
cloud Documentation

Release alpha

Cloudmesh Community

May 02, 2019

PREFACE

1	About	1
1.1	Features	1
1.2	Roadmap for Future Activities	2
1.3	Contact	2
2	Contributors	3
3	Installation	5
3.1	Prerequisites	5
3.2	Installation of mongod	6
3.3	Anaconda and Conda	7
4	Quickstart	9
4.1	Command line	9
4.2	Interactive shell (proposed)	10
4.3	Command scripts	10
4.4	Cache	11
4.5	Manual	11
5	Configuration	13
5.1	MongoDB	13
5.2	Compute Cloud Providers	13
5.3	Storage Providers	17
5.4	Object Store	19
5.5	Batch	19
5.6	REST	19
5.7	Log File (proposed)	19
6	Cloudmesh yaml file	21
6.1	Variables	21
7	Cloudmesh Database	23
7.1	Database Decorator	24
7.2	Database Access	25
7.3	Creating Unique Names	26
7.4	Cloudmesh Attributes	26

8	Cloudmesh Yaml file Encrytion (TODO)	29
8.1	Generating the Key and Certificate	29
8.2	Validate and verify the key	29
8.3	Encryption	30
8.4	Decryption	30
8.5	Cloudmesh Integration	30
8.6	Editing the Configuration file	30
8.7	Adding information to the configuration	31
8.8	Separating the sensitive information	31
9	Virtual Machine Management	33
9.1	Command Line and Shell Interface	33
9.2	Uniform Parameter Management	34
9.3	Virtual machine management	35
9.4	Key management	35
9.5	Security groups	35
9.6	Command Examples	35
9.7	AWS Quickstart	36
10	Cloudmesh Multi Cloud Storage Interface	37
11	Cloudmesh Storage Module	39
11.1	Requirements	39
11.2	AWSS3 Cloudmesh Integration	39
11.3	Pytests	42
11.4	General features	42
11.5	Virtual Directory	43
11.6	Google drive	44
12	Object Storage	45
12.1	Instalation for Users	45
12.2	Instalation for Developers	45
12.3	Cloudmesh Object Storage Interfaces	46
12.4	Create Object Directory	47
12.5	Put	47
12.6	Get	47
12.7	Search	47
12.8	List	47
12.9	Delete	48
13	Cloudmesh Multi Cloud Open API Interface	49
13.1	Pytests	49
14	Infrastructure Workflow	51
14.1	Javascript Interface (proposed)	52
14.2	REST	53
14.3	Resources	53
15	Jupyter Integration (proposed)	55
15.1	API command shell access (proposed)	55
15.2	API calls (ok)	55
16	Batch	57
16.1	Creating a job configuration	58
16.2	Testing the connection	58

16.3	Running the Job	59
16.4	Downloading the Results	59
16.5	Cleaning the remote	60
16.6	Get the list of the jobs and clusters	60
16.7	Modifying the Configuration by Setting Parameters	60
16.8	Removing jobs and clusters	61
17	CMD5 Integartion	63
17.1	Install	63
18	MAMUAL PAGES	65
18.1	Reference Card (proposed)	65
18.2	Commands	66
18.3	Manual Cmd5	66
18.4	Compute Manual Pages	74
18.5	Storage Manual Pages	87
19	DATABASE OBJECTS	93
19.1	Cloudmesh Database	93
20	BENCHMARKS	111
20.1	Benchmarks	111
20.2	AWS EC2 VM Management	111
20.3	AWS S3 File Storage	113
20.4	Azure Blob Storage	114
20.5	115
20.6	Benchmark results for 'box' Storage	115
20.7	115
20.8	AWS EMR Benchmarking	116
21	CODE	119
21.1	Code Documentation	119
21.2	Common	119
21.3	CMD5	131
21.4	Cloudmesh	136
21.5	Management	136
21.6	Mongo	139
21.7	Commands	139
21.8	Inventory	139
21.9	cloudmesh-storage	139
21.10	cloudmesh-objstorage	141
21.11	cloudmesh-cloud	141
21.12	cloudmesh-batch	148
21.13	cloudmesh-emr	148
21.14	Code Conventions	148
21.15	Code Management	148
21.16	Documentation Management	149
21.17	Version Managemt	149
21.18	Pytest	149
22	Cloud Provider Accounts	153
22.1	Amazon Web Services (AWS) Account Creation Tutorial	153
22.2	Azure Blob Storage and Account Creation	164
22.3	Setting Up Your Box Account	168
22.4	Google Storage Providers	172

22.5	Google (What is this?)	177
22.6	VM Providers (outdated)	178
23	Cloudmesh Outdated	183
23.1	Goal (outdated)	183
23.2	Manual: Cloudmesh Multi Service Data Access	184
23.3	Vagrant (outdated)	185
23.4	CM4 Details (outdated)	186
23.5	AWS cm (outdated)	195
23.6	REST Service (outdated)	201
23.7	Virtual Cluster (in progress)	202
24	Indices and tables	209
	Python Module Index	211

Cloudmesh version 4 is an evolution of our previous tool that has been used by hundreds of students and cloud practitioners to interact easily with clouds to create a service meshup to access common cloud services across a number of cloud providers.

It is under active development. It is managed in github at

- Documentation: <https://cloudmesh.github.io/cloudmesh-manual/>
- Code: <https://github.com/cloudmesh/>

It has a variety of repositories that add features to cloudmesh based on needs by the user.

1.1 Features

- useful common programming library to make it easy to interface with the system <https://github.com/cloudmesh/cloudmesh-common>
- extensible cmondline and shell with cmd5 <https://github.com/cloudmesh/cloudmesh-cmd5>
- convenient installer for developers <https://github.com/cloudmesh/cloudmesh-installer>
- MongoDB as a backend for caching <https://github.com/cloudmesh/cloudmesh-cloud>
- Cloud Providers, AWS, Azure, Google, Openstack <https://github.com/cloudmesh/cloudmesh-cloud>
- A cloud workflow plugin <https://github.com/cloudmesh/cloudmesh-workflow>
- AWS emr plugin <https://github.com/cloudmesh/cloudmesh-emr>
- A plugin for Storage providers <https://github.com/cloudmesh/cloudmesh-storage>
- OpenAPI based REST service interfaces

1.2 Roadmap for Future Activities

- A plugin for HPC systems <https://github.com/cloudmesh/cloudmesh-batch>
- Storage: local provider
- Compute: virtual box, ssh, slurm
- A javascript based GUI
- A cloud high throughput broker for pleasantly parallel parameter studies

1.3 Contact

For more info please contact Gregor von Laszewski, laszewski@gmail.com

Gregor von Laszewski E-mail: laszewski@gmail.com Indiana University School of Informatics and Computing 2425 N Milo B Sampson Ln Bloomington, IN 47408

[Google Map](#)

CHAPTER 2

Contributors

Code Contributions can be seen at

- <https://github.com/cloudmesh/cloudmesh-cloud/graphs/contributors>

The original author of cloudmesh is

- **Gregor von Laszewski** (laszewski@gmail.com)

Large contributions have also been made by

- *Fugang Wang*

Many have contributed to cloudmesh in the past. However, this version has received contributions from a subset of them. Previous contributors will be acknowledged once we integrate them in the author script.

Contributors are sorted by the first letter of their combined Firstname and Lastname and if not available by their github ID. Please, note that the authors are identified through git logs in addition to some contributors added by hand. The git repository from which this document is derived contains more than the documents included in this document. Thus not everyone in this list may have directly contributed to this document. However if you find someone missing that has contributed (they may not have used this particular git) please let us know. We will add you. The contributors that we are aware of include:

Anthon van der Neut, Anthony Duer, Ashok, Badi Abdul-Wahid, Bo Feng, Chun-Sheng Wu, Dave DeMeulenaere, Eric Collins, Fugang Wang, Gerald Manipon, Gregor von Laszewski, Jeevan Reddy Rachepalli, Jing Huang, Karthick, Keli Fine, Mallik Challa, Manjunath Sivan, Ritesh Tandon, Rui Li, Sachith Withana, Scott McClary, Tarun Rawat, Tharak Vangalapat, Vafa Andalibi, Yu Luo, Yue, Xiao, amannars, colliner, fugangwang, himanshu3jul, hyspoc, juaco77, kimballXD, manjunathsivan, robludwig, swsachith, xiao yue, zhengyili4321

3.1 Prerequisites

Before you install make sure that you have at minimum python 3.7.2 installed. We recommend that you use a python virtualenv such as `venv` or `pyenv` to isolate the python installed packages as not to interfere with the system installation.

3.1.1 Installation via pip development

The installation via pip is not yet supported for cloudmesh cm. Thus we recommend that you use the source installation instead.

In future cloudmesh version 4 will be installed with

```
$ pip install cloudmesh-cms
$ pip install cloudmesh-cloud
$ pip install cloudmesh-storage
```

Additional packages will include

```
$ pip install cloudmesh-flow
$ pip install cloudmesh-emr
$ pip install cloudmesh-batch
$ pip install cloudmesh-openapi
```

For the time being we recommend you conduct the source install.

3.1.2 Source installation for development

The best way to install cloudmesh from source is to use our installer:

More documentation about it can be found at

- <https://github.com/cloudmesh/cloudmesh-installer>

You install it with

```
$ pip install cloudmesh-installer
```

It is best to create an empty directory and decide which bundles to install

```
$ mkdir cm
$ cd cm
$ cloudmesh-installer bundles
```

Decide which bundles you like to install (let us assume you use storage) and simply say

```
$ cloudmesh-installer git clone storage
$ cloudmesh-installer install storage -e
```

It will take a while to install. On newer machines 1 minute, on older significant longer.

You can then test if

```
$ cms help
```

works. Make sure to stay up to date while issuing the pull command on your bundle

```
$ cloudmesh-installer git pull bundle
$ cloudmesh-installer install storage -e
```

3.2 Installation of mongod

First, you will need to install a `cloudmesh4.yaml` file, if you have not done this before. The easiest way to do so is with the command

```
$ cms help
```

Now you will need to edit the file

```
~/ .cloudmesh/cloudmesh4.yaml
```

and change the password of the mongo entry to something you like, e.g. change the TBD to a real strong password

```
MONGO_PASSWORD: TBD
```

In case you do not have mongod installed, you can do so for macOS and Ubuntu 18.xx by setting the following variable:

```
MONGO_AUTOINSTALL: True
```

Now you can run the `admin mongo install` command. It will not only install mongo, but also add the path to your `.bash_*` file. In case of windows platform, you will have to set the PATH variable manually. To install it simply say.

```
$ cms admin mongo install
```

To create a password protection you then run the command

`bash $ cms admin mongo create` In case of Windows platform, after executing above command, open a new cms session and execute below commands.

```
$ cms admin mongo start
```

Once the mongo db is created it can be started and stoped with

```
$ cms admin mongo start  
$ cms admin mongo stop
```

For cloudmesh to work properly, please start mongo.

3.3 Anaconda and Conda

We also have the base packages available as conda packages on conda hub in the chanel `laszewski`. This includes

- `cloudmesh-common`
- `cloudmesh-cmd5`
- `cloudmesh-sys`

Note that the packages will always be a little bit behind the packages on pypi and especially the source distribution. If you are interested in helping out with the conda packages, let us know.

CHAPTER 4

Quickstart

One of the features up Cloudmesh is to easily start new virtual machines on various clouds. It uses defaults for these clouds that can be changed, but are easily stored in a yaml file located at `~/ .cloudmesh/cloudmesh4.yaml`. This file is created upon first start of the shell. You need to edit it and include some of your cloud information.

A template for the yaml file is located at:

- <https://github.com/cloudmesh/cloudmesh-cloud/blob/master/cloudmesh/etc/cloudmesh4.yaml>

4.1 Command line

It is easy to switch between clouds with the `set` command. After the `set` and specifying the cloud by name many commands will default to that cloud. The exception is the `vm list` command that lists by default all vms on all clouds. In addition the `vm refresh` command will also work on all clouds.

```
$ cms admin mongo create # needs only be done one time
$ cms admin mongo start

$ cms set cloud=vagrant
$ cms vm start
$ cms image list
$ cms flavor list

$ cms set cloud=aws
$ cms vm start
$ cms image list
$ cms flavor list

$ cms set cloud=azure
$ cms vm start
$ cms image list
$ cms flavor list
```

(continues on next page)

(continued from previous page)

```
$ cms set cloud=chameleon
$ cms vm start
$ cms image list
$ cms flavor list

$ cms set cloud=jetstream
$ cms vm start
$ cms image list
$ cms flavor list

$ cms vm refresh

$ cms vm listcms admin mongo stop
```

In case you want a command explicitly apply to one or more clouds or one or more vms, they can be specified by name such as

```
$ cms vm list --name vm[0-100]
$ cms vm list --cloud aws,azure
```

Defaults for the cloud and the name can be specified through set such as

```
$ cms set name=vm[0-100]
$ cms set cloud=aws,azure
```

Using the commands

```
$ cms vm list
```

would then add the appropriate options to the command. To reset the show to all vms set name and cloud to all

```
$ cms set name=all
$ cms set cloud=all
```

4.2 Interactive shell (proposed)

Cloudmesh uses cmd5 for its shell implementation and thus all commands that are typed in in the terminal can also be typed in into a shell that is started with cms

```
$ cms
cms> set cloud=aws
cms> vm start
```

4.3 Command scripts

As we use cmd5 we also have access to piped and named scripts with

```
$ echo script.cms | cms
```

and


```
$ cms --script script.cms
```

4.4 Cache

All information about for example virtual machines are cached locally. The cache for various information sources can be explicitly updated with the `--refresh` flag. Thus the command

```
$ cms vm list --refresh
```

would first execute a refresh while the command

```
$ cms vm list
```

would only read from the local cache

To change the behavior and always do a refresh you can use the command

```
$ cms set refresh=True
```

To switch it off you can say

```
$ cms set refresh=False
```

4.5 Manual

The manual page can be opened with

```
$ cms open doc
```

or in case you start it in the source with

```
$ cms open doc local
```


CHAPTER 5

Configuration

The Configuration of cloudmesh is controled with a yaml file that is placed in `~/ .cloudmesh/cloudmesh4.yaml`. It is created automatically from the templace located at

- <https://github.com/cloudmesh/cloudmesh-cloud/blob/master/cloudmesh/etc/cloudmesh4.yaml>

You can customize the file in your local directory.

5.1 MongoDB

The cache of cloudmesh is managed in a mongo db database with various collections. However the user does not have to manage thes collections as this is done for the user through cloudmesh. Before you can use it it mongo does need to be installed.

If you have not installed mongo, you may try

```
$ cms admin mongo install
```

Next you create the database template with authentication with

```
$ cms admin mongo create
```

Now you are ready to use it in cloudmesh. The mongo db can be started and stoped with the command

```
$cms admin mongo start
$cms admin mongo stop
```

The configuration details are included in the yaml file.

5.2 Compute Cloud Providers

The default yaml file includes templates to configure various clouds. YOU can change these defaults and provide access to your cloud credentials to make the management of cloud virtual machines easier. Templates for AWS,

Azure, Google, OpenStack are provided. Specific templates for Jetstream and Chameleopn cloud are included in the example `cloudmesh4.yaml`. We list each template next.

5.2.1 AWS

It is beyond the scope of this manual to discuss how to get an account on Aws. However we do provide a convenient documentation at

```
cloudmesh:
  ...
  cloud:
    ...
    aws:
      cm:
        active: False
        heading: AWS
        host: aws.amazon.com
        label: aws
        kind: aws
        version: TBD
      default:
        image: 'ami-0f65671a86f061fcd'
        size: 't2.micro'
      credentials:
        region: 'us-west-2'
        EC2_SECURITY_GROUP: 'group1'
        EC2_ACCESS_ID: TBD
        EC2_SECRET_KEY: TBD
        EC2_PRIVATE_KEY_FILE_PATH: '~/.cloudmesh/aws_cert.pem'
        EC2_PRIVATE_KEY_FILE_NAME: 'aws_cert'
```

5.2.2 Azure

It is beyond the scope of this manual to discuss how to get an account on Azure. However we do provide a convenient documentation at

```
cloudmesh:
  ...
  cloud:
    ...
    azure:
      cm:
        active: False
        heading: AWS
        host: azure.mocrosoft.com
        label: Azure
        kind: azure_arm
        version: TBD
      default:
        image: 'Canonical:UbuntuServer:16.04-LTS:latest'
        size: 'Basic_A0'
        resource_group: 'cloudmesh'
        storage_account: 'cmdrive'
        network: 'cmnetwork'
        subnet: 'cmsubnet'
```

(continues on next page)

(continued from previous page)

```

    blob_container: 'vhds'
  credentials:
    AZURE_TENANT_ID: 'xxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxx'
    AZURE_SUBSCRIPTION_ID: 'xxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxx'
    AZURE_APPLICATION_ID: 'xxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxx'
    AZURE_SECRET_KEY: TBD
    AZURE_REGION: 'northcentralus'

```

5.2.3 Google

It is beyond the scope of this manual to discuss how to get an account on Google. However we do provide a convenient documentation at

```

cloudmesh:
  ...
  cloud:
    ...
    google:
      cm:
        active: True
        heading: google
        host: google.cloud.com
        label: google
        kind: google
        version: TBD
      default:
        image: 'Image Name'
        size: 'n1-standard-4'
      credentials:
        datacenter: 'us-central1-a'
        client_email: '<service account>.iam.gserviceaccount.com'
        project: '<Project Name>'
        path_to_json_file: '~/cloudmesh/<file with credentials>'

```

5.2.4 OpenStack

We provide an example on how to use an OpenStack based cloud in cloudmesh. Please add the following to your cloudmesh4.yaml file and replace the values for TBD. Our example uses [Chameleon Cloud](#). This is a cloud for academic research. Certainly you can configure other clouds based on this template. We have successfully used also clouds in Canada (Cybera), Germany (KIT), Indiana University (jetstream). TO get started you can even install your local cloud with devstack and make adjustments. Please remember you can have multiple clouds in the cloudmesh4.yaml file so you could if you have access to them integrate all of them.

Example for chameleon cloud:

- You will need access to a project and add your project number to the credentials.

```

cloudmesh:
  ...
  cloud:
    ...
    chameleon:
      cm:

```

(continues on next page)

(continued from previous page)

```
active: True
heading: Chameleon
host: chameleoncloud.org
label: chameleon
kind: openstack
version: liberty
credentials:
  OS_AUTH_URL: https://openstack.tacc.chameleoncloud.org:5000/v2.0/tokens
  OS_USERNAME: TBD
  OS_PASSWORD: TBD
  OS_TENANT_NAME: CH-819337
  OS_TENANT_ID: CH-819337
  OS_PROJECT_NAME: CH-819337
  OS_PROJECT_DOMAIN_ID: default
  OS_USER_DOMAIN_ID: default
  OS_VERSION: liberty
  OS_REGION_NAME: RegionOne
  OS_KEY_PATH: ~/.ssh/id_rsa.pub
default:
  flavor: m1.small
  image: CC-Ubuntu16.04
  username: cc
```

5.2.5 Virtual Box

Virtualbox has at this time limited functionality, but creation, ssh, and deletion of the virtual box is possible.

You can also integrate virtualbox as part of cloudmesh while providing the following description:

```
cloudmesh:
  ...
  cloud:
    ...
    vbox:
      cm:
        active: False
        heading: Vagrant
        host: localhost
        label: vbox
        kind: vagrant
        version: TBD
      default:
        path: ~/.cloudmesh/vagrant
        image: "generic/ubuntu1810"
      credentials:
        local: True
        hostname: localhost
```

5.2.6 SSH

STUDENT CONTRIBUTE HERE

5.2.7 Local

STUDENT CONTRIBUTE HERE

5.2.8 Docker

STUDENT CONTRIBUTE HERE

5.3 Storage Providers

General description for all storage providers, comment on the `default:` and what that does

5.3.1 AWS S3

It is beyond the scope of this manual to discuss how to get an account on Google. However we do provide a convenient documentation at

In the `cloudmesh4.yaml` file, the `'aws'` section under `'storage'` describes an example configuration or a AWS S3 storage provider. In the `credentials` section under `aws`, specify the access key id and secret access key which will be available in the AWS console under AWS IAM service -> Users -> Security Credentials. Container is the default Bucket which will be used to store the files in AWS S3. Region is the geographic area like `us-east-1` which contains the bucket. Region is required to get a connection handle on the S3 Client or resource for that geographic area. Here is a sample.

TODO: Make credentials more uniform between compute and data

```
storage:
  aws:
    cm:
      heading: aws
      host: amazon.aws.com
      label: aws
      kind: awsS3
      version: TBD
    default:
      directory: /
    credentials:
      access_key_id: *****
      secret_access_key: *****
      container: name of bucket that you want user to be contained in.
      region: Specfiy the default region eg us-east-1
```

5.3.2 Azure

It is beyond the scope of this manual to discuss how to get an account on Google. However we do provide a convenient documentation at

The `cloudmesh4.yaml` file needs to be set up as follows for the `'azureblob'` section under `'storage'`.

```
cloudmesh:
  .....
  storage:
    azureblob:
      cm:
        heading: Azure
        host: azure.com
        label: Azure
        kind: azureblob
        version: TBD
      default:
        directory: /
      credentials:
        account_name: '*****'
        account_key:
→ '*****'
        container: 'azuretest'
```

Configuration settings for credentials in the yaml file can be obtained from Azure portal.

TODO: MOre information via a pointer to a documentation you create needs to be added here

In the yaml file the following values have to be changed

- `account_name` - This is the name of the Azure blob storage account.
- `account_key` - This can be found under ‘Access Keys’ after navigating to the storage account on the Azure portal.
- `container` - This can be set to a default container created under the Azure blob storage account.

5.3.3 Google drive

Due to bugs in the requirements of the google driver code, we have not yet included it in the Provider code. This needs to be fixed before we can do this.

The `cloudmesh4.yaml` file needs to be set up as follows for the ‘gdrive’ section under ‘storage’.

```
storage:
  gdrive:
    cm:
      heading: GDrive
      host: gdrive.google.com
      kind: gdrive
      label: GDrive
      version: TBD
    credentials:
      auth_host_name: localhost
      auth_host_port:
        - ****
        - ****
      auth_provider_x509_cert_url: "https://www.googleapis.com/oauth2/v1/certs"
      auth_uri: "https://accounts.google.com/o/oauth2/auth"
      client_id: *****
      client_secret: *****
      project_id: *****
      redirect_uris:
        - "urn:ietf:wg:oauth:2.0:oob"
```

(continues on next page)

(continued from previous page)

```

    - "http://localhost"
    token_uri: "https://oauth2.googleapis.com/token"
  default:
    directory: TBD

```

5.3.4 Box

It is beyond the scope of this manual to discuss how to get an account on Google. However we do provide a convenient documentation at

In the `cloudmesh4.yaml` file, find the ‘box’ section under ‘storage’. Under credentials, set `config_path` to the path of the configuration file you created as described in the Box chapter:

```

box:
  cm:
    heading: Box
    host: box.com
    label: Box
    kind: box
    version: TBD
  default:
    directory: /
  credentials:
    config_path: *****

```

5.3.5 ADD OTHERS IF MISSING

5.4 Object Store

5.5 Batch

5.6 REST

TBD

5.7 Log File (proposed)

THIS FEATURE IS NOT YET SUPPORTED

Log files are stored by default in `~/ .cloudmesh/log` The directory can be specified in the yaml file.

Cloudmesh yaml file

cpy the file

- <https://github.com/cloudmesh/cloudmesh-cloud/blob/master/cloudmesh/etc/cloudmesh4.yaml>

to ~/.cloudmesh/cloudmesh4.yaml

```
$ put hed the code for thsi wit h git and so on wget or curl
```

make sure the permissions are

Next edit the yaml file and add your credentials.

6.1 Variables

6.1.1 Replacing home

Values in the yaml file that include a ~ or \$HOME will be replaced with the home directory.

Vales starting with . will be replaced with the current working directory.

In addition any value that includes strings such as "{cloudmesh.attribute}" will be replaced with the value from within the yaml file.

For example. ;et us assume the yaml file contains:

from cloudmesh.management.configuration.config import Config

cloudmesh4.yaml:

```
script =  
"""  
cloudmesh:  
  profile:  
    name: Gregor
```

(continues on next page)

(continued from previous page)

```
cloud:
  aws:
    username: "{cloudmesh.grofile.name}"
    key: ~/.ssh/id_rsa
    dir: $HOME
    current: .
```

will result be transformed with

`data = Config()`

to for example

```
cloudmesh:
  profile:
    name: Gregor
  cloud:
    aws:
      username: "Gregor"
      key: /home/gergor/.ssh/id_rsa
      dir: /home/gregor
      current: /home/gregor/github/cm
```

end converted to a dict. The data in the cloudmesh4.yaml file stays unchanegd.

CHAPTER 7

Cloudmesh Database

Cloudmesh stores its status in a database so that you can easily remember which services you used where and have an accurate account of them. We use as a database mongoDB to store this information. To use cloudmesh you simply need to create and start the database service.

First, you need to create a MongoDB database with

```
$ cms admin mongo create
```

Second, you need to start it with below command (for windows platform, open a new command prompt)

```
$ cms admin mongo start
```

Now you can interact with it to find out the status, the stats, and the database listing with the commands

```
$ cms admin mongo status
$ cms admin mongo stats
$ cms admin mongo list
```

To stop it from running use the command

```
$ cms admin mongo stop
```

The database will be started on the information as specified in `~/.cloudmesh/cloudmesh4.yaml`

An example is

```
mongo:
  MONGO_AUTOINSTALL: True
  MONGO_BREWINSTALL: False
  LOCAL: ~/local
  MONGO_HOME: ~/local/mongo
  MONGO_PATH: ~/.cloudmesh/mongodb
  MONGO_LOG: ~/.cloudmesh/mongodb/log
  MONGO_DBNAME: 'cloudmesh'
  MONGO_HOST: '127.0.0.1'
```

(continues on next page)

(continued from previous page)

```

MONGO_PORT: '27017'
MONGO_USERNAME: 'admin'
MONGO_PASSWORD: TBD
MONGO_DOWNLOAD:
  darwin: https://fastdl.mongodb.org/osx/mongodb-osx-ssl-x86_64-4.0.4.tgz
  linux: https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-4.0.4.tgz
  win32: https://fastdl.mongodb.org/win32/mongodb-win32-x86_64-2008plus-ssl-4.0.
↪4-signed.msi
  redhat: https://repo.mongodb.org/yum/redhat/7/mongodb-org/4.0/x86_64/RPMS/
↪mongodb-org-server-4.0.4-1.el7.x86_64.rpm

```

We also provide a convenient install script that downloads the version defined in the yaml file and installs it in the system with the command. In case of windows platform, you will have to set the PATH variable manually after install

```
$ cms admin mongo install
```

7.1 Database Decorator

Cloudmesh comes with a very convenient mechanism to integrate data into MongoDB. All you have to do is to create a list of dictionaries with a function, that returns this dictionary and use a decorator in the function to update the information into the database.

The data base decorator automatically replaces an entry in the database with the dictionary returned by a function.

It is added to a MongoDB collection. The location is determined from the values in the dictionary.

The name of the collection is determined from cloud and kind:

```
cloud-kind
```

In addition each entry in the collection has a name that must be unique in that collection.

In most examples it is best to separate the upload from the actual return class. This way we essentially provide two functions one that provide the dict and another that is responsible for the upload to the database.

Example:

cloudmesh.example.foo contains:

```

class Provider(object)

    def entries(self):
        return {
            "cm" : {
                "kind" : "flavor",
                "driver" : "openstack",
                "cloud" : "foo",
                "created" : "2019-04-01 15:59:39.815993",
                "name" : "m1.xxxlarge",
                "collection" : "chameleon-flavor",
                "modified" : "2019-04-01 16:01:11.720274"
            },

```

cloudmesh.example.bar contains:

```
class Provider(object)

    def entries(self):
        return {
            "cm" : {
                "kind" : "flavor",
                "driver" : "openstack",
                "cloud" : "bar",
                "created" : "2019-04-01 15:59:39.815993",
                "name" : "ml.xxxlarge",
                "collection" : "chameleon-flavor",
                "modified" : "2019-04-01 16:01:11.720274"
            },

```

cloudmesh.example.provider.foo contains:

```
from cloudmesh.example.foo import Provider as FooProvider
from cloudmesh.example.foo import Provider as BarProvider

class Provider(object)

    def __init__(self, provider):
        if provider == "foo":
            provider = FooProvider()
        elif provider == "bar":
            provider = BarProvider()

    @DatabaseUpdate()
    def entries(self):
        provider.entries()
```

Separating the database and the dictionary creation allows the developer to implement different providers but only use one class with the same methods to interact for all providers with the database.

In the combined provider a find function to for example search for entries by name across collections could be implemented.

7.2 Database Access

In addition to the decorator, we have a very simple database class for interacting across a number of collections. This especially is useful for finding informtion.

```
self.database = CmDatabase()
```

Find the entry with the unique name CC-Centos

```
r = self.database.find_name("CC-CentOS7")
pprint(r)
```

Find the entries with either CC-CentOS7 or CC-CentOS7-1811

```
r = self.database.find_names("CC-CentOS7,CC-CentOS7-1811")
pprint(r)
```

Find out how many entries exist with the name CC-CentOS7:

```
r = self.database.name_count("CC-CentOS7")
pprint(r)
```

7.3 Creating Unique Names

Unique names with the format {experiment}-{group}-{user}-{counter} can be created with

```
from cloumesh.management.configuration.name import Name

name = Name(
    experiment="exp",
    group="grp",
    user="gregor",
    kind="vm",
    counter=1)
```

To increae the counter use

```
name.incr()
```

To get the name at the current counter value say

```
str(name)
```

or

```
name.id()
```

The format can be chaned with `schema=` at the initailization. Thus

```
name = Name(
    user='gregor',
    schema='{user}-{counter}',
    counter=1)
```

would create names of the form gergor1, gergor2 and so on.

7.4 Cloudmesh Attributes

Cloudmesh elements in the database will have a special `cm` dictionary with a number of attributes defined in it. The following example showcases such an attribute dict. The attributs can be used to uniquely define an object in the database by cobining the cloud, kind, and name. In addition it contains the date for the object being created first and its update time.

```
"cm" : {
    "name" : "ml.medium",
    "created" : "2019-03-25 07:45:46.905623",
    "updated" : "2019-03-25 07:45:46.905623",
    "cloud" : "chameleon",
    "kind" : "flavor",
    "driver" : "openstack",
    "collection" : "chameleon-flavor"
},
```


Using this information the object can easily be found in the database by name, type or cloud or a combination thereof.

Cloudmesh Yaml file Encrytion (TODO)

THIS SECTION IS UNDER DEVELOPMENT AND THE CONTENT DESCRIBED IS NOT YET WORKING

The cloudmesh yaml file can contain some information to simplify authentication. IN order not to store the file in cleartext we have provided a replacement function for the configuration that allows encryption of the file with your ssh key. IT is important that your ssh key is generated with a passphrase. If you do not, even if you encrypt, the file can be decrypted without a passphrase which is the same as having it in cleartext. SO make sure your ssh-key has a passphrase.

8.1 Generating the Key and Certificate

We can encrypt and decrypt files using a generated random key as follows:

First, you need to create a public-private key with a passphrase. This can be achieved with the `cms key` command; it assumes that you have not yet created a key

```
cms config ssh keygen
```

Alternatively, you can create a key as follows

```
ssh-keygen -t rsa -m pem
```

In case you need to convert your key to a pem certificate, you can do it as follows

```
$ openssl rsa -in ~/.ssh/id_rsa -out ~/.ssh/id_rsa.pem
```

8.2 Validate and verify the key

To validate the key, please use the `cms` command

```
$ cms config check
$ cms config verify
```

8.3 Encryption

To encrypt the file, please use the command

```
$ cms config encrypt
```

This command will encrypt your `cloudmesh4.yaml` file and place it under `~/.cloudmesh/cloudmesh4.yaml.enc`. It will ask you if you like to delete the original yaml file.

8.4 Decryption

To decrypt the file, please use the command

```
$ cms config decrypt
```

This command will decrypt your `cloudmesh4.yaml.enc` file and place it under `~/.cloudmesh/cloudmesh4.yaml.enc.enc`. It will ask you if you like to delete the original yaml file.

8.5 Cloudmesh Integration

We have provided a new `cloudmesh.management.configuration.ConfigCrypt()` That will be integrated in future (once verified it works) into the regular `Config()`

The functionality that is provided by `ConfigCrypt()` includes

- if `cloudmesh4.yaml` and `cloudmesh4.yaml.enc` exist a warning is written that both files exist and it is recommended in production to delete the unencrypted file.
- if only `cloudmesh4.yaml` exist a warning is written that an unencrypted yaml file is used
- if only `cloudmesh4.yaml.enc` exists it is unencrypted and loaded into memory. Please note that `ConfigCrypt()` just as `Config()` is implemented as Borg class so that the decryption and loading is conducted only once.

8.6 Editing the Configuration file

Editing the configuration file can be done by first unencrypting the file with

```
$ TBD
```

Then you can use your favourite editor to make modifications. Let us assume this is emacs.

```
emacs ~/.cloudmesh/cloudmesh4.yaml
```

Once written back quit your editor and encrypt the file with

```
cms config encrypt
```

8.7 Adding information to the configuration

It is also possible to add configurations to the encrypted file while storing the new values in a temporary yaml file.

Let us assume the temporary file `./change.yaml` contains the following information:

```
cloudmesh:
  profile:
    firstname: Gregor
    lastname: von Laszewski
```

Then the command

```
cms config add ./change.yaml
```

Will update the existing `cloudmesh4.yaml` or `cloudmesh4.yaml.enc` file with the provided information. This is taking place regardless if the yaml file is encrypted or not. If both files exist, both files will be modified. A warning is however issued if the unencrypted yaml file exists to remind the user to delete it.

Alternatively the convenient dot notation cloudmesh provides for configuration files can be used. This is done by having the ending `txt` instead of `yaml`. We illustrated this on the following example where the data is stored in `change.txt`

```
cloudmesh.profile.firstname: Gregor
cloudmesh.profile.lastname: von Laszewski
```

```
cms config add ./change.txt
```

8.8 Separating the sensitive information

As it may be beneficial to separate the sensitive from the non sensitive information, we also provide a mechanism for authentication with a merged file. This way you could for example store the sensitive information on a USB key.

For this to work we specify in the yaml file a field called

```
cloudmesh:
  encrypted: ~/.cloudmesh/cloudmesh4-secrets.yaml.enc
```

You can name the file anything you like and you could point it to your location of the USB key.

in this file we store only the sensitive information such as

```
cloudmesh.storage.azure.credentials.AZURE_SUBSCRIPTION_ID: 'xxxxxx-xxxx-xxxx-xxxx-
↳xxxxxxxxxxxxx'
```

Please note that we leverage the convenient dot notation cloudmesh provides for configuration files so we can formulate the value in a single line>

Virtual Machine Management

CLoudmesh v4 contains sophisticated virtual machine management services that makes it easy for the user to manage a large number of virtual machines across clouds with a uniform naming scheme.

For now we will focus on the command line and shell interface.

9.1 Command Line and Shell Interface

The command line and shell interface to manage virtual machines are listed next.

```
vm ping [NAMES] [--cloud=CLOUDS] [N]
vm check [NAMES] [--cloud=CLOUDS]
vm refresh [NAMES] [--cloud=CLOUDS]
vm status [NAMES] [--cloud=CLOUDS]
vm console [NAME] [--force]
vm start [NAMES] [--cloud=CLOUD] [--dryrun]
vm stop [NAMES] [--cloud=CLOUD] [--dryrun]
vm terminate [NAMES] [--cloud=CLOUD] [--dryrun]
vm delete [NAMES] [--cloud=CLOUD] [--dryrun]
vm list [NAMES]
    [--cloud=CLOUDS]
    [--output=OUTPUT]
    [--refresh]
vm boot [--name=NAME]
    [--cloud=CLOUD]
    [--username=USERNAME]
    [--image=IMAGE]
    [--flavor=FLAVOR]
    [--public]
    [--secgroup=SECGROUPs]
    [--key=KEY]
    [--dryrun]
vm boot [--n=COUNT]
```

(continues on next page)

(continued from previous page)

```

    [--cloud=CLOUD]
    [--username=USERNAME]
    [--image=IMAGE]
    [--flavor=FLAVOR]
    [--public]
    [--secgroup=SECGROUPS]
    [--key=KEY]
    [--dryrun]
vm run [--name=NAMES] [--username=USERNAME] [--dryrun] COMMAND
vm script [--name=NAMES] [--username=USERNAME] [--dryrun] SCRIPT
vm ip assign [NAMES]
    [--cloud=CLOUD]
vm ip show [NAMES]
    [--cloud=CLOUD]
    [--output=OUTPUT]
    [--refresh]
vm ip inventory [NAMES]
vm ssh [NAMES] [--username=USER]
    [--quiet]
    [--ip=IP]
    [--key=KEY]
    [--command=COMMAND]
    [--modify-knownhosts]
vm rename [OLDNAMES] [NEWNAMES] [--force] [--dryrun]
vm wait [--cloud=CLOUD] [--interval=SECONDS]
vm info [--cloud=CLOUD]
    [--output=OUTPUT]
vm username USERNAME [NAMES] [--cloud=CLOUD]
vm resize [NAMES] [--size=SIZE]

```

9.2 Uniform Parameter Management

The parameters across the commands are uniformly managed. Most of the plural form allow a parameterized specification such as `a[00-03]`, `a8` which would result in an array `["a0", "a1", "a2", "a3", "a8"]`. This especially applies to clouds as well as virtual machine names.

