Ember CSI plugin Documentation

Release 0.0.2

Gorka Eguileor

Dec 05, 2018

Contents

Contents:

CHAPTER 1

Ember CSI

Multi-vendor CSI plugin driver supporting over 80 storage drivers in a single plugin to provide `block` and `mount` storage to Container Orchestration systems.

- Free software: Apache Software License 2.0
- Documentation: Pending

1.1 Features

This CSI driver is up to date with latest CSI specs including the new snapshots feature recently introduced.

Currently supported features are:

- Create block volume
- Creating snapshots
- Creating a block volume from a snapshot
- Delete block volume
- Deleting snapshots
- Listing volumes with pagination
- Listing snapshots with pagination
- Attaching volumes
- Detaching volumes
- Reporting storage capacity
- Probing the node
- Retrieving the plugin info

1.2 Runtime Dependencies

This driver requires that Cinder v11.0 (OSP-12/Pike) is already installed in the system, how this is accomplished is left to the installer, as there are multiple ways this can be accomplished:

- From OSP repositories
- From RDO repositories
- From github
- From other repositories

Any other basic requirement is already handled by `ember-csi` when installing from PyPi.

Besides the basic dependencies there are also some drivers that have additional requirements that must be met for proper operation of the driver and/or attachment/detachment operations, just like in Cinder.

Some of these Python dependencies for the Controller servicer are:

- DRBD: dbus and drbdmanage
- HPE 3PAR: python-3parclient
- Kaminario: krest
- Pure: purestorage
- Dell EMC VMAX, IBM DS8K: pyOpenSSL
- HPE Lefthad: python-lefthandclient
- Fujitsu Eternus DX: pywbem
- IBM XIV: pyxcli
- RBD: rados and rbd
- Dell EMC VNX: storops
- Violin: vmemclient
- INFINIDAT: infinisdk, capacity, infy.dtypes.wwn, infi.dtypes.iqn

Other backends may also require additional packages, for example LVM on CentOS/RHEL requires the `targetcli` package, so please check with your hardware vendor.

Besides the Controller requirements there are usually requirements for the Node servicer needed to handle the attaching and detaching of volumes to the node based on the connection used to access the storage. For example:

- iSCSI: iscsi-initiator-tools and device-mapper-multipath
- RBD/Ceph: ceph-common package

1.3 Installation

First we need to install the Cinder Python package, for example to install from RDO on CentOS:

```
$ sudo yum install -y centos-release-openstack-pike
$ sudo yum install -y openstack-cinder python-pip
```

Then we just need to install the `ember-csi` package:

```
$ sudo pip install ember-csi
```

Now we should install any additional package required by our backend.

For iSCSI backends we'll want to install:

```
$ sudo yum install iscsi-initiator-utils
$ sudo yum install device-mapper-multipath
$ sudo mpathconf --enable --with_multipathd y --user_friendly_names n --find_
↪multipaths y
```

For RBD we'll also need a specific package:

```
$ sudo yum install ceph-common
```

1.4 Configuration

The CSI driver is configured via environmental variables, any value that doesn't have a default is a required value.

