Bynar Documentation

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Date | Reason for Change | Version |
| Michelle Zhong | 10/8/2019 | Outline the Document | 0.1 |
| Michelle Zhong | 10/9/2019 | Outline the Document Modules, fill in the API section, Config File section, start filling out the Backend Section | 0.2 |
| Michelle Zhong | 10/10/2019 | Reorganize Headers in API section | 0.3 |
|  |  |  |  |

# Table of Contents

[Revision History 2](#_Toc21589836)

[Table of Contents 3](#_Toc21589837)

[API 5](#_Toc21589838)

[Introduction 5](#_Toc21589839)

[Messages 5](#_Toc21589840)

[Enums 5](#_Toc21589841)

[Structs 6](#_Toc21589842)

[Configuration Files 8](#_Toc21589843)

[Introduction 8](#_Toc21589844)

[List of Config Files 8](#_Toc21589845)

[Bynar JSON 8](#_Toc21589846)

[Ceph JSON 9](#_Toc21589847)

[Disk-Manager JSON 9](#_Toc21589848)

[Backend 9](#_Toc21589849)

[Introduction 9](#_Toc21589850)

[Backend Module 9](#_Toc21589851)

[Enums 9](#_Toc21589852)

[Interface 9](#_Toc21589853)

[Ceph 10](#_Toc21589854)

[Structs 10](#_Toc21589855)

[Helper Functions 14](#_Toc21589856)

[Database Schema 15](#_Toc21589857)

[Introduction 15](#_Toc21589858)

[Postgres 15](#_Toc21589859)

[Database Logging 15](#_Toc21589860)

[Introduction 15](#_Toc21589861)

[Logging 15](#_Toc21589862)

[Helper Functions 15](#_Toc21589863)

[Introduction 15](#_Toc21589864)

[Error Module 15](#_Toc21589865)

[Host Information 15](#_Toc21589866)

[Helper Module 15](#_Toc21589867)

[Client 15](#_Toc21589868)

[Introduction 15](#_Toc21589869)

[Client Interface 15](#_Toc21589870)

[Support Tickets 15](#_Toc21589871)

[Introduction 15](#_Toc21589872)

[JIRA Support 15](#_Toc21589873)

[Disk Manager 15](#_Toc21589874)

[Introduction 15](#_Toc21589875)

[Disk Manager 15](#_Toc21589876)

[Disk Testing 15](#_Toc21589877)

[Introduction 15](#_Toc21589878)

[State Machine 15](#_Toc21589879)

[Hardware Testing 15](#_Toc21589880)

[Introduction 15](#_Toc21589881)

[Hardware Tests 15](#_Toc21589882)

[Bynar 15](#_Toc21589883)

[Introduction 15](#_Toc21589884)

[Main Process 15](#_Toc21589885)

# API

## Introduction

This package uses Protobuf version 2 to create Messages that can be sent over the network. Protobuf is a fast and small protocol for serializing structs (or structured data). Serialized messages can be sent between Sockets, unpackaged, and read easily and quickly. The protobuf package automatically generates the rust code needed to create, modify, and destroy Messages as well as their attributes.

## Messages

### Enums

#### DiskType

The type of disk or device

##### Enum Values

|  |  |
| --- | --- |
| Name | Description |
| LOOPBACK | Special loopback device |
| LVM | Logical Volume Device |
| MDRAID | Linux software RAID |
| NVME | Non-Volatile Memory Express, a logical device interface |
| RAM | Ramdisk |
| ROTATIONAL | Regular rotational device |
| SOLID\_STATE | SSD |
| VIRTUAL | Virtual Disk |
| UNKNOWN | Unknown disk |

#### ResultType

A result value

##### Enum Values

|  |  |
| --- | --- |
| Name | Description |
| OK | ok |
| ERR | There is an error |

#### Op

An operation on a disk

##### Enum Values

|  |  |
| --- | --- |
| Name | Description |
| Add | Generic Add Disk command, returns an OpResult |
| AddPartition | Add a Partition Command, returns an OpResult |
| List | List the Disks, returns a list of Disks |
| Remove | Remove a Disk, returns an OpResult |
| SafeToRemove | Checks if a Disk is safe to remove, returns a bool |
| GetCreatedTickets | list created tickets, returns a list of created tickets |

