

Ceph Drill Automation

Table of contents

Ceph Drill Automation	2
Cli Command Reference	5
API Reference	8
Pet	9
Find pet inventories by status	27
Place order for pet	28
Find purchase order by ID	34
Delete purchase order by ID	36
Create user	37
Create list of users with given input array	40
Log user into the system	44
Log current user out	46
Find user	47
Update user	49
Delete user	52
Changes	53

Ceph Drill Automation

Introduction

Automate the process of verifying data integrity during disaster recovery drills in a Ceph storage cluster by using checksums and hashing. This ensures the integrity of RBD images transferred from the primary data center (DC) to the disaster recovery (DR) site without the need to launch a version of the image.

Installation

Note: In this guide we describe using installation using .tar file. You can also clone it from github (<https://github.com/substantialcattle5/>) Install the file using

```
wget localhost.com
```

Then unzip the file

```
tar -xvzf <reqd_release>.tar.gz
```

Then provide enough permissions tfo execute

```
chmod a+x cda.sh
```

Basic workflow

Once installed, you can invoke CLI commands directly from your OS command line through the `cephautomationdrill` executable. See the available commands by entering the following:

```
cda --help
```

Get help on an individual command using the following construct. Substitute any command, like `verify`, `check`, etc., where you see `generate` in the example below to get detailed help on that command:

```
cda verify --help
```

To run a basic drill check, go to the ceph center and run the following command , replacing the names with your own:

```
cda check <pool-name> <image-name>
```

The results would get stored in `checksum` directory,the structure is as follows :

```
checksums/
├── export
│   ├── <poolname>_<imagename>.log
│   ├── <poolname>_<imagename>.log
│   └── <poolname>_<imagename>.log
├── export-diff
│   └── <poolname>_<imagename>.log
└── verify
    └── <poolname>_<imagename>.log
```

CLI command syntax

All `cda` commands follow the same format:

```
cda commandOrAlias requiredArg [optionalArg] [options]
```

For example:

```
cda check pool1 vm_state1
```

Command Overview

Run `cephautomationdrill <command> --help` for any of the following commands to see command-specific options See usage for detailed descriptions for each command.

Command	Aliases	Description
check	c	This command generates a hash of the given snapshot and saves it in a log file.
completion		Generates and/or modifies files based on a schematic.
verify		Generate the autocompletion script for the specified shell
help	h	Help about any command.

Cli Command Reference

check

Automates the Image Drill process.

```
cda export <pool_name> <vm_snapshot_name>
cda export-diff <pool_name> <vm_snapshot_name>
```

Description

Saves a timestamp and checksum to the inbuilt log file of an `ceph_image` which can be later used for validation checks.

1. `export/export-diff` signifies which `rbid` command to perform.
2. `<pool_name>` signifies the pool name where the snapshot is stored.
3. `<vm_snapshot_name>` the snapshot which reqds the checksum operation

verify

Verifies the integrity of a snapshot by comparing its current checksum with the saved checksum from DC and DR log files.

```
cda verify <export/export-diff> <dc_log_file> <dr_log_file> <pool_name>
<image_name>
```

Description

Verifies the integrity of the given snapshot by comparing the current checksum with the ones stored in the DC and DR log files. This ensures the snapshot has not been altered since the last recorded checksum.

1. `<export/export-diff>` specifies whether to use export or export-diff.
2. `<dc_log_file>` specifies the path to the DC log file.

3. `<dr_log_file>` specifies the path to the DR log file.
4. `<pool_name>` signifies the pool name where the snapshot is stored.
5. `<image_name>` the snapshot to verify.

Example :

```
cda verify export /path/to/dc/logfile /path/to/dr/logfile mypool mysnapshot
```

completion

Generates the autocompletion script for the specified shell.

```
cda completion [shell]
```

Description

Generates a script that enables autocompletion for the cda commands in the specified shell.

1. `[shell]` is the optional argument to specify the shell type (e.g., bash, zsh). If not specified, it will generate for the default shell. Example

```
cda completion zsh
cda completion bash
```

help

Displays help information about cda commands.

```
cda help <command>
```

Description


Provides detailed help and usage information for any specific cda command.

1. `<command>` is the command for which help is required. Example

cda help check
cda help verify

API Reference

This is a sample Pet Store Server based on the OpenAPI 3.0 specification.

 A very important note about this API.