| Name | Role | Description | Default | Example | | ————————————— | ———— | —————————————————— ——————————————— | —— | —— | | `CSI_ENDPOINT` | all | IP and port to bind the service | [::]:50051 | 192.168.1.22:50050 | | `CSI_MODE` | all | Role the service should perform: controller, node, all | all | controller | | `X_CSI_STORAGE_NW_IP` | node | IP address in the Node used to connect to the storage | IP resolved from Node's fqdn | 192.168.1.22 | | `X_CSI_NODE_ID` | node | ID used by this node to identify itself to the controller | Node's fqdn | csi_test_node | | `X_CSI_PERSISTENCE_CONFIG` | all | Configuration of the `cinderlib` metadata persistence plugin. | {'storage': 'db', 'connection': 'sqlite:///db.sqlite'} | {'storage': 'db', 'connection': 'mysql+pymysql://root:stackdb@192.168.1.1/cinder?charset=utf8'} | | `X_CSI_EMBER_CONFIG` | all | Global `cinderlib` configuration | {'project_id': 'io.ember-csi', 'user_id': 'io.ember-csi', 'root_helper': 'sudo', 'request_multipath': true, 'plugin_name': 'io.ember-csi'} | {"project_id":"k8s project","user_id":"csi driver","root_helper":"sudo",'plugin_name': 'io.ember-csi.external-ceph'} | | `X_CSI_BACKEND_CONFIG` | controller | Driver configuration | | {"volume_backend_name": "rbd", "volume_driver": "cinder.volume.drivers.rbd.RBDDriver", "rbd_user": "cinder", "rbd_pool": "volumes", "rbd_ceph_conf": "/etc/ceph/ceph.conf", "rbd_keyring_conf": "/etc/ceph/ceph.client.cinder.keyring"} | | `X_CSI_DEFAULT_MOUNT_FS` | node | Default mount filesystem when missing in publish calls | ext4 | btrfs | | `X_CSI_SYSTEM_FILES` | all | All required storage driver-specific files archived in tar, tar.gz or tar.bz2 format| | /path/to/etc-ceph.tar.gz | | `X_CSI_DEBUG_MODE` | all | Debug mode (rpdb, pdb) to use. Disabled by default. | | rpdb | | `X_CSI_ABORT_DUPLICATES` | all | If we want to abort or queue (default) duplicated requests. | false | true |

The only role that has been tested at the moment is the default one, where Controller and Node servicer are executed in the same service (`CSI_MODE=all`), and other modes are expected to have issues at the moment.

The X_CSI_SYSTEM_FILES variable should point to a tar/tar.gz/tar.bz2 file accessible in the Ember CSI driver's filesystem. The contents of the archive will be extracted into '/'. A trusted user such as an operator/administrator with privileged access must create the archive before starting the driver.

e.g.

```
$ tar cvf ceph-files.tar /etc/ceph/ceph.conf /etc/ceph/ceph.client.cinder.keyring
tar: Removing leading `/' from member names
/etc/ceph/ceph.conf
/etc/ceph/ceph.client.cinder.keyring
$ export X_CSI_SYSTEM_FILES=`pwd`/ceph-files.tar
```

1.5 Starting the plugin

Once we have installed `ember-csi` and required dependencies (for the backend and for the connection type) we just have to run the `ember-csi` service with a user that can do passwordless sudo:

```
$ ember-csi
```

1.6 Testing the plugin

There are several examples of running the Ember CSI plugin in the `examples` directory both for a baremetal deployment and a containerized version of the driver.

In all cases we have to run the plugin first before we can test it, and for that we have to check the configuration provided as a test before starting the plugin. By default all examples run the service on port 50051.

1.6.1 Baremetal

For example to test with the LVM driver on our development environment we can just run the following commands from the root of the `ember-csi` project:

Note: The iscsi IP addresses are auto-assigned in the lvm env file. You may change these IP addresses if desired:

```
$ cd tmp
$ sudo dd if=/dev/zero of=ember-volumes bs=1048576 seek=22527 count=1
$ lodevice=`sudo losetup --show -f ./ember-volumes`
$ sudo pvcreate $lodevice
$ sudo vgcreate ember-volumes $lodevice
$ sudo vgscan --cache
$ cd ../examples/baremetal
$ ./run.sh lvm
py27 develop-inst-nodeps: /home/geguileo/code/ember-csi
py27 installed: ...
___ summary ___
  py27: skipped tests
  congratulations :)
Starting Ember CSI v0.0.2 (cinderlib: v0.2.1, cinder: v11.1.2.dev5, CSI spec: v0.
↪2.0)
Supported filesystems are: fat, ext4dev, vfat, ext3, ext2, msdos, ext4, hfsplus,␣
↪cramfs, xfs, ntfs, minix, btrfs
Running backend LVMVolumeDriver v3.0.0
Debugging is OFF
Now serving on [::]:50051...
```

There is also an example of testing a Ceph cluster using a user called "cinder" and the "volumes" pool. For the Ceph/RBD backend, due to a limitation in Cinder, we need to have both the credentials and the configuration in `/etc/ceph` for it to work:

```
$ cd examples/baremetal
$ ./run.sh rbd
Starting Ember CSI v0.0.2 (cinderlib: v0.2.1, cinder: v11.1.2.dev5, CSI spec: v0.
↪2.0)
Supported filesystems are: fat, ext4dev, vfat, ext3, ext2, msdos, ext4, hfsplus,␣
↪cramfs, xfs, ntfs, minix, btrfs
Running backend LVMVolumeDriver v3.0.0
```

```
    Debugging is OFF
    Now serving on [::]:50051...
```

There is also an XtremIO example that only requires the iSCSI connection packages.