#### DatacenterOp

Datacenter API’s, these all require server\_id as a parameter for the operation

##### Enum Values

|  |  |
| --- | --- |
| Name | Description |
| GetDc | Get ? Returns an OpStringResult |
| GetRack | Get the rack of a server, returns an OpStringResult |
| GetRow | Get the row of a server, returns an OpStringResult |
| GetElevation | Get the elevation of a server, returns an OpStringResult |

### Structs

#### Osd

A Ceph OSD object descriptor

##### Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| fsid | Option<String> | OSD File System ID, if one exists |
| id | u64 | OSD ID number |
| block\_device | String | Block Device of the OSD |
| journal | Option<String> | Name of the Journal if the OSD has one set |
| active | bool | Whether or not an OSD is active or a spare |
| used\_space | u64 | How much space in the OSD is currently used |
| total\_space | u64 | Total space in the OSD |

#### Partition

A single partition descriptor

##### Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| uuid | String | The id of the partition |
| first\_lba | u64 | The first logical block address of the partition |
| last\_lba | u64 | The last logical block address of the partition |
| flags | u64 | Flags associated with the partition |
| name | String | The name of the partition |

#### PartitionInfo

A list of Partitions

##### Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| partition | Vec<Partition> | List of partitions |

#### Disk

A disk object descriptor

##### Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| type | DiskType | The type of disk |
| dev\_path | String | ?? Device path? |
| partitions | PartitionInfo | Disk partitions |
| serial\_number | Option<String> | Disk serial number |

#### OpResult

A result of an Op message

##### Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| result | ResultType | Whether the result is ok or Error |
| error\_msg | Option<String> | Error message if there is an error |

#### OpBoolResult

A boolean result of an Op message

##### Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| result | ResultType | Whether Ok or Error |
| value | Option<bool> | A value is set if OK |
| error\_msg | Option<String> | Error message is set if there is an Error |

#### OpStringResult

A String result of an Op message

##### Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| result | ResultType | Whether Ok or Error |
| value | Option<String> | A value is set if OK |
| error\_msg | Option<String> | Error message is set if there is an Error |

#### JiraInfo

A Jira Ticket information descriptor

##### Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| ticket\_id | String | Ticket number |
| server\_name | String | Name of the JIRA server |

#### OpJiraTicketsResult

A Jira ticket result

##### Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| result | ResultType | Whether Ok or Error |
| value | Option<String> | A value is set if OK |
| error\_msg | Option<String> | Error message is set if there is an Error |

#### DatacenterOperation

A Datacenter operation message

##### Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| Op\_type | DatacenterOp | The type of operation to be performed |
| server\_id | String | The ID of the server to be operated on |

#### Operation

A service operation that can be performed

##### Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| Op\_type | Op | The operation type |
| disk | Option<String> | The disk name, used for an Add or Remove |
| simulate | Option<bool> | Whether the operation is a simulation, used for Add, Remove, and SafeToRemove |
| partition\_start | Option<u64> | Optional field for AddPartition, start of a partition |
| partition\_end | Option<u64> | Optional field for AddPartition, end of a partition |
| partition\_name | Option<String> | Optional field for AddPartition, partition name |
| osd\_id | Option<u64> | Optional Ceph related field, the id of an OSD |
| replica\_set | Vector<String> | Host:/dev/disk strings list for gluster replica sets |

# Configuration Files

## Introduction

Bynar uses a set of configuration files to configure different settings. Bynar uses JSON as the format for its configuration files, as JSON files are easily parsed, serialized, and deserialized using the Rust serde and serde-json crates.

## List of Config Files

### Bynar JSON

This config file, bynar.json, is used to configure several different settings, including a Slack webhook, JIRA support, Redfish access, Vault password access, and Postgres database access