We distinguish the following parameterized options

`:-cloud=CLOUDS`: which specifies one or more clouds in parameterized fashion

`:-names=NAMES`: which specifies one or more clouds in parameterized fashion

We distinguish the following regular options

`:-interval=INTERVAL`: a specified interval in seconds

`:-output=OUTPUT`: The output format: txt, csv, table

`:-refresh`: To update the state of the vms specified with clouds and names

`:-username=USERNAME`: The username to be used for connecting with the vm

`:-quiet`: do not print debug messages

`:-dryrun`: do not execute the command, but just print what would happen

`:-ip=IP`: specify a public IP

`:-key=KEY`: start the vm with the keypair name

9.3 Virtual machine management

Virtual machines can be

- Created
- Started
- Stopped
- Suspended
- Resumed
- Destroyed

Default behavior such as a key management naming scheme as well as ip adress and security management is conveniently provided

9.4 Key management

Access to the virtual machien is governed by SSH keys. The default key can be uploaded to the cloud with the key command. The name of the key in the cloud can be used to associate it with virtual machines so that this key can be used to log into the VM

9.5 Security groups

A security group acts as a virtual firewall for the instance. When we launch a instance, we want to attach security Groups for controlling the traffic in and out of the VM.

9.6 Command Examples

9.6.1 Ping

```
vm ping [NAMES] [--cloud=CLOUDS] [N]
```

9.6.2 Check

```
vm check [NAMES] [--cloud=CLOUDS]
```

9.6.3 Refersh

```
vm refresh [NAMES] [--cloud=CLOUDS]
```

9.6.4 Status

```
vm status [NAMES] [--cloud=CLOUDS]
```

9.6.5 Console

vm console [NAME] [--force]

9.6.6 Start

vm start [NAMES] [--cloud=CLOUD] [--dryrun]

9.6.7 Stop

vm stop [NAMES] [--cloud=CLOUD] [--dryrun]

9.6.8 Terminate

vm terminate [NAMES] [--cloud=CLOUD] [--dryrun]

9.6.9 Delete

vm delete [NAMES] [--cloud=CLOUD] [--dryrun]

9.7 AWS Quickstart

vm boot --name=test_cloudmesh --cloud=aws

vm status --name=test_cloudmesh --cloud=aws (check to see if test_cloudmesh is running)

vm ping --name=test_cloudmesh --cloud=aws

vm check --name=test_cloudmesh --cloud=aws

vm ssh --name=test_cloudmesh --cloud=aws

vm stop --name=test_cloudmesh --cloud=aws

vm start --name=test_cloudmesh --cloud=aws

vm terminate --name=test_cloudmesh --cloud=aws

CHAPTER 10

Cloudmesh Multi Cloud Storage Interface

Cloudmesh multiple cloud storage services is independent of the APIs and interfaces used to access these services. In other words, an abstraction layer between data and the proprietary APIs is used to place that data in any given cloud storage service.

Provides a interface to manage all cloud storage in one place. it helps you access and search all of your files in one place so you don't need to sign into several accounts.

Cloudmesh Storage Module

Note: Do not modify the shield, once we release the storage module they will work

Version

11.1 Requirements

Please note that several packages are available which are pointed to in the installation documentation.

|| Links || ———— | ——— | || Documentation | <https://cloudmesh.github.io/cloudmesh-manual> || Code | <https://github.com/cloudmesh/cloudmesh-storage> || Installation Instructions | <https://github.com/cloudmesh-installer> |

An dynamically extensible CMD based command shell. For en extensive documentation please see

- <https://github.com/cloudmesh-community/book/blob/master/vonLaszewski-cloud.epub?raw=true>

where we also document how to use pyenv virtualenv.

11.2 AWSS3 Cloudmesh Integration

AWS S3 file storage has been integrated with cloudmesh library and is available for use via commandline. As a first step we need to modify `cloudmesh4.yaml` config file. Under 'storage' section, we need to add the aws section to specify the parameters used to store files in AWS S3.

In the credentials section under aws, specify the access key id and secret access key which will be available in the AWS console under AWS IAM service -> Users -> Security Credentials.

Container is the default bucket which will be used to store the files in AWS S3. Region is the geographic area like `us-east-1` which contains the bucket. Region is required to get a connection handle on the S3 Client or resource for that geographic area.

Here is a sample.

```
cloudmesh:
...
storage:
  aws:
    cm:
      heading: aws
      host: amazon.aws.com
      label: aws
      kind: awsS3
      version: TBD
    default:
      directory: TBD
  credentials:
    access_key_id: *****
    secret_access_key: *****
    container: name of bucket that you want user to be contained in.
    region: Specfiy the default region eg us-east-1
```

The Cloudmesh command line library offers six functions under storage command: get, put, search, list, create directory, and delete. Once you have installed Cloudmesh, type `cms` into the command line to start the cms shell.

```
$ cms
+-----+
| / _ _ | | _ _ | | _ _ | | _ _ | | _ _ | | _ _ | | _ _ | | _ _ | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| \ _ _ | | \ _ _ | | \ _ _ | | \ _ _ | | \ _ _ | | \ _ _ | | \ _ _ |
+-----+
|                                     Cloudmesh CMD5 Shell                                     |
+-----+

cms>
```

To view the docopt for storage command, type in

```
cms> help storage
```

Help command gives a detail level understanding of what each command does and how to use the command line to interact with different storage providers and different parameters / options available in a particular command. For eg to invoke AWS S3 service, we need to pass `awss3` as parameter to storage and suffix with the function call with the function parameters.

```
cms> storage --storage='aws' list ''
```

Alternatively, storage command can also be called directly without starting the cms shell.

```
$ cms storage --storage='aws' list ''
```

11.2.1 Storage functions overview

11.2.2 Create dir

This command helps to create a new directory on AWS S3. You must specify the full path of the new directory you would like to create.

```
$ cms storage --storage='aws' create dir /base_path/targetdir
```

11.2.3 Put

The put command uploads files from your local host to the S3.

```
$ cms storage --storage='aws' put ~/.cloudmesh/storage/sourcedir /base_path/targetdir_
↪--recursive
```

Source for this command could be either a file or directory.

If you specify a file as the source, the file will be uploaded if no such file exists on the cloud or updated if a copy already exists on the cloud.

If the source is a directory, you can choose to specify the recursive option to upload the files in the sub-directories in the source as well to the target directory in S3. If the recursive option is not specified, only the files in the source directory will be uploaded to the target directory and the sub-directories will be ignored.

11.2.4 Get

The get command downloads files from S3 to your local host.

```
$ cms storage --storage='aws' get /base_container/sourcedir ~/.cloudmesh/storage/
↪targetdir --recursive
```

Source for this command could be either a file or directory.

If you specify a file as the source, you need to specify the full path of file including the file name where you want the file to be downloaded. In case you do not specify the file name and only give the target directory, then the file will be downloaded with the same name as present on S3.

If the source is a directory, you can choose to specify the recursive option to download files in the sub-directories in the source as well to the target directory in your local host. If the recursive option is not specified, only the files in the source directory will be downloaded to the target directory and the sub-directories will be ignored.

11.2.5 Search

The search command helps to search for a particular file within a directory.

If recursive options is specified, Cloudmesh will search for the file in all sub-directories of the original directory as well.

To search for a file at the root, pass an empty string or / as the target dir.

```
$ cms storage --storage='aws' search /base_path/targetdir testfile.txt --recursive
```

Note that for the Box storage provider, objects are only indexed every 5 to 10 minutes and will not show up in a search until they have been indexed.

11.2.6 List

The list command lists all the contents of a cloud directory. If the recursive option is specified, it will list the contents of all sub-directories as well.

```
$ cms storage --storage='aws' list /base_path/targetdir --recursive
```

11.2.7 Delete

The delete command can delete files or folders from your cloud file storage. Deleting a folder will delete its contents as well (including the sub-directories).

```
$ cms storage --storage='aws' delete /base_path/targetdir --recursive
```

11.3 Pytests

11.3.1 Generic Tests

We have developed a number of simple pytests that can be called. To see the list of Pytests go to our directory

- <https://github.com/cloudmesh/cloudmesh-storage/tree/master/tests>

We also developed a general pytest that works accross providers and can be invoked as follows

```
$ cms set storage=box
$ pytest -v --capture=no tests/test_storage.py

$ cms set storage=azure
$ pytest -v --capture=no tests/test_storage.py

$ cms set storage=gdrive
$ pytest -v --capture=no tests/test_storage.py

$ cms set storage=awss3
$ pytest -v --capture=no tests/test_storage.py
```

11.3.2 Provider Specific Pytests

Open a terminal and navigate to the cloudmesh-storage directory. Enter the following command to run pytests:

```
$ pytest -v --capture=no tests/test_storage_box.py
$ pytest -v --capture=no tests/test_azure.py
$ pytest -v --capture=no tests/test_storage_aws.py
```

TODO: rename to

- test_storage_azure.py

11.4 General features

How to set up the authentication to a specific service is discussed in later sections

TODO: Provide a simple programming example with the general provider

11.4.1 Command Line Interface

TBD

```
$ cms set storage=azure
$ cms storage list
```

11.4.2 Programming Interface

TBD

Cloudmesh Storage provides a simple programming API interface that you can use. We highlight a simple example for storing and retrieving a file from a storage provider.

We assume the files at the given path exist

```
import cloudmesh.storage.provider.Provider as Provider
from cloudmesh.common.util import path_expand
from pprint import pprint

provider = Provider(service="azure")
src = path_expand("~/cloudmesh/storage/test/a/a.txt")
dst = "/"
result = provider.put(src, dst)
# The result will be a dict of the information which you can print with
pprint(result)
```

11.4.3 Pytests

Script to test the GDrive service can be accessed under tests folder using the following pytest command.

TODO rename to test_storage_gdrive.py

```
$ pytest -v --capture=no tests/test_gdrive.py
```

11.5 Virtual Directory

The virtual directory has been developed to mirror the linux directory commands. File links in the virtual directory point to files on storage providers, which can be retrieved using the virtual directory.

11.5.1 Configuration

The credentials for the virtual directory are the same as for the admin mongo command. See the Mongo section for details.

11.5.2 Pytests

The vdir command can be tested as follows:

```
$ pytest -v --capture=no tests/test_vdir.py
```

11.6 Google drive

The Google Drive API needs the following two 2 credentials files.

- `client_secret.json`
- `google-drive-credentials.json`

If we run the Google Drive Provider.py for the **First time** then the required keys, tokens are taken from the `cloudmesh4.yaml` file and creates a `client_secret.json` file in the following path `~/.cloudmesh/gdrive/`

The `Authentication.py` creates a `.credentials` folder under the following path `~/.cloudmesh/gdrive/` if it doesn't exist and creates a `google-drive-credentials.json` file under the following folder `~/.cloudmesh/gdrive/.credentials/`

So, for the **First time** browser will be opened up automatically and asks for the Google Drive(gmail) credentials i.e., login email and password. If you provide these 2 then the Authentication step is completed and then it will create the `google-drive-credentials.json` and place it in `~/.cloudmesh/gdrive/.credentials/` folder.

These steps are to be followed for the first time or initial run. Once it is done then our program is set. After these steps then the program will run automatically by using these credentials stored in the respective files.

11.6.1 Note

The Google Drive API accepts these 2 files in the form of **.json file format** and not in the form of a dictionary.

11.6.2 Links

Link for additional information:

- <https://github.com/cloudmesh-community/sp19-516-130/blob/master/gdrive.md>

12.1 Instalation for Users

At this time we do not offer this ~~but it will be~~

~~pip install cloudmesh-community~~

Gregor will set this up, so please do not do this yourself.

12.2 Instalation for Developers

This install only works if you use ssh-keys with github

```
mkdir cm
cd cm
pip install cloudmesh-installer
#
# if you have not uploaded your ssh key to git do so. One option is
# cloudmesh-installer git key
#
cloudmesh-installer clone storage
cloudmesh-installer install storage -e
git clone git@github.com:cloudmesh/cloudmesh-objstorage.git
cd cloudmesh-objstorage
pip install -e .
```

12.3 Cloudfmesh Object Storage Interfaces

12.3.1 Object Storage with ASW S3

Object Storage is one of the feature in AWS S3 and this feature integrated with cloudmesh library and is available for use via commandline.

Follow the below steps:

- Modify `cloudmesh4.yaml` config file in 'cloudmesh-objstorage' section. User need to add required object storage parameters to communicate with cloud(AWS S3)
- In the credentials section under `awss3`, add the parameter values of `access_key_id` and `secret_access_key`, these credentials will be gained from appropriate cloud vendor(For ex: AWS), in the case of AWS, these will be available which will be available in the AWS console under `AWS IAM service -> Users -> Security Credentials`.

Here is a sample.

```
cloudmesh:
...
objstorage:
  awss3:
    cm:
      heading: aws
      host: amazon.aws.com
      label: aws
      kind: awss3
      version: 1.0
    default:
      directory: AWS
  credentials:
    access_key_id: *****
    secret_access_key: *****
```

The Cloudmesh command line library offers several functions as part of objstorage command: get, put, search, list, create, and delete. Once you have installed Cloudmesh, type `cms` into the command line to start the cms shell.

[illegible]

To view the docopt for objstorage command, type in

```
cms> help objstorage
```

Help command gives a detail level understanding of what each command does and how to use the command line to interact with different object storage providers and different parameters / options available in a particular command.

In this, default object storage invokes AWS S3 service, we need to pass `awss3` as parameter to storage and suffix with the function call with the function parameters.

```
cms> objstorage --objstorage='aws3' list ''
```

Alternatively, `objstorage` command can also be called directly without starting the `cms` shell.

```
$ cms objstorage --objstorage='awss3' list ''
```

12.3.2 Objstorage Functionality

12.4 Create Object Directory

This command helps to create a new bucket before storage an object on AWS S3. You must specify the full path of the new directory you would like to create.

```
$ cms objstorage --objstorage='awss3' create bucket_name /base_path/
```

12.5 Put

The `put` command uploads object from your local system to AWS S3 object storage

```
$ cms objstorage --objstorage='awss3' put ~/.cloudmesh/objstorage/src /base_path/dest_
↪--recursive
```

12.6 Get

The `put` command retrieve or download a object from AWS S3 object storage

```
$ cms objstorage --objstorage='awss3' get /bucket_name/src ~/.cloudmesh/objstorage/
↪dest --recursive
```

12.7 Search

The advantage of search command to search a given object in specified bucket location

```
$ cms objstorage --objstorage='aws3' search //bucket_name/dest "<<objectname>>" --
↪recursive
```

12.8 List

The `list` command lists all the contents of a cloud object details. If the recursive option is specified, it will list the contents of all the nested objects information

```
$ cms objstorage --objstorage='awss3' list /bucket_name/dest --recursive
```

12.9 Delete

The delete command can delete objects on cloud storage. Once object deletes it will never be rollback and delete applicable to nested objects when function `--recursive` used. Deleting a folder will delete its contents as well (including the sub-directories).

```
$ cms objstorage --objstorage='awss3' delete /bucket_name/est --recursive
```

Cloudmesh Multi Cloud Open API Interface

Cloudmesh-storage also provides an OpenAPI specification that allows users to run the multi cloud storage services using a single REST service in addition to the command line interface.

The REST service is developed independent of the storage service provider and users can switch between providers by setting the `storage` variable as follows:

```
$ cms set storage='azureblob'
```

Note: `azureblob` can be replaced with the desired service in the above command.

To invoke the server, navigate to the OpenAPI folder in the `cloudmesh-storage` directory and use:

```
$ cms openapi server start ./openapi_storage.yaml
```

to start up the server on the default address and port. Once the server is started all cloudmesh-storage functions can be accessed using the following endpoints.

```
http://localhost:8080/cloudmesh/storage/v1/put
http://localhost:8080/cloudmesh/storage/v1/get
http://localhost:8080/cloudmesh/storage/v1/list
http://localhost:8080/cloudmesh/storage/v1/search
http://localhost:8080/cloudmesh/storage/v1/delete
http://localhost:8080/cloudmesh/storage/v1/create_dir
```

All of the options described in the `cloudmesh-storage` section are available in the OpenAPI specification as arguments. For example, to list the files from a specific directory on the service, the following URL can be visited:

```
http://localhost:8080/cloudmesh/storage/v1/list?service={storage}&directory=
↪%2fapitest&recursive=True
```

13.1 Pytests

A generic pytest is also developed which is available in the following directory

- <https://github.com/cloudmesh/cloudmesh-storage/tree/master/cloudmesh/storage/spec/tests>

The generic pytest that works accross providers and can be invoked as follows

```
$ cms set storage=azureblob
$ pytest -v --capture=no tests/test_openapi_storage.py

$ cms set storage=box
$ pytest -v --capture=no tests/test_openapi_storage.py

$ cms set storage=gdrive
$ pytest -v --capture=no tests/test_openapi_storage.py

$ cms set storage=awss3
$ pytest -v --capture=no tests/test_openapi_storage.py
```

Infrastructure Workflow

Cloudmesh supports an infrastructure [workflow](#) where users can specify python functions and map their execution on cloud infrastructure. The workflow feature allows you to define Python functions on a workflow class, and specify when to execute them via the command line or in a YAML file. You can then visualize the result of your workflow execution.

An example is given below.

```
from cloudmesh.flow.FlowDecorator import BaseWorkFlow

class MyFlow(BaseWorkFlow):
    def a(self):
        print("in a!")
        time.sleep(5)
    def b(self):
        print("in b!")
        time.sleep(10)
    def c(self):
        print("in c!")
        time.sleep(10)
```

This allows you to define functions in your workflow file. Then you can write a specification for the sequence to execute your functions:

```
(a; b || c)
```

Where

- a ; b is executed sequentially
- b || c is executed in parallel.

Finally, after execution the results are stored in MongoDB to be visualized or consumed by later functions in the series.

14.1 Javascript Interface (proposed)

We are looking for someone that would chose as its project to include a rendering of some DAG in javascript. The javascript library must be free to use. Nodes and edges must be able to be labeled.

A promissing start for a Javascript library is

- http://visjs.org/network_examples.html
- <http://visjs.org/examples/network/events/interactionEvents.html>

This project is only recommended for someone that knows javascript already.

You will do the rest of the project in python. It is important that the functions be specified in python and not just Javascript. The focus is not on specifying the DAG with a GUI, but to visualizing it at runtime with status updates

Here is another summary that we posted earlier and is probably better as it has a dict return

So what we want to do is something i have done previously somewhere with graphviz, but instead of using graphviz we use java script. W want to define tasks that depend on each other. The tasks are defined as python functions. The dependencieas are specified via a simple graph string

```
def a (); print("a"); sleep(1) ; return {"status": "done", "color":"green", shape:
↪"circle", label="a"}

def b (); print("b"); sleep(2); return{"status": "done", "color":"green", shape:
↪"circle", label="b"}

def c (); print("c"); sleep(3); return{"status": "done", "color":"green", shape:
↪"circle", label="c"}

w = workflow("a; b | c")

; = sequential

| = parallel

w.run()
```

While executing the javascript would change dynamically the state and color after a calculation is completed. The workflow should also be able to be specified in yaml

Here just one idea:

```
tasks:
  task:
    name: a
    parameter:
      x: "int"
      y:: "int"
    calculation: f(x,y)
    entry:
      color: green
      label: a
      value: x (this is a python variable local to the function
```

(continues on next page)

(continued from previous page)

```
    shape: circle
    return:
    color: green
    label: a
    value: x (this is a python variable local to the function)
    shape: circle
```

Naturally at one point $f(x,y)$ will be cloud related such as starting a vm and executing a command in teh vm

Followup:

We added a value to the return. Values can be any object.

```
def a():
    x = 10
    return {"status": "done",
            "color": "green",
            "shape": "circle",
            "label": "c",
            "value": x}
```

14.2 REST

An OpenAPI specification for this is to be defined.

14.3 Resources

- <https://github.com/xflr6/graphviz>
- <http://visjs.org/examples/network/events/interactionEvents.html>

CHAPTER 15

Jupyter Integration (proposed)

As cloudmesh provides an API but also is available as command shell it is very easy to integrate it into jupyter

In this section we describe how to do this.

Any cms command can be run via the shell

```
[1] !cms set cloud=AWS
[2] !cms vm start
```

15.1 API command shell access (proposed)

To use a more pythonic approach you can do

```
import cloudmesh

script = """
cms set cloud=AWS
cms vm start
"""

cloudmesh.shell(script)
```

15.2 API calls (ok)

To use the specific API calls, look at the manual or the tests. To list for example the flavors of a cloud you can use:

```
from cloudmesh.compute.libcloud.Provider import Provider
from cloudmesh.common.Printer import Printer
from pprint import pprint
```

(continues on next page)

(continued from previous page)

```
provider = Provider(name="chameleon")
images= provider.images()

pprint (images)
```

To print the information in a nice table you can also use

```
print(Printer.flatwrite(images,
                        sort_keys=("name", "extra.minDisk"),
                        order=["name", "extra.minDisk", "updated", "driver"],
                        header=["Name", "MinDisk", "Updated", "Driver"])
)
```

The printer has a flatwrite method included that first converts the dict into a flat dict, where each attribute is changed to a single level dict by using a period to indicate the indentation of the dicts in case dict of dicts are used as in our example

The purpose of this sub-command is to facilitate job submission on clusters that use SLURM as their workload manager. Note that this tool assumes that the SLURM file is properly prepared by the user and does not modify the SLURM script in any way. Similar to other sub-commands `batch` has several sub-commands itself:

```
cms batch create-job JOB_NAME --slurm-script=SLURM_SCRIPT_PATH --input-type=INPUT_
↪TYPE --slurm-cluster=SLURM_CLUSTER_NAME --job-script-path=SCRIPT_PATH --remote-
↪path=REMOTE_PATH --local-path=LOCAL_PATH [--argfile-path=ARGUMENT_FILE_PATH] [--
↪outfile-name=OUTPUT_FILE_NAME] [--suffix=SUFFIX] [--overwrite]
cms batch run-job JOB_NAME
cms batch fetch JOB_NAME
cms batch test-connection SLURM_CLUSTER_NAME
cms batch set-param slurm-cluster CLUSTER_NAME PARAMETER VALUE
cms batch set-param job-metadata JOB_NAME PARAMETER VALUE
cms batch list slurm-clusters [DEPTH [default:1]]
cms batch list jobs [DEPTH [default:1]]
cms batch remove slurm-cluster CLUSTER_NAME
cms batch remove job JOB_NAME
cms batch clean-remote JOB_NAME
```

The main options are:

- `create-job`: used for creating a job configuration (this does not run the job automatically)
- `run-job`: used for running a job configuration that is previously created.
- `test-connection`: used for testing the connection to a SLURM cluster
- `set-param`: used for setting a parameter in any configuration key
- `list`: used for listing possible instances of an entity
- `remove`: used for removing a cluster or job
- `clean-remote`: used for cleaning the files of a job from a cluster

Each of these sub-commands are reviewed in the following sections with examples.

16.1 Creating a job configuration

As can be seen, this sub-command has the most number of arguments and is the vital part of the `batch` tool. The parameters are all self-explanatory, but we will review the important ones here:

- `--slurm-script`: defines the path to the SLURM script that is going to be submitted to the SLURM cluster.
- `--input-type`: defines the type of input for the application that is going to be run on the cluster. This is important because if the program takes a file name as an argument, that file has to be transferred to the cluster as well. Possible values for this parameter is either `params` or `params+file`. Note that if you pass `params+file` then you have to specify the `--argfile-path` as well where you define the path to the argument file.
- `--slurm-cluster`: defines the name of the cluster that is previously defined in cloudmesh yaml file.
- `--job-script-path`: defines the path to the file that is going to be run on the SLURM cluster
- `--remote-path`: defines the path on SLURM cluster on which the job files are going to be copied, run and collected.
- `--local-path`: defines the local path for saving the results.

Consider the following example :

```
$ cms batch create-job SlurmTest1 --slurm-script=./1_argsin_stdout.slurm --input-
→type=params --slurm-cluster=slurm-taito --job-script-path=./1_argsin_stdout_script.
→sh --remote-path=~/.tmp --local-path=./batch/sample_scripts/out --overwrite
```

This will create a job that looks like this in the `slurm_batch` configuration file placed in the workspace directory:

```
slurm_cluster:
  slurm-taito:
    name: taito
    credentials:
      sshconfigpath: ~/vms/sshconfig_slurm
job-metadata:
  SlurmTest1:
    suffix: _20181206_19275141
    slurm_cluster_name: slurm-taito
    input_type: params+file
    raw_remote_path: ~/.tmp
    slurm_script_path: ./4_filein_fileout.slurm
    job_script_path: ./4_filein_fileout_script.sh
    argfile_path: ./test-script-argument
    argfile_name: test-script-argument
    script_name: 4_filein_fileout_script.sh
    slurm_script_name: 4_filein_fileout.slurm
    remote_path: ~/.tmp/job_20181206_19275141/
    remote_script_path: ~/.tmp/job_20181206_19275141/4_filein_fileout_script.sh
    remote_slurm_script_path: ~/.tmp/job_20181206_19275141/4_filein_fileout.slurm
    local_path: ../batch/sample_scripts/out
```

16.2 Testing the connection

Note that the cluster information is already extracted and added to this file. Therefore unlike `vcluster`, there is no need to add the cluster manually. So far, we have just added and updated the configuration and the job is neither submitted nor run in the cluster. Before doing that, let's try to test our connection to the cluster:


```
$ cms batch test-connection slurm-taito
Slurm Cluster taito is accessible.
```

16.3 Running the Job

Now that we are sure that the ssh connection works fine, let's try to run the job:

```
$ cms batch run-job SlurmTest1
Remote job ID: 32846209
```

Despite the short output, this command does a lot of work behind the seen including:

- Creating the proper folder structure in the remote
- Copying the SLURM script, as well as the job script and the argument files if any.
- Submitting the job
- Keeping the job ID and save it in the configuration file so that the results can be fetched later

Just for the demonstration purpose, let's check the remote folder in the cluster and you will see that all of the files as well as the results will be available there:

```
@taito-login3:~/tmp/job_20181206_19301175$ ll
total 28
drwxr-xr-x 2 4096 Dec  7 02:36 ./
drwx----- 3 4096 Dec  7 02:35 ../
-rwxr-xr-x 1 238 Dec  7 02:35 4_filein_fileout.slurm*
-rw-r--r-- 1 0 Dec  7 02:36 4_filein_fileout.slurm.e32846209
-rw-r--r-- 1 117 Dec  7 02:36 4_filein_fileout.slurm.o32846209
-rwxr-xr-x 1 48 Dec  7 02:35 4_filein_fileout_script.sh*
-rw-r--r-- 1 35 Dec  7 02:35 test-script-argument
-rw----- 1 35 Dec  7 02:36 test-script-output
```

16.4 Downloading the Results

Now that the results are ready we can fetch the results using the following command:

```
$ cms batch fetch SlurmTest1
collecting results
Results collected from taito for jobID 32846209
waiting for other results if any...
All of the remote results collected.
```

Using this, the results will be downloaded in the local path specified in the configuration file:

```
out$ ll job_20181206_19301175/
total 1M
drwxr-xr-x 2 corriel 1M Dec  6 19:40 ./
drwxr-xr-x 3 corriel 1M Dec  6 19:40 ../
-rw-r--r-- 1 corriel 0M Dec  6 19:40 4_filein_fileout.slurm.e32846209
-rw-r--r-- 1 corriel 1M Dec  6 19:40 4_filein_fileout.slurm.o32846209
-rw----- 1 corriel 1M Dec  6 19:40 test-script-output
```

16.5 Cleaning the remote

Now that you are done, you can easily clean the remote using:

```
$ cms batch clean-remote SlurmTest1
Job SlurmTest1 cleaned successfully.
```

16.6 Get the list of the jobs and clusters

Naturally after working with the `batch` for a while, several jobs and clusters will be accumulated in the configuration file. You can get the list of current jobs and clusters using the following commands:

```
$ cms batch list slurm-clusters
slurm-taito:
  name
  credentials
$ cms batch list jobs
SlurmTest1:
  suffix
  slurm_cluster_name
  input_type
  raw_remote_path
  slurm_script_path
  job_script_path
  argfile_path
  argfile_name
  script_name
  slurm_script_name
  remote_path
  remote_script_path
  remote_slurm_script_path
  local_path
  jobIDs
```

It is also possible to increase the depth of the information by adding the desired depth as the next parameter:

```
$ cms batch list slurm-clusters 2
slurm-taito:
  name:
    taito
  credentials:
    sshconfigpath:
      ~/vms/sshconfig_slurm
```

16.7 Modifying the Configuration by Setting Parameters

In case you want to modify or add a configuration parameter, there is no need to directly modify the file. Indeed you can use the `set-param` command to set a key for both jobs and slurm-clusters. In the next example we will add a `test-key` and `test-value` parameter to the `slurm-taito` cluster:

```
$ cms batch set-param slurm-cluster slurm-taito test-key test-value
slurm-cluster parameter test-key set to test-value successfully.

$ cms batch list slurm-clusters 2
slurm-taito:
  name:
    taito
  credentials:
    sshconfigpath:
      ~/vms/sshconfig_slurm
  test-key:
    test-value
```

16.8 Removing jobs and clusters

Finally, when you are done with a job, or when a cluster is not accessible anymore, you can easily remove them from the batch configuration file using the following:

```
$ cms baremove slurm-cluster slurm-taito
Slurm-cluster slurm-taito removed successfully.
```

similarly, you can remove a obsolete job using the following command:

```
$ cms batch remove job SlurmTest1
Job SlurmTest1 removed successfully.
```


17.1 Install

TODO verify if pip install is different from python setup.py

```
pip install cmd5  
pip install -e .
```

```
git clone https://github.com/cloudmesh/cloudmesh.common.git  
git clone https://github.com/cloudmesh/cloudmesh.cmd5.git  
git clone https://github.com/cloudmesh/cloudmesh.bar.git  
git clone https://github.com/cloudmesh/cloudmesh.sys.git  
cd ~/github/cloudmesh.common  
python setup.py install  
pip install .  
cd ~/github/cloudmesh.cmd5  
python setup.py install  
pip install .  
cd ~/github/cloudmesh.sys  
python setup.py install  
pip install .
```


18.1 Reference Card (proposed)

18.1.1 Shell

Table 1: Shell

Command	Description
cms help	help
cms man	manual pages
cms script.cm	execute cm commands in script

18.1.2 Shell commands that expire after a session

Table 2: Shell

Command	Description
cms color on	sets the shell color
cms color off	switches off the color
cms refresh on	automatic refresh from the clouds
cms refresh off	data is only read from the database. Useful for managing thousands of VMs or limit your access to the cloud.
var a=xyx	declares a variable
var user-name=cloudmesh.profile.username	reads the variable from the cloudmesh.yaml file
var time=now	gets the time and store it in the variable time

18.1.3 Clouds

Table 3: Cloud

Command	Description
cms image list	list images
cms flavor list	list flavors
cms vm list	list vms
cms vm boot	boot vm
cms vm boot --cloud=kilo	boot vm on cloud kilo
cms default cloud=kilo	set default cloud to kilo
cms select image	select interactively the default image (not implemented yet).
cms select flavor	select interactively the default flavor (not implemented yet).
cms select cloud	select interactively the default cloud (not implemented yet).

18.1.4 Comet

18.1.5 HPC

Table 4: HPC

Command	Description
cms help	Help
cms hpc queue <batch>	info about the queue <batch>
cms hpc info	information about the queues on the HPC resource
cms hpc run uname -a	runs the command uname
cms hpc run list	prints the ids of previously run jobs
cms hpc run list	prints the ids of previously run jobs
cms hpc run list 11	prints the information regarding the job with the id 11

18.2 Commands

EOF data image objstorage secgroup storage version admin default info open set sys vm banner echo inventory pause
shell url workflow batch emr key plugin sleep var clear flavor login q source vbox config flow man queue ssh vcluster
container help network quit stopwatch vdir

18.3 Manual Cmd5

18.3.1 admin

```
Usage:
  admin mongo install [--brew] [--download=PATH]
  admin mongo create
  admin mongo status
  admin mongo stats
  admin mongo version
  admin mongo start
  admin mongo stop
```

(continues on next page)

(continued from previous page)

```

admin mongo backup FILENAME
admin mongo load FILENAME
admin mongo security
admin mongo password PASSWORD
admin mongo list
admin rest status
admin rest start
admin rest stop
admin status
admin system info
admin yaml cat
admin yaml check

```

The admin command performs some administrative functions, such as installing packages, → software and services.

It also is used to start services and configure them.

Arguments:

FILENAME the filename for backups

Options:

-f specify the file

Description:

Mongo DB

MongoDB is managed through a number of commands.

The configuration is read from ~/.cloudmesh/cloudmesh4.yaml

First, you need to create a MongoDB database with

```
cms admin mongo create
```

Second, you need to start it with

```
cms admin mongo start
```

Now you can interact with it to find out the status, the stats, and the database listing with the commands

```

cms admin mongo status
cms admin mongo stats
cms admin mongo list

```

To stop it from running use the command

```
cms admin mongo stop
```

System information about your machine can be returned by

```
cms admin system info
```

This can be very useful in case you are filing an issue or bug.

18.3.2 banner

```
Usage:
  banner [-c CHAR] [-n WIDTH] [-i INDENT] [-r COLOR] TEXT...

Arguments:
  TEXT...  The text message from which to create the banner
  CHAR     The character for the frame.
  WIDTH    Width of the banner
  INDENT   indentation of the banner
  COLOR    the color

Options:
  -c CHAR  The character for the frame. [default: #]
  -n WIDTH The width of the banner. [default: 70]
  -i INDENT The width of the banner. [default: 0]
  -r COLOR The color of the banner. [default: BLACK]

Prints a banner form a one line text message.
```

18.3.3 clear

```
Usage:
  clear

Clears the screen.
```

18.3.4 default

```
Usage:
  default list [--context=CONTEXT] [--format=FORMAT]
  default delete --context=CONTEXT
  default delete KEY [--context=CONTEXT]
  default KEY [--context=CONTEXT]
  default KEY=VALUE [--CONTEXT=CONTEXT]

Arguments:
  KEY     the name of the default
  VALUE   the value to set the key to

Options:
  --context=CONTEXT  the name of the context
  --format=FORMAT    the output format. Values include
                    table, json, csv, yaml.

Description:
  Cloudmesh has the ability to manage easily multiple
  clouds. One of the key concepts to manage multiple clouds
  is to use defaults for the cloud, the images, flavors,
  and other values. The default command is used to manage
  such default values. These defaults are used in other commands
```

(continues on next page)

(continued from previous page)

if they are **not** overwritten by a command parameter.

The current default values can be listed **with**

```
default list --all
```

Via the default command you can **list**, **set**, get **and** delete default values. You can **list** the defaults **with**

```
default list
```

A default can be **set with**

```
default KEY=VALUE
```

To look up a default value you can say

```
default KEY
```

A default can be deleted **with**

```
default delete KEY
```

To be specific to a cloud you can specify the name of the cloud **with** the `--cloud=CLOUD` option. The **list** command can **print** the information **in** various formats iv specified.