1.6.2 Containerized

There is a sample `Dockerfile` included in the project that has been used to create the `akrog/ember-csi` container available in the docker hub.

There are two bash scripts, one for each example, that will run the CSI driver on a container, be aware that the container needs to run as privileged to mount the volumes.

For the RBD example we need to copy our "ceph.conf" and "ceph.client.cinder.keyring" files, assuming we are using the "cinder" user into the example/docker directory replacing the existing ones:

```
    $ cd examples/docker
    $ ./rbd.sh
    Starting Ember CSI v0.0.2 (cinderlib: v0.2.1, cinder: v11.1.0, CSI spec: v0.2.0)
    Supported filesystems are: cramfs, minix, ext3, ext2, ext4, xfs, btrfs
    Running backend LVMVolumeDriver v3.0.0
    Debugging is ON with rpdb
    Now serving on [::]:50051...
```

1.6.3 CSC

Now that we have the service running we can use the CSC tool to run commands simulating the Container Orchestration system.

Due to the recent changes in the CSI spec not all commands are available yet, so you won't be able to test the snapshot commands.

Checking the plugin info:

```
    $ csc identity plugin-info -e tcp://127.0.0.1:50051
    "io.ember-csi"      "0.0.2" "cinder-driver"="RBDDriver"     "cinder-driver-
↪supported"="True"        "cinder-driver-version"="1.2.0" "cinder-version"="11.1.0" ␣
↪     "cinderlib-version"="0.2.1"     "persistence"="DBPersistence"
```

Checking the node id:

```
    $ csc node get-id -e tcp://127.0.0.1:50051
    localhost.localdomain

    $ hostname -f
    localhost.localdomain
```

Checking the current backend capacity:

```
    $ csc controller get-capacity -e tcp://127.0.0.1:50051
    24202140712
```

Creating a volume:

```
    $ csc controller create-volume --cap SINGLE_NODE_WRITER,block --req-bytes␣
↪2147483648 disk -e tcp://127.0.0.1:50051
    "5ee5fd7c-45cd-44cf-af7b-06081f680f2c"  2147483648
```

Listing volumes:

```
    $ csc controller list-volumes -e tcp://127.0.0.1:50051
    "5ee5fd7c-45cd-44cf-af7b-06081f680f2c"  2147483648
```

Store the volume id for all the following calls:

```
    $ vol_id=`csc controller list-volumes -e tcp://127.0.0.1:50051|awk '{ print␣
↪gensub("\"","","g",$1)}'`
```

Attaching the volume to `tmp/mnt/publish` on baremetal as a block device:

```
    $ touch tmp/mnt/{staging,publish}

    $ csc controller publish --cap SINGLE_NODE_WRITER,block --node-id `hostname -f`
↪$vol_id -e tcp://127.0.0.1:50051
    "5ee5fd7c-45cd-44cf-af7b-06081f680f2c"  "connection_info"="{\"connector\": {\
↪"initiator\": \"iqn.1994-05.com.redhat:aa532823bac9\", \"ip\": \"127.0.0.1\", \
↪"platform\": \"x86_64\", \"host\": \"localhost.localdomain\", \"do_local_attach\":␣
↪false, \"os_type\": \"linux2\", \"multipath\": false}, \"conn\": {\"driver_volume_
↪type\": \"rbd\", \"data\": {\"secret_uuid\": null, \"volume_id\": \"5ee5fd7c-45cd-
↪44cf-af7b-06081f680f2c\", \"auth_username\": \"cinder\", \"secret_type\": \"ceph\",␣
↪\"name\": \"volumes/volume-5ee5fd7c-45cd-44cf-af7b-06081f680f2c\", \"discard\":␣
↪true, \"keyring\": \"[client.cinder]\\n\\tkey =␣
↪AQCQPetaof03IxAAoHZJD6kGxiMQfLdn3QzdlQ==\\n\", \"cluster_name\": \"ceph\", \"hosts\
↪": [\"192.168.1.22\"], \"auth_enabled\": true, \"ports\": [\"6789\"]}}}"

    $ csc node stage --pub-info connection_info="irrelevant" --cap SINGLE_NODE_WRITER,
↪block --staging-target-path `realpath tmp/mnt/staging` $vol_id -e tcp://127.0.0.
↪1:50051
    5ee5fd7c-45cd-44cf-af7b-06081f680f2c

    $ csc node publish --cap SINGLE_NODE_WRITER,block --pub-info connection_info=
↪"irrelevant" --staging-target-path `realpath tmp/mnt/staging` --target-path␣
↪`realpath tmp/mnt/publish` $vol_id -e tcp://127.0.0.1:50051
    5ee5fd7c-45cd-44cf-af7b-06081f680f2c
```