|  |  |  |
| --- | --- | --- |
| Name | Description | Example Value |
| proxy | Proxy web server? | “https://my.proxy” |
| manager\_host | The host ip of the bynar disk manager | “localhost” |
| manager\_port | The port of the Bynar disk manager | 5555 |
| slack\_webhook | Slack webhook to access Slack API | "<https://hooks.slack.com/services/ID>" |
| slack\_channel | Slack channel to post messages to | “#my-channel" |
| slack\_botname | Name of the Bot to post messages under | "my-bot" |
| jira\_user | JIRA username to create tickets under | “test\_user” |
| jira\_password | JIRA password | “user\_pass” |
| jira\_host | JIRA host to create tickets under | “https://tickets.jira.com” |
| jira\_issue\_type | JIRA issue type name to create tickets of | “3” |
| jira\_priority | JIRA priority value of tickets created | “4” |
| jira\_project\_id | JIRA project id to create tickets under | “MyProject” |
| jira\_ticket\_assignee | User created JIRA tickets are assigned to | “assignee\_username” |
| redfish\_ip | IP address of a Redfish instance | “localhost” |
| redfish\_username | Username to access Redfish instance | “redfish\_user” |
| redfish\_password | Password to access Redfish | “redfish\_pass” |
| redfish\_port | Port of the Redfish instance | 4443 |
| vault\_endpoint | Hashicorp vault endpoint | “https://my\_vault.com” |
| vault\_token | Hashicorp vault token to access the vault | “token\_234464562” |
| database | List of Database parameters |  |
| database:username | Username to access database with | “postgres” |
| database:password | Password to access database with | “” |
| database:port | Port of the database | 5432 |
| database:dbname | Name of the database | “bynar” |
| database:endpoint | Database endpoint | “some.endpoint” |

### Ceph JSON

This config file, ceph.json, is used to tell Bynar where the ceph.conf file is, what user to use when running Ceph commands, and what journal devices are known?

|  |  |  |
| --- | --- | --- |
| Name | Description | Example Value |
| config\_file | The path to the ceph.conf file | “/etc/ceph/ceph.conf” |
| user\_id | User to use when running Ceph commands | “admin” |
| journal\_devices | Journal device list |  |
| journal\_devices:device | Path of the device | “/dev/sda” |
| journal\_devices:partition\_id | Partition ID number | 1 |

### Disk-Manager JSON

This config file, disk-manager.json is used to tell Bynar what the backend storage system is

|  |  |  |
| --- | --- | --- |
| Name | Description | Example Value |
| backend | The backend type of the storage system | "ceph” |

# Backend

## Introduction

Different distributed storage clusters have different ways of adding and removing disks, the backend module seeks to create an interface to the different backends

## Backend Module

A Generic Module for interfacing with different storage backends

### Enums

#### BackendType

##### Enum Values

|  |  |
| --- | --- |
| Name | Description |
| Ceph | Ceph is the backend type |
| Gluster | GlusterFS is the backend type |

##### Trait Implementations

###### FromStr

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Inputs | Description | Outputs |
| from\_str | s: &str | Converts a string to a BackendType. Return Ok(BackendType) if successful or an Error if the string is not a BackendType | BynarResult<BackendType> |

###### Clone, Debug, Deserialize

### Interface

#### Backend

##### Trait Function Definition

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Inputs | Description | Outputs |
| add\_disk | device: &Path  id: Option<u64>  simulate: bool | Add a disk at path *device*, *id* an optional OSD id for Ceph clusters to ensure the OSD is set to that id, if *simulate* is passed no action is taken. Returns Ok(()) if successful or an Error if one occurs | BynarResult<()> |
| remove\_disk | device: &Path  simulate: bool | Remove a disk at path *device* from a cluster. If *simulate* is passed no action is taken. Returns Ok(()) if successful or an Error if one occurs | BynarResult<()> |
| safe\_to\_remove | device: &Path  simulate: bool | Check if safe to remove a disk from a cluster at path *device*. If *simulate* passed then return true. Returns Ok(true) if successful and safe, Ok(false) if successful and not safe to remove, or an Error if one occurs | BynarResult<bool> |

##### Public Functions

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Inputs | Description | Outputs |
| load\_backend | backend\_type: &BackendType  config\_dir: Option<&Path> | Given a BackendType, *backend\_type,* and a config file directory from *config\_dir*, return Ok(Backend) if successful or Error if one occurs. | BynarResult<Box<dyn Backend>> |

## Ceph

The Ceph backend implementation

### Structs

#### CephBackend

This is a public struct object defining a Ceph cluster

##### Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| cluster\_handle | Rados | A handle to the ceph librados |
| config | CephConfig | Handle for the Ceph Configuration File |
| version | CephVersion | The Ceph Version |