Examples:

```
default list --all
  lists all default values

default list --cloud=kilo
  lists the defaults for the cloud with the name kilo

default image=xyz
  sets the default image for the default cloud to xyz

default image=abc --cloud=kilo
  sets the default image for the cloud kilo to xyz

default image
  list the default image of the default cloud

default image --cloud=kilo
  list the default image of the cloud kilo

default delete image
  deletes the value for the default image in the
  default cloud

default delete image --cloud=kilo
  deletes the value for the default image in the
  cloud kilo
```

18.3.5 echo

```
Usage:
  echo  [-r COLOR] TEXT

Arguments:
  TEXT    The text message to print
  COLOR   the color

Options:
  -r COLOR The color of the text. [default: BLACK]

Prints a text in the given color
```

18.3.6 info

```
Usage:
  info [path|commands|files|cloudmesh]

Description:
  info
  provides internal info about the shell and its packages
```

18.3.7 pause

```
Usage:
  pause [MESSAGE]

Arguments:
  MESSAGE message to be displayed

Description:
  Displays the specified text then waits for the user to press RETURN.
```

18.3.8 plugin

```
Usage:
  plugin install PLUGIN [-s]
  plugin uninstall PLUGIN
  plugin list
  plugin ? [--format=FORMAT]

Arguments:
  PLUGIN the name of the plugin

Description:
  plugin available
    lists the available plugins
  plugin list
    lists the plugin
  plugin install
```

(continues on next page)

(continued from previous page)

```
    installs the given plugin
plugin uninstall
    uninstalls the given plugin
```

18.3.9 q

Usage:
quit

Description:
Action to be performed when quit **is** typed

18.3.10 quit

Usage:
quit

Description:
Action to be performed when quit **is** typed

18.3.11 shell

Usage:
shell COMMAND

Arguments:
COMMAND the command to be executed

Description:
shell COMMAND executes the command

18.3.12 sleep

Usage:
sleep SECONDS

Clears the screen.

18.3.13 stopwatch

Usage:
stopwatch start TIMER
stopwatch stop TIMER
stopwatch **print** [TIMER]

Arguments:

(continues on next page)

(continued from previous page)

TIMER the name of the timer

Description:

THIS IS NOT YET WORKING
starts **and** stops named timers **and** prints them

18.3.14 sys

Usage:

```
sys upload
sys commit MESSAGE
sys command generate NAME
sys version VERSION
```

This command does some useful things.

Arguments:

```
MESSAGE the message to commit
NAME     the command to generate
VERSION  the version number
```

Options:

```
-f specify the file
```

Description:

```
cms sys command generate my
    This requires that you have checked out
```

```
./cloudmesh-common
./cloudmesh-cmd5
./cloudmesh-sys
```

When you execute **in** . this command
will generate a sample directory tree **for**
the command 'my'.

You can than modify

```
cloudmesh.my/cloudmesh/my/command/my.py
```

to define your own cmd5 add on commands.
You install the command **with**

```
cd cloudmesh.my; pip install .
```

The commands 'version', 'commit' **and** 'upload'
are only to be used by Gregor.

```
cms version
```

The version command adds a new version to the
VERSION file **for** cmd5, common, **and** sys.
This helps to keep the versions aligned across
these modules.

```
cms commit
```

(continues on next page)

(continued from previous page)

The `commit` command adds a new version **and** commits

`cms upload`

The `upload` command uploads the new version to pypi

18.3.15 var

Usage:

```
var list
var clear
var delete NAME
var NAME=VALUE
var NAME
```

Arguments:

```
NAME      the name of the variable
VALUE     the value of the variable
FILENAME  the filename of the variable
```

Description:

Manage persistent variables

`var NAME=VALUE`

```
sets the variable with the name to the value
if the value is one of data, time, now it will be
replaced with the value at this time, the format will be
date      2017-04-14
time      11:30:33
now       2017-04-14 11:30:41
```

It will wbe replaced accordingly

The value can also refer to another variable name.

In this case the current value will be copied in the named variable. As we use the `$` sign it is important to distinguish shell variables from cms variables while using proper quoting.

Examples include:

```
cms var a=\$b
cms var 'a=$b'
cms var a=val.b
```

The previous command copy the value from `b` to `a`. The `val` command was added to avoid quoting.

18.3.16 version

Usage:

```
version pip [PACKAGE]
version [--format=FORMAT] [--check=CHECK]
```

Options:

(continues on next page)

(continued from previous page)

```
--format=FORMAT  the format to print the versions in [default: table]
--check=CHECK    boolean tp conduct an additional check [default: True]
```

Description:

```
version
    Prints out the version number
version pip
    Prints the contents of pip list
```

Limitations:

Package names must **not** have a **.** **in** them instead you need to use **-**
Thus to query **for** cloudmesh-cmd5 use

```
cms version pip cloudmesh-cmd5
```

18.4 Compute Manual Pages

18.4.1 batch

Usage:

```
batch job create
    --name=NAME
    --cluster=CLUSTER
    --script=SCRIPT
    --executable=EXECUTABLE
    --destination=DESTINATION
    --source=SOURCE
    [--companion-file=COMPANION_FILE]
    [--outfile-name=OUTPUT_FILE_NAME]
    [--suffix=SUFFIX]
    [--overwrite]
batch job run [--name=NAMES] [--output=OUTPUT]
batch job fetch [--name=NAMES]
batch job remove [--name=NAMES]
batch job clean [--name=NAMES]
batch job set [--name=NAMES] PARAMETER=VALUE
batch job list [--name=NAMES] [--depth=DEPTH]
batch connection_test --job=JOB
batch cluster list [--cluster=CLUSTERS] [--depth=DEPTH]
batch cluster remove [--cluster=CLUSTERS]
batch cluster set [--cluster=CLUSTERS] PARAMETER=VALUE
```

Arguments:

```
FILE    a file name
INPUT_TYPE  tbd
```

Options:

```
-f          specify the file
--depth=DEPTH [default: 1]
--output=OUTPUT [default: table]
```

Description:

(continues on next page)

(continued from previous page)

This command allows to submit batch jobs to queuing systems hosted **in** an HBC center **as** a service directly form your commandline.

We assume that a number of experiments are conducted **with** possibly running the script multiple times. Each experiment will save the batch script **in** its own folder.

The output of the script can be saved **in** a destination folder. A virtual directory **is** used to coordinate **all** saved files.

The files can be located due to the use of the virtual directory on multiple different data **or** file services

Authentication to the Batch systems **is** done viw the underlaying HPC center authentication. We assume that the user has an account to submit on these systems.

(SSH, 2 factor, XSEDE-account) TBD.

Experiments:

experiments are jobs that can be run multiple times **and** create **input** **and** output file sin them

cloudmesh:

 experiment:

 job:

```

      name: {cloudmesh.profile.user.name}-01
      directory: ~/experiment/{experiment.job.name}
      output:  {cloudmesh.experiment.job.name}/output
      input:  ~/experiment/{experiment.job.name}/input
      script: script.sh
      source ,,,
      destination: {cloudmesh.experiment.job.directory}

```

- queue associates **with** server (cloud)
- job could be run on queue **and is** associated **with** one **or** multiple servers
- experiment **is** same **as** job, but gives some facility to run it multiple times

I do **not** know what companion file **is**

Examples:

```
batch job run [--name=NAMES] [--output=OUTPUT]
```

 runs jobs **with** the given names

LOTS OF DOCUMENTATION MISSING HERE

```

  [--companion-file=COMPANION_FILE]
  [--outfile-name=OUTPUT_FILE_NAME]
  [--suffix=SUFFIX] [--overwrite]

```

18.4.2 flavor

Usage:

```
flavor list [NAMES] [--cloud=CLOUD] [--refresh] [--output=OUTPUT]
```

Options:

```
--output=OUTPUT  the output format [default: table]
--cloud=CLOUD    the cloud name
--refresh        refreshes the data before displaying it
```

Description:

This lists out the flavors present **for** a cloud

Examples:

```
cm flavor refresh
cm flavor list
cm flavor list --output=csv
cm flavor list 58c9552c-8d93-42c0-9dea-5f48d90a3188 --refresh
```

please remember that a uuid **or** the falvor name can be used to identify a flavor.

18.4.3 image

Usage:

```
image list [NAMES] [--cloud=CLOUD] [--refresh] [--output=OUTPUT]
```

Options:

```
--output=OUTPUT  the output format [default: table]
--cloud=CLOUD    the cloud name
--refresh        live data taken from the cloud
```

Description:

```
cm image list
cm image list --output=csv
cm image list 58c9552c-8d93-42c0-9dea-5f48d90a3188 --refresh
```

18.4.4 key

Usage:

```
key -h | --help
key list --cloud=CLOUDS [--output=OUTPUT]
key list --source=ssh [--dir=DIR] [--output=OUTPUT]
key list --source=git [--output=OUTPUT] [--username=USERNAME]
key list [NAMES] [--output=OUTPUT]
key load --filename=FILENAME [--output=OUTPUT]
key add [NAME] [--source=FILENAME]
key add [NAME] [--source=git]
key add [NAME] [--source=ssh]
key get NAME [--output=OUTPUT]
key default --select
```

(continues on next page)

(continued from previous page)

```

key delete (NAMES | --select | --all) [--dryrun]
key delete NAMES --cloud=CLOUDS [--dryrun]
key upload [NAMES] [--cloud=CLOUDS] [--dryrun]
key upload [NAMES] [VMS] [--dryrun]
key group upload [--group=GROUPNAMES] [--cloud=CLOUDS] [--dryrun]
key group add [--group=GROUPNAMES] [--cloud=CLOUDS] [--dryrun]
key group add --file=FILENAME
key group delete [--group=GROUPNAMES] [NAMES] [--dryrun]
key group list [--group=GROUPNAMES] [--output=OUTPUT]
key group export --group=GROUPNAMES --filename=FILENAME

```

Arguments:

VMS	Parameterized list of virtual machines
CLOUDS	The clouds
NAME	The name of the key.
SOURCE	db, ssh, all
KEYNAME	The name of a key. For key upload it defaults to the default key_↵ ↵name.
OUTPUT	The format of the output (table, json, yaml)
FILENAME	The filename with full path in which the key is located

Options:

--dir=DIR	the directory with keys [default: ~/.ssh]
--output=OUTPUT	the format of the output [default: table]
--source=SOURCE	the source for the keys [default: cm]
--username=USERNAME	the source for the keys [default: none]
--name=KEYNAME	The name of a key

Description:

Please note that some values are read **from the** cloudmesh4.yaml file. One such value **is** cloudmesh.profile.user

Manages public keys **is** an essential component of accessing virtual machine sin the cloud. There are a number of sources where you can find public keys. This includes teh ~/.ssh directory **and for** example github. To **list** these keys the following **list** functions are provided.

```

key list --source=git [--username=USERNAME]
  lists all keys in git for the specified user. If the
  name is not specified it is read from cloudmesh4.yaml
key list --source=ssh [--dir=DIR] [--output=OUTPUT]
  lists all keys in the directory. If the directory is not
  specified the default will be ~/.ssh
key list NAMES
  lists all keys in the named virtual machines.

```

The keys will be uploaded into cloudmesh **with** the add command under the given name. If the name **is not** specified the name cloudmesh.profile.user **is** assumed.

```

key add --ssh
  adds the default key in ~/.ssh/id_rsa.pub

```

(continues on next page)

(continued from previous page)

```
key add NAME --source=FILENAME
  adds the key specified by the filename with the given name
key add NAME --git --username=username
  adds a named github key from a user with the given github
  username.
```

Once the keys are uploaded to github, they can be listed

```
key list [NAME] [--output=OUTPUT]
  list the keys loaded to cloudmesh in the given format:
  json, yaml, table is default. The NAME can be
  specified and if omitted the name cloudmesh.profile.user
  is assumed.
```

```
key get NAME
  Retrieves the key indicated by the NAME parameter from
  cloudmesh and prints its details.
```

```
key default --select
  Select the default key interactively
```

```
key delete NAMES
  deletes the keys. This may also have an impact on groups
```

```
key rename NAME NEW
  renames the key from NAME to NEW.
```

Group management of keys **is** an important concept **in** cloudmesh, allowing multiple users to be added to virtual machines. The keys must be uploaded to cloudmesh **with** a name so they can be used **in** a group. The `--dryrun` option executes the command without uploading the information to the clouds. If no groupname **is** specified the groupname default **is** assumed. If no cloudnames are specified, **all** active clouds are assumed. active clouds can be **set in** the cloudmesh4.yaml file.

```
key group delete [GROUPNAMES] [NAMES] [--dryrun]
  deletes the named keys from the named groups.
```

```
key group list [GROUPNAMES] [--output=OUTPUT]
  list the key names and details in the group.
```

```
key group upload [GROUPNAMES] [CLOUDS] [--dryrun]
  uploads the named groups to the specified clouds.
```

In some cases you may want to store the public keys **in** files. For this reason we support the following commands.

```
key group add --group=GROUPNAME --file=FILENAME
  the command adds the keys to the given group. The keys are
  written in the files in yaml format.
```

```
key group export --group=GROUPNAMES --filename=FILENAME
  the command exports the keys to the given group. The keys are
  written in the files in yaml format.
```

The yaml **format is as** follows:

(continues on next page)

(continued from previous page)

```

cloudmesh:
  keys:
    NAMEOFKEY:
      name: NAMEOFKEY
      key: ssh-rsa AAAA..... comment
      group:
        - GROUPNAME
    ...

```

If a key **is** included **in** multiple groups they will be added to the grouplist of the key

18.4.5 network

Usage:

```

network get fixed [ip] [--cloud=CLOUD] FIXED_IP
network get floating [ip] [--cloud=CLOUD] FLOATING_IP_ID
network reserve fixed [ip] [--cloud=CLOUD] FIXED_IP
network unreserve fixed [ip] [--cloud=CLOUD] FIXED_IP
network associate floating [ip] [--cloud=CLOUD] [--group=GROUP]
                        [--instance=INS_ID_OR_NAME] [FLOATING_IP]
network disassociate floating [ip] [--cloud=CLOUD] [--group=GROUP]
                        [--instance=INS_ID_OR_NAME] [FLOATING_IP]
network create floating [ip] [--cloud=CLOUD] [--pool=FLOATING_IP_POOL]
network delete floating [ip] [--cloud=CLOUD] [--unused] [FLOATING_IP]
network list floating pool [--cloud=CLOUD]
network list floating [ip] [--cloud=CLOUD] [--unused] [--instance=INS_ID_OR_NAME]
↪ [IP_OR_ID]
network create cluster --group=demo_group
network -h | --help

```

Options:

-h	help message
--unused	unused floating ips
--cloud=CLOUD	Name of the IaaS cloud e.g. india_openstack_grizzly.
--group=GROUP	Name of the group in Cloudmesh
--pool=FLOATING_IP_POOL	Name of Floating IP Pool
--instance=INS_ID_OR_NAME	ID or Name of the vm instance

Arguments:

IP_OR_ID	IP Address or ID of IP Address
FIXED_IP	Fixed IP Address, e.g. 10.1.5.2
FLOATING_IP	Floating IP Address, e.g. 192.1.66.8
FLOATING_IP_ID	ID associated with Floating IP, e.g. 185c5195-e824-4e7b-8581-703abec4bc01

Examples:

```

network get fixed ip --cloud=india 10.1.2.5
network get fixed --cloud=india 10.1.2.5
network get floating ip --cloud=india 185c5195-e824-4e7b-8581-703abec4bc01
network get floating --cloud=india 185c5195-e824-4e7b-8581-703abec4bc01
network reserve fixed ip --cloud=india 10.1.2.5
network reserve fixed --cloud=india 10.1.2.5
network unreserve fixed ip --cloud=india 10.1.2.5
network unreserve fixed --cloud=india 10.1.2.5

```

(continues on next page)

(continued from previous page)

```

network associate floating ip --cloud=india --instance=albert-001 192.1.66.8
network associate floating --cloud=india --instance=albert-001
network associate floating --cloud=india --group=albert_group
network disassociate floating ip --cloud=india --instance=albert-001 192.1.66.8
network disassociate floating --cloud=india --instance=albert-001 192.1.66.8
network create floating ip --cloud=india --pool=albert-f01
network create floating --cloud=india --pool=albert-f01
network delete floating ip --cloud=india 192.1.66.8 192.1.66.9
network delete floating --cloud=india 192.1.66.8 192.1.66.9
network list floating ip --cloud=india
network list floating --cloud=india
network list floating --cloud=india --unused
network list floating --cloud=india 192.1.66.8
network list floating --cloud=india --instance=323c5195-7yy34-4e7b-8581-703abec4b
network list floating pool --cloud=india
network create cluster --group=demo_group

```

18.4.6 open

Usage:

```

open chameleon baremetal tacc
open chameleon baremetal uc
open chameleon vm
open chameleon openstack
open FILENAME
open doc local
open doc

```

Arguments:

```

FILENAME  the file to open in the cwd if . is
           specified. If file in in cwd
           you must specify it with ./FILENAME

           if the FILENAME is doc then the documentation from the Web
           is opened.

```

Description:

```

Opens the given URL in a browser window.

open chameleon baremetal tacc
    starts horizon for baremetal for chameleon cloud at TACC

open chameleon baremetal uc
    starts horizon for baremetal for chameleon cloud at UC

open chameleon vm
    starts horizon for virtual machines

```

18.4.7 secgroup

Usage:

```
secgroup list [--output=OUTPUT]
secgroup list --cloud=CLOUD [--output=OUTPUT]
secgroup list GROUP [--output=OUTPUT]
secgroup add GROUP RULE FROMPORT TOPORT PROTOCOL CIDR
secgroup delete GROUP [--cloud=CLOUD]
secgroup delete GROUP RULE
secgroup upload [GROUP] [--cloud=CLOUD]
```

Options:

```
--output=OUTPUT Specify output format, in one of the following:
                    table, csv, json, yaml, dict. The default value
                    is 'table'.
--cloud=CLOUD      Name of the IaaS cloud e.g. kilo, chameleon.
                    The clouds are defined in the yaml file.
                    If the name "all" is used for the cloud all
                    clouds will be selected.
```

Arguments:

```
RULE              The security group rule name
GROUP             The label/name of the security group
FROMPORT          Starting port of the rule, e.g. 22
TOPORT            Ending port of the rule, e.g. 22
PROTOCOL          Protocol applied, e.g. TCP,UDP,ICMP
CIDR              IP address range in CIDR format, e.g.,
                    129.79.0.0/16
```

Examples:

```
secgroup list
secgroup list --cloud=kilo
secgroup add my_new_group webapp 8080 8080 tcp 0.0.0.0/0
secgroup delete my_group my_rule
secgroup delete my_unused_group --cloud=kilo
secgroup upload --cloud=kilo
```

Description:

```
security_group command provides list/add/delete
security_groups for a tenant of a cloud, as well as
list/add/delete of rules for a security group from a
specified cloud and tenant.
Security groups are first assembled in a local database.
Once they are defined they can be added to the clouds.
secgroup list [--output=OUTPUT]
    lists all security groups and rules in the database
secgroup list GROUP [--output=OUTPUT]
    lists a given security group and its rules defined
    locally in the database
secgroup list --cloud=CLOUD [--output=OUTPUT]
    lists the security groups and rules on the specified clouds.
secgroup add GROUP RULE FROMPORT TOPORT PROTOCOL CIDR
    adds a security rule with the given group and the details
    of the security rules
secgroup delete GROUP [--cloud=CLOUD]
    Deletes a security group from the local database. To make
    the change on the remote cloud, using the 'upload' command
    afterwards.
    If the --cloud parameter is specified, the change would be
```

(continues on next page)

(continued from previous page)

```

    made directly on the specified cloud
    secgroup delete GROUP RULE
        deletes the given rule from the group. To make this change
        on the remote cloud, using 'upload' command.
    secgroup upload [GROUP] [--cloud=CLOUD...]
        uploads a given group to the given cloud. If the cloud is
        not specified the default cloud is used.
        If the parameter for cloud is "all" the rules and groups
        will be uploaded to all active clouds.
        This will synchronize the changes (add/delete on security
        groups, rules) made locally to the remote cloud(s).

```

18.4.8 ssh

Usage:

```

    ssh table
    ssh list [--output=OUTPUT]
    ssh cat
    ssh register NAME PARAMETERS
    ssh ARGUMENTS
        conducts a ssh login on a machine while using a set of
        registered machines specified in ~/.ssh/config

```

Arguments:

NAME	Name or ip of the machine to log in
list	Lists the machines that are registered and the commands to login to them
PARAMETERS	Register te resource and add the given parameters to the ssh config file. if the resource exists, it will be overwritten. The information will be written in ~/.ssh/config

Options:

-v	verbose mode
--output=OUTPUT	the format in which this list is given formats includes table, json, yaml, dict [default: table]
--user=USER	overwrites the username that is specified in ~/.ssh/config
--key=KEY	The keyname as defined in the key list or a location that contains a public key

Description:

```

    ssh list
        lists the hostnames that are present in the
        ~/.ssh/config file
    ssh cat
        prints the ~/.ssh/config file
    ssh table
        prints contents of the ~/.ssh/config file in table format
    ssh register NAME PARAMETERS
        registers a host i ~/.ssh/config file
        Parameters are attribute=value pairs
        Note: Note yet implemented
    ssh ARGUMENTS

```

(continues on next page)

(continued from previous page)

```

executes the ssh command with the given arguments
Example:
    ssh myhost
        conducts an ssh login to myhost if it is defined in
        ~/.ssh/config file

```

18.4.9 vbox

Usage:

```

vbox version [--output=OUTPUT]
vbox image list [--output=OUTPUT]
vbox image find KEYWORDS...
vbox image add NAME
vbox image delete NAME
vbox vm info NAME
vbox vm list [--output=OUTPUT] [-v]
vbox vm delete NAME
vbox vm ip [NAME] [--all]
vbox vm create [NAME] [--memory=MEMORY] [--image=IMAGE] [--port=PORT] [--
↪script=SCRIPT] | list)
vbox vm boot [NAME] [--memory=MEMORY] [--image=IMAGE] [--port=PORT] [--
↪script=SCRIPT] | list)
vbox vm ssh [NAME] [-e COMMAND]

```

18.4.10 vcluster

Usage:

```

vcluster create cluster CLUSTER_NAME --clusters=CLUSTERS_LIST [--
↪computers=COMPUTERS_LIST] [--debug]
vcluster destroy cluster CLUSTER_NAME
vcluster create runtime-config CONFIG_NAME PROCESS_NUM in:params out:stdout [--
↪fetch-proc-num=FETCH_PROCESS_NUM [default=1]] [--download-later [default=True]] [--
↪debug]
vcluster create runtime-config CONFIG_NAME PROCESS_NUM in:params out:file [--fetch-
↪proc-num=FETCH_PROCESS_NUM [default=1]] [--download-later [default=True]] [--debug]
vcluster create runtime-config CONFIG_NAME PROCESS_NUM in:params+file out:stdout [--
↪fetch-proc-num=FETCH_PROCESS_NUM [default=1]] [--download-later [default=True]] [--
↪debug]
vcluster create runtime-config CONFIG_NAME PROCESS_NUM in:params+file
↪out:stdout+file [--fetch-proc-num=FETCH_PROCESS_NUM [default=1]] [--download-later
↪[default=True]] [--debug]
vcluster set-param runtime-config CONFIG_NAME PARAMETER VALUE
vcluster destroy runtime-config CONFIG_NAME
vcluster list clusters [DEPTH [default:1]]
vcluster list runtime-configs [DEPTH [default:1]]
vcluster run-script --script-path=SCRIPT_PATH --job-name=JOB_NAME --vcluster-
↪name=CLUSTER_NAME --config-name=CONFIG_NAME --arguments=SET_OF_PARAMS --remote-
↪path=REMOTE_PATH --local-path=LOCAL_PATH [--argfile-path=ARGUMENT_FILE_PATH] [--
↪outfile-name=OUTPUT_FILE_NAME] [--suffix=SUFFIX] [--overwrite]

```

(continues on next page)

(continued from previous page)

```
vcluster fetch JOB_NAME
vcluster clean-remote JOB_NAME PROCESS_NUM
vcluster test-connection CLUSTER_NAME PROCESS_NUM
```

This command does some useful things.

Arguments:

FILE a file name

Options:

-f specify the file

18.4.11 vm

Usage:

```
vm ping [NAMES] [--cloud=CLOUDS] [--count=N] [--processors=PROCESSORS]
vm check [NAMES] [--cloud=CLOUDS] [--processors=PROCESSORS]
vm status [NAMES] [--cloud=CLOUDS]
vm console [NAME] [--force]
vm start [NAMES] [--cloud=CLOUD] [--dryrun]
vm stop [NAMES] [--cloud=CLOUD] [--dryrun]
vm terminate [NAMES] [--cloud=CLOUD] [--dryrun]
vm delete [NAMES] [--cloud=CLOUD] [--dryrun]
vm refresh [--cloud=CLOUDS]
vm list [NAMES]
    [--cloud=CLOUDS]
    [--output=OUTPUT]
    [--refresh]
vm boot [--name=VMNAMES]
    [--cloud=CLOUD]
    [--username=USERNAME]
    [--image=IMAGE]
    [--flavor=FLAVOR]
    [--public]
    [--secgroup=SECGROUPs]
    [--key=KEY]
    [--dryrun]
vm boot [--n=COUNT]
    [--cloud=CLOUD]
    [--username=USERNAME]
    [--image=IMAGE]
    [--flavor=FLAVOR]
    [--public]
    [--secgroup=SECGROUPS]
    [--key=KEY]
    [--dryrun]
vm run [--name=VMNAMES] [--username=USERNAME] [--dryrun] COMMAND
vm script [--name=NAMES] [--username=USERNAME] [--dryrun] SCRIPT
vm ip assign [NAMES]
    [--cloud=CLOUD]
vm ip show [NAMES]
    [--group=GROUP]
    [--cloud=CLOUD]
    [--output=OUTPUT]
    [--refresh]
```

(continues on next page)

(continued from previous page)

```

vm ip inventory [NAMES]
vm ssh [NAMES] [--username=USER]
    [--quiet]
    [--ip=IP]
    [--key=KEY]
    [--command=COMMAND]
    [--modify-knownhosts]
vm rename [OLDNAMES] [NEWNAMES] [--force] [--dryrun]
vm wait [--cloud=CLOUD] [--interval=SECONDS]
vm info [--cloud=CLOUD]
    [--output=OUTPUT]
vm username USERNAME [NAMES] [--cloud=CLOUD]
vm resize [NAMES] [--size=SIZE]

```

Arguments:

OUTPUT	the output format
COMMAND	positional arguments, the commands you want to execute on the server(e.g. <code>ls -a</code>) separated by <code>';</code> , you will get a return of executing result instead of login to the server, note that type in <code>--</code> is suggested before you input the commands
NAME	server name. By default it is set to the name of last vm from database .
NAMES	server name. By default it is set to the name of last vm from database .
KEYPAIR_NAME	Name of the vm keypair to be used to create VM. Note this is not a path to key.
NEWNAMES	New names of the VM while renaming.
OLDNAMES	Old names of the VM while renaming.

Options:

<code>--output=OUTPUT</code>	the output format [default: table]
<code>-H --modify-knownhosts</code>	Do not modify <code>~/.ssh/known_hosts</code> file when ssh'ing into a machine
<code>--username=USERNAME</code>	the username to login into the vm. If not specified it will be guessed from the image name and the cloud
<code>--ip=IP</code>	give the public ip of the server
<code>--cloud=CLOUD</code>	give a cloud to work on, if not given, selected or default cloud will be used
<code>--count=COUNT</code>	give the number of servers to start
<code>--detail</code>	for table, a brief version is used as default, use this flag to print detailed table
<code>--flavor=FLAVOR</code>	give the name or id of the flavor
<code>--group=GROUP</code>	give the group name of server
<code>--secgroup=SECGROUP</code>	security group name for the server
<code>--image=IMAGE</code>	give the name or id of the image
<code>--key=KEY</code>	specify a key to use, input a string which is the full path to the private key file
<code>--keypair_name=KEYPAIR_NAME</code>	Name of the vm keypair to be used to create VM. Note this is not a path to key.
<code>--user=USER</code>	give the user name of the server that you want to use to login
<code>--name=NAME</code>	give the name of the virtual machine
<code>--force</code>	rename/ delete vms without user's confirmation

(continues on next page)

(continued from previous page)

```
--command=COMMAND
    specify the commands to be executed
```

Description:

commands used to boot, start **or** delete servers of a cloud

vm default [options...]

Displays default parameters that are **set for** vm boot either on the default cloud **or** the specified cloud.

vm boot [options...]

Boots servers on a cloud, user may specify flavor, image .etc, otherwise default values will be used, see how to **set** default values of a cloud: cloud help

vm start [options...]

Starts a suspended **or** stopped vm instance.

vm stop [options...]

Stops a vm instance .

vm delete [options...]

Delete servers of a cloud, user may delete a server by its name **or id**, delete servers of a group **or** servers of a cloud, give prefix **and/or range** to find servers by their names. Or user may specify more options to narrow the search

vm floating_ip_assign [options...]

assign a public ip to a VM of a cloud

vm ip show [options...]

show the ips of VMs

vm ssh [options...]

login to a server **or** execute commands on it

vm **list** [options...]

same **as** command "**list vm**", please refer to it

vm status [options...]

Retrieves status of last VM booted on cloud **and** displays it.

vm refresh [--cloud=CLOUDS]

this command refreshes the data **for** virtual machines, images **and** flavors **for** the specified clouds.

vm ping [NAMES] [--cloud=CLOUDS] [--count=N] [--processors=PROCESSORS]
pings the specified virtual machines, **while** using at most N pings.
The ping **is** executed **in** parallel.

If names are specifies the ping **is** restricted to the given names **in** parameter **format**. If clouds are specified, names that are **not in** these clouds are ignored. If the name **is set in** the variables this name **is** used.

Tip:

(continues on next page)

(continued from previous page)

```

give the VM name, but in a hostlist style, which is very
convenient when you need a range of VMs e.g. sample[1-3]
=> ['sample1', 'sample2', 'sample3']
sample[1-3,18] => ['sample1', 'sample2', 'sample3', 'sample18']

```

Quoting commands:

```
cm vm login gvonlasz-004 --command="uname -a"
```

Limitations:

Azure: rename **is not** supported

18.4.12 workflow

Usage:

```

workflow refresh [--cloud=CLOUD] [-v]
workflow list [ID] [NAME] [--cloud=CLOUD] [--output=OUTPUT] [--refresh] [-v]
workflow add NAME LOCATION
workflow delete ID
workflow status [NAMES]
workflow show ID
workflow save NAME WORKFLOWSTR
workflow run NAME
workflow service start
workflow service stop
This lists out the workflows present for a cloud

```

Options:

```

--output=OUTPUT  the output format [default: table]
--cloud=CLOUD    the cloud name
--refresh        refreshes the data before displaying it
                  from the cloud

```

Examples:

```

cm workflow refresh
cm workflow list
cm workflow list --format=csv
cm workflow show 58c9552c-8d93-42c0-9dea-5f48d90a3188 --refresh
cm workflow run workflow1

```

18.5 Storage Manual Pages

18.5.1 objstorage

Usage:

```

objstorage [--service=SERVICE] create dir DIRECTORY
objstorage [--service=SERVICE] copy SOURCE DESTINATION [--recursive]
objstorage [--service=SERVICE] get SOURCE DESTINATION [--recursive]
objstorage [--service=SERVICE] put SOURCE DESTINATION [--recursive]
objstorage [--service=SERVICE] list SOURCE [--recursive] [--output=OUTPUT]
objstorage [--service=SERVICE] delete SOURCE

```

(continues on next page)

(continued from previous page)

```
objstorage [--service=SERVICE] search DIRECTORY FILENAME [--recursive] [--
↳output=OUTPUT]
```

This command does some useful things.

Arguments:

```
SOURCE      BUCKET | OBJECT  can be a source bucket or object name or file
DESTINATION  BUCKET | OBJECT  can be a destination bucket or object name or file
DIRECTORY    DIRECTORY refers to a folder or bucket on the cloud service for ex:
↳awss3
```

Options:

```
-h, --help
--service=SERVICE  specify the cloud service name like aws-s3
```

Description:

commands used to upload, download, list files on different cloud objstorage_
↳services.

```
objstorage put [options..]
Uploads the file specified in the filename to specified cloud from the_
↳SOURCEDIR.
```

```
objstorage get [options..]
Downloads the file specified in the filename from the specified cloud to_
↳the DESTDIR.
```

```
objstorage delete [options..]
Deletes the file specified in the filename from the specified cloud.
```

```
objstorage list [options..]
lists all the files from the container name specified on the specified_
↳cloud.
```

```
objstorage create dir [options..]
creates a folder with the directory name specified on the specified cloud.
```

```
objstorage search [options..]
searches for the source in all the folders on the specified cloud.
```

Example:

```
set objstorage=s3object
objstorage put SOURCE DESTINATION --recursive
is the same as
objstorage --service=s3object put SOURCE DESTINATION --recursive
```

18.5.2 storage

Usage:

```
storage [--storage=SERVICE] create dir DIRECTORY
storage [--storage=SERVICE] get SOURCE DESTINATION [--recursive]
storage [--storage=SERVICE] put SOURCE DESTINATION [--recursive]
storage [--storage=SERVICE] list SOURCE [--recursive] [--output=OUTPUT]
storage [--storage=SERVICE] delete SOURCE
storage [--storage=SERVICE] search DIRECTORY FILENAME [--recursive] [--
↳output=OUTPUT]
```

(continues on next page)

(continued from previous page)

```

storage [--storage=SERVICE] sync SOURCE DESTINATION [--name=NAME] [--async]
storage [--storage=SERVICE] sync status [--name=NAME]
storage config list [--output=OUTPUT]

```

This command does some useful things.

Arguments:

```

SOURCE          SOURCE can be a directory or file
DESTINATION     DESTINATION can be a directory or file
DIRECTORY       DIRECTORY refers to a folder on the cloud service

```

Options:

```

--storage=SERVICE  specify the cloud service name like aws or
                    azure or box or google

```

Description:

commands used to upload, download, list files on different cloud storage services.

storage put [options..]

Uploads the file specified in the filename to specified cloud from the SOURCEDIR.

storage get [options..]

Downloads the file specified in the filename from the specified cloud to the DESTDIR.

storage delete [options..]

Deletes the file specified in the filename from the specified cloud.

storage list [options..]

lists all the files from the container name specified on the specified cloud.

storage create dir [options..]

creates a folder with the directory name specified on the specified cloud.

storage search [options..]

searches for the source in all the folders on the specified cloud.

sync SOURCE DESTINATION

puts the content of source to the destination.

If --recursive is specified this is done recursively from the source

If --async is specified, this is done asynchronously

If a name is specified, the process can also be monitored with the status command by name.

If the anme is not specified all date is monitored.

sync status

The status for the asynchronous sync can be seen with this command

(continues on next page)

(continued from previous page)

```
config list
    Lists the configures storage services in the yaml file
```

Example:

```
set storage=azureblob
storage put SOURCE DESTINATION --recursive

is the same as
storage --storage=azureblob put SOURCE DESTINATION --recursive
```

18.5.3 vdir

Usage:

```
vdir mkdir DIR
vdir cd [DIR]
vdir ls [DIR]
vdir add [FILEENDPOINT] [DIR_AND_NAME]
vdir delete [DIR_OR_NAME]
vdir status [DIR_OR_NAME]
vdir get NAME DESTINATION
```

Arguments:

DIR	a directory name
FILEENDPOINT	location of file
DIR_AND_NAME	path of file link
DIR_OR_NAME	name of directory or link
DESTINATION	directory to download to
NAME	name of link

Options:

-f	specify the file
----	------------------

Description:

A virtual directory **is** explained **in** our NIST documentation. It contains a number of links that point to other storage services on which the file **is** stored. The links include the provider, the name of the provider **and** its type are identified **in** the ~/.cloudmesh4.yaml file.

the location **is** identified **as**

```
{provider}:{directory}/{filename}
```

A cloudmesh directory can be used to uniquely specify the file:

```
cm:
  name: the unique name of the file
  kind: vdir
  cloud: local
  directory: directory
  filename: filename
  directory: directory
  provider: provider
  created: date
```

(continues on next page)

(continued from previous page)

```
modified: date
```

```
vdir get NAME DESTINATION
```

locates the file **with** the name on a storage provider,
and fetches it **from there**.

DATABASE OBJECTS

19.1 Cloudmesh Database

Cloudmesh has a database in which a local copy of information about objects that are stored in the cloud is maintained. The objects contain all information of the cloud that can be retrieved with the raw provider but are enhanced with a cloudmesh attribute dict. Potential security related attributes, will however be removed from it so they are not stored in the database.