Attaching the volume to `tmp/mnt/publish` on container as a block device:

```
    $ touch tmp/mnt/{staging,publish}

    $ csc controller publish --cap SINGLE_NODE_WRITER,block --node-id `hostname -f`
↪$vol_id -e tcp://127.0.0.1:50051
    "5ee5fd7c-45cd-44cf-af7b-06081f680f2c"  "connection_info"="{\"connector\": {\
↪"initiator\": \"iqn.1994-05.com.redhat:aa532823bac9\", \"ip\": \"127.0.0.1\", \
↪"platform\": \"x86_64\", \"host\": \"localhost.localdomain\", \"do_local_attach\":␣
↪false, \"os_type\": \"linux2\", \"multipath\": false}, \"conn\": {\"driver_volume_
↪type\": \"rbd\", \"data\": {\"secret_uuid\": null, \"volume_id\": \"5ee5fd7c-45cd-
↪44cf-af7b-06081f680f2c\", \"auth_username\": \"cinder\", \"secret_type\": \"ceph\",␣
↪\"name\": \"volumes/volume-5ee5fd7c-45cd-44cf-af7b-06081f680f2c\", \"discard\":␣
↪true, \"keyring\": \"[client.cinder]\\n\\tkey =␣
↪AQCQPetaof03IxAAoHZJD6kGxiMQfLdn3QzdlQ==\\n\", \"cluster_name\": \"ceph\", \"hosts\
↪": [\"192.168.1.22\"], \"auth_enabled\": true, \"ports\": [\"6789\"]}}}"
```

```
    $ csc node stage --pub-info connection_info="irrelevant" --cap SINGLE_NODE_WRITER,
↪block --staging-target-path /mnt/staging $vol_id -e tcp://127.0.0.1:50051
    5ee5fd7c-45cd-44cf-af7b-06081f680f2c

    $ csc node publish --cap SINGLE_NODE_WRITER,block --pub-info connection_info=
↪"irrelevant" --staging-target-path /mnt/staging --target-path /mnt/publish $vol_id -
↪e tcp://127.0.0.1:50051
    5ee5fd7c-45cd-44cf-af7b-06081f680f2c
```

Detaching the volume on baremetal:

```
    $ csc node unpublish --target-path `realpath tmp/mnt/publish` $vol_id -e tcp://
↪127.0.0.1:50051
    5ee5fd7c-45cd-44cf-af7b-06081f680f2c

    $ csc node unstage --staging-target-path `realpath tmp/mnt/staging` $vol_id -e␣
↪tcp://127.0.0.1:50051
    5ee5fd7c-45cd-44cf-af7b-06081f680f2c

    $ csc controller unpublish --node-id `hostname -f` $vol_id -e tcp://127.0.0.
↪1:50051
    5ee5fd7c-45cd-44cf-af7b-06081f680f2c
```

Detaching the volume on container:

```
    $ csc node unpublish --target-path /mnt/publish $vol_id -e tcp://127.0.0.1:50051
    5ee5fd7c-45cd-44cf-af7b-06081f680f2c

    $ csc node unstage --staging-target-path /tmp/mnt/staging $vol_id -e tcp://127.0.
↪0.1:50051
    5ee5fd7c-45cd-44cf-af7b-06081f680f2c

    $ csc controller unpublish --node-id `hostname -f` $vol_id -e tcp://127.0.0.
↪1:50051
    5ee5fd7c-45cd-44cf-af7b-06081f680f2c
```

Deleting the volume:

```
    $ csc controller delete-volume $vol_id -e tcp://127.0.0.1:50051
```

If we want to use the mount interface instead of the block one, we can also do it making sure we create directories instead of files and replacing the `block` word with `mount,ext4` if we want an `ext4` filesystem.