Pet

Update an existing pet

PUT /pet

Update an existing pet by Id

Request parameters

Body

application/json Pet **required**

Child attributes

id integer int64

name string **required**

category Category

Child attributes

id integer int64

name string

photoUrls Array of string **required**

tags Array of Tag

Child attributes

id integer int64

name string

status Enum

pet status in the store

Enum elements

available, pending, sold

JSON example

```
{
  "id": 10,
  "name": "doggie",
  "category": {
    "id": 1,
    "name": "Dogs"
  },
  "photoUrls": [
    "example"
  ],
  "tags": [
    {
      "id": 55,
      "name": "example"
    }
  ],
  "status": "available"
}
```

Responses

200 Successful operation

Content type application/json

Successful operation

application/json Pet

Child attributes

id integer int64

name string **required**

category Category

Child attributes

id integer int64

name string

photoUrls Array of string **required**

tags Array of Tag

Child attributes

id integer int64

name string

status Enum

pet status in the store

Enum elements

available, pending, sold

JSON example

```
{
  "id": 10,
  "name": "doggie",
  "category": {
    "id": 1,
    "name": "Dogs"
  },
  "photoUrls": [
    "example"
  ],
  "tags": [
    {
      "id": 55,
      "name": "example"
    }
  ],
  "status": "available"
}
```

400 Invalid ID supplied

Content type

Invalid ID supplied

404 Pet not found

Content type

Pet not found

405 Validation exception

Content type

Validation exception

Add a new pet to the store

POST /pet

Add a new pet to the store

Request parameters

Body

application/json Pet **required**

Child attributes

id integer int64

name string **required**

category Category

Child attributes

id integer int64

name string

photoUrls Array of string **required**

tags Array of Tag

Child attributes

id integer int64

name string

status Enum

pet status in the store

Enum elements

available, pending, sold

JSON example

```
{
  "id": 10,
  "name": "doggie",
  "category": {
    "id": 1,
    "name": "Dogs"
  },
  "photoUrls": [
    "example"
  ],
  "tags": [
    {
      "id": 55,
      "name": "example"
    }
  ],
  "status": "available"
}
```

Responses

200 Successful operation

Content type application/json

Successful operation

application/json Pet

Child attributes

id integer int64

name string **required**

category Category

Child attributes

id integer int64

name string

photoUrls Array of string **required**

tags Array of Tag

Child attributes

id integer int64

name string

status Enum

pet status in the store

Enum elements

available, pending, sold

JSON example

```
{
  "id": 10,
  "name": "doggie",
  "category": {
    "id": 1,
    "name": "Dogs"
  },
  "photoUrls": [
    "example"
  ],
  "tags": [
    {
      "id": 55,
      "name": "example"
    }
  ],
  "status": "available"
}
```

405 Invalid input

Content type

Invalid input

Find pets by status

GET /pet/findByStatus

Multiple status values can be provided with comma separated strings

Request parameters

Query

status Enum

Status values that need to be considered for filter

Enum elements

available, pending, sold

Responses

200 successful operation

Content type application/json

successful operation

application/json Array of Pet

Child attributes

id integer int64

name string **required**

category Category

Child attributes

id integer int64

name string

photoUrls Array of string **required**

tags Array of Tag

Child attributes

id integer int64

name string

status Enum

pet status in the store

Enum elements

available, pending, sold

JSON example

```
[
  {
    "id": 10,
    "name": "doggie",
    "category": {
      "id": 1,
      "name": "Dogs"
    },
    "photoUrls": [
      "example"
    ],
    "tags": [
      {
        "id": 55,
        "name": "example"
      }
    ]
  },
]
```

```
    "status": "available"
  }
]
```

400 Invalid status value

Content type

Invalid status value

Finds pets by tags

GET /pet/findByTags

Multiple tags can be provided with comma separated strings. Use tag1, tag2, tag3 for testing.

Request parameters

Query

tags Array of string

Tags to filter by

Responses

200 successful operation

Content type application/json

successful operation

application/json Array of Pet

Child attributes

id integer int64

name string **required**

category Category

Child attributes

id integer int64

name string

photoUrls Array of string **required**

tags Array of Tag

Child attributes

id integer int64

name string

status Enum

pet status in the store

Enum elements

available, pending, sold

JSON example

```
[  
  {
```

```
[
  {
    "id": 10,
    "name": "doggie",
    "category": {
      "id": 1,
      "name": "Dogs"
    },
    "photoUrls": [
      "example"
    ],
    "tags": [
      {
        "id": 55,
        "name": "example"
      }
    ],
    "status": "available"
  }
]
```

400 Invalid tag value

Content type

Invalid tag value

Find pet by ID

GET /pet/{petId}

Returns a single pet

Request parameters

Path

petId integer int64 **required**

ID of pet to return

Responses

200 successful operation

Content type application/json

successful operation

application/json Pet

Child attributes

id integer int64

name string **required**

category Category

Child attributes

id integer int64

name string

photoUrls Array of string **required**

tags Array of Tag

Child attributes

id integer int64

name string

status Enum

pet status in the store

Enum elements

available, pending, sold

JSON example

```
{
  "id": 10,
  "name": "doggie",
  "category": {
    "id": 1,
    "name": "Dogs"
  },
  "photoUrls": [
    "example"
  ],
  "tags": [
    {
      "id": 55,
      "name": "example"
    }
  ],
  "status": "available"
}
```