This dict looks like

```
"cm": {
  "kind": the kind of the provider
  "cloud": the cloud or service name, will be renamed to service in future)
  "name": a unique name of the object
}
```

We list in the next section examples of such data objects

19.1.1 Virtual Machines

Openstack

The compute provider kind is `openstack`. The Provider is located at

- <https://github.com/cloudmesh/cloudmesh-cloud/blob/master/cloudmesh/compute/virtualbox/Provider.py>

This provider should not be used, but you need to use the general provider at

- <https://github.com/cloudmesh/cloudmesh-cloud/blob/master/cloudmesh/compute/vm/Provider.py>

Flavor

```
{
  "_id" : ObjectId("5ca798a7dc64f18b19e644e1"),
  "id" : "1",
  "name" : "ml.tiny",
  "ram" : 512,
  "disk" : 1,
  "bandwidth" : null,
  "price" : 0.0,
  "extra" : {},
  "vcpus" : 1,
  "ephemeral_disk" : 0,
  "swap" : "",
  "cm" : {
    "kind" : "flavor",
    "driver" : "openstack",
    "cloud" : "chameleon",
    "created" : "2019-04-05 18:04:23.621043",
    "name" : "ml.tiny",
    "collection" : "chameleon-flavor",
    "modified" : "2019-04-06 06:50:23.894035"
  },
  "updated" : "2019-04-06 06:50:23.867695"
}
```

Image

```
{
  "_id" : ObjectId("5ca798abdc64f18b19e644e9"),
  "id" : "4c8e6dac-97f0-4224-b7a2-0daef96b5c9f",
  "name" : "CC-Ubuntu16.04",
  "extra" : {
    "visibility" : null,
    "updated" : "2019-03-25T21:21:06Z",
    "created" : "2019-03-25T21:20:51Z",
    "status" : "ACTIVE",
    "progress" : 100,
    "metadata" : {
      "build-repo-commit" : "4ba0beb418de52f0d3bf93a94392b662d653c073",
      "build-variant" : "base",
      "build-os-base-image-revision" : "20190325",
      "build-os" : "ubuntu-xenial",
      "build-tag" : "jenkins-cc-ubuntu16.04-builder-34",
      "build-repo" : "https://github.com/ChameleonCloud/CC-Ubuntu16.04"
    },
    "os_type" : null,
    "serverId" : null,
    "minDisk" : 0,
    "minRam" : 0
  },
  "cm" : {
    "kind" : "image",
    "driver" : "openstack",
    "cloud" : "chameleon",

```

(continues on next page)

(continued from previous page)

```

    "created" : "2019-04-05 18:04:27.417958",
    "updated" : "2019-03-25T21:21:06Z",
    "name" : "CC-Ubuntu16.04",
    "collection" : "chameleon-image",
    "modified" : "2019-04-06 06:50:33.747656"
  }
}

```

VM

```

{
  "_id" : ObjectId("5ca798acdc64f18b19e6454e"),
  "id" : "f531d2df-c472-4b32-8239-0e3969d33ebb",
  "name" : "exp-grp-gregor-vm-1",
  "state" : "running",
  "public_ips" : [],
  "private_ips" : [
    "192.168.0.249"
  ],
  "size" : null,
  "created_at" : ISODate("2019-04-01T11:05:56.000-04:00"),
  "image" : null,
  "extra" : {
    "addresses" : {
      "CH-819337-net" : [
        {
          "OS-EXT-IPS-MAC:mac_addr" : "fa:16:3e:9d:ca:c2",
          "version" : 4,
          "addr" : "192.168.0.249",
          "OS-EXT-IPS:type" : "fixed"
        }
      ]
    },
    "hostId" : "64472a496451a2d599c215a8e86275191c9e3fb9d53790de35bbb6dc",
    "access_ip" : "",
    "access_ipv6" : "",
    "tenantId" : "CH-819337",
    "userId" : "tg455498",
    "imageId" : "4c8e6dac-97f0-4224-b7a2-0daef96b5c9f",
    "flavorId" : "3",
    "uri" : "http://openstack.tacc.chameleoncloud.org:8774/v2/CH-819337/servers/
↪f531d2df-c472-4b32-8239-0e3969d33ebb",
    "service_name" : "nova",
    "metadata" : {},
    "password" : null,
    "created" : "2019-04-01T15:05:56Z",
    "updated" : "2019-04-01T15:06:06Z",
    "key_name" : "gregor",
    "disk_config" : "MANUAL",
    "config_drive" : "",
    "availability_zone" : "nova",
    "volumes_attached" : [],
    "task_state" : null,
    "vm_state" : "active",

```

(continues on next page)

(continued from previous page)

```

        "power_state" : 1,
        "progress" : 0,
        "fault" : null
    },
    "cm" : {
        "kind" : "node",
        "driver" : "openstack",
        "cloud" : "chameleon",
        "updated" : "2019-04-06 06:50:35.592158",
        "name" : "exp-grp-gregor-vm-1",
        "created" : "2019-04-05 18:04:28.376784",
        "collection" : "chameleon-node",
        "modified" : "2019-04-06 06:50:35.596479"
    }
}

```

19.1.2 Azure AzProvider

The compute provider kind is azure. The Provider is located at

- <https://github.com/cloudmesh/cloudmesh-cloud/blob/master/cloudmesh/compute/virtualbox/Provider.py>

This provider should not be used, but you need to use the general provider at

- <https://github.com/cloudmesh/cloudmesh-cloud/blob/master/cloudmesh/compute/azure/AzProvider.py>

Flavor

Image

VM

```

{
    "_id" : ObjectId("5cbe0513b6ac5a154ef64a26"),
    "additionalCapabilities" : null,
    "availabilitySet" : null,
    "diagnosticsProfile" : null,
    "hardwareProfile" : {
        "vmSize" : "Standard_DS1_v2"
    },
    "id" : null,
    "identity" : null,
    "instanceView" : null,
    "licenseType" : null,
    "location" : "eastus",
    "name" : "testvm1",
    "networkProfile" : {
        "networkInterfaces" : [
            {
                "id" : null,
                "primary" : null,
                "resourceGroup" : "test"
            }
        ]
    }
}

```

(continues on next page)

(continued from previous page)

```

    },
    "osProfile" : {
        "adminPassword" : null,
        "adminUsername" : "ubuntu",
        "allowExtensionOperations" : true,
        "computerName" : "testvm1",
        "customData" : null,
        "linuxConfiguration" : {
            "disablePasswordAuthentication" : true,
            "provisionVmAgent" : true,
            "ssh" : {
                "publicKeys" : [
                    {
                        "keyData" : "ssh-rsa ...."
                        "path" : "/home/ubuntu/.ssh/authorized_keys"
                    }
                ]
            }
        },
        "requireGuestProvisionSignal" : true,
        "secrets" : [],
        "windowsConfiguration" : null
    },
    "plan" : null,
    "provisioningState" : "Succeeded",
    "resourceGroup" : "test",
    "resources" : null,
    "storageProfile" : {
        "dataDisks" : [],
        "imageReference" : {
            "id" : null,
            "offer" : "UbuntuServer",
            "publisher" : "Canonical",
            "sku" : "18.04-LTS",
            "version" : "latest"
        },
        "osDisk" : {
            "caching" : "ReadWrite",
            "createOption" : "FromImage",
            "diffDiskSettings" : null,
            "diskSizeGb" : null,
            "encryptionSettings" : null,
            "image" : null,
            "managedDisk" : {
                "id" : null,
                "resourceGroup" : "test",
                "storageAccountType" : null
            },
            "name" : "testvm1_OsDisk_1_a6a6a6a7639468d88e7b018385e225f",
            "osType" : "Linux",
            "vhd" : null,
            "writeAcceleratorEnabled" : null
        }
    },
    "tags" : {},
    "type" : "Microsoft.Compute/virtualMachines",
    "vmId" : "aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa",

```

(continues on next page)

(continued from previous page)

```
"zones" : null,
"cm" : {
  "kind" : "node",
  "driver" : "azure",
  "cloud" : "az",
  "name" : "testvm1",
  "collection" : "az-node",
  "created" : "2019-04-22 18:16:51.552324",
  "modified" : "2019-04-22 18:16:51.552324"
}
}
```

19.1.3 Azure MS Azure Library Provider

The compute provider kind is MISSING. The Provider is located at

-

This provider should not be used, but you need to use the general provider at

- <https://github.com/cloudmesh/cloudmesh-cloud/blob/master/cloudmesh/compute/azure/AzProvider.py>

Flavor

Image

VM

19.1.4 AWS Libcloud Provider

The compute provider kind is aws. The Provider is located at

- <https://github.com/cloudmesh/cloudmesh-cloud/blob/master/cloudmesh/compute/virtualbox/Provider.py>

This provider should not be used, but you need to use the general provider at

- <https://github.com/cloudmesh/cloudmesh-cloud/blob/master/cloudmesh/compute/libcloud/Provider.py>

Flavor

```
{'bandwidth': None,
 'cm': {'cloud': 'aws',
        'created': '2019-04-25 11:01:21.939851',
        'driver': 'aws',
        'kind': 'flavor',
        'name': 't2.micro'},
 'disk': 0,
 'id': 't2.micro',
 'name': 't2.micro',
 'price': 0.012,
 'ram': 1024,
 'updated': '2019-04-25 11:01:21.939851'}
```


Image

```
{'cm': {'cloud': 'aws',
        'created': '2019-04-25 11:10:31.912143',
        'driver': 'aws',
        'kind': 'image',
        'name': 'memsql-cloudformation_6.7.11-5d2517b77a_1.5.3_1.0.6 '
                '20190208-212347',
        'updated': '2019-04-25 11:10:31.912155'},
'id': 'ami-0496a382c868777a4',
'name': 'memsql-cloudformation_6.7.11-5d2517b77a_1.5.3_1.0.6 '
        '20190208-212347'}
```

VM

```
{'cm': {'cloud': 'aws',
        'created': '2019-04-25 06:47:39+00:00',
        'driver': 'aws',
        'kind': 'node',
        'name': 't0',
        'updated': '2019-04-25 10:55:45.394053'},
'created_at': datetime.datetime(2019, 4, 25, 6, 47, 39, tzinfo=<libcloud.utils.
→iso8601.Utc object at 0x1119225c0>),
'id': 'i-032d5c07fcfaf5b8b',
'image': None,
'name': 't0',
'private_ips': [],
'public_ips': [],
'size': None,
'state': 'running'}
```

19.1.5 AWS Boto3 Provider

The compute provider kind is MISSING. The Provider is located at

-

This provider should not be used, but you need to use the general provider at

- <https://github.com/cloudmesh/cloudmesh-cloud/blob/master/cloudmesh/compute/azure/AzProvider.py>

Flavor

Image

VM

19.1.6 Storage

AwsS3

The storage provider kind is awss3. The Provider is located at

- <https://github.com/cloudmesh/cloudmesh-storage/tree/master/cloudmesh/storage/provider/awss3>

This provider should not be used, but you need to use the general provider at

- <https://github.com/cloudmesh/cloudmesh-storage/blob/master/cloudmesh/storage/Provider.py>

Box

The storage provider kind is `box`. The Provider is located at

- <https://github.com/cloudmesh/cloudmesh-storage/blob/master/cloudmesh/storage/provider/box/Provider.py>

This provider should not be used, but you need to use the general provider at

- <https://github.com/cloudmesh/cloudmesh-storage/blob/master/cloudmesh/storage/Provider.py>

Directory

```
{'_object_id': '71983743142',
  'cm': {
    'cloud': 'box',
    'kind': 'storage',
    'name': 'test01'
  },
  'etag': '0',
  'id': '71983743142',
  'name': 'test01',
  'sequence_id': '0',
  'type': 'folder'}
```

File

```
{'_object_id': '432543586295',
  'cm': {
    'cloud': 'box',
    'kind': 'storage',
    'name': 'test.txt'
  },
  'etag': '285',
  'id': '432543586295',
  'name': 'test.txt',
  'sequence_id': '285',
  'sha1': 'bca20547e94049e1ffea27223581c567022a5774',
  'type': 'file'}}
```

Azure Blob

The storage provider kind is `azureblob`. The Provider is located at

- <https://github.com/cloudmesh/cloudmesh-storage/blob/master/cloudmesh/storage/provider/azureblob/Provider.py>

This provider should not be used, but you need to use the general provider at

- <https://github.com/cloudmesh/cloudmesh-storage/blob/master/cloudmesh/storage/Provider.py>

Directory

Note that directory in Azure Blob storage is a virtual folder. An empty directory cannot be created and hence cloudmesh-storage creates a default marker file `dummy.txt` to create the directory.

```
{'cm': {'cloud': 'azureblob',
        'created': '2',
        'kind': 'storage',
        'name': 'dirtest/dummy.txt',
        'size': 1,
        'status': 'exists',
        'updated': '2'},
 'content': None,
 'deleted': False,
 'metadata': None,
 'name': 'dirtest/dummy.txt',
 'properties': {'append_blob_committed_block_count': None,
                 'blob_tier': None,
                 'blob_tier_change_time': None,
                 'blob_tier_inferred': False,
                 'blob_type': 'BlockBlob',
                 'content_length': 1,
                 'content_range': None,
                 'deleted_time': None,
                 'etag': '0x8D6CA68C2D61B73',
                 'page_blob_sequence_number': None,
                 'remaining_retention_days': None,
                 'server_encrypted': True},
 'snapshot': None}
```

File

```
{'cm': {'cloud': 'azureblob',
        'created': '2',
        'kind': 'storage',
        'name': 'a/a/al.txt',
        'size': 19,
        'status': 'exists',
        'updated': '2'},
 'content': None,
 'deleted': False,
 'metadata': None,
 'name': 'a/a/al.txt',
 'properties': {'append_blob_committed_block_count': None,
                 'blob_tier': None,
                 'blob_tier_change_time': None,
                 'blob_tier_inferred': False,
                 'blob_type': 'BlockBlob',
                 'content_length': 19,
                 'content_range': None,
                 'deleted_time': None,
                 'etag': '0x8D6CA57263B4AEA',
                 'page_blob_sequence_number': None,
                 'remaining_retention_days': None,
                 'server_encrypted': True},
 'snapshot': None}
```

AWSS3 the one from cloudmesh-cloud

The storage provider kind is `awss3`. The Provider is located at

- <https://github.com/cloudmesh/cloudmesh-storage/blob/master/cloudmesh/storage/provider/awss3/Provider.py>

This provder should not be used, but you need to use the general provider at

- <https://github.com/cloudmesh/cloudmesh-storage/blob/master/cloudmesh/storage/Provider.py>

Directory

File

AWSS3 the one from cloudmesh-objstore

It is unclear what the difference to AWSS3 the one from cloudmesh-cloud is. Please explain. If it's the same, let us know and we should merge.

The storage provider kind is `objstorage`. The Provider is located at

- UNCLER IF IT DUPLICATES `awss3`

This provder should not be used, but you need to use the general provider at

- <https://github.com/cloudmesh/cloudmesh-storage/blob/master/cloudmesh/storage/Provider.py>

Directory

File

Google Drive

The storage provider kind is `gdrive`. The Provider is located at

- <https://github.com/cloudmesh/cloudmesh-storage/blob/master/cloudmesh/storage/provider/gdrive/Provider.py>

This provder should not be used, but you need to use the general provider at

- <https://github.com/cloudmesh/cloudmesh-storage/blob/master/cloudmesh/storage/Provider.py>

Directory

File

Local

This has not been tested yet, so be careful as it could remove local dir trees. We may need to add an option `-force` for this provider and always ask if we want to delete the files while showing them first. This could even be a reason to introduce it in all providers.

The storage provider kind is `local`. The Provider is located at

- <https://github.com/cloudmesh/cloudmesh-storage/blob/master/cloudmesh/storage/provider/local/Provider.py>

This provder should not be used, but you need to use the general provider at

- <https://github.com/cloudmesh/cloudmesh-storage/blob/master/cloudmesh/storage/Provider.py>

Directory

File

19.1.7 Workflow

```
/* 2 */
{
  "_id" : ObjectId("5cbc414e20b39a69d014efec"),
  "name" : "b",
  "dependencies" : [
    "c"
  ],
  "workflow" : "workflow",
  "cm" : {
    "kind" : "flow",
    "cloud" : "workflow",
    "name" : "b",
    "collection" : "workflow-flow",
    "created" : "2019-04-21 10:09:18.887115",
    "modified" : "2019-04-21 10:09:18.887115"
  },
  "kind" : "flow",
  "cloud" : "workflow"
}
```

19.1.8 EMR

Cluster Listing

```
{
  "_id" : ObjectId("5cae2f2176cd490cac627e04"),
  "cm" : {
    "cloud" : "aws",
    "kind" : "emr cluster list",
    "name" : "emr",
    "collection" : "aws-emr cluster list",
    "created" : "2019-04-10 18:00:01.850862",
    "modified" : "2019-04-10 18:00:55.341485"
  },
  "data" : [
    {
      "Id" : "j-XXXXXXXXXXXX",
      "Name" : "NAME",
      "Status" : {
        "State" : "TERMINATED",
        "StateChangeReason" : {
          "Code" : "USER_REQUEST",
          "Message" : "Terminated by user request"
        }
      },
      "Timeline" : {
```

(continues on next page)

(continued from previous page)

```

        "CreationDateTime" : ISODate("2019-04-
↪04T01:38:26.595Z"),
        "EndTime" : ISODate("2019-04-04T01:43:53.
↪907Z")
    },
    },
    "NormalizedInstanceHours" : 0
},
...
]
}

```

Instance Listing

```

{
  "_id" : ObjectId("5cae36b976cd490de715a25a"),
  "cm" : {
    "cloud" : "aws",
    "kind" : "emr instance list",
    "name" : "j-XXXXXXXXXXXX",
    "collection" : "aws-emr instance list",
    "created" : "2019-04-10 18:32:25.886579",
    "modified" : "2019-04-10 18:32:39.244152"
  },
  "data" : [
    {
      "Id" : "ci-XXXXXXXXXXXX",
      "Ec2InstanceId" : "i-XXXXXXXXXXXX",
      "PublicDnsName" : "ec2-54-193-70-173.us-west-1.compute.
↪amazonaws.com",
      "PublicIpAddress" : "54.193.70.173",
      "PrivateDnsName" : "ip-172-31-10-13.us-west-1.compute.internal
↪",
      "PrivateIpAddress" : "172.31.10.13",
      "Status" : {
        "State" : "TERMINATED",
        "StateChangeReason" : {
          "Code" : "INSTANCE_FAILURE",
          "Message" : "Instance was terminated."
        },
        "Timeline" : {
          "CreationDateTime" : ISODate("2019-04-
↪04T01:39:20.733Z"),
          "ReadyDateTime" : ISODate("2019-04-
↪04T01:42:11.677Z"),
          "EndTime" : ISODate("2019-04-04T01:43:53.
↪810Z")
        }
      },
      "InstanceGroupId" : "ig-XXXXXXXXXXXX",
      "Market" : "ON_DEMAND",
      "InstanceType" : "m4.xlarge",
      "EbsVolumes" : [
        {
          "Device" : "/dev/sdb",

```

(continues on next page)

(continued from previous page)

```

        "VolumeId" : "vol-0255d0ae88dbbe78f"
      },
      {
        "Device" : "/dev/sdc",
        "VolumeId" : "vol-0092cf772c4cb49d6"
      }
    ]
  }
  ...
]
}

```

Cluster Description

```

{
  "_id" : ObjectId("5cae3acd76cd490f74997607"),
  "cm" : {
    "cloud" : "aws",
    "kind" : "emr cluster description",
    "name" : "j-2KJT8GO4RV4VR",
    "collection" : "aws-emr cluster description",
    "created" : "2019-04-10 18:49:49.923422",
    "modified" : "2019-04-10 18:49:49.923422"
  },
  "data" : {
    "Id" : "j-XXXXXXXXXXXX",
    "Name" : "NAME",
    "Status" : {
      "State" : "TERMINATED",
      "StateChangeReason" : {
        "Code" : "USER_REQUEST",
        "Message" : "Terminated by user request"
      },
      "Timeline" : {
        "CreationDateTime" : ISODate("2019-04-04T01:38:26.595Z"),
        "EndDateTime" : ISODate("2019-04-04T01:43:53.907Z")
      }
    },
    "Ec2InstanceAttributes" : {
      "RequestedEc2SubnetIds" : [ ],
      "Ec2AvailabilityZone" : "us-west-1c",
      "RequestedEc2AvailabilityZones" : [ ],
      "IamInstanceProfile" : "EMR_EC2_DefaultRole",
      "EmrManagedMasterSecurityGroup" : "sg-XXXXXXXXXXXX",
      "EmrManagedSlaveSecurityGroup" : "sg-XXXXXXXXXXXX"
    },
    "InstanceCollectionType" : "INSTANCE_GROUP",
    "ReleaseLabel" : "emr-5.22.0",
    "AutoTerminate" : false,
    "TerminationProtected" : false,
    "VisibleToAllUsers" : true,
    "Applications" : [
      {
        "Name" : "Spark",

```

(continues on next page)

(continued from previous page)

```

        "Version" : "2.4.0"
      },
      {
        "Name" : "Hadoop",
        "Version" : "2.8.5"
      }
    ],
    "Tags" : [ ],
    "ServiceRole" : "EMR_DefaultRole",
    "NormalizedInstanceHours" : 0,
    "MasterPublicDnsName" : "ec2-54-193-70-173.us-west-1.compute.
↪amazonaws.com",
    "Configurations" : [ ],
    "ScaleDownBehavior" : "TERMINATE_AT_TASK_COMPLETION",
    "KerberosAttributes" : {
      }
    }
  }
}

```

Copy File Request

```

{
  "_id" : ObjectId("5cae3c7576cd49129fde9fd7"),
  "cm" : {
    "cloud" : "aws",
    "kind" : "emr copy file request",
    "name" : "test.md",
    "collection" : "aws-emr copy file request",
    "created" : "2019-04-10 18:56:53.568585",
    "modified" : "2019-04-10 18:56:53.568585"
  },
  "data" : {
    "StepIds" : [
      "s-XXXXXXXXXXXX"
    ],
    "ResponseMetadata" : {
      "RequestId" : "077a0182-5bc2-11e9-a7ff-118a03244614",
      "HTTPStatusCode" : 200,
      "HTTPHeaders" : {
        "x-amzn-requestid" : "077a0182-5bc2-11e9-a7ff-
↪118a03244614",
        "content-type" : "application/x-amz-json-1.1",
        "content-length" : "30",
        "date" : "Wed, 10 Apr 2019 18:54:11 GMT"
      },
      "RetryAttempts" : 0
    }
  }
}

```


File Upload

```
{
  "_id" : ObjectId("5cae3c3576cd49120a3a0513"),
  "cm" : {
    "cloud" : "aws",
    "kind" : "emr file upload",
    "name" : "test.md",
    "collection" : "aws-emr file upload",
    "created" : "2019-04-10 18:55:49.612897",
    "modified" : "2019-04-10 18:55:49.612897"
  },
  "data" : {
    "file" : "LICENSE",
    "bucket" : "BUCKET NAME",
    "bucketname" : "FILE NAME"
  }
}
```

Run File Request

```
{
  "_id" : ObjectId("5cae3c8e76cd4913333930cc"),
  "cm" : {
    "cloud" : "aws",
    "kind" : "emr run file request",
    "name" : "main.py",
    "collection" : "aws-emr run file request",
    "created" : "2019-04-10 18:57:18.893398",
    "modified" : "2019-04-10 18:57:18.893398"
  },
  "data" : {
    "StepIds" : [
      "s-XXXXXXXXXXXXX"
    ],
    "ResponseMetadata" : {
      "RequestId" : "16944707-5bc2-11e9-ab7c-333f036e49a6",
      "HTTPStatusCode" : 200,
      "HTTPHeaders" : {
        "x-amzn-requestid" : "16944707-5bc2-11e9-ab7c-
↪333f036e49a6",
        "content-type" : "application/x-amz-json-1.1",
        "content-length" : "30",
        "date" : "Wed, 10 Apr 2019 18:54:36 GMT"
      },
      "RetryAttempts" : 0
    }
  }
}
```

Start Cluster Request

```
{
  "_id" : ObjectId("5cae3ba776cd4910291c2bbf"),
```

(continues on next page)

(continued from previous page)

```

    "cm" : {
      "cloud" : "aws",
      "kind" : "emr start cluster request",
      "name" : "temp12",
      "collection" : "aws-emr start cluster request",
      "created" : "2019-04-10 18:53:27.965266",
      "modified" : "2019-04-10 18:53:27.965266"
    },
    "data" : {
      "cluster" : "j-XXXXXXXXXXXXXX",
      "name" : "NAME"
    }
  }
}

```

Stop Cluster Request

```

{
  "_id" : ObjectId("5cae3c9c76cd4913c74b5060"),
  "cm" : {
    "cloud" : "aws",
    "kind" : "emr stop cluster request",
    "name" : "j-XXXXXXXXXXXXXX",
    "collection" : "aws-emr stop cluster request",
    "created" : "2019-04-10 18:57:32.779245",
    "modified" : "2019-04-10 18:57:32.779245"
  },
  "data" : {
    "name" : "j-XXXXXXXXXXXXXX"
  }
}

```

Step List

```

{
  "_id" : ObjectId("5cae3ac276cd490f2ae1c8b7"),
  "cm" : {
    "cloud" : "aws",
    "kind" : "emr step list",
    "name" : "j-2KJT8GO4RV4VR",
    "collection" : "aws-emr step list",
    "created" : "2019-04-10 18:49:38.921115",
    "modified" : "2019-04-10 18:49:38.921115"
  },
  "data" : [
    {
      "Id" : "s-XXXXXXXXXXXXXX",
      "Name" : "Run main.py",
      "Config" : {
        "Jar" : "command-runner.jar",
        "Properties" : {

        },
        "Args" : [

```

(continues on next page)

(continued from previous page)

```

        "spark-submit",
        "s3://BUCKET/FILENAME.py"
    ],
    },
    "ActionOnFailure" : "CANCEL_AND_WAIT",
    "Status" : {
        "State" : "CANCELLED",
        "StateChangeReason" : {
            "Message" : "Job terminated"
        },
        "Timeline" : {
            "CreationDateTime" : ISODate("2019-04-
↪04T01:42:11.574Z")
        }
    }
    },
    ...
]
}

```

19.1.9 HPC

Batch

Queue

Job

19.1.10 Keys

```

{
    "_id" : ObjectId("5ca79c92dc64f1905d924234"),
    "name" : "gregor",
    "fingerprint" : "aa:aa:bb:11:22:33:88:98:13:74:8a:3b:6a:5a:b2:5d",
    "public_key" : "ssh-rsa xxxxxxxxx ... gregor@nowhere test",
    "private_key" : null,
    "extra" : {},
    "cm" : {
        "kind" : "key",
        "driver" : "openstack",
        "cloud" : "chameleon",
        "name" : "gregor",
        "collection" : "chameleon-key",
        "created" : "2019-04-05 18:21:06.898856",
        "modified" : "2019-04-06 06:50:30.975625"
    }
}

```


20.1 Benchmarks

Put your benchmarks in separate files into this directory.

20.2 AWS EC2 VM Management

Benchmark results for AWS EC2 File vm management under cloudmesh-storage.

PING ms : 19 DOWNLOAD Mbps : 54.05 UPLOAD Mbps : 5.90

Machine Attribute	Time/s
mac_version	10.14.2
machine	('x86_64',)
node	('hyspocMacBookPro.local',)
platform	Darwin-18.2.0-x86_64-i386-64bit
processor	('i386',)
processors	Darwin
python	3.7.2 (default, Feb 8 2019, 11:44:32)
	[Clang 10.0.0 (clang-1000.11.45.5)]

(continues on next page)

(continued from previous page)

```

| release          | ('18.2.0',) |
↪
| sys              | darwin      |
↪
| system           | Darwin      |
↪
| user             | hyspoc      |
↪
| version           | Darwin Kernel Version 18.2.0: Mon Nov 12 20:24:46 PST 2018;
↪root:xnu-4903.231.4~2/RELEASE_X86_64 |
| win_version      |
↪
+-----+
↪-----+
+-----+-----+-----+-----+-----+-----+
↪+-----+
| timer            | time | node | system | mac_version |
↪| win_version |
+-----+-----+-----+-----+-----+-----+
↪+-----+
| cms vm boot dryrun | 1.88 | ('hyspocMacBookPro.local',) | Darwin | 10.14.2 |
↪|
| cms vm boot       | 4.28 | ('hyspocMacBookPro.local',) | Darwin | 10.14.2 |
↪|
| cms vm list        | 4.24 | ('hyspocMacBookPro.local',) | Darwin | 10.14.2 |
↪|
| cms vm status      | 1.82 | ('hyspocMacBookPro.local',) | Darwin | 10.14.2 |
↪|
| cms vm stop dryrun | 1.83 | ('hyspocMacBookPro.local',) | Darwin | 10.14.2 |
↪|
| cms vm stop        | 2.88 | ('hyspocMacBookPro.local',) | Darwin | 10.14.2 |
↪|
| cms vm ping        | 3.84 | ('hyspocMacBookPro.local',) | Darwin | 10.14.2 |
↪|
| cms vm check       | 3.32 | ('hyspocMacBookPro.local',) | Darwin | 10.14.2 |
↪|
| cms vm run dryrun  | 1.83 | ('hyspocMacBookPro.local',) | Darwin | 10.14.2 |
↪|
| cms vm script dryrun | 1.82 | ('hyspocMacBookPro.local',) | Darwin | 10.14.2 |
↪|
| cms vm start dryrun | 1.82 | ('hyspocMacBookPro.local',) | Darwin | 10.14.2 |
↪|
| cms vm start       | 2.82 | ('hyspocMacBookPro.local',) | Darwin | 10.14.2 |
↪|
| cms vm delete dryrun | 1.84 | ('hyspocMacBookPro.local',) | Darwin | 10.14.2 |
↪|
| cms vm terminate dryrun | 1.82 | ('hyspocMacBookPro.local',) | Darwin | 10.14.2 |
↪|
| cms vm terminate   | 2.31 | ('hyspocMacBookPro.local',) | Darwin | 10.14.2 |
↪|
| cms vm delete      | 2.31 | ('hyspocMacBookPro.local',) | Darwin | 10.14.2 |
↪|
+-----+-----+-----+-----+-----+-----+
↪+-----+

```

20.3 AWS S3 File Storage

Benchmark results for AWS S3 File storage under cloudmesh-storage.

PING ms : 3 DOWNLOAD Mbps : 46.86 UPLOAD Mbps : 35.13

Machine	Attribute	Time/s			
mac_version					
machine	AMD64				
node	DESKTOP-CNS55VM				
platform	Windows-10-10.0.17134-SP0				
processor	Intel64 Family 6 Model 78 Stepping 3, GenuineIntel				
processors	Windows				
release	10				
sys	win32				
system	Windows				
version	10.0.17134				
win_version	('10', '10.0.17134', 'SP0', 'Multiprocessor Free')				
timer	time	node	system	mac_version	
win_version					
PUT file	0.29	DESKTOP-CNS55VM	Windows		
↳('10', '10.0.17134', 'SP0',					
↳'Multiprocessor Free')					
GET file	0.35	DESKTOP-CNS55VM	Windows		
↳('10', '10.0.17134', 'SP0',					
↳'Multiprocessor Free')					
LIST Directory	3.52	DESKTOP-CNS55VM	Windows		
↳('10', '10.0.17134', 'SP0',					
↳'Multiprocessor Free')					
CREATE DIR	0.36	DESKTOP-CNS55VM	Windows		
↳('10', '10.0.17134', 'SP0',					
↳'Multiprocessor Free')					
SEARCH file	1.08	DESKTOP-CNS55VM	Windows		
↳('10', '10.0.17134', 'SP0',					
↳'Multiprocessor Free')					
DELETE Directory	0.49	DESKTOP-CNS55VM	Windows		
↳('10', '10.0.17134', 'SP0',					
↳'Multiprocessor Free')					
PUT Directory --recursive	32.34	DESKTOP-CNS55VM	Windows		
↳('10', '10.0.17134', 'SP0',					
↳'Multiprocessor Free')					
GET Directory --recursive	35.97	DESKTOP-CNS55VM	Windows		
↳('10', '10.0.17134', 'SP0',					

(continues on next page)

(continued from previous page)

↪ 'Multiprocessor Free')					
DELETE Sub-directory	0.64	DESKTOP-CNS55VM	Windows		
↪ ('10', '10.0.17134', 'SP0',					
↪ 'Multiprocessor Free')					
LIST Directory --recursive	11.55	DESKTOP-CNS55VM	Windows		
↪ ('10', '10.0.17134', 'SP0',					
↪ 'Multiprocessor Free')					
LIST Sub-directory --recursive	1.15	DESKTOP-CNS55VM	Windows		
↪ ('10', '10.0.17134', 'SP0',					
↪ 'Multiprocessor Free')					
SEARCH file --recursive	1.63	DESKTOP-CNS55VM	Windows		
↪ ('10', '10.0.17134', 'SP0',					
↪ 'Multiprocessor Free')					
SEARCH file under a sub-dir --r	0.31	DESKTOP-CNS55VM	Windows		
↪ ('10', '10.0.17134', 'SP0',					
↪ 'Multiprocessor Free')					
SEARCH file under root dir --r	1.53	DESKTOP-CNS55VM	Windows		
↪ ('10', '10.0.17134', 'SP0',					
↪ 'Multiprocessor Free')					
+-----+-----+-----+-----+-----+					
↪-----+					

20.4 Azure Blob Storage

```
Internet Speedtest:
ping: 11 ms
download: 129.68 Mbps
upload: 11.73 Mbps
```

Benchmark results for Azure Blob storage under cloudmesh-storage.

```
# #####
# Benchmark results for 'azureblob' Storage
# #####

+-----+-----+
| Machine Attribute | Time/s |
+-----+-----+
| mac_version      |         |
| machine          | ('x86_64',) |
| node            | ('TESTUSER-VirtualBox',) |
| platform        | Linux-4.15.0-47-generic-x86_64-with-debian-buster-sid |
| processor       | ('x86_64',) |
| processors      | Linux |
| python          | 3.7.2 (default, Feb 11 2019, 00:01:16) |
|                 | [GCC 7.3.0] |
| release         | ('4.15.0-47-generic',) |
```

(continues on next page)


```

| sys | linux |
| system | Linux |
| user | TESTUSER |
| version | #50-Ubuntu SMP Wed Mar 13 10:44:52 UTC 2019 |
| win_version |
+-----+-----+
+-----+-----+
| timer | time | node | system | mac_ |
| version | win_version |
+-----+-----+
+-----+-----+
| PUT file | 0.41 | ('TESTUSER-VirtualBox',) | Linux |
| GET file | 0.18 | ('TESTUSER-VirtualBox',) | Linux |
| LIST Directory | 0.44 | ('TESTUSER-VirtualBox',) | Linux |
| CREATE DIR | 0.72 | ('TESTUSER-VirtualBox',) | Linux |
| SEARCH file | 0.42 | ('TESTUSER-VirtualBox',) | Linux |
| DELETE Directory | 0.49 | ('TESTUSER-VirtualBox',) | Linux |
| PUT Directory --recursive | 3.69 | ('TESTUSER-VirtualBox',) | Linux |
| GET Directory --recursive | 1.78 | ('TESTUSER-VirtualBox',) | Linux |
| DELETE Sub-directory | 0.79 | ('TESTUSER-VirtualBox',) | Linux |
| LIST Directory --recursive | 0.5 | ('TESTUSER-VirtualBox',) | Linux |
| LIST Sub-directory --recursive | 0.5 | ('TESTUSER-VirtualBox',) | Linux |
| SEARCH file --recursive | 0.49 | ('TESTUSER-VirtualBox',) | Linux |
| SEARCH file under a sub-dir --r | 0.44 | ('TESTUSER-VirtualBox',) | Linux |
| SEARCH file under root dir --r | 0.48 | ('TESTUSER-VirtualBox',) | Linux |
+-----+-----+
+-----+-----+

```

20.7

```
+-----+-----+ | Machine Ar-  
ribute | Time/s | +-----+  
| mac_version | 10.13.6 || machine | x86_64 || node | KeliFinsMacBook || platform | Darwin-17.7.0-x86_64-i386-
```

```

64bit || processor | i386 || processors | Darwin || release | 17.7.0 || sys | darwin || system | Darwin || user | alan12fine ||
version | Darwin Kernel Version 17.7.0: Thu Dec 20 21:47:19 PST 2018; root:xnu-4570.71.22~1/RELEASE_X86_64 |
| win_version || +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| timer | time | node | system | mac_version
| win_version | +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| test setup | 0.03 | KeliFinsMacBook | Darwin | 10.13.6 || | test config | 0.0 | KeliFinsMacBook | Darwin | 10.13.6 || | test provider | 0.0 |
KeliFinsMacBook | Darwin | 10.13.6 || | box put | 5.13 | KeliFinsMacBook | Darwin | 10.13.6 || | box get | 2.39 |
KeliFinsMacBook | Darwin | 10.13.6 || | box list | 1.02 | KeliFinsMacBook | Darwin | 10.13.6 || | box create dir | 1.18 |
| KeliFinsMacBook | Darwin | 10.13.6 || | box delete | 549.25 | KeliFinsMacBook | Darwin | 10.13.6 || +-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

20.8 AWS EMR Benchmarking

Benchmark results for AWS EMR using cloudmesh-emr.

Speedtest.net results: Ping: 14ms Download: 23.32 Mbps. Upload: 6.01 Mbps.

Benchmark results via StopWatch:

Machine	Attribute	Time/s
BUG_REPORT_URL	"https://bugs.launchpad.net/ubuntu/"	
DISTRIB_CODENAME	bionic	
DISTRIB_DESCRIPTION	"Ubuntu 18.04.2 LTS"	
DISTRIB_ID	Ubuntu	
DISTRIB_RELEASE	18.04	
HOME_URL	"https://www.ubuntu.com/"	
ID	ubuntu	
ID_LIKE	debian	
NAME	"Ubuntu"	
PRETTY_NAME	"Ubuntu 18.04.2 LTS"	
PRIVACY_POLICY_URL	"https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"	
SUPPORT_URL	"https://help.ubuntu.com/"	
UBUNTU_CODENAME	bionic	
VERSION	"18.04.2 LTS (Bionic Beaver)"	
VERSION_CODENAME	bionic	
VERSION_ID	"18.04"	

(continues on next page)

(continued from previous page)