For example these would be the commands for the baremetal attach:

```
    $ mkdir tmp/mnt/{staging_dir,publish_dir}

    $ csc controller publish --cap SINGLE_NODE_WRITER,mount,ext4 --node-id `hostname -
↪f` $vol_id -e tcp://127.0.0.1:50051

    $ csc node stage --pub-info connection_info="irrelevant" --cap SINGLE_NODE_WRITER,
↪mount,ext4 --staging-target-path `realpath tmp/mnt/staging_dir` $vol_id -e tcp://
↪127.0.0.1:50051
    5ee5fd7c-45cd-44cf-af7b-06081f680f2c

    $ csc node publish --pub-info connection_info="irrelevant" --cap SINGLE_NODE_
↪WRITER,mount,ext4 -staging-target-path `realpath tmp/mnt/staging_dir` --target-path␣
↪`realpath tmp/mnt/publish_dir` $vol_id -e tcp://127.0.0.1:50051
```

```
    5ee5fd7c-45cd-44cf-af7b-06081f680f2c
```

1.7 Capable operational modes

The CSI spec defines a set of `AccessModes` that CSI drivers can support, such as single writer, single reader, multiple writers, single writer and multiple readers.

This CSI driver currently only supports `SINGLE_MODE_WRITER`, although it will also succeed with the `SINGLE_MODE_READER_ONLY` mode and mount it as read/write.

1.8 Debugging

The first tool for debugging is the log that displays detailed information on the driver code used by *ember-CSI*. We can enable INFO or DEBUG logs using the `X_CSI_EMBER_CONFIG` environmental variable.

To enable logs, defaulting to INFO level, we must set the `disable_logs` key to `false`. If we want them at DEBUG levels, we also need to set `debug` to `true`.

For baremetal, enablig DEBUG log levels can be done like this:

```
    export X_CSI_EMBER_CONFIG={"project_id":"io.ember-csi","user_id":"io.ember-csi",
↪"root_helper":"sudo","plugin_name": "io.ember-csi","disable_logs":false,"debug
↪":true}
```

For containers we can just add the environmental variable to a file and import into our run using `--env-file` or adding it to our command line with `-e`.

In both cases it should not have the `export` command:

```
    X_CSI_EMBER_CONFIG={"project_id":"io.ember-csi","user_id":"io.ember-csi","root_
↪helper":"sudo","plugin_name": "io.ember-csi","disable_logs":false,"debug":true}
```

Besides this basic debugging level, the Ember CSI plugin also supports live debugging when run in the baremetal and when running as a container.

There are two mechanisms that can be used to debug the driver: with `pdb`, and with `rpdb`.

The difference between them is that `pdb` works with stdin and stdout, whereas `rpdb` opens port 4444 to accept remote connections for debugging.

Debugging the Ember CSI plugin requires enabling debugging on the plugin before starting it, and then one it is running we have to turn it on.

Enabling debugging is done using the `X_CSI_DEBUG_MODE` environmental variable. Setting it to `pdb` or `rpdb` will enable debugging. The plugin has this feature disabled by default, but our *latest* and *master* containers have it enabled by default with `rpdb`.

Once we have the plugin running with the debugging enable (we can see it in the start message) we can turn it on and off using the `SIGUSR1` signal, and the service will output the change with a *Debugging is ON* or *Debugging is OFF* message.

After turning it *ON* the plugin will stop for debugging on the next GRPC request. Going into interactive mode if using `pdb` or opening port 4444 if using `rpdb`. When using `rpdb` we'll see the following message on the plugin: *pdb is running on 127.0.0.1:4444*

Sending the signal to toggle ON/OFF the debugging is quite easy. For baremetal we can do:

```
$ pkill -USR1 ember-csi
```

And for the container (assuming its named `ember-csi` like in the examples) we can do:

```
$ docker kill -sUSR1 ember-csi
```

If we are using `rpdb` then we'll have to connect to the port:

```
$ nc 127.0.0.1 4444
```

1.9 Troubleshooting

1.9.1 CSC commands timeout

If you have a slow backend or a slow data network connection, and you are creating mount volumes, then you may run into "context deadline exceeded" errors when running the node staging command on the volume.