400 Invalid ID supplied

Content type

Invalid ID supplied

404 Pet not found

Content type

Pet not found

Update a pet in the store with form data

POST /pet/{petId}

Request parameters

Path

petId integer int64 **required**

ID of pet that needs to be updated

Query

name string

Name of pet that needs to be updated

status string

Status of pet that needs to be updated

Responses

405 Invalid input

Content type

Invalid input

Delete a pet

DELETE /pet/{petId}

Request parameters

Header

api_key string

Path

petId integer int64 **required**

Pet id to delete

Responses

400 Invalid pet value

Content type

Invalid pet value

Upload an image

POST /pet/{petId}/uploadImage

Request parameters

Path

petId integer int64 **required**

ID of pet to update

Query

additionalMetadata string

Additional Metadata

Body

application/octet-stream string binary

Responses

200 successful operation

Content type application/json

successful operation

application/json ApiResponse

Child attributes

code integer int32

type string

message string

JSON example

```
{
  "code": 81,
  "type": "example",
  "message": "example"
}
```

Find pet inventories by status

GET /store/inventory

Returns a map of status codes to quantities

Responses

200 successful operation

Content type application/json

successful operation

application/json object

Child attributes

JSON example

```
{}
```

Place order for pet

v1

POST /store/order/v1

Place a new order in the store

Request parameters

Body

application/json Order

Child attributes

id integer int64

petId integer int64

quantity integer int32

shipDate string date-time

status Enum

Order Status

Enum elements

placed, approved, delivered

complete boolean

JSON example

```
{
  "id": 10,
  "petId": 198772,
  "quantity": 7,
  "shipDate": "1989-04-22T10:33:53Z",
  "status": "approved",
  "complete": false
}
```

Responses

200 successful operation

Content type application/json

successful operation

application/json Order

Child attributes

id integer int64

petId integer int64

quantity integer int32

shipDate string date-time

status Enum

Order Status

Enum elements

placed, approved, delivered

complete boolean

JSON example

```
{
  "id": 10,
  "petId": 198772,
  "quantity": 7,
  "shipDate": "1989-04-22T10:33:53Z",
  "status": "approved",
  "complete": false
}
```

405 Invalid input

Content type

Invalid input

v2

POST /store/order/v2

Place a new order in the store with v2 endpoint

Request parameters

Body

application/json Order [required](#)

Child attributes

id integer int64

petId integer int64

quantity integer int32

shipDate string date-time

status Enum

Order Status

Enum elements

placed, approved, delivered

complete boolean

JSON example

```
{
  "id": 10,
  "petId": 198772,
  "quantity": 7,
  "shipDate": "1989-04-22T10:33:53Z",
  "status": "approved",
  "complete": false
}
```

Responses

201 Order successfully placed

Content type application/json

Order successfully placed

application/json Order

Child attributes

id integer int64

petId integer int64

quantity integer int32

shipDate string date-time

status Enum

Order Status

Enum elements

placed, approved, delivered

complete boolean

JSON example

```
{
  "id": 10,
  "petId": 198772,
  "quantity": 7,
  "shipDate": "1989-04-22T10:33:53Z",
  "status": "approved",
  "complete": false
}
```

400 Bad request

Content type

Bad request

405 Invalid input

Content type

Invalid input

Find purchase order by ID

GET /store/order/{orderId}

For valid response try integer IDs with value less than 5 or > 10. Other values will generate exceptions.

Request parameters

Path

orderId integer int64 **required**

ID of order that needs to be fetched

Responses

200 successful operation

Content type application/json

successful operation

application/json Order

Child attributes

id integer int64

petId integer int64

quantity integer int32

shipDate string date-time

status Enum

Order Status

Enum elements

placed, approved, delivered

complete boolean

JSON example

```
{  
  "id": 10,  
  "petId": 198772,  
  "quantity": 7,  
  "shipDate": "1989-04-22T10:33:53Z",  
  "status": "approved",  
  "complete": false  
}
```

400 Invalid ID supplied

Content type

Invalid ID supplied

404 Order not found

Content type

Order not found

Delete purchase order by ID

DELETE /store/order/{orderId}

For valid response try integer IDs with value < 1000. Anything above 1000 or nonintegers will generate API errors

Request parameters

Path

orderId integer int64 **required**
ID of the order that needs to be deleted

Responses

400 Invalid ID supplied

Content type

Invalid ID supplied

404 Order not found

Content type

Order not found

Create user

POST /user

This can only be done by the logged in user.