```

| mac_version      |
↪ |
| machine          | ('x86_64',)
↪ |
| node            | ('vb',)
↪ |
| platform         | Linux-4.18.0-17-generic-x86_64-with-Ubuntu-18.04-bionic
↪ |
| processor        | ('x86_64',)
↪ |
| processors       | Linux
↪ |
| python           | 3.6.7 (default, Oct 22 2018, 11:32:17)
↪ |
|                  | [GCC 8.2.0]
↪ |
| release          | ('4.18.0-17-generic',)
↪ |
| sys              | linux
↪ |
| system           | Linux
↪ |
| user             | anthony
↪ |
| version          | #18~18.04.1-Ubuntu SMP Fri Mar 15 15:27:12 UTC 2019
↪ /
| win_version      |
↪ |
+-----+
↪---+
+-----+-----+-----+-----+-----+-----+
| timer          | time | node   | system | mac_version | win_version |
+-----+-----+-----+-----+-----+-----+
| List Clusters  | 0.53 | ('vb',) | Linux  |              |              |
| Start Cluster  | 0.41 | ('vb',) | Linux  |              |              |
| List Instances | 0.26 | ('vb',) | Linux  |              |              |
| Describe Cluster | 0.22 | ('vb',) | Linux  |              |              |
| List Steps     | 0.25 | ('vb',) | Linux  |              |              |
| Copy File      | 0.36 | ('vb',) | Linux  |              |              |
| Run Program    | 0.32 | ('vb',) | Linux  |              |              |
| Stop Cluster   | 0.26 | ('vb',) | Linux  |              |              |
+-----+-----+-----+-----+-----+-----+

```


21.1 Code Documentation

21.2 Common

cloudmesh-common contains a number of useful methods that you can reuse to develop your code. They avoid reimplementing and duplication among the different contributors. Please use these methods instead of reimplementing them.

21.2.1 DEBUG

21.2.2 Variable

21.2.3 Util

Useful utility functions

`cloudmesh.common.util.Heading(txt=None, c='#')`

Prints a message to stdout with ##### surrounding it. This is useful for nosetests to better distinguish them.

Parameters

- **c** – uses the given char to wrap the header
- **txt** (*string*) – a text message to be printed

`cloudmesh.common.util.auto_create_requirements(requirements)`

creates a requirement.txt file from the requirements in the list. If the file exists, it get changed only if the requirements in the list are different from the existing file

Parameters **requirements** – the requirements in a list

`cloudmesh.common.util.auto_create_version(class_name, version, filename='__init__.py')`
creates a version number in the `__init__.py` file. it can be accessed with `__version__` :param class_name: :param version: :param filename: :return:

`cloudmesh.common.util.backup_name(filename)`

Parameters `filename` (*string*) – given a filename creates a backup name of the form `filename.bak.1`. If the filename already exists the number will be increased as much as needed so the file does not exist in the given location. The filename can consists a path and is expanded with `~` and environment variables.

Return type `string`

`cloudmesh.common.util.banner(txt=None, c='#', debug=True, label=None, color=None)`
prints a banner of the form with a frame of `#` around the `txt`:

```
#####  
# txt  
#####
```

Parameters

- **color** – prints in the given color
- **label** – adds a label
- **debug** – prints only if debug is true
- **txt** (*string*) – a text message to be printed
- **c** (*character*) – the character used instead of `c`

`cloudmesh.common.util.convert_from_unicode(data)`
converts unicode data to a string :param data: the data to convert :return:

`cloudmesh.common.util.copy_files(files_glob, source_dir, dest_dir)`

Parameters

- **files_glob** – *.yaml
- **source_dir** – source directiry
- **dest_dir** – destination directory

Returns

`cloudmesh.common.util.exponential_backoff(fn, sleeptime_s_max=1800)`
Calls `fn` until it returns `True`, with an exponentially increasing wait time between calls

`cloudmesh.common.util.generate_password(length=8, lower=True, upper=True, number=True)`
generates a simple password. We should not really use this in production. :param length: the length of the password :param lower: True of lower case characters are allowed :param upper: True if upper case characters are allowed :param number: True if numbers are allowed :return:

`cloudmesh.common.util.grep(pattern, filename)`
Very simple grep that returns the first matching line in a file. String matching only, does not do REs as currently implemented.

`cloudmesh.common.util.path_expand(text)`
returns a string with expanded variable.

Parameters

- **text** – the path to be expanded, which can include ~ and environment \$ variables
- **text** – string

`cloudmesh.common.util.readfile(filename)`

returns the content of a file :param filename: the filename :return:

`cloudmesh.common.util.search(lines, pattern)`

return all lines that match the pattern #TODO: we need an example

Parameters

- **lines** –
- **pattern** –

Returns

`cloudmesh.common.util.str_banner(txt=None, c='#', debug=True)`

prints a banner of the form with a frame of # around the txt:

```
#####
# txt
#####
```

Parameters

- **debug** (*boolean*) – return "" if not in debug
- **txt** (*string*) – a text message to be printed
- **c** (*character*) – the character used instead of c

`cloudmesh.common.util.tempdir(*args, **kwargs)`

A contextmanager to work in an auto-removed temporary directory

Arguments are passed through to tempfile.mkdtemp

example:

```
>>> with tempdir() as path:
...     pass
```

`cloudmesh.common.util.writefile(filename, content)`

writes the content into the file :param filename: the filename :param content: the content :return:

`cloudmesh.common.util.yn_choice(message, default='y', tries=None)`

asks for a yes/no question.

Parameters

- **tries** – the number of tries
- **message** – the message containing the question
- **default** – the default answer

21.2.4 Dotdict

A convenient dot dict class

`a = dotdict({"argument": "value"})`

```
print (a.argument)
```

```
class cloudmesh.common.dotdict.dotdict
    dot.notation access to dictionary attributes
```

21.2.5 Locations

class that specifies where we read the cloudmesh.yaml file from

```
cloudmesh.common.locations.config_dir_setup(filename)
    sets the config file and makes sure the directory exists if it has not yet been created. :param filename: :return:
```

```
cloudmesh.common.locations.config_file(filename)
    The location of the config file: ~/.cloudmesh/filename. ~ will be expanded :param filename: the filename
```

```
cloudmesh.common.locations.config_file_prefix()
    The prefix of the configuration file location
```

```
cloudmesh.common.locations.config_file_raw(filename)
    The location of the config file: ~/.cloudmesh/filename. ~ will NOT be expanded :param filename: the filename
```

21.2.6 Parameter

21.2.7 FlatDict

```
class cloudmesh.common.FlatDict.FlatDict(d)
    A data structure to manage a flattened dict. It is initialized by passing the dict at time of initialization.
```

keys() → a set-like object providing a view on D's keys

values() → an object providing a view on D's values

```
cloudmesh.common.FlatDict.flatten(d, parent_key=", sep='__')
    flattens the dict into a one dimensional dictionary
```

Parameters

- **d** – multidimensional dict
- **parent_key** – replaces from the parent key
- **sep** – the separation character used when fattening. the default is __

Returns the flattened dict

```
cloudmesh.common.FlatDict.key_prefix_replace(d, prefix, new_prefix="")
    replaces the list of prefix in keys of a flattened dict
```

Parameters

- **d** – the flattened dict
- **prefix**(list of str) – a list of prefixes that are replaced with a new prefix. Typically this will be ""
- **new_prefix** – The new prefix. By default it is set to ""

Returns the dict with the keys replaced as specified

21.2.8 Printer

Convenient methods and classes to print tables.

class `cloudmesh.common.Printer.Printer`

A simple Printer class with convenient methods to print dictionary, tables, csv, lists

classmethod attribute (*d*, *header=None*, *order=None*, *sort_keys=True*, *output='table'*)

prints a attribute/key value table :param d: A a dict with dicts of the same type.

Each key will be a column

Parameters

- **order** – The order in which the columns are printed. The order is specified by the key names of the dict.
- **header** (*A list of string*) – The Header of each of the columns
- **sort_keys** (*string or a tuple of string (for sorting with multiple columns)*) – Key(s) of the dict to be used for sorting. This specify the column(s) in the table for sorting.
- **output** – the output format table, csv, dict, json

classmethod csv (*d*, *order=None*, *header=None*, *sort_keys=True*)

prints a table in csv format

Parameters *d* (*dict*) – A a dict with dicts of the same type.

:param order:The order in which the columns are printed. The order is specified by the key names of the dict.

Parameters

- **header** (*list or tuple of field names*) – The Header of each of the columns
- **sort_keys** (*bool*) – TODO: not yet implemented

Returns a string representing the table in csv format

classmethod dict (*d*, *order=None*, *header=None*, *output='table'*, *sort_keys=True*, *show_none=""*)

TODO :param d: A a dict with dicts of the same type. :type d: dict :param order:The order in which the columns are printed.

The order is specified by the key names of the dict.

Parameters

- **header** (*list or tuple of field names*) – The Header of each of the columns
- **output** (*string*) – type of output (table, csv, json, yaml or dict)
- **sort_keys** (*bool*) –
- **show_none** (*bool*) – prints None if True for None values otherwise ""

Returns

```
classmethod dict_table (d, order=None, header=None, sort_keys=True, show_none="",  
                        max_width=40)
```

prints a pretty table from an dict of dicts :param d: A a dict with dicts of the same type.

Each key will be a column

Parameters

- **order** – The order in which the columns are printed. The order is specified by the key names of the dict.
- **header** (*A list of string*) – The Header of each of the columns
- **sort_keys** (*string or a tuple of string (for sorting with multiple columns)*) – Key(s) of the dict to be used for sorting. This specify the column(s) in the table for sorting.
- **show_none** (*bool*) – prints None if True for None values otherwise ""
- **max_width** (*int*) – maximum width for a cell

```
classmethod flatwrite (table, order=None, header=None, output='table', sort_keys=True,  
                      show_none="", sep='.')
```

writes the information given in the table :param table: the table of values :param order: the order of the columns :param header: the header for the columns :param output: the format (default is table, values are raw, csv, json, yaml, dict :param sort_keys: if true the table is sorted :param show_none: passed along to the list or dict printer :param sep: uses sep as the separator for csv printer :return:

```
classmethod list (l, order=None, header=None, output='table', sort_keys=True, show_none="")
```

Parameters

- **l** – l is a list not a dict
- **order** –
- **header** –
- **output** –
- **sort_keys** –
- **show_none** –

Returns

```
classmethod print_list (l, output='table')
```

prints a list :param l: the list :param output: the output, default is a table :return:

```
classmethod row_table (d, order=None, labels=None)
```

prints a pretty table from data in the dict. :param d: A dict to be printed :param order: The order in which the columns are printed.

The order is specified by the key names of the dict.

Parameters labels – The array of labels for the column

```
classmethod write (table, order=None, header=None, output='table', sort_keys=True,  
                  show_none="")
```

writes the information given in the table :param table: the table of values :param order: the order of the columns :param header: the header for the columns :param output: the format (default is table, values are raw, csv, json, yaml, dict :param sort_keys: if true the table is sorted :param show_none: passed along to the list or dict printer :return:

21.2.9 Stopwatch

Class for starting and stopping named timers.

This class is based on a similar java class in cyberaide, and java cog kit.

class cloudmesh.common.StopWatch.**StopWatch**

A class to measure times between events.

classmethod benchmark (*sysinfo=True*)

prints out all timers in a convenient benchmark tabble :return: :rtype:

classmethod clear ()

clear start and end timer_start

classmethod get (*name*)

returns the time of the timer.

Parameters **name** (*string*) – the name of the timer

Return type the elapsed time

classmethod keys ()

returns the names of the timers

classmethod print (**args*)

prints a timer. The first argument is the label if it exists, the last is the timer :param args: label, name :return:

classmethod start (*name*)

starts a timer with the given name.

Parameters **name** (*string*) – the name of the timer

classmethod stop (*name*)

stops the timer with a given name.

Parameters **name** (*string*) – the name of the timer

21.2.10 Console

Printing messages in a console

class cloudmesh.common.console.**Console**

A simple way to print in a console terminal in color. Instead of using simply the print statement you can use special methods to indicate warnings, errors, ok and regular messages.

Example Usage:

```
Console.warning("Warning")
Console.error("Error")
Console.info("Info")
Console.msg("msg")
Console.ok("Success")
```

One can switch the color mode off with:

```
Console.color = False
Console.error("Error")
```

The color will be switched on by default.

```
static TODO (message, prefix=True, traceflag=True)
    prints an TODO message :param message: the message :param prefix: if set to true it prints TODO: as
    prefix :param traceflag: if true the stack trace is retrieved and printed :return:

static cprint (color, prefix, message)
    prints a message in a given color :param color: the color as defined in the theme :param prefix: the prefix
    (a string) :param message: the message :return:

static debug_msg (message)
    print a debug message :param message: the message :return:

classmethod error (message, prefix=True, traceflag=False)
    prints an error message :param message: the message :param prefix: a prefix for the message :param
    traceflag: if true the stack trace is retrieved and printed :return:

static get (name)
    returns the default theme for printing console messages :param name: the name of the theme :return:

static info (message)
    prints an informational message :param message: the message :return:

static msg (*message)
    prints a message :param message: the message to print :return:

static ok (message)
    prints an ok message :param message: the message< :return:

classmethod set_debug (on=True)
    sets debugging on or of :param on: if on debugging is set :return:

static set_theme (color=True)
    defines if the console messages are printed in color :param color: if True its printed in color :return:

static txt_msg (message, width=79)
    prints a message to the screen :param message: the message to print :param width: teh width of the line
    :return:

static warning (message)
    prints a warning :param message: the message :return:
```

```
cloudmesh.common.console.indent (text, indent=2, width=128)
    indents the given text by the indent specified and wrapping to the given width
```

Parameters

- **text** – the text to print
- **indent** – indent characters
- **width** – the width of the text

Returns

21.2.11 Logger

simple logging convenience framework

```
cloudmesh.common.logger.LOGGER (filename)
    creates a logger with the given name.
```

You can use it as follows:

```
log = cloudmesh.common.LOGGER(__file__)
log.error("this is an error")
log.info("this is an info")
log.warning("this is a warning")
```

`cloudmesh.common.logger.LOGGING_OFF(log)`

Switches logging off :param log: the logger for which we switch logging off

`cloudmesh.common.logger.LOGGING_ON(log)`

Switches logging on :param log: the logger for which we switch logging on

21.2.12 Error

A simple framework to handle error messages

class `cloudmesh.common.error.Error`

A class to print error messages

classmethod `debug(msg)`

prints a debug message. :param msg: the message :return:

classmethod `exit(msg)`

call a system exit :param msg: :return:

classmethod `info(msg)`

prints an info msg. :param msg: the message :return:

classmethod `msg(error=None, debug=True, trace=True)`

prints the error message :param error: the error message :param debug: only prints it if debug is set to true
:param trace: if true prints the trace :return:

classmethod `traceback(error=None, debug=True, trace=True)`

prints the trace :param error: a message preceding the trace :param debug: prints it if debug is set to true
:param trace: :return:

classmethod `warning(msg)`

prints a warning message. :param msg: :return:

21.2.13 Shell

A convenient method to execute shell commands and return their output. Note: that this method requires that the command be completely execute before the output is returned. FOr many activities in cloudmesh this is sufficient.

class `cloudmesh.common.Shell.Shell`

The shell class allowing us to conveniently access many operating system commands. TODO: This works well on Linux and OSX, but has not been tested much on Windows

classmethod `VBoxManage(*args)`

executes VBoxManage with the given arguments :param args: :return:

classmethod `bash(*args)`

executes bash with the given arguments :param args: :return:

classmethod `blockdiag(*args)`

executes blockdiag with the given arguments :param args: :return:

classmethod `brew(*args)`

executes bash with the given arguments :param args: :return:

classmethod cat (*args)
executes cat with the given arguments :param args: :return:

classmethod check_output (*args, **kwargs)
Thin wrapper around `subprocess.check_output()`

classmethod check_python ()
checks if the python version is supported :return: True if it is supported

classmethod cm (*args)
executes cm with the given arguments :param args: :return:

command = {'darwin': {}, 'linux': {}, 'windows': {}}
TODO

how do we now define dynamically functions based on a list that we want to support

what we want is where args are multiple unlimited parameters to the function

def f(args...): name = get the name from f a = list of args...

cls.execute(cmd, arguments=a, capture=True, verbose=False)

commands = ['ps', 'ls',] for c in commands:

generate this command and add to this class dynamically

or do something more simple

ls = cls.execute('cmd', args...)

classmethod command_exists (name)
returns True if the command exists :param name: :return:

classmethod dialog (*args)
executes dialof with the given arguments :param args: :return:

classmethod execute (cmd, arguments="", shell=False, cwd=None, traceflag=True, wither-
ror=True)
Run Shell command

Parameters

- **witherror** – if set to False the error will not be printed
- **traceflag** – if set to true the trace is printed in case of an error
- **cwd** – the current working directory in which the command is supposed to be executed.
- **shell** – if set to true the subprocess is called as part of a shell
- **cmd** – command to run
- **arguments** – we do not know yet

Returns

classmethod fgrep (*args)
executes fgrep with the given arguments :param args: :return:

classmethod find_cygwin_executables ()
find the executables in cygwin

classmethod find_lines_with (lines, what)
returns all lines that contain what :param lines: :param what: :return:

classmethod get_python ()
returns the python and pip version :return: python version, pip version

```

classmethod git (*args)
    executes git with the given arguments :param args: :return:

classmethod grep (*args)
    executes grep with the given arguments :param args: :return:

classmethod head (*args)
    executes head with the given arguments :param args: :return:

classmethod keystone (*args)
    executes keystone with the given arguments :param args: :return:

classmethod kill (*args)
    executes kill with the given arguments :param args: :return:

classmethod ls (*args)
    executes ls with the given arguments :param args: :return:

classmethod mkdir (directory)
    creates a directory with all its parents in ots name :param directory: the path of the directory :return:

classmethod mongod (*args)
    executes mongod with the given arguments :param args: :return:

classmethod nosetests (*args)
    executes nosetests with the given arguments :param args: :return:

classmethod nova (*args)
    executes nova with the given arguments :param args: :return:

classmethod operating_system ()
    the name of the os :return: the name of the os

classmethod pandoc (*args)
    executes vagrant with the given arguments :param args: :return:

classmethod ping (host=None, count=1)
    execute ping :param host: the host to ping :param count: the number of pings :return:

classmethod pip (*args)
    executes pip with the given arguments :param args: :return:

classmethod ps (*args)
    executes ps with the given arguments :param args: :return:

classmethod pwd (*args)
    executes pwd with the given arguments :param args: :return:

classmethod rackdiag (*args)
    executes rackdiag with the given arguments :param args: :return:

classmethod remove_line_with (lines, what)
    returns all lines that do not contain what :param lines: :param what: :return:

classmethod rm (*args)
    executes rm with the given arguments :param args: :return:

classmethod rsync (*args)
    executes rsync with the given arguments :param args: :return:

classmethod scp (*args)
    executes scp with the given arguments :param args: :return:

```

```
classmethod sh (*args)
    executes sh with the given arguments :param args: :return:

classmethod sort (*args)
    executes sort with the given arguments :param args: :return:

classmethod ssh (*args)
    executes ssh with the given arguments :param args: :return:

classmethod sudo (*args)
    executes sudo with the given arguments :param args: :return:

classmethod tail (*args)
    executes tail with the given arguments :param args: :return:

classmethod terminal_type ()
    returns darwin, cygwin, cmd, or linux

unzip (source_filename, dest_dir)
    unzips a file into the destination directory :param source_filename: the source :param dest_dir: the destination directory :return:

classmethod vagrant (*args)
    executes vagrant with the given arguments :param args: :return:

classmethod which (command)
    returns the path of the command with which :param command: the command :return: the path

class cloudmesh.common.Shell.Subprocess (cmd, cwd=None, stderr=-1, stdout=-1, env=None)
    Executes a command. This class should not be directly used, but instead you should use Shell.

exception cloudmesh.common.Shell.SubprocessError (cmd, returncode, stderr, stdout)
    Manages the formatting of the error and stdout. This command should not be directly called. Instead use Shell

cloudmesh.common.Shell.main ()
    a test that should actually be added into a nosetest :return:
```

21.2.14 Run

21.2.15 DB

```
class cloudmesh.db.strdb.YamlDB (path)
    A YAML-backed Key-Value database to store strings

clear ()
    Truncate the database

close ()
    This is a NoOP for backwards compatibility
```

21.2.16 SSH

authorized key management.

```
class cloudmesh.common.ssh.authorized_keys.AuthorizedKeys
    Class to manage authorized keys.

add (pubkey)
    add a public key. :param pubkey: the filename to the public key :return:
```


classmethod `load(path)`

load the keys from a path

Parameters `path` – the filename (path) in which we find the keys

Returns

remove (`pubkey`)

Removes the public key TODO: this method is not implemented :param pubkey: the filename of the public key :return:

`cloudmesh.common.ssh.authorized_keys.get_fingerprint_from_public_key(pubkey)`

Generate the fingerprint of a public key

Parameters `pubkey` (`str`) – the value of the public key

Returns fingerprint

Return type `str`

Managing `~/.ssh/config`

class `cloudmesh.common.ssh.ssh_config.ssh_config(filename=None)`

Managing the config in `.ssh`

execute (`name, command`)

execute the command on the named host :param name: the name of the host in config :param command: the command to be executed :return:

generate (`key='india', host='india.futuresystems.org', username=None, force=False, verbose=False`)

adds a host to the config file with given parameters. #TODO: make sure this is better documented :param key: the key :param host: the host :param username: the username :param force: not used :param verbose: prints debug messages :return:

list ()

list the hosts in the config file :return:

load ()

list the hosts defined in the ssh config file

local (`command`)

execute the command on the localhost :param command: the command to execute :return:

login (`name`)

login to the host defines in `.ssh/config` by name :param name: the name of the host as defined in the config file :return:

names ()

The names defined in `~/.ssh/config` :return: the names

status ()

executes a test with the given ssh config if a login is possible. TODO: not yet implemented

username (`host`)

returns the username for a given host in the config file :param host: the hostname :return: the username

21.3 CMD5

`cloudmesh.shell.command.basecommand(func)`

A decorator to create a function with docopt arguments. It also generates a help function

```
@command def do_myfunc(self, args):
    """ docopts text """ pass

will create

def do_myfunc(self, args, arguments): """ docopts text """ ...
def help_myfunc(self, args, arguments): ... prints the docopt text ...
```

Parameters **func** – the function for the decorator

`cloudmesh.shell.command.command(func)`

A decorator to create a function with docopt arguments. It also generates a help function

```
@command def do_myfunc(self, args):
    """ docopts text """ pass

will create

def do_myfunc(self, args, arguments): """ docopts text """ ...
def help_myfunc(self, args, arguments): ... prints the docopt text ...
```

Parameters **func** – the function for the decorator

`cloudmesh.shell.command.map_parameters(arguments, *args)`

This command is useful to map parameters with – to regular argument dicts for easier processing.

Parameters

- **arguments** –
- **args** –

Returns

an example is

`map_parameters(arguments, 'active', 'cloud')`

where `–active=ACTIVE` is mapped to `arguments["active"]` and `–cloud=CLOUD` is mapped to `arguments["cloud"]`

as `arguments` is a dotdict, they can then for example be called as

`arguments.cloud`

`class cloudmesh.shell.shell.CMShell(completekey='tab', stdin=None, stdout=None)`

The command shell that inherits all commands from `PluginCommand`

`do_EOF(args)`

```
Usage:
EOF

Description:
Command to the shell to terminate reading a script.
```

`do_help(arg)`

```
Usage:
    help
    help COMMAND

Description:
    List available commands with "help" or detailed help with
    "help COMMAND".
```

do_info(args)

```
Usage:
    info [path|commands|files|cloudmesh]

Description:
    info
        provides internal info about the shell and its packages
```

do_plugin(args)

```
Usage:
    plugin install PLUGIN [-s]
    plugin uninstall PLUGIN
    plugin list
    plugin ? [--format=FORMAT]

Arguments:
    PLUGIN    the name of the plugin

Description:
    plugin available
        lists the available plugins
    plugin list
        lists the plugin
    plugin install
        installs the given plugin
    plugin uninstall
        uninstalls the given plugin
```

do_q(args)

```
Usage:
    quit

Description:
    Action to be performed when quit is typed
```

do_quit(args)

```
Usage:
    quit

Description:
    Action to be performed when quit is typed
```

do_shell (*args*)

```
Usage:
    shell COMMAND

Arguments:
    COMMAND    the command to be executed

Description:
    shell COMMAND    executes the command
```

do_version (*args*)

```
Usage:
    version pip [PACKAGE]
    version [--format=FORMAT] [--check=CHECK]

Options:
    --format=FORMAT    the format to print the versions in [default: table]
    --check=CHECK      boolean tp conduct an additional check [default: True]

Description:
    version
        Prints out the version number
    version pip
        Prints the contents of pip list

Limitations:
    Package names must not have a . in them instead you need to use -
    Thus to query for cloudmesh-cmd5 use

    cms version pip cloudmesh-cmd5
```

emptyline ()

Called when an empty line is entered in response to the prompt.

If this method is not overridden, it repeats the last nonempty command entered.

help_help ()

```
Usage:
    help
    help COMMAND

Description:
    List available commands with "help" or detailed help with
    "help COMMAND".
```

onecmd (*line*)

Interpret the argument as though it had been typed in response to the prompt.

This may be overridden, but should not normally need to be; see the `precmd()` and `postcmd()` methods for useful execution hooks. The return value is a flag indicating whether interpretation of commands by the interpreter should stop.

postcmd (*stop, line*)

Hook method executed just after a command dispatch is finished.

precmd (*line*)

Hook method executed just before the command line is interpreted, but after the input prompt is generated and issued.

preloop ()

adds the banner to the preloop

class cloudmesh.shell.shell.**Plugin**

Some simple methods to manage dynamic namespace plugins for cloudmesh.

classmethod **class_name** (*command*)

creates the default filename in which the module is defined :param command: the name of the command
:return: cloudmesh.ext.command.<command>+command.<Command>

classmethod **classes** ()

list of the commands in the cloudmesh namespace under cloudmesh.ext.command :return: list of the commands

classmethod **load** (*commands=None*)

Parameters **commands** – If None the commands will be found from import cloudmesh Otherwise the commands can be explicitly specified with

```
commands = [ 'cloudmesh.ext.command.bar.BarCommand',
               'cloudmesh.ext.command.foo.FooCommand', ]
```

A namespace package must exists. Foo and Bar are just examples

Returns the classes of the command

classmethod **modules** ()

list of cloudmesh modules in the cloudmesh namespace :return: list of modules

classmethod **name** (*command*)

creates a name for a modules starting with **do_** :param command: returns a tuple with the module location and the do_function :return:

cloudmesh.shell.shell.**PluginCommandClasses**

alias of cloudmesh.shell.shell.CommandProxyClass

cloudmesh.shell.shell.**main** ()

cms.

Usage: cms -help cms [-echo] [-debug] [-nosplash] [-i] [COMMAND ...]

Arguments: COMMAND A command to be executed

Options:

--file=SCRIPT	-f	SCRIPT Executes the script
-i		After start keep the shell interactive, otherwise quit [default: False]
--nosplash		do not show the banner [default: False]

cloudmesh.shell.shell.**print_list** (*elements*)

prints the element of a list :param elements: the elements to be printed

21.4 Clouddmesh

21.5 Management

21.5.1 Configuration

class `cloudmesh.management.configuration.counter.Counter` (`counter_file_path='~/cloudmesh/counter.yaml'`)

A counter is used to keep track of some value that can be increased and is associated with a user. Typically it is used to increment the vm id or the job id.

decr (`name='counter'`)
increments the counter by one :return:

get (`name='counter'`)
returns the value of the counter :param name: name of the counter :return: the value of the counter

incr (`name='counter'`)
increments the counter by one :return:

set (`name='counter', value=None`)
sets a counter associated with a particular user :param name: name of the counter :param value: the value :return:

We use a uniform naming convention method. The name is defined by different kinds of objects. The name is a string its syntax is defined in a yaml file located at `~/cloudmesh/name.yaml`

```
order:
- experiment
- group
- user
- kind
- counter
schema: '{experiment}-{group}-{user}-{kind}-{counter}'
experiment: exp
group: grp
user: gregor
kind: container
counter: 2
```

This file is automatically generated if it does not exists by a simple *Name* object that can include an ordered number of dictionary keys such as

Experiment is an experiment that all cloud objects can be placed under.

Group A group formulates a number of objects that logically build an entity, such as a number of virtual machines building a cluster

User A user name that may control the group

Kind A kind that identifies which kind of resource this is

The last is a counter which is always increased and written into this file in order to assure that the latest value is safely included in it.

A typical use is

```
n = Name(experiment="exp",
         group="grp",
```

(continues on next page)

(continued from previous page)

```

        user="gregor",
        kind="vm",
        counter=1)

n.incr()
counter = n.get()

```

Which will return

```
exp-grp-gregor-vm-1
```

```
class cloudmesh.management.configuration.name.Name (order=None, **kwargs)
```

```
class cloudmesh.management.configuration.SSHkey.SSHkey
```

```
get_from_git (user, keyname=None)
    gets the key from github
```

Parameters

- **keyname** – the keyname
- **user** – the github username

Returns an array of public keys

Return type `list`

```
set_permissions (path)
```

Sets the permissions of the path assuming the path is a public or private key :param path: :return:

21.5.2 Printer

21.5.3 Names

We use a uniform naming convention method. The name is defined by different kinds of objects. The name is a string its syntax is defined in a yaml file located at `~/ .cloudmesh/name.yaml`

```

order:
- experiment
- group
- user
- kind
- counter
schema: '{experiment}-{group}-{user}-{kind}-{counter}'
experiment: exp
group: grp
user: gregor
kind: container
counter: 2

```

This file is automatically generated if it does not exists by a simple *Name* object that can include an ordered number of dictionary keys such as

Experiment is an experiment that all cloud objects can be placed under.

Group A group formulates a number of objects that logically build an entity, such as a number of virtual machines building a cluster

User A user name that may control the group

Kind A kind that identifies which kind of resource this is

The last is a counter which is always increased and written into this file in order to assure that the latest value is safely included in it.

A typical use is

```
n = Name(experiment="exp",
        group="grp",
        user="gregor",
        kind="vm",
        counter=1)

n.incr()
counter = n.get()
```

Which will return

```
exp-grp-gregor-vm-1
```

```
class cloudmesh.management.configuration.name.Name (order=None, **kwargs)
```

21.5.4 Script

A convenient method to execute shell commands and return their output. Note: that this method requires that the command be completely executed before the output is returned. For many activities in cloudmesh this is sufficient.

```
class cloudmesh.management.script.Script
```

Executing a script defined by a simple text parameter

```
static run (script, live=False, debug=False)
```

run the specified script line by line.

TODO: at one point this should be moved to cloudmesh.common

Parameters

- **script** – The script
- **debug** – If true the output of the script is printed

Returns

```
class cloudmesh.management.script.SystemPath
```

Managing the System path in the .bashrc or .bash_profile files

```
static add (path)
```

Adds a path to the ~/.bashrc or ~/.bash_profile files.

TODO: Windows is not implemented yet.

Parameters **path** – The path to be added

Returns

```
cloudmesh.management.script.find_process (name)
```

find a process by name

Parameters **name** – the name of the process

Returns A list of dicts in which the attributes pid, command, and created are available and the name matches the specified name argument.

TODO: at one point this should be moved to cloudmesh.common

Return a list of processes matching 'name'.

21.5.5 Debug

21.6 Mongo

21.6.1 MongoDB

21.6.2 Controler

21.7 Commands

21.8 Inventory

21.9 cloudmesh-storage

class cloudmesh.storage.spec.tests.test_openapi_storage.**Test_cloud_storage**

see: <https://github.com/cloudmesh/cloudmesh-common/blob/master/cloudmesh/common/run/background.py>
 the code in the link has not been tested make this function execute the server in the background not in a terminal, get the pid and kill it after the test is done `UNAME := $(shell uname) ifeq ($(UNAME), Darwin)`
 define terminal

`osascript -e 'tell application "Terminal" to do script "cd $(PWD); $1"'`

`endif endif ifeq ($(UNAME), Linux) define terminal`

`gnome-terminal --command 'bash -c "cd $(PWD); $1"'`

`endif endif`

class cloudmesh.storage.provider.local.Provider.**Provider**(service=None, con-
fig='~/cloudmesh/cloudmesh4.yaml')

cloudmesh:

a:

cm: active: False heading: Local A host: localhost label: local_a kind: local version: 1.0

default: directory: .

credentials: directory: ~/cloudmesh/storage/a

default location is credentials.directory / default.directory

create_dir (directory=None)

creates a directory

Parameters **directory** – the name of the directory

Returns dict

create_dir_from_filename (*filename=None*)

creates a directory

Parameters **directory** – the name of the directory for the filename

Returns dict

delete (*source=None, recursive=False*)

deletes the source

Parameters

- **source** – the source which either can be a directory or file
- **recursive** – in case of directory the recursive refers to all subdirectories in the specified source

Returns dict

get (*source=None, destination=None, recursive=False*)

gets the source and copies it in destination

Parameters

- **source** – the source which either can be a directory or file
- **destination** – the destination which either can be a directory or file
- **recursive** – in case of directory the recursive refers to all subdirectories in the specified source

Returns dict

put (*source=None, destination=None, recursive=False*)

puts the source on the service

Parameters

- **source** – the source which either can be a directory or file
- **destination** – the destination which either can be a directory or file
- **recursive** – in case of directory the recursive refers to all subdirectories in the specified source

Returns dict

search (*directory=None, filename=None, recursive=False*)

gets the destination and copies it in source

Parameters

- **service** – the name of the service in the yaml file
- **directory** – the directory which either can be a directory or file
- **recursive** – in case of directory the recursive refers to all subdirectories in the specified source

Returns dict

tree (*directory=None*)

Prints a visual representation of the files and directories :param directory: :type directory: :return: :rtype:

`cloudmesh.storage.provider.local.Provider.creation_date` (*path_to_file*)

Try to get the date that a file was created, falling back to when it was last modified if that isn't possible. See <http://stackoverflow.com/a/39501288/1709587> for explanation.

21.10 cloudmesh-objstorage

21.11 cloudmesh-cloud

class cloudmesh.security.encrypt.**EncryptFile** (*filename, secret*)

keys must be generated with

```
ssh-keygen -t rsa -m pem openssl rsa -in id_rsa -out id_rsa.pem
```

check_passphrase ()

this does not work with pem

cecks if the ssh key has a password :return:

pem_verify ()

this does not work :return:

Managing ~/.ssh/config

class cloudmesh.security.ssh_config.**ssh_config** (*filename=None*)

Managing the config in .ssh

execute (*name, command*)

execute the command on the named host :param name: the name of the host in config :param command: the command to be executed :return:

generate (*key='india', host='india.futuresystems.org', username=None, force=False, verbose=False*)

adds a host to the config file with given parameters. #TODO: make sure this is better documented :param key: the key :param host: the host :param username: the username :param force: not used :param verbose: prints debug messages :return:

list ()

list the hosts in the config file :return:

load ()

list the hosts defined in the ssh config file

local (*command*)

execute the command on the localhost :param command: the command to execute :return:

login (*name*)

login to the host defines in .ssh/config by name :param name: the name of the host as defined in the config file :return:

names ()

The names defined in ~/.ssh/config :return: the names

status ()

executes a test with the given ssh config if a login is possible. TODO: not yet implemented

username (*host*)

returns the username for a given host in the config file :param host: the hostname :return: the username

authorized key management.

class cloudmesh.security.authorized_keys.**AuthorizedKeys**

Class to manage authorized keys.

add (*pubkey*)

add a public key. :param pubkey: the filename to the public key :return:

classmethod `load(path)`

load the keys from a path

Parameters `path` – the filename (path) in which we find the keys

Returns

remove (`pubkey`)

Removes the public key TODO: this method is not implemented :param pubkey: the filename of the public key :return:

`cloudmesh.security.authorized_keys.get_fingerprint_from_public_key(pubkey)`

Generate the fingerprint of a public key

Parameters `pubkey` (`str`) – the value of the public key

Returns fingerprint

Return type `str`

class `cloudmesh.management.configuration.SSHkey.SSHkey`

get_from_git (`user`, `keyname=None`)

gets the key from github

Parameters

- **keyname** – the keyname
- **user** – the github username

Returns an array of public keys

Return type `list`

set_permissions (`path`)

Sets the permissions of the path assuming the path is a public or private key :param path: :return:

We use a uniform naming convention method. The name is defined by different kinds of objects. The name is a string its syntax is defined in a yaml file located at `~/.cloudmesh/name.yaml`

```
order:
- experiment
- group
- user
- kind
- counter
schema: '{experiment}-{group}-{user}-{kind}-{counter}'
experiment: exp
group: grp
user: gregor
kind: container
counter: 2
```

This file is automatically generated if it does not exists by a simple *Name* object that can include an ordered number of dictionary keys such as

Experiment is an experiment that all cloud objects can be placed under.

Group A group formulates a number of objects that logically build an entity, such as a number of virtual machines building a cluster

User A user name that may control the group

Kind A kind that identifies which kind of resource this is

The last is a counter which is always increased and written into this file in order to assure that the latest value is safely included in it.

A typical use is

```
n = Name(experiment="exp",
         group="grp",
         user="gregor",
         kind="vm",
         counter=1)

n.incr()
counter = n.get()
```

Which will return

```
exp-grp-gregor-vm-1
```

```
class cloudmesh.management.configuration.name.Name (order=None, **kwargs)
```

```
class cloudmesh.management.configuration.counter.Counter (counter_file_path='~/cloudmesh/counter.yaml')
```

A counter is used to keep track of some value that can be increased and is associated with a user. Typically it is used to increment the vm id or the job id.