##### Implementation

|  |
| --- |
| Function Definition |
| new(config\_dir: Option<&Path>) -> BynarResult<()>  DESCRIPTION: Create a new CephBackend  PARAMETERS: config\_dir – the directory of the ceph.json file or NONE if in the .config directory of the HOME directory  RETURNS: Ok(CephBackend) on success, Error otherwise  IMPLEMENTATION: Get the ceph.json file from the config\_dir parameter. If successful, create the CephConfig object from the ceph.json file. Using the CephConfig object, connect to the specified Ceph instance using the specified user id to get the librados handle. Using the Rados handle, get the Ceph version string and convert it into a CephVersion object. If all steps are successful return a new CephBackend object with the CephConfig, Rados handle, and CephVersion. |
| add\_bluestore\_osd(&self, dev\_path:&Path, id:Option<u64>, simulate: bool) -> BynarResult<()>  DESCRIPTION: Add a bluestore OSD to the Ceph Cluster  PARAMETERS: dev\_path – the device path of the OSD  id– the OSD id of the OSD to add  simulate – if passed skip execution of the function  RETURNS: Ok(()) on success, Error otherwise  IMPLEMENTATION: Find a journal device that has enough free space? Create a new osd and get its osd\_id (if id is not NONE then the new osd id should match id. Create an osd\_fsid, and use it, the osd id, the device path, and the journal to create an lvm. Create a mount point path for the drive if necessary. Write the osd fsid to a file. Resolve the created lvm name to a true device path and chown it so ceph can use it. Symlink the lvm device name to the mount point’s /block, and if a journal device with enough space was found, symlink the journal to the mount point’s /block.wal and change the permissions so ceph can use it. Write activate monmap out by getting the map, and creating a file activate.monmap. Lookup the ceph user id and change all the permissions on the created files so ceph can use them. Create a ceph authorization entry, get the keyring created and save it. Format the osd with the osd filesystem. Use the ceph bluestore tool, and add the osd to the crush. Enable the osd, and then initialize the osd. If all steps are successful return (), else it error’d out somewhere. |
| add\_filestore\_osd(&self, dev\_path:&Path, id:Option<u64>, simulate:bool) -> BynarResult<()>  DESCRIPTION: Add a new /dev/ path as an osd, with xfs, for Jewel or earlier  PARAMETERS: dev\_path – the device path of the OSD  id– the OSD id of the OSD to add  simulate – if passed skip execution of the function  RETURNS: Ok(()) on success, Error otherwise  IMPLEMENTATION: Format the drive with the Xfs filesystem. Prove the drive by getting the device info and checking if it has a filesystem id. Create a new osd and get its id, which should be the same as the input id if one was input. Create the mount point path and mount the drive. Select a journal with enough space (if there is one, can be None). Format the osd with the osd filesystem. Create a ceph authorization entry, get the authorization key and save the keyring. Add the osd to the crush, add the osd to the fstab, then init the osd. If all steps are successful return (), else it error’d out somewhere. |
| change\_permissions(&self, paths: &[&Path], perms: &Passwd) -> BynarResult<()>  DESCRIPTION: change permissions of many files at once  PARAMETERS: paths – the paths of the files to change the permissions of  perms – the group and owner permissions to change the file permissions to  RETURNS: Ok(()) on success, Error otherwise  IMPLEMENTATION: loop through the paths and chown each path to the input permission values. If all steps are successful return (), else it error’d out somewhere. |
| create\_lvm(&self, osd\_fsid: &uuid::Uuid, new\_osd\_id: u64, dev\_path: &Path, journal\_device: Option<&JournalDevice>) -> BynarResult<(PathBuf, u64)>  DESCRIPTION: Create the lvm device and return the path and size of it  PARAMETERS: osd\_fsid – the osd filesystem id  new\_osd\_id – the id of the osd  dev\_path – the path to the device of the osd  journal\_device – an optional journal device ? Dunno what it’s used for...  RETURNS: Ok(PathToLvm,Size) on success, Error otherwise  IMPLEMENTATION: probe the device for its udev info. create a volume group name, and logical volume name, and use them to create the logical volume device name. Initialize a new LVM, and scan it. create the volume group on the LVM, then add the device path to the volume group by extending it and writing. create a linear logical volume in the volume group, create its tags. If all steps are successful return the path to the lvm device name and the volume group size, else it error’d out somewhere. |