This is just a 60 seconds timeout, and we can easily fix this by increasing allowed timeout for the command to complete. For example to 5 minutes with `-t5m` or to 1 hour if we are manually debugging things on the server side with `-t1h`.

1.9.2 Staging fails in container using iSCSI

When I try to stage a volume using a containerized *Node* I see the error "ERROR root VolumeDeviceNotFound: Volume device not found at .".

Turning the DEBUG log levels shows me login errors:

```
    2018-07-03 11:14:57.258 1 WARNING os_brick.initiator.connectors.iscsi [req-
↪0e77bf32-a29b-40d1-b359-9e115435a94a io.ember-csi io.ember-csi - - -] Failed to␣
↪connect to iSCSI portal 192.168.1.1:3260.
    2018-07-03 11:14:57.259 1 WARNING os_brick.initiator.connectors.iscsi [req-
↪0e77bf32-a29b-40d1-b359-9e115435a94a io.ember-csi io.ember-csi - - -] Failed to␣
↪login iSCSI target iqn.2008-05.com.something:smt00153500071-514f0c50023f6c01 on␣
↪portal 192.168.1.1:3260 (exit code 12).: ProcessExecutionError: Unexpected error␣
↪while running command.
```

And looking into the host's journal (where the `iscsid` daemon is running) I can see `Kmod` errors:

```
Jul 03 13:15:02 think iscsid[9509]: Could not insert module . Kmod error -2
```

This seems to be cause by some kind of incompatibility between the host and the container's iSCSI modules. We currently don't have a solution other than using a CentOS 7 host system.

1.10 Support

For any questions or concerns please file an issue with the ember-csi project or ping me on IRC (my handle is geguileo and I hang on the #openstack-cinder channel in Freenode).

1.11 TODO

There are many things that need to be done in this POC driver, and here's a non exhaustive list:

- Support for NFS volumes
- Support for Kubernetes CRDs as the persistence storage
- Unit tests
- Functional tests
- Improve received parameters checking
- Make driver more resilient
- Test driver in Kubernetes
- Review some of the returned error codes
- Support volume attributes via volume types
- Look into multi-attaching
- Support read-only mode
- Report capacity based on over provisioning values
- Configure the private data location

CHAPTER 2

Installation

2.1 Stable release

To install Ember CSI plugin, run this command in your terminal:

```
$ pip install ember_csi
```

This is the preferred method to install Ember CSI plugin, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for Ember CSI plugin can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/akrog/ember_csi
```

Or download the tarball:

```
$ curl  -OL https://github.com/akrog/ember_csi/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use Ember CSI plugin in a project:

```
import ember_csi
```

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/akrog/ember-csi/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

4.1.4 Write Documentation

Ember CSI plugin could always use more documentation, whether as part of the official Ember CSI plugin docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/akrog/ember-csi/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *ember-csi* for local development.

1. Fork the *ember-csi* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/ember-csi.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv ember-csi
$ cd ember-csi/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 ember_csi tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/akrog/ember-csi/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_ember_csi
```

CHAPTER 5

Credits

5.1 Development Lead

- Gorka Eguileor <gorka@eguileor.com>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

6.1 0.0.x (2018-XX-XX)

6.1.1 Features

- Support for mount filesystems
- Check presence of REQUIRED fields
- Log requests and responses
- Support defining default mount filesystem
- Show supported CSI spec version on start
- Support live debugging of running driver
- Add CRD metadata persistence plugin for Kubernetes
- Support queuing duplicated requests

6.1.2 Bugs

- Fix issues receiving duplicated RPC calls
- Fix UUID warning
- Check staging and publishing targets
- Exit on binding error

6.2 0.0.2 (2018-06-19)

- Use cinderlib v0.2.1 instead of github branch

6.3 0.0.1 (2018-05-18)

- First release on PyPI.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search