```
decr (name='counter')
```

increments the counter by one :return:

```
get (name='counter')
```

returns the value of the counter :param name: name of the counter :return: the value of the counter

```
incr (name='counter')
```

increments the counter by one :return:

```
set (name='counter', value=None)
```

sets a counter associated with a particular user :param name: name of the counter :param value: the value :return:

A convenient method to execute shell commands and return their output. Note: that this method requires that the command be completely executed before the output is returned. For many activities in cloudmesh this is sufficient.

```
class cloudmesh.management.script.Script
```

Executing a script defined by a simple text parameter

```
static run (script, live=False, debug=False)
```

run the specified script line by line.

TODO: at one point this should be moved to cloudmesh.common

Parameters

- **script** – The script
- **debug** – If true the output of the script is printed

Returns

```
class cloudmesh.management.script.SystemPath
```

Managing the System path in the .bashrc or .bash_profile files

```
static add (path)
```

Adds a path to the ~/ .bashrc or ~/ .bash_profile files.

TODO: Windows is not implemented yet.

Parameters `path` – The path to be added

Returns

`cloudmesh.management.script.find_process(name)`

find a process by name

Parameters `name` – the name of the process

Returns A list of dicts in which the attributes pid, command, and created are available and the name matches the specified name argument.

TODO: at one point this should be moved to cloudmesh.common

Return a list of processes matching 'name'.

`cloudmesh.vbox.command.vbox.defaults()`

default values :return: a number of default values for memory, image, and script :rtype: dotdict

class `cloudmesh.display.Display`

class `cloudmesh.compute.azure.AzProvider.Provider` (*name=None*, *configuration='~/cloudmesh/cloudmesh4.yaml'*)

az commands

<https://docs.microsoft.com/en-us/cli/azure/reference-index?view=azure-cli-latest>

create the

<https://docs.microsoft.com/en-us/azure/active-directory/develop/howto-create-service-principal-portal>

create (*name=None*, *image=None*, *size=None*, *timeout=360*, ***kwargs*)

creates a named node

Parameters

- **name** – the name of the node
- **image** – the image used
- **size** – the size of the image
- **timeout** – a timeout in seconds that is invoked in case the image does not boot. The default is set to 3 minutes.
- **kwargs** – additional arguments passed along at time of boot

Returns

destroy (*name=None*)

Destroys the node :param name: the name of the node :return: the dict of the node

info (*name=None*)

gets the information of a node with a given name

Parameters `name` –

Returns The dict representing the node including updated status

list ()

list all nodes id

Returns an array of dicts representing the nodes

rename (*name=None*, *destination=None*)

rename a node

Parameters

- **destination** –
- **name** – the current name

Returns the dict with the new name

resume (*name=None*)

resume the named node

Parameters **name** – the name of the node

Returns the dict of the node

start (*name=None*)

start a node

Parameters **name** – the unique node name

Returns The dict representing the node

stop (*name=None*)

stops the node with the given name

Parameters **name** –

Returns The dict representing the node including updated status

suspend (*name=None*)

suspends the node with the given name

Parameters **name** – the name of the node

Returns The dict representing the node

class cloudmesh.abstractclass.State.State (*name=None*)

class cloudmesh.mongo.DataBaseDecorator.DatabaseAlter (***kwargs*)

The data base decorator utomatically replaces an entry in the database with the dictionary returned by a function.

It is added to a MongoDB collection. The location is determined from the values in the dictionary.

The name of the collection is determined from cloud and kind:

cloud-kind

In addition each entry in the collection has a name that must be unique in that collection.

IN most examples it is pest to separate the upload from the actual return class. This way we essentially provide two functions one that provide the dict and another that is responsible for the upload to the database.

Example:

cloudmesh.example.foo contains:

```
class Provider(object)
```

```
    def entries(self):
```

```
        return {
```

```
            "cm": { "cloud": "foo", "kind": "entries", "name": "test01" "test": "hello" }
```

```
        } "cloud": "foo", "kind": "entries", "name": "test01" "test": "hello" }
```

cloudmesh.example.bar contains:

```
class Provider(object)
```

```
def entries(self):
    return { "cloud": "bar", "kind": "entries", "name": "test01" "test": "hello"}

cloudmesh.example.provider.foo:

from cloudmesh.example.foo import Provider as FooProvider from cloudmesh.example.foo import
Provider as BarProvider

class Provider(object)

    def __init__(self, provider):
        if provider == "foo": provider = FooProvider()
        elif provider == "bar": provider = BarProvider()

    @DatabaseUpdate def entries(self):
        provider.entries()
```

Separating the database and the dictionary creation allows the developer to implement different providers but only use one class with the same methods to interact for all providers with the database.

In the combined provider a find function to for example search for entries by name across collections could be implemented.

```
class cloudmesh.mongo.DataBaseDecorator.DatabaseUpdate (**kwargs)
```

The data base decorator utomatically replaces an entry in the database with the dictionary returned by a function.

It is added to a MongoDB collection. The location is determined from the values in the dictionary.

The name of the collection is determined from cloud and kind:

```
cloud-kind
```

In addition each entry in the collection has a name that must be unique in that collection.

IN most examples it is pest to separate the upload from the actual return class. This way we essentially provide two functions one that provide the dict and another that is responsible for the upload to the database.

Example:

cloudmesh.example.foo contains:

```
class Provider(object)

    def entries(self):
        return {
            "cm": { "cloud": "foo", "kind": "entries", "name": "test01" "test": "hello"}
        } "cloud": "foo", "kind": "entries", "name": "test01" "test": "hello" }
```

cloudmesh.example.bar contains:

```
class Provider(object)

    def entries(self):
        return { "cloud": "bar", "kind": "entries", "name": "test01" "test": "hello" }
```

cloudmesh.example.provider.foo:

```
from cloudmesh.example.foo import Provider as FooProvider from cloudmesh.example.foo import
Provider as BarProvider

class Provider(object)
```



```
def __init__(self, provider):
    if provider == "foo": provider = FooProvider()
    elif provider == "bar": provider = BarProvider()
    @DatabaseUpdate def entries(self):
        provider.entries()
```

Separating the database and the dictionary creation allows the developer to implement different providers but only use one class with the same methods to interact for all providers with the database.

In the combined provider a find function to for example search for entries by name across collections could be implemented.

```
class cloudmesh.data.api.storage.StorageProviderABC.StorageProviderABC
```

Abstract Base Class for supported cloud providers.

```
delete (name)
```

delete a file from the provider

Parameters **name** – the cloud file entry being deleted

```
exists (name)
```

if a file exists in the remote storage provider

Parameters **name** – a file name to check

```
get (source, destination)
```

get a file stored with this provider

Parameters

- **source** – the cloud file entry being retrieved
- **destination** – download destination

Returns the downloaded cloud file binary

```
put (source)
```

upload a file

Parameters **source** – a CloudFile. todo

Returns a CloudFile with resource information filled in

```
class cloudmesh.data.api.db.DBProviderABC.DBProviderABC
```

Abstract Base Class for supported database providers.

```
add (cloud_file)
```

add a new CloudFile to the database

Parameters **cloud_file** – a CloudFile. todo

Returns a CloudFile with resource information filled in

```
delete (cloud_file)
```

delete a file from the database

Parameters **cloud_file** – the cloud file entry being deleted

```
list_files ()
```

get a list of stored files

Returns a list of CloudFiles

update (*cloud_file*)

update a file

Parameters *cloud_file* – the cloud file entry being updated

Returns the updated CloudFile

21.12 cloudbatch

21.13 cloudbatch-emr

21.14 Code Conventions

- Python lint: All code must be formatted with the pyCharm inspect method which will suggest a reformat with pyCharm and allows good error and python issue detection. Please fix as many as you can.
- Printing errors:

In case you need to print errors please do not use print, but use

```
from cloudbatch.common.console import Console

Console.error("this is an example")
```

- Printing debug messages in verbose mode

In case you like to do debug messages use

```
from cloudbatch.DEBUG import VERBOSE

VERBOSE("this is an example")
'''
```

- Managing debug and verbose mode

Verbosity and debugging can be controlled with

```
cms set verbose=10
cms set debug=True
cms set trace=True
```

anything that is smaller than 10 will be printed.

21.15 Code Management

At this time we recommend and require that you use pyCharm community edition or professional to edit your code. Before committing we like that you run `Inspect Code` on all files that you commit and fix as many errors as possible including PEP8 format suggestions. It also notifies you of issues you may not think about while doing other code inspection.

The reason we ask you to do so is that pycharms code inspection is very good, and that if everyone uses pycharm the format of the code is uniform and we do not run in to formatting issues.

This will make the review of any code contributed much easier.

Naturally you can use a different editor for your work, but we still ask you to use pycharm to fix formatting and code inspection before you commit.

Typically we run a code inspection every week.

21.16 Documentation Management

To increase readability of the documentation we ask you to try to use 80 character line limits if possible. This is important for better editing experience in github. A good editor to do this with is emacs with its `Esc-q` command and pycharm with its `Edit-Wrap Line to column` or `paragraph` features. On macOS this can be called with `CONTROL-SHIFT-COMMAND-W` or `CONTROL-SHIFT-COMMAND-P`

21.17 Version Managemt

This is only done by Gregor

To create a development version we say

```
$ make build
```

To increase the patch number, say

```
$ make patch
```

To increase the minor number

```
$ make minor
```

The major number will stay to 4, so this is not changed

To create a release say

```
$ make release
```

After the release is done the minor number will be increased and the buld number will be reset.

21.18 Pytest

Pytest is a utility to unit test python code.

We use nosetests and not `__main__` to test all functionality so they can me automatically run and reports can be generated. A project that does not have a sufficient number of tests to make sure the module works can not be accepted.

21.18.1 Installation

The nose module can be installed with the help of pip utility

```
$ pip install pytest
```

This will install the pytest module in the current Python distribution, which means the test can be run using this utility as well as using `-m` switch. All tests are included in the folder `tests`.

For example for the `cloudmesh-cloud` module they are in

- <https://github.com/cloudmesh/cloudmesh-cloud/tree/master/tests>

Best is to add a number to identify in which order they are run

```
+cm
+ cloudmesh
+ tests
- test_01_topic1.py
- test_02_topic2.py
- test_03_topic2.py
```

21.18.2 Test Specification and Execution

A simple example for a test is

- https://github.com/cloudmesh/cloudmesh-cloud/blob/master/tests/test_key.py

Note that all test python programs have specific function names of the form

```
def test_number_topic (self)
```

The number is defined to order them and is typically something like `001`, note the leading spaces. The topic is a descriptive term on what we test.

Each test starts with a setup function `def setup(self)` we declare a setup that is run prior to each test being executed. Other functions will use the setup prior to execution.

A function includes one or multiple asserts that check if a particular test succeeds and reports this to nose to expose the information if a test succeeds or fails, when running it

Note that all nosetest functions start with a `HEADING()` in the body which conveniently prints a banner with the function name and thus helps in debugging in case of errors.

Invocation is simply done with the comment lines you see on top that you will include.

in our case the test is called `test_key.py` so we include on the top

```
#####
# pytest -v --capture=no tests/test_key.py
# pytest -v tests/test_key.py
# pytest -v --capture=no -v --nocapture tests/test_key.py:Test_key.<METHODNAME>
#####
```

You can then execute the test with either command. More information is printed with the command

Make sure that you place this comment in your tests.

The following is our simple nosetests for key. The file is stored at `tests/test_key.py`

First, we import the needed classes and methods we like to test. We define a class, and then we define the methods, such as the setup and the actual tests.

you run it with

```
$ pytest -v --capture=no tests/test_key.py
```

```
#####
# pytest -v --capture=no tests/test_key.py
# pytest -v tests/test_key.py
# pytest -v --capture=no -v --nocapture tests/test_key.py:Test_key.<METHODNAME>
#####
from pprint import pprint
from cloudmesh.common.Printer import Printer
from cloudmesh.common.util import HEADING
from cloudmesh.management.configuration.SSHkey import SSHkey
from cloudmesh.management.configuration.config import Config

@pytest.mark.incremental
class TestKey:

    def setup(self):
        self.sshkey = SSHkey()

    def test_01_key(self):
        HEADING()
        pprint(self.sshkey)
        print(self.sshkey)
        print(type(self.sshkey))
        pprint(self.sshkey.__dict__)

        assert self.sshkey.__dict__ is not None

    def test_02_git(self):
        HEADING()
        config = Config()
        username = config["cloudmesh.profile.github"]
        print("Username:", username)
        keys = self.sshkey.get_from_git(username)
        pprint(keys)
        print(Printer.flatwrite(keys,
                                sort_keys=("name"),
                                order=["name", "fingerprint"],
                                header=["Name", "Fingerprint"])
              )

        assert len(keys) > 0
```

The output with `pytest tests/test_key.py` does not provide any detail, but just reports if tests fail or succeed.

```
-----
Ran 2 tests in 0.457s
OK
```

The output with `nosetests -v tests/test_key.py` results in

```
tests.test_key.TestName.test_01_key ... ok
tests.test_key.TestName.test_02_git ... ok
-----
```

(continues on next page)

(continued from previous page)