| create\_lvm\_tags(&self, lv: &LogicalVolume<\_,\_>, lv\_dev\_name: &Path, osd\_fsid: &uuid::Uuid, new\_osd\_id:u64, info:&block\_utils::Device, journal\_device:Option<&JournalDevice)->BynarResult<()>  DESCRIPTION: Add the lvm tags that ceph requires to identify the osd  PARAMETERS: lv – the logical volume  lv\_dev\_name – the path to the logical volume device  osd\_fsid – the osd filesystem id  new\_osd\_id – the id of the osd  info – the device info  journal\_device – an optional journal device ? Dunno what it’s used for...  RETURNS: Ok(()) on success, Error otherwise  IMPLEMENTATION: create the lvm tags. If there is a journal device input, add a tag for the wal\_device and add the wal\_uuid. Once all tags are created add them to the logical volume. If all steps are successful return (), else it error’d out somewhere. |
| remove\_bluestore\_osd(&self, dev\_path:&Path, simulate:bool) -> BynarResult<()>  DESCRIPTION: Remove a bluestore OSD to the Ceph Cluster  PARAMETERS: dev\_path – the device path of the OSD  simulate – if passed skip execution of the function  RETURNS: Ok(()) on success, Error otherwise  IMPLEMENTATION: Initialize an lvm and scan it for volume groups and LVM metadata. Get the volume group that the device is associated with, if it cannot find the volume group, check if it is a filestore and if so fall back. otherwise, open the volume group and list all logical volumes in the volume group. List the tags to get the osd id and osd fsid. Set the osd as out, remove it from the crush, delete the authorization key, stop the osd, and remove it. Then, wipe the disk. remove all the logical volumes associated with the volume group, remove the volume group, and remove the physical volume and erase the physical volume. Then disable the osd. If all steps are successful return (), else it error’d out somewhere. |
| remove\_filestore\_osd(&self, dev\_path: &Path, simulate: bool) -> BynarResult<()>  DESCRIPTION: Remove a bluestore OSD to the Ceph Cluster  PARAMETERS: dev\_path – the device path of the OSD  simulate – if passed skip execution of the function  RETURNS: Ok(()) on success, Error otherwise  IMPLEMENTATION: get the mountpoint of the dev path and get the osd\_id. Set the osd as out, remove it from the crush, delete the osd auth key, and remove the osd. Then, wipe the disk by erasing the block device. If all steps are successful return (), else it error’d out somewhere. |
| resolve\_lvm\_device(&self, lv\_dev\_name: &Path) -> BynarResult<PathBuf>  DESCRIPTION: Resolve the lvm device name to an absolute path, since the lvm device name is a symlink, so it needs to be resolved to an absolute path to do anything with it.  PARAMETERS: lv\_dev\_name – the device name of the lvm  RETURNS: Ok(Lvm Absolute Path) on success, Error otherwise  IMPLEMENTATION: read the symlink. If it is a relative path, get its parent and the relative path to its parent, and canonicalize it, which returns the canonical, absolute form of a path with all intermediate components normalized and symbolic links resolved. If all steps are successful return the absolute path, else it error’d out somewhere. |
| select\_journal(&self) -> BynarResult<Option<JournalDevice>>  DESCRIPTION: Find a journal device that has enough free space if there is one  PARAMETERS:  RETURNS: Ok(Some(JournalDevice)) or Ok(None) on success, Error otherwise  IMPLEMENTATION: get the journal size from the Rados config. Convert it from MB to bytes. Get the journal devices from the ceph.json and sort them by the number of partitions. Iterate over the journal devices and remove the devices that are too small, and take the first journal with enough space. If all steps are successful, return Ok(Some(JournalWithEnoughSpace)) or Ok(None) if there are no journals with enough space, else it error’d out somewhere. |

