1       Running   ␣
↪0          4m
openebs-provisioner-1174174075-f2ss6                              1/1       Running   ␣
↪0          4m
percona                                                           1/1       Running   ␣
↪0          2m
pvc-016e9a68-71c1-11e7-9fea-000c298ff5fc-ctrl-2825810277-rjmxh    1/1       Running   ␣
↪0          2m
```

```
pvc-016e9a68-71c1-11e7-9fea-000c298ff5fc-rep-2644468602-92lfg     1/1       Running
↪0          2m
pvc-016e9a68-71c1-11e7-9fea-000c298ff5fc-rep-2644468602-rm8mz     1/1       Running
↪0          2m
```

Note:

It may take some time for the pods to start as the images must be pulled and instantiated. This is also dependent on the network speed.

11.3 Run a Database Client Container to Generate SQL Load

To test the pod, you can run a Kubernetes job, in which a mysql client container runs a load generation script (which in turn performs simple sql queries) to simulate storage traffic. Run the following procedure on any node in the Kubernetes cluster.

Get the IP address of the percona application pod. You can obtain this by executing kubectl describe on the percona pod.

```
name@Master:~$ kubectl describe pod percona | grep IP
IP:             10.44.0.3
```

Edit the following line in sql-loadgen job yaml to pass the desired load duration and percona pod IP as arguments. In this example, the job performs sql queries on pod with IP address 10.44.0.3 for 300s.

```
args: ["-c", "timelimit -t 300 sh MySQLLoadGenerate.sh 10.44.0.3 > /dev/null 2>&1;
↪exit 0"]
```

Run the load generation job using the following command.

```
kubectl apply -f sql-loadgen.yaml
```

11.4 View Performance and Storage Consumption Statistics Using mayactl

Performance and capacity usage statistics on the OpenEBS storage volume can be viewed by executing the following *mayactl* command inside the maya-apiserver pod.

Start an interactive bash console for the maya-apiserver container using the following command.

```
kubectl exec -it maya-apiserver-1633167387-5ss2w /bin/bash
```

Lookup the storage volume name using the *vsm-list* command

```
name@Master:~$ kubectl exec -it maya-apiserver-1633167387-5ss2w /bin/bash

root@maya-apiserver-1633167387-5ss2w:/# maya vsm-list
Name                                    Status
pvc-016e9a68-71c1-11e7-9fea-000c298ff5fc  Running
```

Get the performance and capacity usage statistics using the *vsm-stats* command.

```
root@maya-apiserver-1633167387-5ss2w:/# maya vsm-stats pvc-016e9a68-71c1-11e7-9fea-
↪000c298ff5fc
-------------------------------------
 IQN     : iqn.2016-09.com.openebs.jiva:pvc-016e9a68-71c1-11e7-9fea-000c298ff5fc
 Volume  : pvc-016e9a68-71c1-11e7-9fea-000c298ff5fc
 Portal  : 10.109.70.220:3260
 Size    : 5G

      Replica|   Status|   DataUpdateIndex|
             |         |                  |
    10.36.0.3|   Online|              4341|
    10.44.0.2|   Online|              4340|

------------ Performance Stats ----------

   r/s|   w/s|   r(MB/s)|   w(MB/s)|   rLat(ms)|   wLat(ms)|   rBlk(KB)|   wBlk(KB)|
     0|    14|     0.000|    14.000|      0.000|     71.325|          0|       1024|

------------ Capacity Stats -------------
   Logical(GB)|   Used(GB)|
      0.074219|   0.000000|
```

The above command can be invoked using the *watch* command by providing a desired interval to continuously monitor statistics.

```
watch -n 1 maya vsm-stats pvc-016e9a68-71c1-11e7-9fea-000c298ff5fc
```

CHAPTER 12

Jupyter

12.1 Running Jupyter on OpenEBS

This section provides detailed instructions on how to run a jupyter pod on OpenEBS storage in a Kubernetes cluster and uses a *jupyter ui editor* to generate load in order to illustrate input/output traffic on the storage.

12.2 Run Jupyter Pod with OpenEBS Storage

Use OpenEBS as persistent storage for the jupyter pod by selecting an OpenEBS storage class in the persistent volume claim. A sample jupyter pod yaml (with container attributes and pvc details) is available in the OpenEBS git repository (which was cloned in the previous steps).