```
Ran 2 tests in 1.072s
```

```
OK
```

During development phase you want to use `nosetests -v --nocapture tests/test_key.py`

Which prints all print statements also

22.1 Amazon Web Services (AWS) Account Creation Tutorial

Amazon Web Services provides a wide variety of cloud-based products including analytics, application integration, AR and VR, cost management, blockchain, business applications, compute, customer engagement, database, developer tools, end user computing, game tech, IoT, machine learning, management and governance, media services, migration and transfer, mobile, networking and content delivery, robotics, satellite, security, identity and compliance, and storage. Here at cloudmesh, we develop services through providers to support your utilization of many of these products.

- **Amazon Elastic Compute Cloud (EC2)** Amazon EC2 is web service that enables users to perform elastic web-scalable computing while having complete control over instances. It is integrated with most AWS services such as Amazon S3, RDS, and VPC.
- **Amazon Simple Storage Service (S3)** Amazon S3 an object storage service that offers a wide range of storage classes.

This page is a step-by-step guide on how to create an AWS account through the AWS webpage.

22.1.1 Step-by-Step Guide

First, we go to the AWS website: <https://aws.amazon.com>. Click on `Create an AWS Account`.



1 **Sign up for an AWS account**
Instantly get access to the AWS Free Tier

2 **Learn with 10-Minute Tutorials**
Explore and learn with simple tutorials

3 **Start building with AWS**
Begin building with step-by-step guides to help you launch your AWS project

Explore Our Products

Open "https://portal.aws.amazon.com/gp/aws/developer/registration/index.html?nc2=h_ct&src=header_signup" in a new tab



Image

This will direct you to the account creation page. Fill out your information and click `Continue`.

The screenshot shows the 'Create an AWS account' page. On the left, there's a heading 'AWS Accounts Include 12 Months of Free Tier Access' and a note about including use of Amazon EC2, Amazon S3, and Amazon DynamoDB. On the right, there's a registration form with fields for 'Email address', 'Password', 'Confirm password', and 'AWS account name'. The 'AWS account name' field contains the text 'cloudmesh_tutorial'. Below the form is a yellow 'Continue' button and a link to 'Sign in to an existing AWS account'. At the bottom, there's a copyright notice for 2019 Amazon Web Services, Inc. and links to 'Privacy Policy' and 'Terms of Use'.

Image

At this page, you will need to fill out your contact information. You can choose `Professional` or `Personal` as your account type. Here in this tutorial, we selected `Personal`. Read the *AWS Customer Agreement*, and check the box if agreed. Click on `Create Account` and `Continue` to continue.

portal.aws.amazon.com

Contact Information

All fields are required.

Please select the account type and complete the fields below with your contact details.

Account type ⓘ

☐ Professional ☒ Personal

Full name

cloudmesh_tutorial

Phone number

Country/Region

United States

Address

700 N Woodlawn Ave

Apartment, suite, unit, building, floor, etc.

City

Bloomington

State / Province or region

Indiana

Postal code

47408

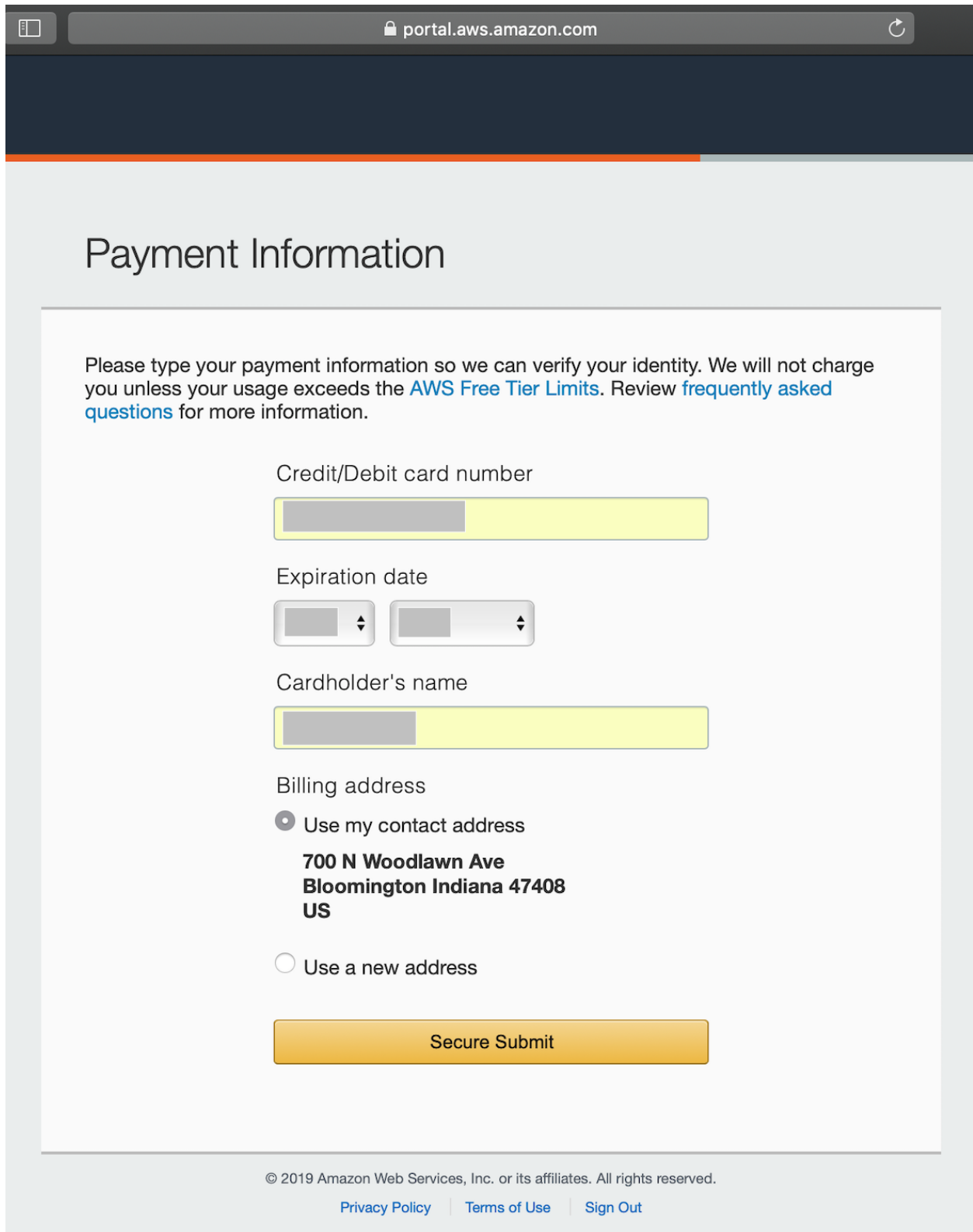
☒ Check here to indicate that you have read and agree to the terms of the [AWS Customer Agreement](#)

Create Account and Continue

Image

Fill out your payment information and proceed.

We will not charge you unless your usage exceeds the AWS Free Tier Limits. - Amazon AWS



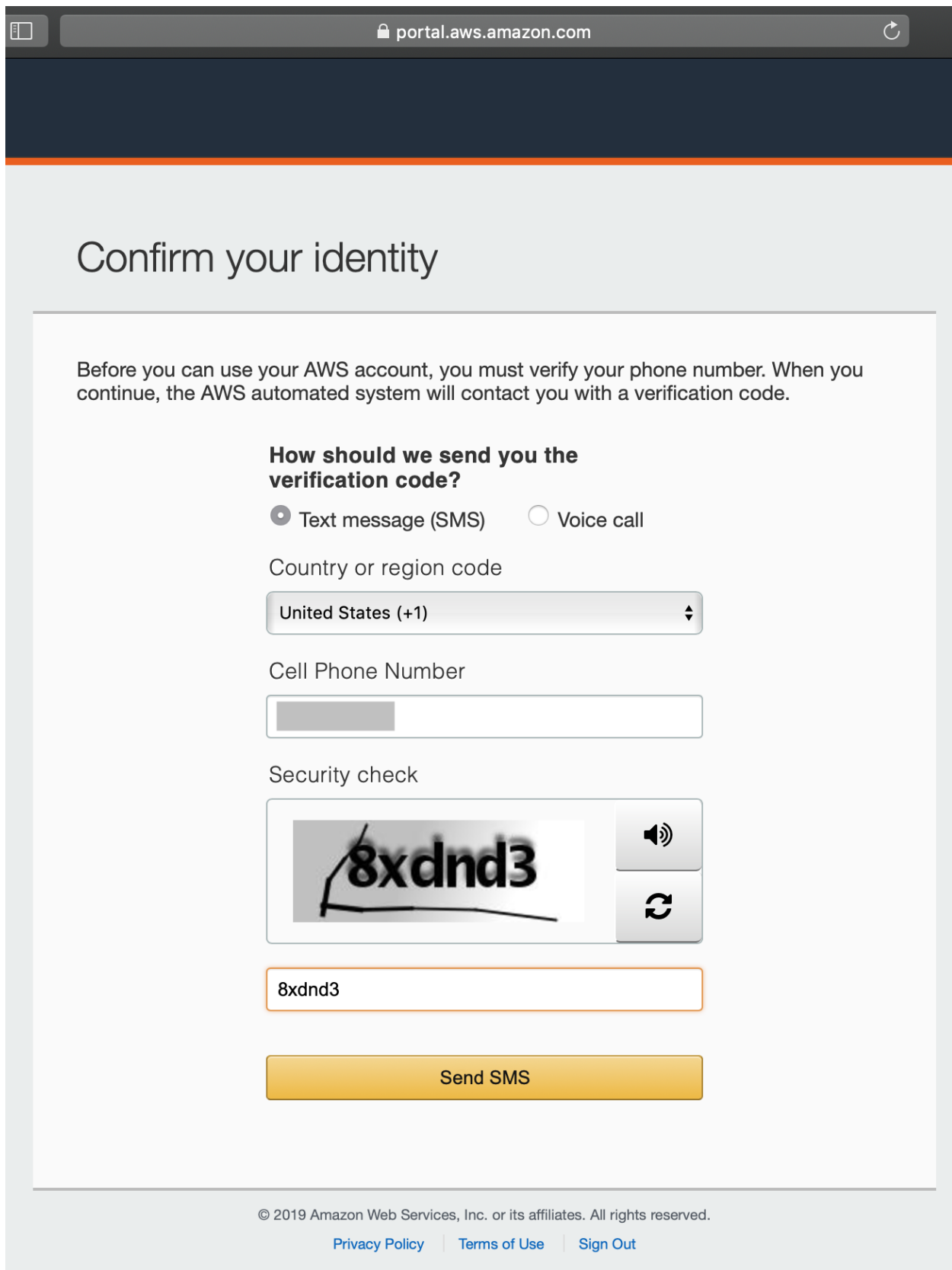
The screenshot shows a web browser window with the address bar displaying 'portal.aws.amazon.com'. The page title is 'Payment Information'. Below the title, a message states: 'Please type your payment information so we can verify your identity. We will not charge you unless your usage exceeds the [AWS Free Tier Limits](#). Review [frequently asked questions](#) for more information.' The form contains the following fields and options:

- Credit/Debit card number:** A single-line text input field.
- Expiration date:** Two separate dropdown menus for month and year.
- Cardholder's name:** A single-line text input field.
- Billing address:** A section with two radio button options:
 - ☒ Use my contact address
 - ☐ Use a new address
- Address details:** Below the radio buttons, the address is displayed as:
700 N Woodlawn Ave
Bloomington Indiana 47408
US
- Secure Submit:** A large orange button at the bottom of the form.

At the bottom of the page, there is a copyright notice: '© 2019 Amazon Web Services, Inc. or its affiliates. All rights reserved.' and three links: [Privacy Policy](#), [Terms of Use](#), and [Sign Out](#).

Image

Next you need to confirm your identity. You can choose either `Text message (SMS)` or `Voice call` to receive your verification code. Here we choose `Text message (SMS)`. Enter your phone number and the security check code and click `Send SMS`.



The screenshot shows a web browser window with the address bar displaying 'portal.aws.amazon.com'. The page title is 'Confirm your identity'. Below the title, a message states: 'Before you can use your AWS account, you must verify your phone number. When you continue, the AWS automated system will contact you with a verification code.' The form asks 'How should we send you the verification code?' with two radio buttons: 'Text message (SMS)' (selected) and 'Voice call'. Below this is a dropdown menu for 'Country or region code' showing 'United States (+1)'. A text input field for 'Cell Phone Number' is present but empty. A 'Security check' section displays a distorted image of the text '8xdnd3' with a hand-drawn line through it. To the right of the image are icons for audio playback and a refresh button. Below the image, a text input field contains the characters '8xdnd3'. A large orange button labeled 'Send SMS' is at the bottom of the form. The footer contains the copyright notice '© 2019 Amazon Web Services, Inc. or its affiliates. All rights reserved.' and links for 'Privacy Policy', 'Terms of Use', and 'Sign Out'.

portal.aws.amazon.com

Confirm your identity

Before you can use your AWS account, you must verify your phone number. When you continue, the AWS automated system will contact you with a verification code.

How should we send you the verification code?

☒ Text message (SMS) ☐ Voice call

Country or region code

United States (+1)

Cell Phone Number

Security check

8xdnd3

8xdnd3

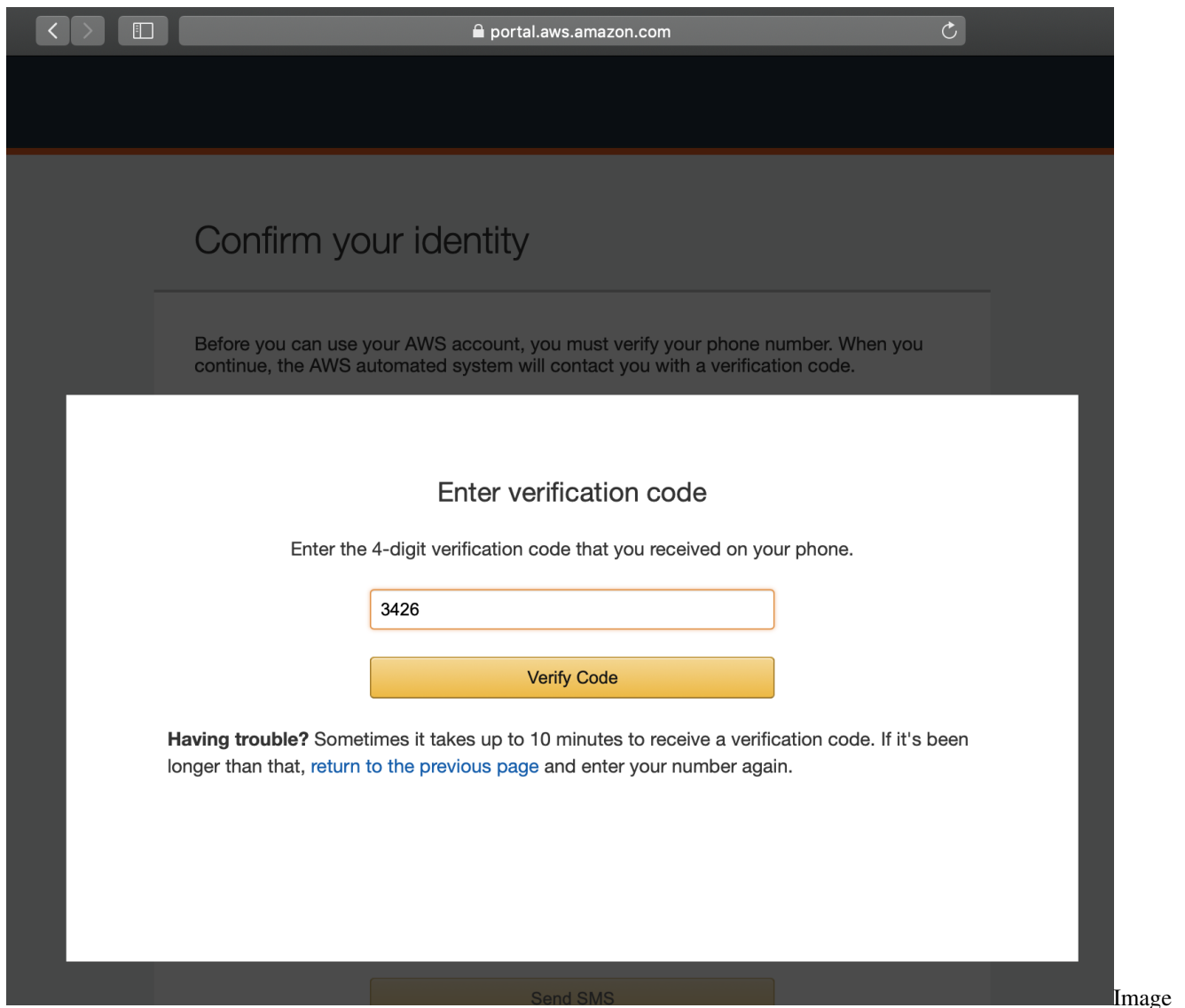
Send SMS

© 2019 Amazon Web Services, Inc. or its affiliates. All rights reserved.

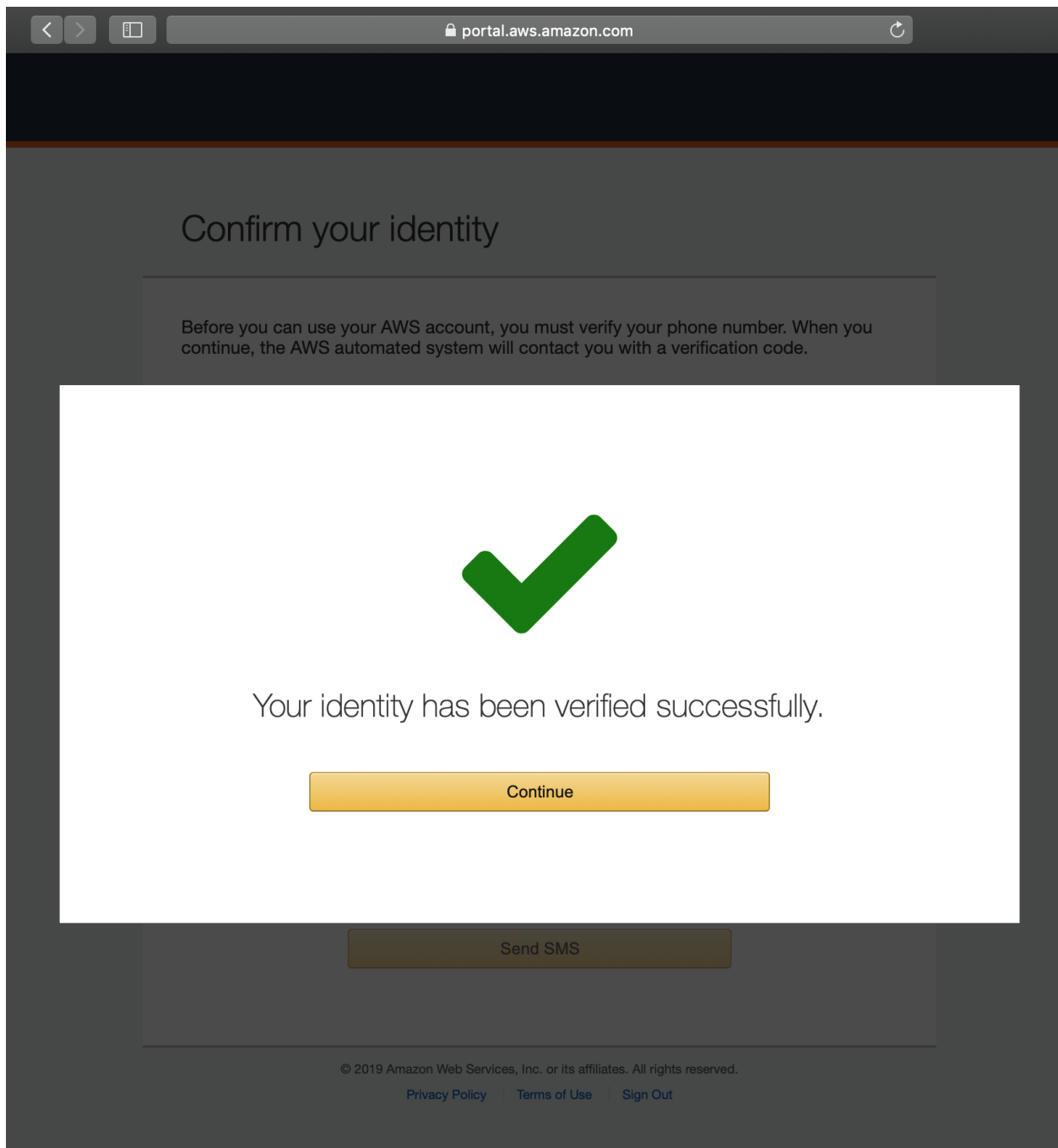
[Privacy Policy](#) | [Terms of Use](#) | [Sign Out](#)

Image

Enter the 4-digit verification code you received from your phone, and click on `Verify Code`.

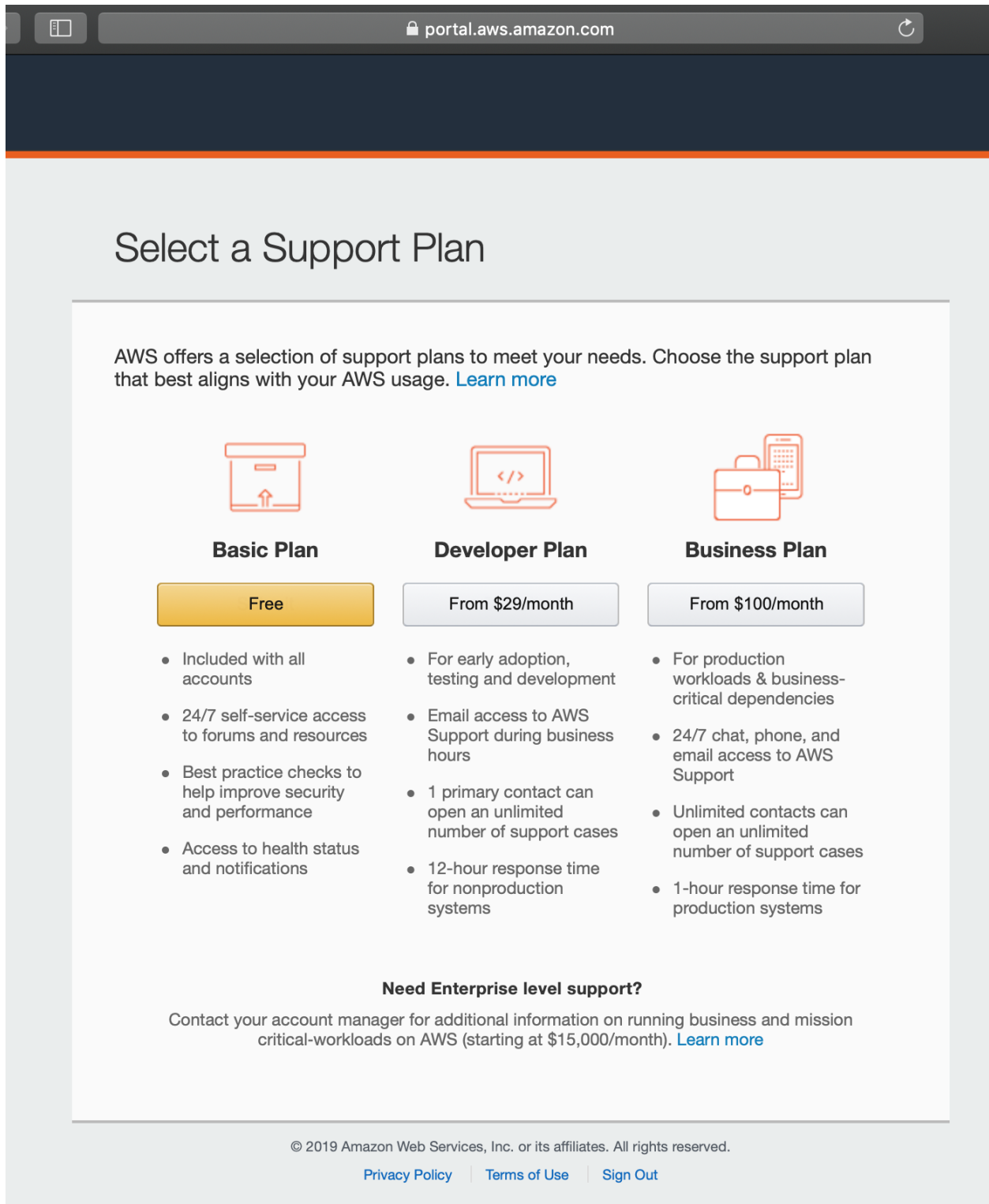


If you entered the correct verification code, you will see this page. Click on `Continue`.



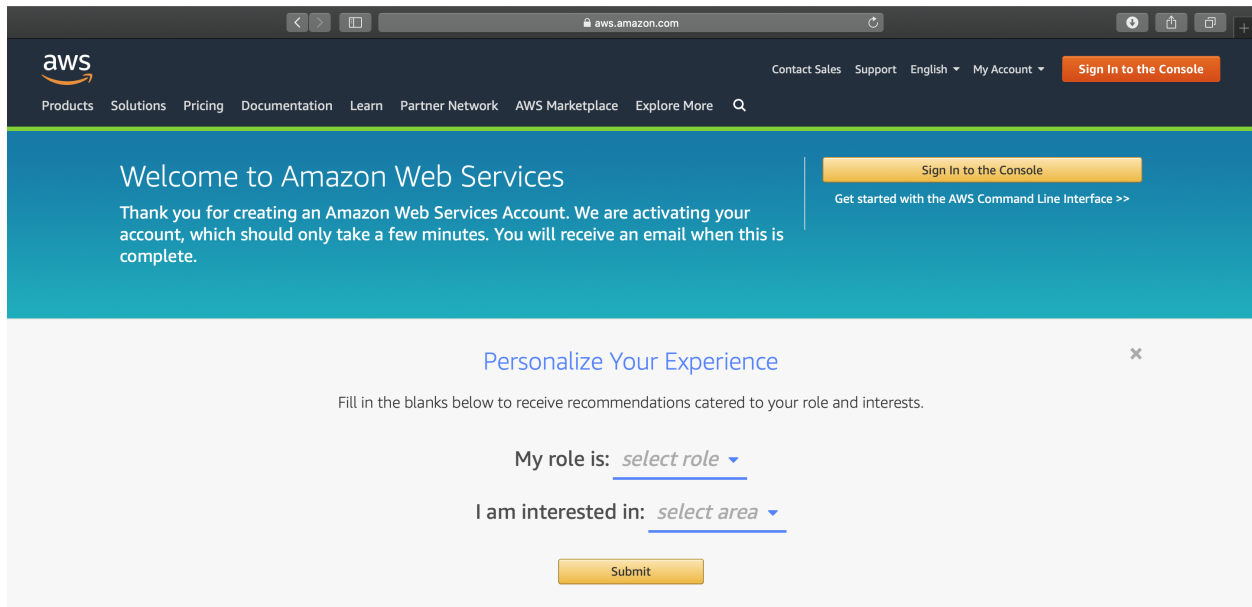
Image

You will need to choose your support plan. For the tutorial, we chose Amazon's free tier `Basic Plan`.



Image

Congratulations! You have successfully created an AWS account. Now you can click on Sign In to the Console to sign in.

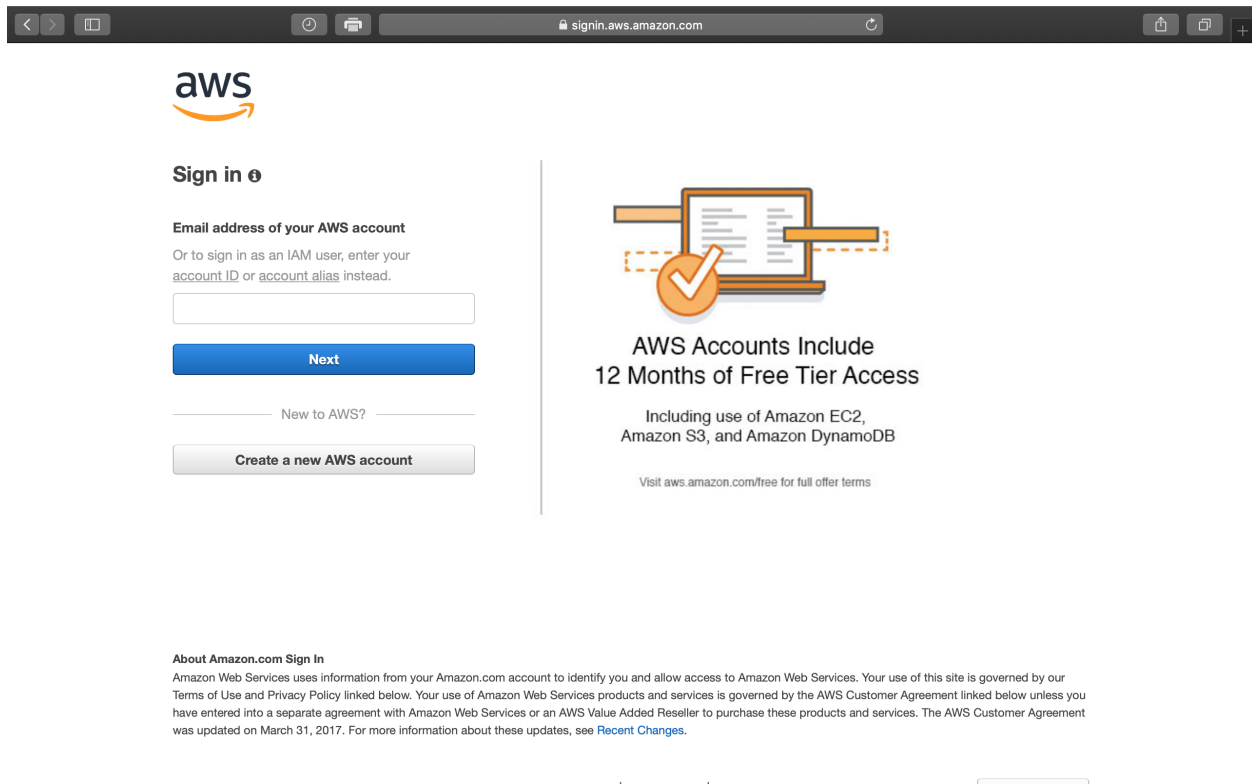


The screenshot shows the AWS Welcome page in a web browser. The browser's address bar displays `aws.amazon.com`. The page features the AWS logo and a navigation menu with links to Products, Solutions, Pricing, Documentation, Learn, Partner Network, AWS Marketplace, and Explore More. A prominent orange button labeled "Sign In to the Console" is located in the top right corner. Below the navigation bar, a large teal banner contains the text "Welcome to Amazon Web Services" and a message: "Thank you for creating an Amazon Web Services Account. We are activating your account, which should only take a few minutes. You will receive an email when this is complete." To the right of this message is another "Sign In to the Console" button and a link "Get started with the AWS Command Line Interface >>". Below the banner, a section titled "Personalize Your Experience" includes a sub-header "Fill in the blanks below to receive recommendations catered to your role and interests." and two dropdown menus: "My role is: select role" and "I am interested in: select area". A "Submit" button is positioned below these dropdowns.



Image

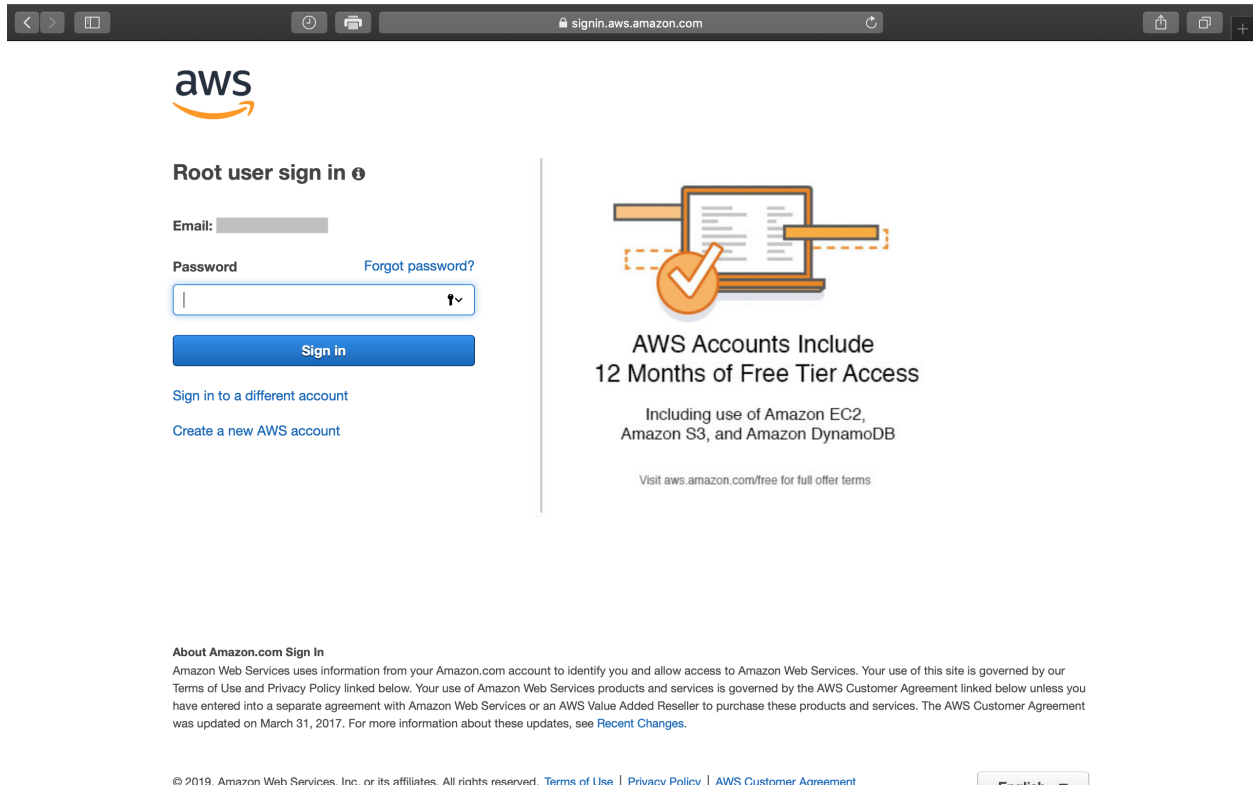
Enter the email address you used for registration, and click on Next.



The screenshot shows the AWS Sign In page in a web browser. The browser's address bar displays `signin.aws.amazon.com`. The page features the AWS logo and a "Sign in" heading. Below the heading, there are two main sections. The left section, titled "Email address of your AWS account", includes a sub-header "Or to sign in as an IAM user, enter your account ID or account alias instead." and a text input field. Below the input field is a blue "Next" button. The right section, titled "AWS Accounts Include 12 Months of Free Tier Access", includes a sub-header "Including use of Amazon EC2, Amazon S3, and Amazon DynamoDB" and a link "Visit aws.amazon.com/free for full offer terms". At the bottom of the page, there is a section titled "About Amazon.com Sign In" with a paragraph of text and a link "Recent Changes".

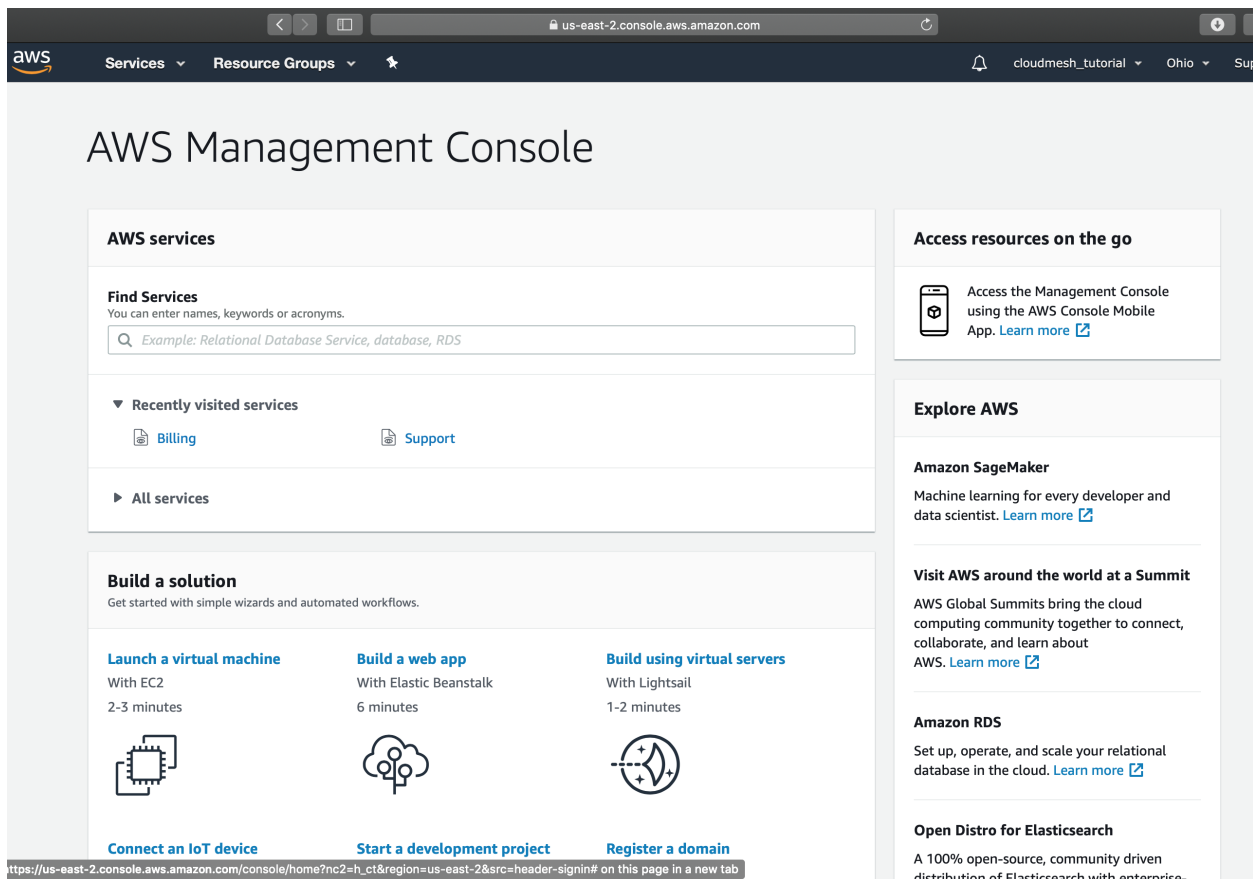
Image

Enter the password you used for registration, and click on Sign In.



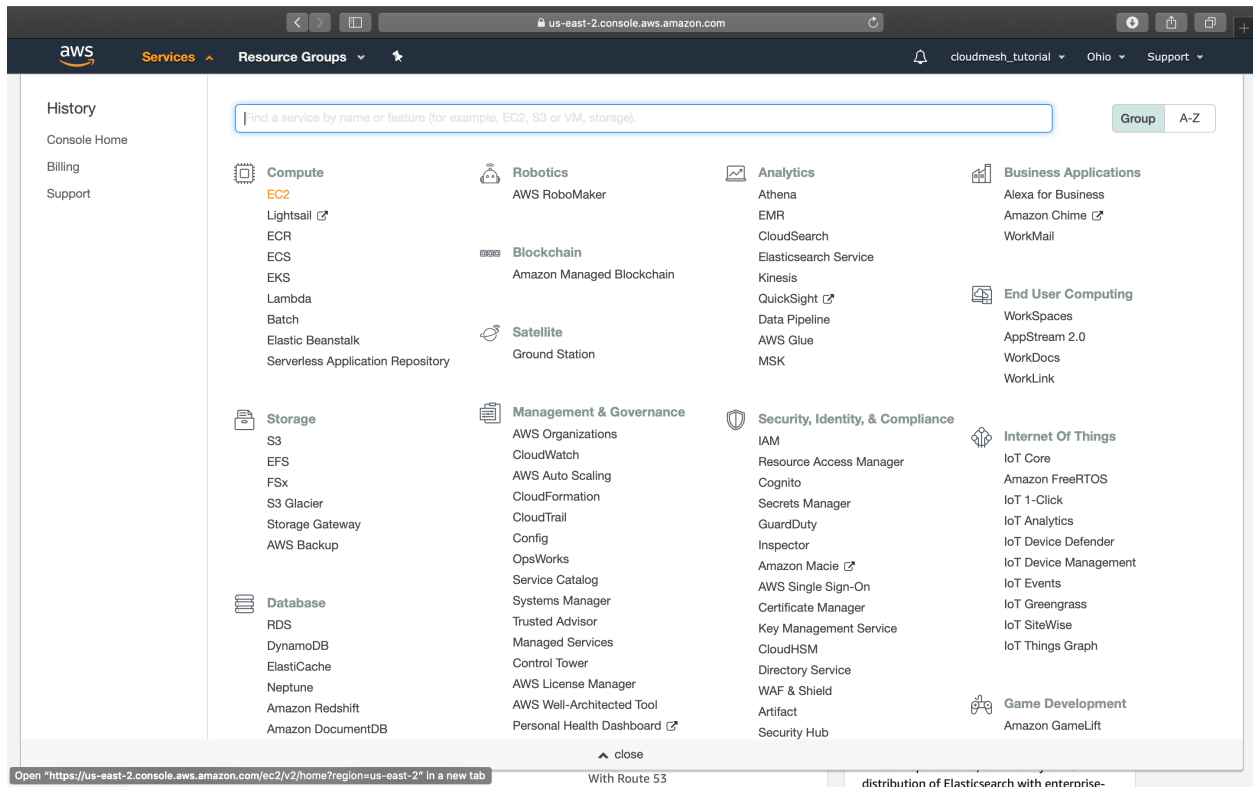
Image

Now you've successfully signed in to the AWS Management Console.



Image

You can click on `Services` to explore AWS services through their webpage.



Image

You can also start managing your account and instances through our cloudmesh services! :)

22.1.2 References

- https://aws.amazon.com/?nc2=h_lg
- https://aws.amazon.com/ec2/?nc2=h_m1
- <https://aws.amazon.com/s3/?c=23&pt=1>

22.2 Azure Blob Storage and Account Creation

22.2.1 Azure Blob Storage

Azure Storage is Microsoft's cloud storage solution for modern data storage scenarios. Azure Blob storage is Microsoft's object storage solution for the cloud. Blob storage is optimized for storing massive amounts of data.

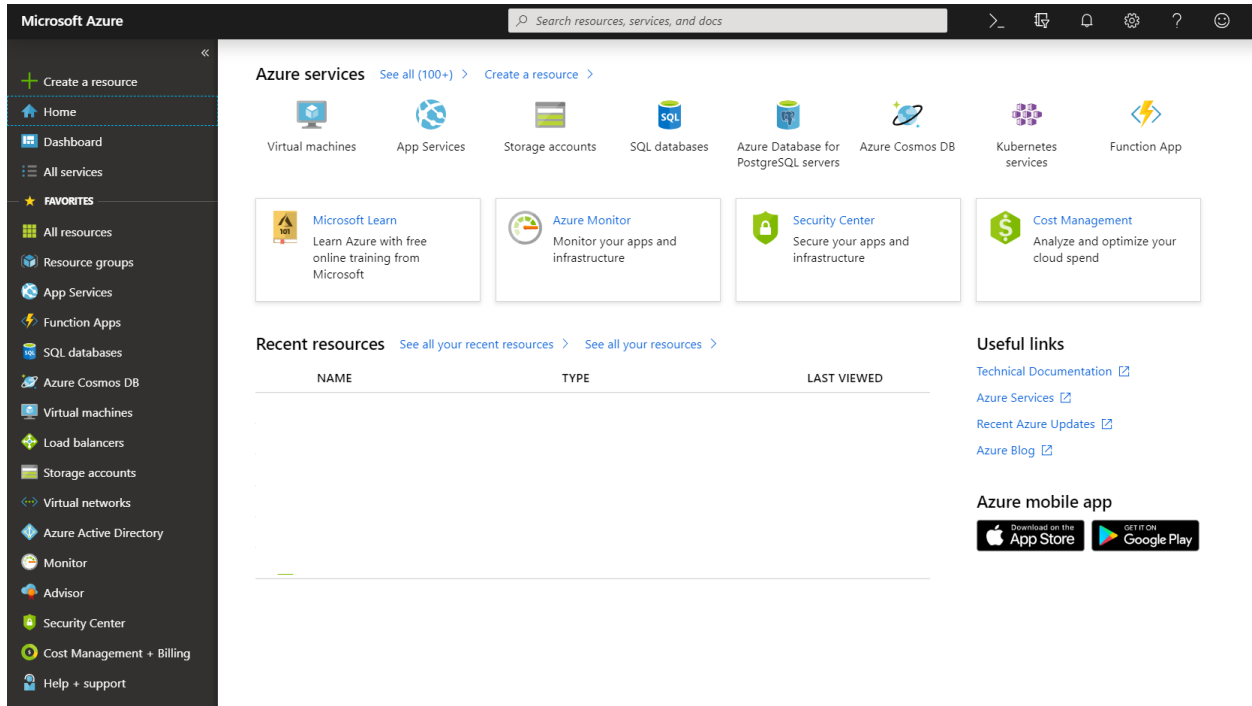
Blob storage offers three types of resources:

- The storage account
- A container in the storage account
- A blob in a container

22.2.2 Azure Storage account creation

Following are the steps to create Azure storage account

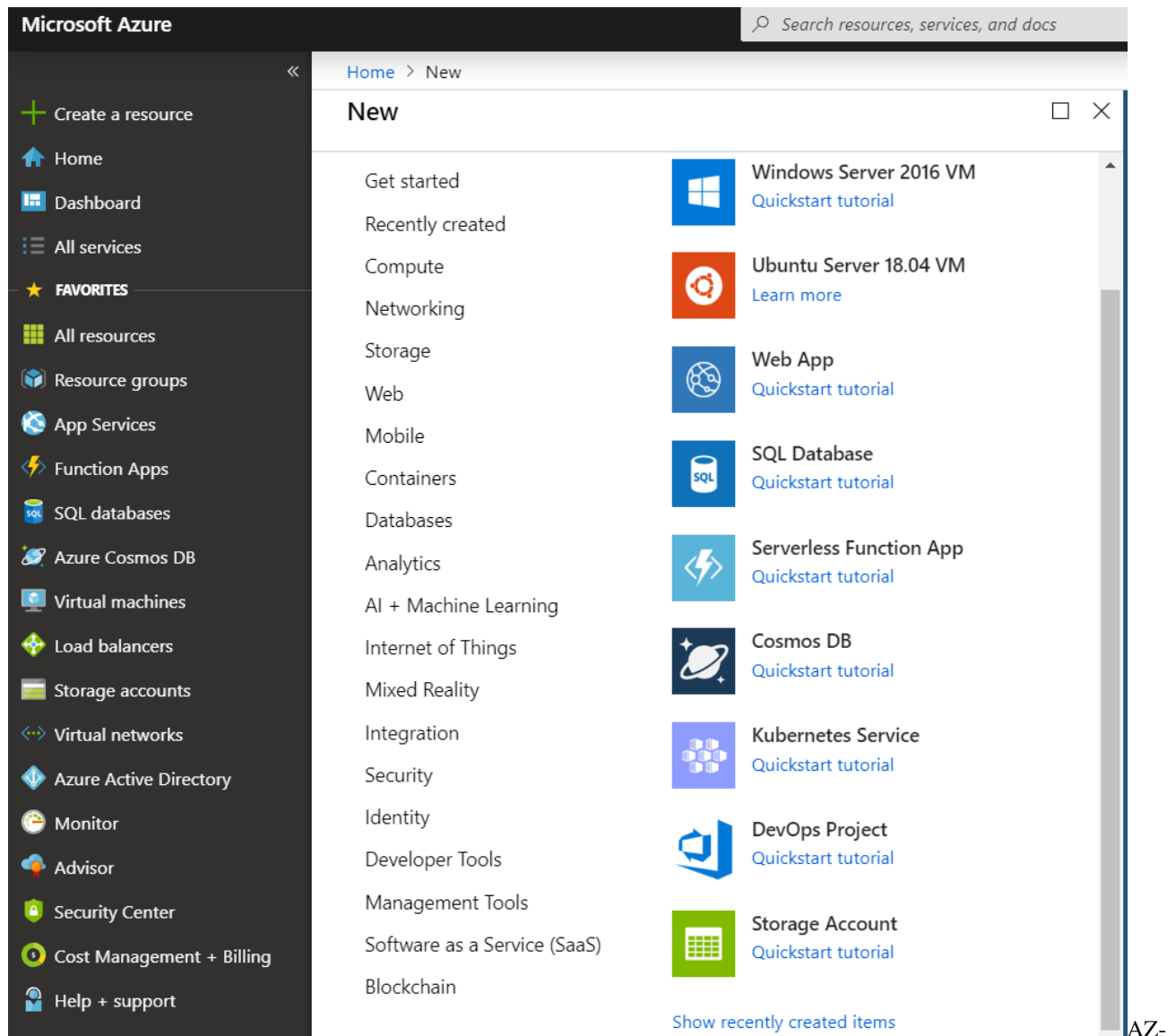
- Prerequisites
Create a free Azure cloud services subscription account.
- Log in to the [Azure Portal](#).



Portal {#fig:az-portal}

AZ-

- In the Azure portal, click on Create a resource on the top left corner.



Resource{#fig:az-resource}

- Select Storage Account from the options shown

Microsoft Azure

Search resources, services, and docs

Home > New > Create storage account

Create storage account

Basics Advanced Tags Review + create

Azure Storage is a Microsoft-managed service providing cloud storage that is highly available, secure, durable, scalable, and redundant. Azure Storage includes Azure Blobs (objects), Azure Data Lake Storage Gen2, Azure Files, Azure Queues, and Azure Tables. The cost of your storage account depends on the usage and the options you choose below. [Learn more](#)

PROJECT DETAILS

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

* Subscription

* Resource group [Create new](#)

INSTANCE DETAILS

The default deployment model is Resource Manager, which supports the latest Azure features. You may choose to deploy using the classic deployment model instead. [Choose classic deployment model](#)

* Storage account name

* Location

Performance ☒ Standard ☐ Premium

Account kind

Replication

[Review + create](#) [Previous](#) [Next : Advanced >](#)

AZ-

Account{#fig:az-account}

- Select the subscription in which to create the storage account.
- Under the Resource group field, select Create new. Enter a name for your new resource group.
- Next, enter a name for your storage account.
- Select a location for your storage account, or use the default location.
- Select create

After the completion of above steps, Azure blob storage service will be ready for use. As a first step, a Container should be created in the Blob storage. A container organizes a set of blobs, similar to a directory in a file system. A default Container should be set in the `cloudmesh4.yaml` file, details of which are outlined [here](#)

IMPORTANT NOTE:

The free Azure account needs to be upgrade to a pay-as-you-go subscription after first 30 days to get continued access to free products—some for the first 12 months.

Refer to more details here - <https://azure.microsoft.com/en-us/free/>

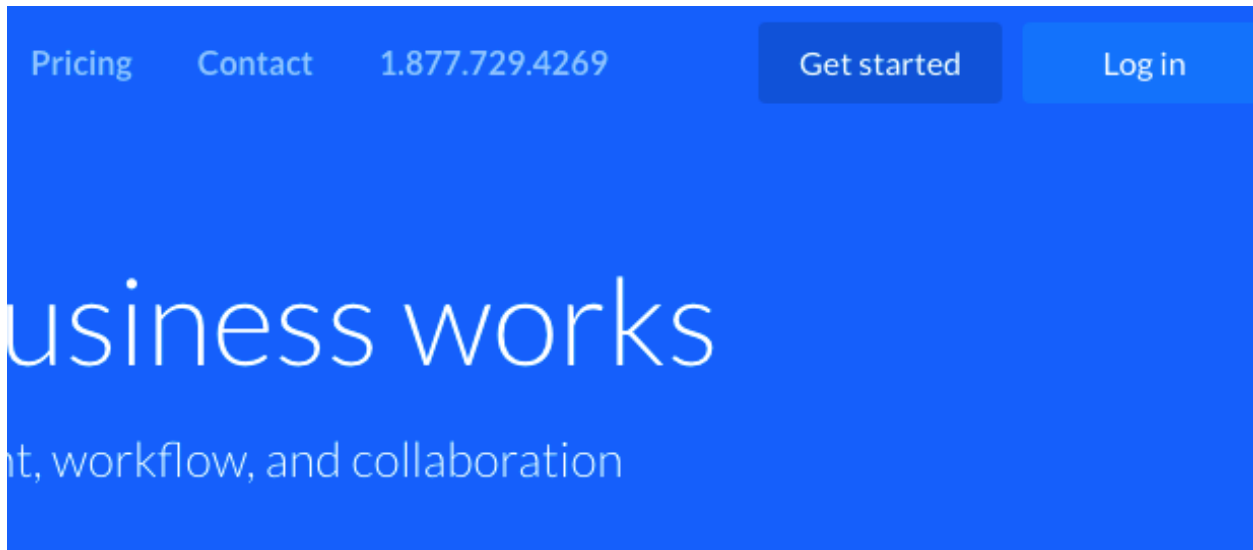
22.2.3 References

- <https://docs.microsoft.com/en-us/azure/storage/common/storage-introduction>
- <https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blobs-overview>

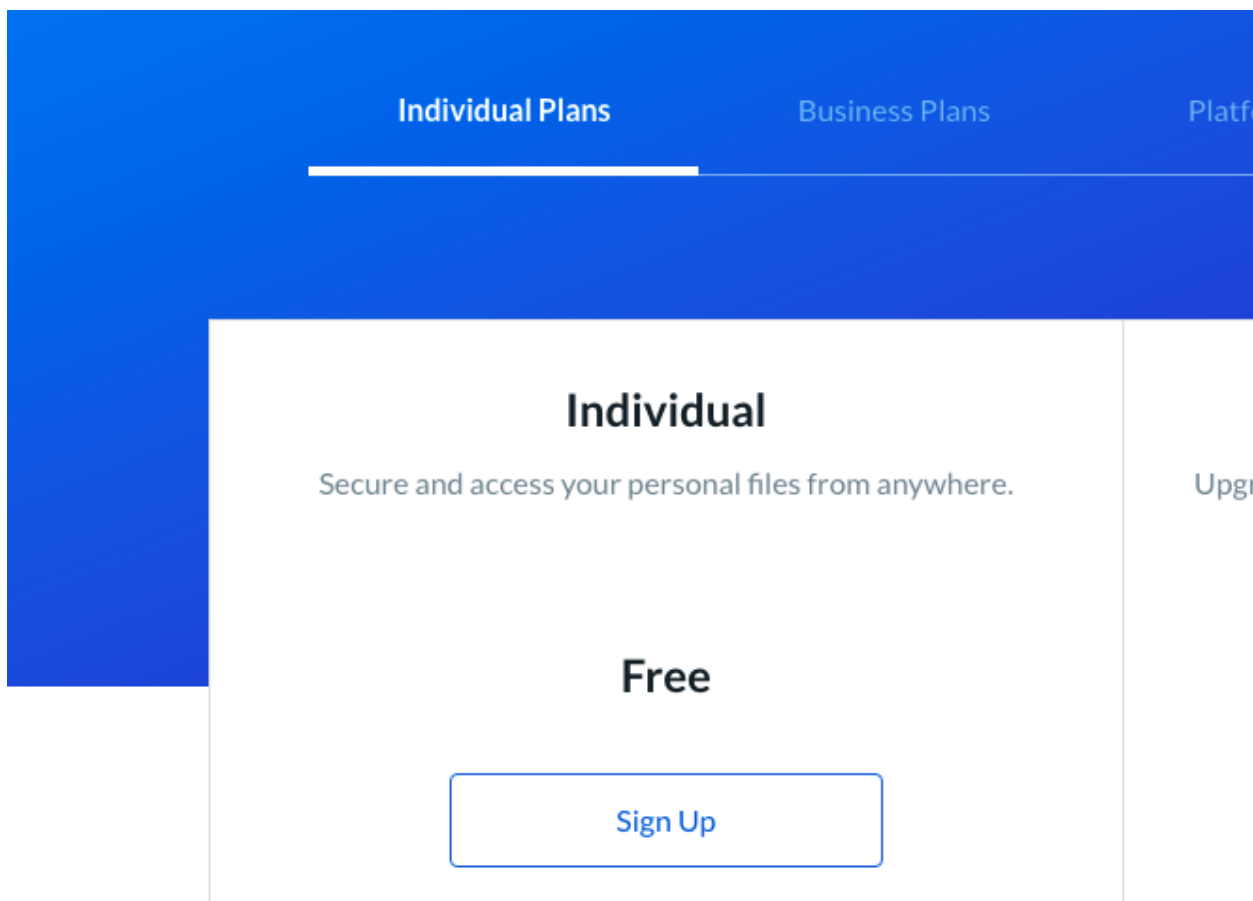
22.3 Setting Up Your Box Account

22.3.1 Sign up

In the top right hand corner of the box [homepage](#) click on the Get Started button.



From the plans page, select the Individual tab and then click on the free option.



Fill out the required information and click `Submit`. You will receive a confirmation email with a link to verify your account.

Your Information

Full Name

Email Address

Password


 

Confirm Password

Phone Number

Country

 ☐

I'm not a robot



reCAPTCHA
Privacy - Terms

Submit

Once you have verified your account and signed in, you will be taken to a page that asks you about how you are using Box. You may fill this out or click `Skip this` and go straight to Box below the `Next` button.



`Skip this and go straight to Box`

`skip`

22.3.2 Creating an app

Navigate to the [developer console](#) and select `Create New App`. You will need to select what type of application you are building and an authentication method for your app and then enter an app name (you can change this later). Once your app has been created, click `View App`. You will then need to click the profile button in the top right corner of the page, and go to `Account Settings`. Scroll down to the `Authentication` section and click `Require 2-step verification for unrecognized logins`, then follow the prompts.

22.3.3 Authentication with JWT

In the `Configuration` panel of the `Developer Console`, scroll down to the section titled `Add and Manage Public Keys` and click `Generate a Public/Private Keypair`:

Add and Manage Public Keys

Generate an RSA keypair to sign and authenticate the JWT request made by your app or upload your own public key.

[Learn to generate your own RSA keypair.](#)

Note: Box does not store your private key, so make sure you save the downloaded file if you are using our generate button.

Add a Public Key

Generate a Public/Private Keypair

Add Key

Once you have generated a keypair, a `config.json` file will automatically download. Save this file in a secure location as you will need it for authentication purposes.

Box

22.3.4 References

- <https://developer.box.com/reference>

22.4 Google Storage Providers

22.4.1 Google Drive

Google Drive is a file storing platform where an user can store all his/her files in the google drive. Here files can be of any form ranging from documents to audio / video or image files. In free account each user will be given around 15 GB of free data space to be stored. We can create folders and subfolders in the Google Drive to store our data.

Each file will be stored in Google cloud with a unique URL and it's up to the user to make the file sharable or not. Google Drive is reliable and if an user has different devices and if he/she wants to access those files then Google Drive is needed in this case as he can have access to his file as all his files are stored in the cloud. The user does not need to install any kind of software in order to view these files.

22.4.2 Google Docs

Google docs is especially designed for viewing or editing or sharing the documents like Docs, Sheets, Slides, Forms. No need to install any software to access or edit these. And google doc can be sharable with editable option. There is an automatic mechanism to convert Microsoft documents to Google Docs.

- **Google Docs:** Google docs is a broader term for Google sheets, Google slides and Google forms.
- **Google Sheets:** Just like Microsoft excel sheet Google sheets has almost all of the functionalities. Google sheets can be shared with other people and can concurrently work on it and can edit it. We can change the font size, type as we want. We can use the formulas to calculate some mathematical expressions. This can be readily transformed to .csv or .xlsx format.
- **Google Slides:** Just like Microsoft PowerPoint presentation, Google has Google slides. We can do small animations, transformations of slides. This can be shared with other people to edit this on real time basis. We can change the font size, type of these as we want.
- **Google Forms:** Out of all Google docs this is the most powerful tool when anyone wants to collect data from other people. One can make a Google form and can share it via the link. The one who opens this link will see a form to fill. We can add many different types of survey questions with multiple choice or Multiple options, or text entries or date entries or choose from a list entry. This google forms can be used to conduct surveys within a close group like teachers, students or employees.

In a broader sense Google docs is just a subset of Google Drive

22.4.3 Python Google Drive API

Step-by-step process

Before writing the Python interface for Google Drive, we need to setup an email account, with that email account we will get a set of google services and one of them is Google Drive with 15 GB overall storage.

After that we need to go through the Google Drive Quick start guide:

<<https://developers.google.com/drive/api/v3/quickstart/python> >

There we can see Enable API option as shown in the next picture:

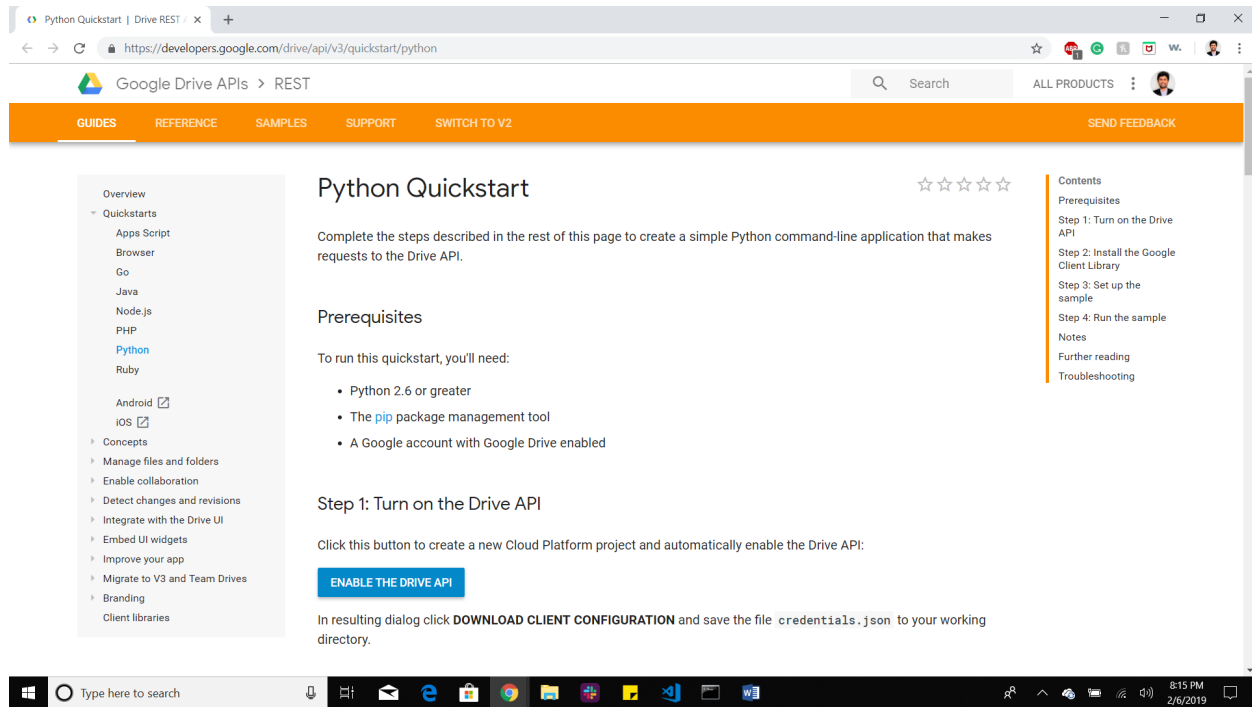
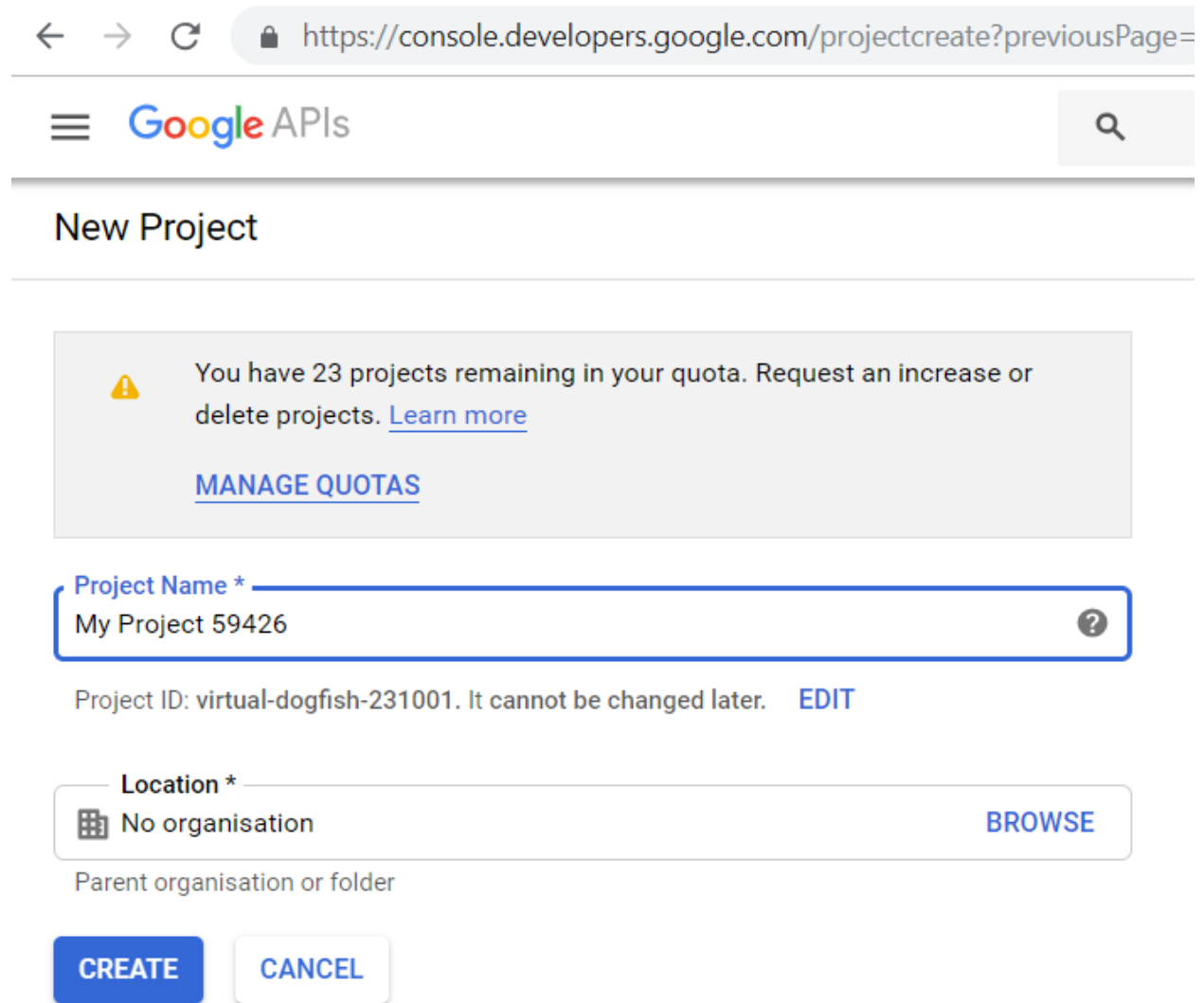


Image1


Once we enable that we will get `credentials.json` file where all of our credentials are stored that can be used to communicate with our Google Drive through Python Interface. After that, we will be redirected to a page where we need to create our own project as shown in the next picture:



← → ↻ 🔒 https://console.developers.google.com/projectcreate?previousPage=


≡ Google APIs 🔍

New Project

 You have 23 projects remaining in your quota. Request an increase or delete projects. [Learn more](#)
[MANAGE QUOTAS](#)

Project Name *
My Project 59426 ?

Project ID: virtual-dogfish-231001. It cannot be changed later. [EDIT](#)

Location *
 No organisation [BROWSE](#)

Parent organisation or folder

[CREATE](#) [CANCEL](#)

image2

As we see next we need to select Google Drive API from here

← → ↻ <https://console.developers.google.com/apis/credentials/wizard?project=my-project-43289>

Unlock more of Google Cloud Platform by upgrading now (\$300.00 credit and 354 days left in your free trial).

☰ Google APIs My Project 43289 🔍

API APIs & Services

Dashboard

Library

Credentials

Credentials

Add credentials to your project

- Find out what kind of credentials you need

We'll help you set up the correct credentials.
If you want you can skip this step and create an [API key](#), [client ID](#) or [service account](#).

Which API are you using?

Different APIs use different auth platforms and some credentials can be restricted to only call certain APIs.

Google Drive API

Where will you be calling the API from?

Credentials can be restricted using details of the context from which they're called. Some credentials are unsafe to use in certain contexts.

 - Web browser (Javascript)
 - Web server (e.g. node.js, Tomcat)
 - Android
 - iOS
 - Chrome application
 - Other UI (e.g. Windows, CLI tool)
 - Other non-UI (e.g. cron job, daemon)

What credentials do I need?
- Get your credentials

Cancel

After that, we need to obtain the `client_secret` file as shown next: (The file that is downloaded as `client_id.json` needs to be renamed as `client_secret.json`)

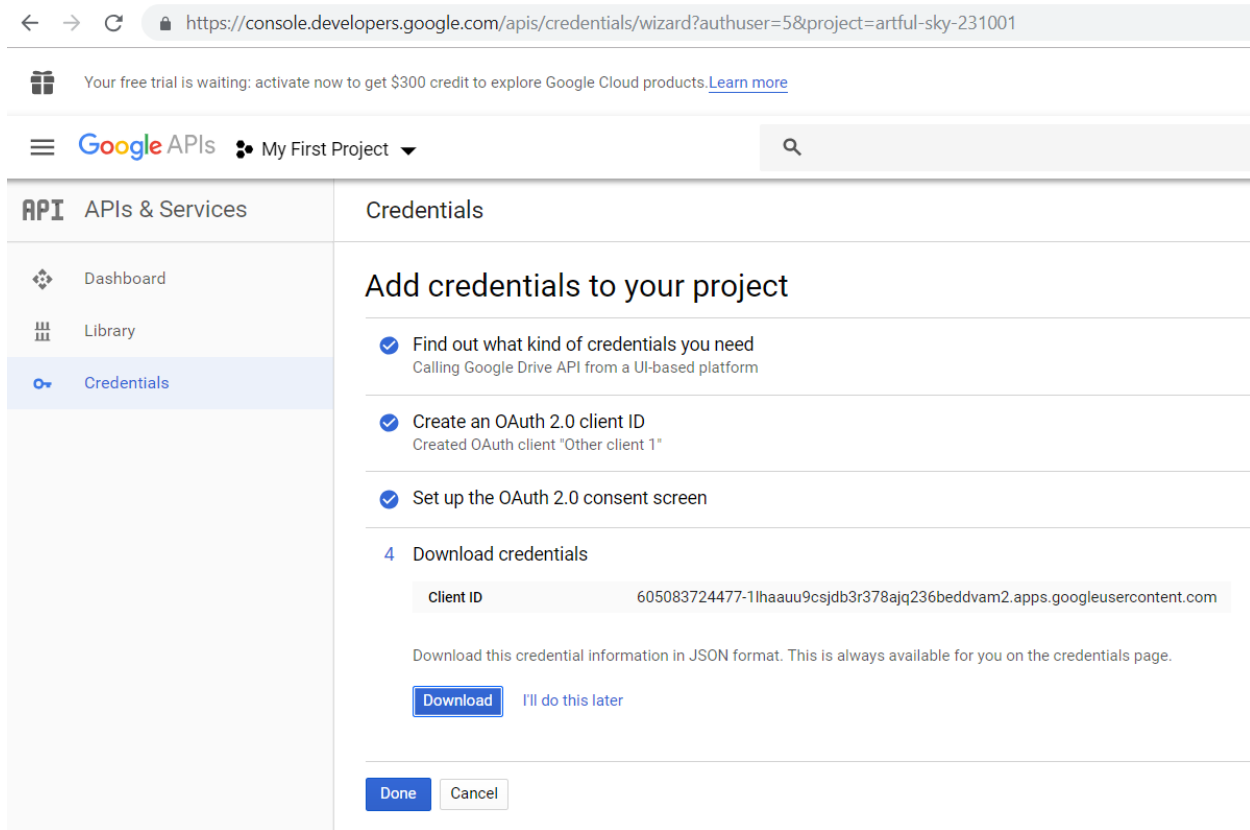
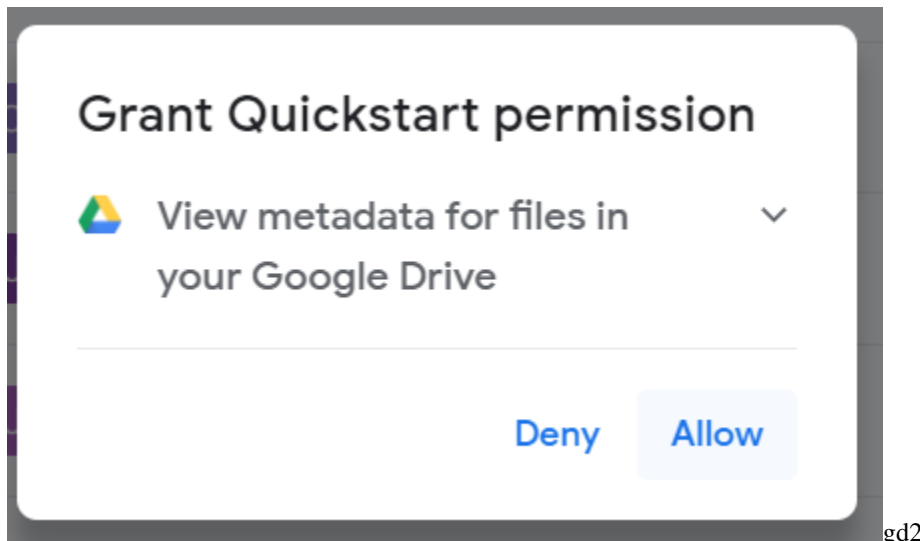


image3

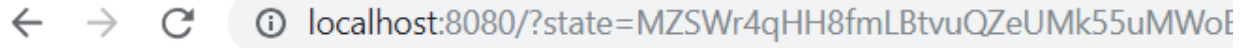
After this we need to click Done otherwise it would not set the Google Drive API

After this if we run Authentication.py we will be redirected to our default browser to put our our login id and password and after that it asks to authenticate our credentials. If we allow that as shown next:



gd2

We will get the screen something like given next (as the authentication pipeline has been completed).



The authentication flow has completed, you may close this window.

gd3

If the authentication flow is completed then the Authentication.py will create a google-drive-credentials.json file in .credentials folder. This file can be used for future purposes. If we delete this file then the Authentication.py will again ask for login id and password and again create that file automatically.

So, now with the client_secret.json, google-drive-credentials.json and with Authentication.py and Provider.py our setup is ready

Once all these steps are done correctly, then we can use the Python program interface to transfer the files between our Python program and Google Drive.

22.4.4 References

- <https://www.cloudwards.net/how-does-google-drive-work/>
- <https://whatis.techtarget.com/definition/Google-Docs>
- <https://www.techopedia.com/definition/13626/google-docs>
- <https://www.technokids.com/blog/apps/reasons-to-use-google-forms-with-your-students/>
- <https://developers.google.com/drive/api/v3/quickstart/python>
- <https://github.com/samlopezf/google-drive-api-tutorial>
- <https://developers.google.com/drive/api/v3/manage-uploads>

22.5 Google (What is this?)

THIS WAS FOUND IN AN UNRELATED DOCUMENT IN A USERS DOCUMENTATION

- client_secret.json
- google-drive-credentials.json

If we run the Google Drive Provider.py for the **First time** then the required keys, tokens are taken from the cloudmesh4.yaml file and creates a client_secret.json file in the following path ~/.cloudmesh/gdrive/

The Authentication.py creates a .credentials folder under the following path ~/.cloudmesh/gdrive/ if it doesn't exist and creates a google-drive-credentials.json file under the following folder ~/.cloudmesh/gdrive/.credentials/

So, for the **First time** browser will be opened up automatically and asks for the Google Drive(gmail) credentials i.e., login email and password. If you provide these 2 then the Authentication step is completed and then it will create the google-drive-credentials.json and place it in ~/.cloudmesh/gdrive/.credentials/ folder.

These steps are to be followed for the first time or initial run. Once it is done then our program is set. After these steps then the program will run automatically by using these credentials stored in the respective files.

22.5.1 Note

The Google Drive API accepts these 2 files in the form of **.json file format** and not in the form of a dictionary.

22.5.2 Links

Link for additional information:

- <https://github.com/cloudmesh-community/sp19-516-130/blob/master/gdrive.md>

22.6 VM Providers (outdated)

Cm4 works straight forward with a number of providers under the assumption you have accounts on these frameworks. We demonstrate here how to start a single vm on each of these providers and list the started vms. Defaults from the configuration file are used to select images and flavors. These defaults can naturally be changed.

22.6.1 General Cloud Providers Access

We are using the python library [Apache Libcloud](#) to interact with cloud service providers. Currently, in `cms`, we could access:

- [AWS](#)
- [AZURE](#)
- any cloud service providers using *OpenStack*. For example, [Chameleon](#) and [Jetstream](#)

By using the *Apache Libcloud* API, we could do these operations for nodes in above cloud service providers:

- Start the node
- Stop the node
- Resume the node
- Suspend the node
- Destroy the node
- Create the node

Improvement: Sometimes adjustments to nodes are necessary (switch between different images/OS and service sizes). Cm4 also allow users to customize their instances across multiple providers by using refactor functions to support their management tasks.

- Resize the node
- Rebuild(with different image) the node
- Rename the node
- Revert previous operations to the node

22.6.2 General Interface