Request parameters

Body

application/json User

Child attributes

id integer int64

username string

firstName string

lastName string

email string

password string

phone string

userStatus integer int32

User Status

JSON example

```
{  
  "id": 10,  
  "username": "theUser",  
  "firstName": "John",  
  "lastName": "James",  
  "email": "john@email.com",  
  "password": "12345",  
  "phone": "12345",  
  "userStatus": 1  
}
```

Responses

default successful operation

Content type application/json

successful operation

application/json User

Child attributes

id integer int64

username string

firstName string

lastName string

email string

password string

phone string

userStatus integer int32

User Status

JSON example

```
{  
  "id": 11,  
  "username": "theUser",  
  "firstName": "John",  
  "lastName": "Doe",  
  "email": "john@email.com",  
  "password": "12345",  
  "phone": "12345",  
  "userStatus": 1  
}
```


Create list of users with given input array

POST /user/createWithList

Creates list of users with given input array

Request parameters

Body

application/json Array of User

Child attributes

id integer int64

username string

firstName string

lastName string

email string

password string

phone string

userStatus integer int32

User Status

JSON example

```
[
  {
    "id": 10,
    "username": "theUser",
    "firstName": "John",
    "lastName": "James",
    "email": "john@email.com",
    "password": "12345",
    "phone": "12345",
    "userStatus": 1
  }
]
```

Responses

200 Successful operation

Content type application/json

Successful operation

application/json User

Child attributes

id integer int64

username string

firstName string

lastName string

email string

password string

phone string

userStatus integer int32

User Status

JSON example

```
{  
  "id": 10,  
  "username": "theUser",  
  "firstName": "John",  
  "lastName": "James",  
  "email": "john@email.com",  
  "password": "12345",  
  "phone": "12345",  
  "userStatus": 1  
}
```

default successful operation

Content type

successful operation

400 Invalid username supplied

Content type application/json

Invalid username supplied

application/json ErrorResponse

Child attributes

code integer int32

details Array of object

Child attributes

typeUrl string

value string

message string

JSON example

```
{
  "code": 0,
  "details": [
    {
      "typeUrl": "string",
      "value": "string"
    }
  ],
  "message": "string"
}
```

Log user into the system

GET /user/login

Request parameters

Query

username string

The user name for login

password string

The password for login in clear text

Responses

200 successful operation

Content type application/json

successful operation

application/json string

X-Rate-Limit (Header) integer int32 **required**

calls per hour allowed by the user

X-Expires-After (Header) string date-time **required**

date in UTC when token expires

JSON example

"example"

400 Invalid username/password supplied

Content type

Invalid username/password supplied

Log current user out

GET /user/logout

Responses

default successful operation

Content type

successful operation

Find user

GET /user/{username}

Request parameters

Path

username string **required**

The name that needs to be fetched. Use user1 for testing.

Responses

200 successful operation

Content type application/json

successful operation

application/json User

Child attributes

id integer int64

username string

firstName string

lastName string

email string

password string

phone string

userStatus integer int32

User Status

JSON example

```
{  
  "id": 10,  
  "username": "theUser",  
  "firstName": "John",  
  "lastName": "James",  
  "email": "john@email.com",  
  "password": "12345",  
  "phone": "12345",  
  "userStatus": 1  
}
```

400 Invalid username supplied

Content type

Invalid username supplied

404 User not found

Content type

User not found

Update user

PUT /user/{username}

This can only be done by the logged in user.

Request parameters

Path

username string **required**

name that needs to be updated

Body

application/json User

Child attributes

id integer int64

username string

firstName string

lastName string

email string

password string

phone string

userStatus integer int32

User Status

JSON

```
{
  "id": 10,
  "username": "theUser",
  "firstName": "John",
  "lastName": "James",
  "email": "john@email.com",
  "password": "12345",
  "phone": "12345",
  "userStatus": 1
}
```

JavaScript

```
const userPayload = {
  id: 10,
  username: "theUser",
  firstName: "John",
  lastName: "James",
  email: "john@email.com",
  password: "12345",
  phone: "12345",
  userStatus: 1
};
console.log(userPayload);
```

Responses

default successful operation

Content type

successful operation

Delete user

DELETE /user/{username}

This can only be done by the logged in user.

Request parameters

Path

username string **required**

The name that needs to be deleted

Responses

400 Invalid username supplied

Content type

Invalid username supplied

404 User not found

Content type

User not found

Changes

September 1, 2023

Method	Changes
/pet/gift (Pet)	Deprecated and removed from the documentation
/pet/{petId} (Pet)	<ul style="list-style-type: none">• added the <code>status</code> field• removed the <code>state</code> field