```
name@Master:~$ cat demo-jupyter-openebs.yaml

...

kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: jupyter-data-vol-claim
spec:
  storageClassName: openebs-jupyter
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5G
...
```

Apply the jupyter pod yaml using the following command.

```
name@Master:~$ kubectl apply -f demo-jupyter-openebs.yaml
deployment "jupyter-server" created
persistentvolumeclaim "jupyter-data-vol-claim" created
service "jupyter-service" created
```

The above command creates the following, which can be verified using the corresponding kubectl commands.

- Launches a Jupyter Server, with the specified notebook file from github (kubectl get deployments)
- Creates an OpenEBS Volume and mounts to the Jupyter Server Pod (/mnt/data) (kubectl get pvc) (kubectl get pv) (kubectl get pods)
- Exposes the Jupyter Server to external world via the http://<NodeIP>:32424 (NodeIP is any of the nodes external IP) (kubectl get pods)

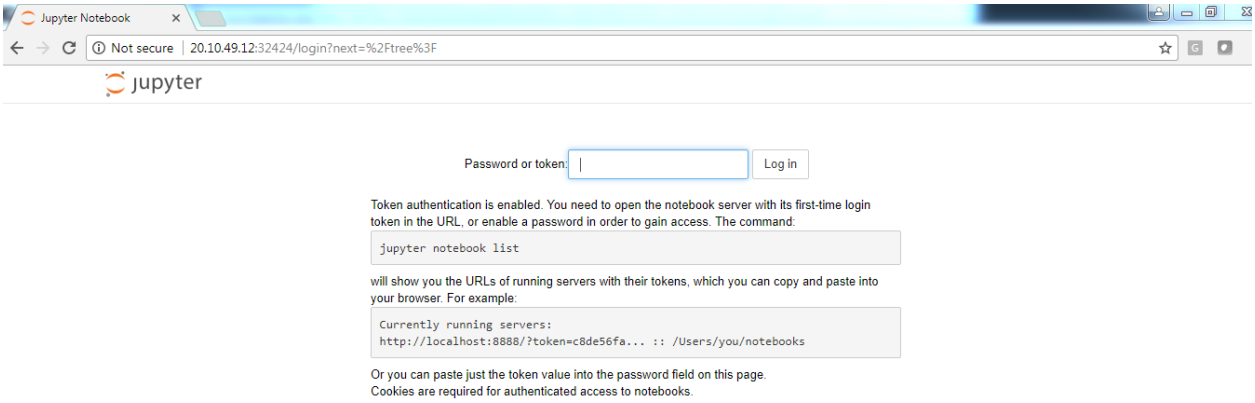
Verify that the OpenEBS storage pods, that is, the jiva controller and jiva replicas are created and the jupyter pod is running succesfully using the following commands.

```
name@Master:~$ kubectl get pods
NAME                                                              READY     STATUS    ␣
↪RESTARTS   AGE
jupyter-server-2764185079-s371g                                   1/1       Running   ␣
↪0          13m
maya-apiserver-1633167387-845fd                                   1/1       Running   ␣
↪0          15d
openebs-provisioner-1174174075-c78sj                              1/1       Running   ␣
↪1          15d
pvc-5467cfe7-a29e-11e7-b4df-000c298ff5fc-ctrl-2903536303-75h3j    1/1       Running   ␣
↪0          13m
pvc-5467cfe7-a29e-11e7-b4df-000c298ff5fc-rep-2383373508-bh0d3     1/1       Running   ␣
↪0          13m
pvc-5467cfe7-a29e-11e7-b4df-000c298ff5fc-rep-2383373508-s1kzz     1/1       Running   ␣
↪0          13m
```

Note:

It may take some time for the pods to start as the images must be pulled and instantiated. This is also dependent on the network speed.

The jupyter server dashboard can be accessed on the Kubernetes node port as in the following screen.



CHAPTER 13

Crunchy Postgres

13.1 Running Crunchy Postgres on OpenEBS

The following steps bring up a postgresql stateful set with one master and one replica on OpenEBS storage. This example uses centos-based postgresql containers from crunchy data to illustrate the same.

Download the files to your host, which has access to kubectl using the following commands.