##### Trait Implementation

###### Backend

|  |
| --- |
| Trait Function Definition |
| add\_disk(&self, device: &Path, id: Option<u64>, simulate: bool) -> BynarResult<()>  DESCRIPTION: Add a disk to the Cluster  PARAMETERS: device – the device path of the disk to add  id – an optional id to give the osd  simulate – if passed, skip the evaluation of this function  RETURNS: Ok(()) on success, Error otherwise  IMPLEMENTATION: If the Ceph Version is >= Luminous, then run add\_bluestore\_osd. Otherwise, run add\_filestore\_osd. If all steps are successful return (), else it error’d out somewhere. |
| remove\_disk(&self, device:&Path, simulate: bool) -> BynarResult<()>  DESCRIPTION: remove a disk from the Cluster  PARAMETERS: device – the device path of the disk to add  simulate – if passed skip execution of the function  RETURNS: Ok(()) on success, Error otherwise  IMPLEMENTATION: check if the Ceph Version is >= Luminous. If so, run remove\_bluestore\_osd. Otherwise, run remove\_filestore\_osd. If all steps are successful return (), else it error’d out somewhere. |
| safe\_to\_remove(&self, \_device:&Path, \_simulate:bool) -> BynarResult<()>  DESCRIPTION: check if a disk is safe to remove from the cluster  PARAMETERS: device – the unused device path of the disk to remove  simulate – if passed skip execution of the function  RETURNS: Ok(True) or Ok(False)on success, Error otherwise  IMPLEMENTATION: Create a DiagMap and run an exhaustive check. If all steps are successful, then return true if the Status is Safe, return false if the Status is NonSafe or Unknown, otherwise the function error’d out somewhere. |

#### JournalDevice

A Journal Device

##### Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| device | PathBuf | The device name? Device path??? |
| partition\_id | Option<u32> | The id of the partition |
| partition\_uuid | Option<uuid::Uuid> | The user? Unique? id of the partition |
| num\_partitions | Option<usize> | The number of partitions in the Journal Device |

##### Implementation

|  |
| --- |
| Function Definition |
| update\_num\_partitions(&mut self) -> BynarResult<()>  DESCRIPTION: Discover the number of partitions on the device and update the num\_partitions field  PARAMETERS:  RETURNS: Ok(CephBackend) on success, Error otherwise  IMPLEMENTATION: use GPT to get the number of partitions from the partition table, and update the num\_partitions field. If all steps are successful, then return (), else the function error’d out somewhere |

##### Trait Implementation

###### Display

|  |
| --- |
| Trait Function Definition |
| fmt(&self, f: &mut fmt::Formatter) -> fmt::Result  DESCRIPTION: format the journal device for printing as a string/displaying as a string  PARAMETERS: f: some formatter  RETURNS: Ok(()) on success, fmt::Result error type otherwise  IMPLEMENTATION: if there is a partition\_id, display the device and the id, otherwise just display the device. |

###### Clone, Debug, Deserialize, PartialEq

#### CephConfig

The ceph configuration object descriptor

##### Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| config\_file | String | The location of the ceph.conf file |
| user\_id | String | The cephx user to connect to the Ceph service with |
| journal\_devices | Option<Vec<JournalDevice>> | The /dev/xxx devices to use for journal partitions. Bynar will create new partitions on these devices as needed if no journal\_partition\_id is given. |

##### Trait Implementation

###### Deserialize, Debug

### Helper Functions

|  |
| --- |
| Helper Function Definition |
| choose\_ceph\_config(config\_dir: Option<&Path>) -> BynarResult<PathBuf>  DESCRIPTION: get the path of the ceph.json file.  PARAMETERS: config\_dir – an optional path to the configuration directory  RETURNS: Ok(ceph.json path) on success, Error otherwise  IMPLEMENTATION: check if a config\_dir was provided. If so, check the directory for a ceph.json file. If a config\_dir is not provided, check in the Home directory under the .config directory for the ceph.json file. If the function was successful return Ok(ceph.json path) else the function error’d somewhere. |
| Get\_osd\_id\_from\_path |
|  |
|  |
|  |
|  |
|  |

# Database Schema

## Introduction

## Postgres

# Database Logging

## Introduction

### Logging

# Helper Functions

## Introduction

### Error Module

### Host Information

### Helper Module

# Client

## Introduction

### Client Interface

# Support Tickets

## Introduction

### JIRA Support

# Disk Manager

## Introduction

### Disk Manager

# Disk Testing

## Introduction

### State Machine

# Hardware Testing

## Introduction

### Hardware Tests

# Bynar

## Introduction

### Main Process