```
cd $HOME
git clone https://github.com/openebs/openebs.git
cd openebs/k8s/demo/crunchy-postgres
```

The size of the OpenEBS persistent storage volume is 400M by default. You can edit the size in the storage class section of the *set.json* specification file.

```
cat set.json
..
"volumeClaimTemplates": [
{
"metadata": {
"name": "pgdata"
},
"spec": {
"accessModes": [
"ReadWriteOnce"
],
"storageClassName": "openebs-basic",
"resources": {
"requests": {
"storage": "400M"
}
}
}
}
..
```

Run the StatefulSet using the following command. The files are available with default images and credentials (*set.json*). The following command will automatically create the OpenEBS volumes required for master and replica postgresql containers.

```
./run.sh
```

Volume details can be inspected using the following standard kubectl commands.

```
kubectl get pvc
kubectl get pv
```

13.2 References

The k8s spec files are based on the files provided by CrunchyData StatefulSet with Dynamic Provisioner.

Kubernetes Blog for running Clustered PostgreSQL using StatefulSet.

CHAPTER 14

Troubleshooting Overview

This section provides information to help you troubleshoot common problems.

The list of OpenEBS issues and their workarounds are available either at the OpenEBS Slack channel or at the OpenEBS GitHub repository. Go to the following links to view the list of OpenEBS issues.

- **OpenEBS issues on Slack**
- **OpenEBS issues on GitHub**

If your problem is not answered in any of the above URLs, you can refer to the documentation where the Components and Architecture section explains the OpenEBS architecture and how each component works, while the *Setup* section provides practical instructions for setting up OpenEBS.

14.1 Slack

The OpenEBS team hangs out on Slack in the #openebs-users channel. To participate in discussions with the OpenEBS team go to Slack OpenEBS Community.

Slack requires registration, but the OpenEBS team welcomes anyone to register at http://slack.openebs.io/. Feel free to come and ask any questions.

14.2 Bugs and Feature Requests

If you have what looks like a bug, or you would like to make a feature request, please use the GitHub Issue Tracking System.

Before you file an issue, please search existing issues to see if your issue is already covered.

CHAPTER 15

Changelog

You can try out OpenEBS v0.4.0 on your Kubernetes Cluster using the quick start guide.

15.1 Downloads for v0.4.0

The following OpenEBS v0.4.0 containers are available at the Docker Hub.

- openebs/jiva:0.4.0 : Storage Controller
- openebs/m-apiserver:0.4.0 : OpenEBS Maya API Server along with the latest maya cli.
- openebs/openebs-k8s-provisioner:0.4.0 : Dynamic OpenEBS Volume Provisioner for Kubernetes.

15.2 New v0.4.0 Features

- Maya CLI Support for managing snapshots for OpenEBS Volumes
- Maya CLI Support for obtaining the capacity usage statistics from OpenEBS Volumes
- OpenEBS Volume - Dynamic Provisioner is merged into kubernetes-incubator/external-storage project
- OpenEBS Maya API Server uses the Kubernetes scheduler logic to place OpenEBS Volume Replicas on different nodes
- OpenEBS Maya API Server can be customized by providing ENV options through K8s YAML file for default replica count and jiva image to be used
- OpenEBS user documentation is avilable at http://openebs.readthedocs.io/en/latest/
- OpenEBS now supports deployment on AWS, along with previously supported Google Cloud and On-premise setups
- OpenEBS Vagrant boxes are upgraded to support Kubernetes version 1.7.5
- OpenEBS can now be deployed within a minikube setup

15.3 Issues Fixed in v0.4.0

- #166 (https://github.com/edorid): openebs-k8s-provisioner goes into crashloopbackoff, during the first volume creation
- #176 (https://github.com/maikotz): OpenEBS PV is unreachable after one of the replica becames unreachable.

15.4 CI Updates with v0.4.0

- Support for on-premise Jenkins CI for performing e2e tests
- iSCSI compliance tests are run as part of the CI
- CI can now be extended using a framework developer for running storage benchmark tests with vdbench or fio.
- CI has been extended to run Percona Benchmarking tests on Kubernetes.

15.5 Deprecated with v0.4.0

The maya cli options (setup-omm, setup-osh, omm-status, osh-status) to setup and manage dedicated OpenEBS setup is removed. Starting with v0.4.0, only hyperconvergence with Kubernetes is supported.

15.6 Notes for Contributors

- OpenEBS user documentation is currently being moved into *openebs/openebs/documentation*
- OpenEBS developer documentation is currently being added to *openebs/openebs/contribute*
- The deployment and e2e functionality will continue to be located in *openebs/k8s* and *openebs/e2e* respectively.
- openebs/maya will act as a single repository for hosting different OpenEBS Storage Control plane (orchestration) components.
- New /metrics handlers are being added to OpenEBS components to allow integration into tools like Prometheus.
- *openebs/maya/cmd/maya-agent* which will be deployed as a deamon-set running along-side kubelet is being developed. maya-agent will augument the kubelet with storage management functionality.

CHAPTER 16

OpenEBS Contribution Guide

You can access the latest documents at http://openebs.readthedocs.io/en/latest/.

Automated builds are setup for OpenEBS documentation at https://readthedocs.org/projects/openebs/

Sphinx is used for building the OpenEBS documentation - http://www.sphinx-doc.org/en/stable/tutorial.html

1. If you would like to give feedback on existing content, create an issue on documentation (see, *Creating an Issue*).
2. If you would like to contribute to new content,
   - create an issue (see, *Creating an Issue*),
   - create your own branch (see, *Creating a Branch*),
   - work on the content by creating an RST file, and
   - create a pull request (see, *Creating a Pull Request*).

16.1 Review Process for Documentation Issues and Pull Requests

1. OpenEBS Lead receives documentation issues/pull requests raised by Documentation/Internal contributors.
2. Lead will tag issues to relevant labels which start with the name **documentation** and assign reviewers and approvers such as feature owner, doc person, and approver.
3. The assignee works on the issue either developing content in RST files or editing the content.
4. The assignee can get the content from other collaborators of the OpenEBS project either as rst/md file or as a comment in the issue. The assignee can also use the "OpenEBS Slack" channel to collect additional information from the community.

   Documentation content is located under documentation/source in reStructured (rst) files. documentation/source/index.rst contains the high level documentation structure (Table of Contents), which links to the content provided in other rst files either in the same directory or in child directories.

   Note:

Before editing the files, familiarize yourself with the reStructured markup.

5. Once you are done with your edits and ready for review, you must create a PR (see *Creating a Pull Request*).
6. The documents must be approved. (see, *PR Approval Process*).

16.2 Creating an Issue

1. Go to https://github.com/openebs/openebs/issues.
2. Click **New issue**.
3. Add a short description in the **Title**.
4. You can enter a detailed description in the edit box below.
5. Click **Submit new issue**.

16.3 Creating a Branch

1. Create your openebs fork from (https://github.com/openebs/openebs). If you already have a forked openebs, rebase with the master to get the latest changes.
2. Create a branch on the openebs fork using the following command. :: git checkout -b <issue name>-#<issue number>

16.4 Creating a Pull Request

1. Go to your fork on Github.
2. Under Branch: master select the branch you created.
3. Click **New pull request**.
4. Add "Fixes #<*Issue number*>" in the commit message.

16.5 PR Approval Process

1. Once the assignee is ready with the final draft, the reviewers have to approve the content.
2. Open the pull request and click on **Review changes**.
3. Click **Approve** and **Submit review**.
4. The approver sees that the document is approved by all the reviewers and closes the issue. The issue gets merged and the documentation is available at http://openebs.readthedocs.io/en/latest/.