```
$ cms set cloud=<cloudname as defined in the ~/.cloudmesh/cloudmesh4.yaml>
$ cms vm start
$ cms vm list

$ cms flavor="medium"
$ cms image="ubuntu18.04"

$ cms vm start
```

22.6.3 Explicit Use with Options

```
$ cms vm start --cloud=chameleon --image=ubuntu18.04 --flavor=medium --key=~/.ssh/id_
↪rsa.bub
```

22.6.4 Vagrant

TODO

```
$ cms set cloud=vagrant
$ cms vm start
$ cms vm list
```

22.6.5 AWS

Setup and Configuration

Amazon Web Service (AWS) provided by Amazon is a secure cloud service platform, users could start any instances with selected images.

Before users use the **cms** platform to access **EC2**, they have to finish these preparations:

1. EC2 account, more information is [here](#)
2. Log in the EC2 account, update your **Access Key**.

Access Keys has two parts: **Access Key ID** and **Secret Access Key**. These **Access Keys** are the only way you could authentically access the AWS through AWS API requests. (create new Access Key: Account (right upper corner) > My Security Credentials

Access Keys > Create New Access Key)

3. **Private Key file** is a key pairs to encrypt and decrypt login information. While using **Private Key file**, there is no need to use username or password to login the instance of AWS. For sshing the instance, the ssh client would use the **private key file** instead of credential information. (create new key pairs: Network & Security (left column bar) > Key Pairs > Create Key Pair)
4. **Security Group** acts as a virtual firewall for the instance. When you launch a instance, we have to attach the **Security Group** to it for controlling the traffic in and out. So before you are using any nodes in AWS, you have to pre-define the **Security Group** that you will use. (create new Security Group: Network & Security (left column bar) > Security Group > Create Security Group)
5. **Region** is the service location where you start the instance. AWS hosts services in different regions, you should select the region where you want to start you instance.

When you finish all above things, you should update information into the block 'aws' of **cloudmesh4.yaml** file in **ETC** folder

EC2 provides On-Demand Pricing cloud services based on different CPU, Memory and Storage selections. Please visit this [page](#) for more information. In default setting, we use the latest **Ubuntu** image filled in default.image field. If you want to use other images, please update the Image ID into it.

```
$ cms set cloud=aws
$ cms vm start
$ cms vm list
```

22.6.6 Azure

Uses LibCloud's Azure ARM Compute Driver

Setup and Configuration

Install Azure CLI

Download and install according to your platform.

Make sure subscription is registered for compute services

```
az provider register --namespace Microsoft.Compute
```

Service principal

Full documentation on creating service principals. The Azure ARM Driver does not appear to support certificate based principals at this time.

Create Principal

```
az ad sp create-for-rbac --name cm-admin-pw --password <SECRET>
```

Add Owner role.

```
az role assignment create --assignee <APP_ID> --role Owner
```

Note: <APP_ID> is provided in the output when the principal is created

```
$ cms set cloud=azure
$ cms vm start
$ cms vm list
```

22.6.7 OpenStack

OpenStack is an Infrastructure service that allows users to utilize computing resource in cloud service platform through virtual environments.

Chameleon Cloud provides an OpenStack installation of version 2015.1 (Kilo) using the KVM virtualization technology at the KVM@TACC site. It is important to make sure you are visiting the **KVM@TACC** site so as to get proper installation. Learn more [here](#) to properly set up your account before proceed to your journey with **cms**.

Jetstream

TODO

```
$ cms set cloud=jetstream
$ cms vm start
$ cms vm list
```

Chameleon Cloud

```
$ cms set cloud=chameleon
$ cms vm start
$ cms vm list
```

Cybera

TODO

```
$ cms set cloud=cybera
$ cms vm start
$ cms vm list
```

DevStack

TODO

```
$ cms set cloud=devstack
$ cms vm start
$ cms vm list
```


23.1 Goal (outdated)

The goal is to have a configuration file in which we add a number of computers that you can use to execute tasks via ssh calls remotely. We will use no fancyful ssh library, but just subprocess. As this task requires possibly more than you can do in a single week, you need to decide which task you like to work on.

- a) develop a documentation so that the program can be managed via a command line. Use docopts for that. You are not allowed to use other tools
- b) develop a yaml file in which we manage the remote machines and how you get access to them. This includes how many jobs on the machine can be executed in parallel.
- c) develop a task mechanism to manage and distribute the jobs on the machine using subprocess and a queue. Start with one job per machine,
 - c.1) take c and do a new logic where each machine can take multiple jobs
- d) develop a mechanism to start n vms via vagrant
- e) develop a test program that distributes a job to the machines calculates the job and fetches the result back. This is closely related to c, but instead of integrating it in c the movement of the data to and from the job is part of a separate mechanism. It is essentially the status of the calculation. Once all results are in do the reduction into a single result. Remember you could do result calculations in parallel even if other results are not there i
- f) advanced: develop a string based formulation of the tasks while providing the task in a def and using the chars | for parallel, ; for sequential and + for adding results

For example

```
def a():  
    string to be executed via ssh on a remote machine  
  
def b():  
  
(a | b | c); d; a+ b+ c +d
```

23.2 Manual: Cloudmesh Multi Service Data Access

23.2.1 Database Providers

A central database provider keeps track of files stored with multiple cloud services.

Local

BUG: we will not use files, this class needs to be eliminated, and instead mongo is to be used

The `LocalDBProvider` uses a folder on the local file system or network share to store each cloud file entry as a yaml file.

MongoDB

Todo

23.2.2 Storage Providers

Storage providers are services that allow storing files.

Local

The `LocalStorageProvider` uses a folder on the local file system or network share to act as a “cloud” storage provider.

Azure Blob Storage

See Libcloud’s [Azure Blobs Storage Driver Documentation](#) for instructions on how to setup a storage account and generate access keys.

23.2.3 Getting Started

The default data section in `cloudmesh.yaml` is setup to use a local database and storage provider.

Download

```
git clone https://github.com/cloudmesh/cloudmesh-cloud
cd cm
pip install -r requirements.txt
cd data
```

Add a file to the default storage service

```
$ cms data add test/files/hello.txt
```

If you’re using an unmodified `cloudmesh.yaml` local test directories are set as the default “service”. An entry for the added file will appear in the local db folder `cloudmesh-cloud/test/data/db` and the file will be stored in `cm4/test/data/storage`.

Note: Network shares can also be used with the local storage provider.

List all files

```
cms data add ls
```

Download file

```
cms data get hello.txt ../test
```

Delete file

```
cms data del hello.txt
```

23.3 Vagrant (outdated)

This has to be reimplemented for Python 3

```
cms set cloud=vagrant
```

See <https://github.com/cloudmesh/vagrant>

```
Usage:
cms vbox version [--output=OUTPUT]
cms vbox image list [--output=OUTPUT]
cms vbox image find NAME
cms vbox image add NAME
cms vbox vm list [--output=OUTPUT] [-v]
cms vbox vm delete NAME
cms vbox vm config NAME
cms vbox vm ip NAME [--all]
cms vbox create NAME ([--memory=MEMORY]
                        [--image=IMAGE]
                        [--script=SCRIPT] | list)
cms vbox vm boot NAME ([--memory=MEMORY]
                      [--image=IMAGE]
                      [--port=PORT]
                      [--script=SCRIPT] | list)
cms vbox vm ssh NAME [-e COMMAND]
```

For each named vbox a directory is created in which a Vagrant file is placed that than is used to interact with the virtual box. The location of the directory is ~/.cloudmesh/vagrant/NAME.

If you set however the cloud to vbox you can save yourself the vbox command in consecutive calls and just use

```
Usage:
cms version [--output=OUTPUT]
cms image list [--output=OUTPUT]
cms image find NAME
cms image add NAME
cms vm list [--output=OUTPUT] [-v]
cms vm delete NAME
cms vm config NAME
cms vm ip NAME [--all]
cms create NAME ([--memory=MEMORY]
                 [--image=IMAGE]
                 [--script=SCRIPT] | list)
```

(continues on next page)

(continued from previous page)

```
cms vm boot NAME ([--memory=MEMORY]
                  [--image=IMAGE]
                  [--port=PORT]
                  [--script=SCRIPT] | list)
cms vm ssh NAME [-e COMMAND]
```

23.4 CM4 Details (outdated)

In cloudmesh, we are using the **Python** tool to implement a program that could remotely control cloud nodes provided by different organizations and run experiments in parallel.

The goal of *cloudmesh* is to provide a platform that users could directly control the nodes they have, like AWS, Azure, and OPENSTACK instances. Users could decide to start, stop, destroy, create, resume, and suspend different nodes without accessing the **Console** interfaces of providers. Then users could install experiment environment, software, and other required tools in these running nodes. Finally, an experiment could be executed in running nodes by sending the commands from *cloudmesh* platform. Meanwhile, we embed the NoSQL database **MongoDB** into cloudmesh for managing the nodes and experiments.

23.4.1 Extra: Vagrant

TODO: update the link

Please refer to [here](#) to see how to setup Vagrant with cloudmesh.

23.4.2 What we have implemented

- the function to install `cms` and its required packages
- the function to manage the virtual machines from cloud service providers (Azure, AWS, and Openstack)
- the function to use *MongoDB* for saving data

The Preparation for installing cloudmesh (David)

- `requirements.txt` : the required packages
- `setup.py`
- `cloudmesh-cloud/command/command.py` : the python class defines the interface for the command-line `cms`

```
$ cms
Usage:
  cms admin mongo install [--brew] [--download=PATH]
  cms admin mongo status
  cms admin mongo start
  cms admin mongo stop
  cms admin mongo backup FILENAME
  ...
```


The Configuration files and some relative function classes (Sachith)

The cloudmesh4.yaml file contains all the configurations required for CM4 to run. By default it's located in the Cloudmesh home directory (~/.cloudmesh/cloudmesh4.yaml).

Use the Configurations file

To use the configurations in CM4, you need to import the Config class and use the config object provided by that class to access and manipulate the configuration file.

Getting the config object

```
from cloudmesh.cloud.configuration.config import Config
config = Config().data["cloudmesh"]
```

Getting values

To get values from the configurations, you can call level by level from top-down config.

```
MONGO_HOST = config["data"]["mongo"]["MONGO_HOST"]
```

Using the Counter file

CM4 keeps track of all the VMs running using counters for each VM. The counter file is located at

```
~/.cloudmesh/counter.yaml
```

Using the counter

```
from cloudmesh.cloud.configuration.counter import Counter
counter = Counter()
```

Incrementing and Decrementing the counter values

```
# to update a specific VM counter
counter.incr("<VM_NAME>")
counter.decr("<VM_NAME>")

# to update the total vm counter
counter.incr()
counter.decr()
```

Getting and Setting the counter values

```
# to update a specific VM counter
counter.get("<VM_NAME>")
counter.set("<VM_NAME>", "value")
```

The MongoDB Database in `cloudmesh` (Yu)

We add the database into `cloudmesh` with two reasons:

- provide the information of nodes in different providers.
- record the experiment executed through `cloudmesh`, easy for next re-execution.

Every time the user use the `cloudmesh` platform, the server would access the running MongoDB database, querying the nodes' information, showing relative metadata, and then updating all necessary data.

The *MongoDB* would finish below tasks:

- saving all information:
 1. the nodes' information queried from cloud service, like name, id, status, and other metadata about this node.
 2. saving the executing or executed experiment information, like which node we run the experiment, the input, the command, and the output.
 3. saving the group information users defined.
- updating any changes:
 1. the changes updated on the nodes, like stop running node, or start stopped node.
 2. the changes updated on the [`cloudmesh4.yaml`], like add new nodes.
 3. when the experiment is done, output and experiment status would be updated.
 4. new group is created while using `cms` will be updated
- return required information:
 1. return the node information, group information, and experiment information when `cms` queries them.

Data Scheme in MongoDB

There are three types of documents in MongoDB:

- Node information in `cloud` collection. Different cloud service providers would return different schemas of node information. It is hard to manipulate different nodes' information into same schema, so we decide to dump the return message into MongoDB without any changes.
- Node's experiment status in `status` collection. The document in `status` collection is going to save the information of experiments executed in a node.

```
{'_id': node_id,
 'status': status,
 'currentJob': job_id,
 'history' : the history of executed experiments in this node}
```

- Experiment information in `job` collection.

```
{'_id': experiment_id,
  'name': name,
  'status': status,
  'input': input_info,
  'output': output_info,
  'description': description,
  'commands': commands}
```

- Group information in group collection.

```
{'cloud': cloud,
  'name': name,
  'size': size,
  'vms': list_vms}
```

Security in MongoDB

For data security purpose, we enable the MongoDB security functionality in `cms`.

When users first time start the *MongoDB*, they have to add an account and open an port to access all database in MongoDB. Because we save all nodes' information into MongoDB including the *Authorization* information. If your MongoDB is open to everyone, it is easy for hacker to steal your information. So you are required to set the *username* and *password* for the security purpose.

If you want to learn more about the *Security* in MongoDB, you can visit this [page](#) or visit the brief introduction about the MongoDB

Here is a quick reference about how to [enable MongoDB Security](#) option.

Install MongoDB Into Local

If you want to know how to install MongoDB into local, you can review [Install MongoDB](#)

And if you want to use `cms` to help you install MongoDB, you have to update the information required for installing MongoDB into `[cloudmesh4.yaml]` file.

The `cloudmesh-cloud/cmmongo/MongoDBController.py` has the functions to install MongoDB for Linux and Darwin system.

The logic of installing MongoDB is:

```
1. install prepared tools
2. download the MongoDB .tgz tarball
3. extract file from tarball
4. update the PATH environment
5. create data and log directories
6. create the mongodb configuration file
```

When we finish installing MongoDB to local, we have to:

```
7. run the MongoDB
8. add new role for user to access the database
9. stop MongoDB
10. enable the security setting in configuration file
```

Insert and Update Documents in MongoDB

We have different documents in different collections. The operations in `cloudmesh-cloud/vm/Vm.py` will call `mongoDB.py` to accomplish inserting and updating the document.

```
insert_cloud_document(document) : insert the document into 'cloud' collection
insert_status_document(document) : insert the document into 'status' collection
insert_job_document(document) : insert the document into 'job' collection
insert_group_document(document) : insert the document into 'group' collection
update_document(collection_name, key, value, info) : update the new information into
↳the
                                                    the document queried by 'key :
↳value'
                                                    in collection_name' collection
find_document(collection_name, key, value) : get a document by 'key : value' from
                                                    'collection_name' collection
find(collection_name, key, value) : get documents satisfied with 'key : value' from
                                                    'collection_name' collection
delete_document(collection_name, key, value) : delete the document satisfied with
↳'key : value'
                                                    from 'collection_name' collection
```

The Virtual Machine Provider

Execute Command in MongoDB

To grant users more power in manipulating their local MongoDB database, we also add functions for users to execute their customized mongoDB command as if they can use mongoDB client through terminal by `db_command(command)` function. And in order to handle the various exceptions and errors which might occur when executing the command, we also add the `db_connection()` function to help contain those unexpected results.

```
db_command(command): issue a command string to virtual mongoDB shell
db_connection: test connection to local mongoDB host
```

4. The Virtual Machine Provider

In `cloudmesh`, we developed the `cloudmesh-cloud/vm/Vm.py` class to implement the operations for different virtual machines from AWS, Azure, and Chameleon by using the python library *Apache Libcloud* to interact with cloud service providers.

The basic functions are:

```
1. start(vm_name) : start the virtual machine with specified name
2. stop(vm_name, deallocate) : stop the virtual machine with specified name
3. resume(vm_name) : resume the suspended virtual machine with specified name
4. suspend(vm_name) : suspend the running virtual machine with specified name
5. destroy(vm_name) : destroy the virtual machine with specified name
6. list() : list all virtual machine in your cloud service account
7. status(vm_name) : show the working status of virtual machine with specified name
8. info(vm_name) : show all information about the virtual machine with specified name
9. get_public_ips(vm_name) : return the public ip of the virtual machine with
↳specified name
10. set_public_ip(vm_name, public_ip): set the public ip for the virtual machine with
↳specified name
```

(continues on next page)

(continued from previous page)

```
11. remove_public_ip(vm_name) : remove the public ip from virtual machine with
↳specified name
```

Below we list some sample of running these functions for virtual machines in AWS, Azure and Openstack.

AWS VM Operations (Yu)

Before using the AWS Vm code, user has to update their AWS information into `cloudmesh4.yaml` file in *etc* folder.

The *Libcloud* library has enough methods to support the operations for managing virtual machines in AWS. We use a `cloudmesh-cloud/vm/Aws.py` to create the driver based on the configuration to connect to AWS.

Inherit the *Libcloud* library, we did some modifications on `AWSDriver` to extend the operation. The `create_node` method would create a virtual machine in AWS based on the configuration of `cloudmesh4.yaml` file

Here are some samples for running these operations by using `cloudmesh-cloud`:

First, user would create the virtual machine in AWS.

```
$ cms vm create
Collection(Database(MongoClient(host=['127.0.0.1:27017'], document_class=dict, tz_
↳aware=False, connect=True), 'cloudmesh'), 'cloud')
Thread: updating the status of node
Created base-cloudmesh-yuluo-4
PING 52.39.13.229 (52.39.13.229): 56 data bytes

--- 52.39.13.229 ping statistics ---
1 packets transmitted, 0 packets received, 100.0% packet loss
```

then **MongoDB** will have below record in `cloud` collection.

```
{ "_id" : ObjectId("5c09c65f56c5a939942a9911"),
  "id" : "i-01ca62f33728f4931",
  "name" : "base-cloudmesh-yuluo-4",
  "state" : "running",
  "public_ips" : [ "52.39.13.229" ],
  ...}
```

If user want to stop the virtual machine, then he has to type below command with virtual machine name. The code will return you the virtual machine from MongoDB record with a thread updating the new information. When the thread is done, you can use `status` method to check the status of the virtual machine.

```
$ cms vm stop --vms=base-cloudmesh-yuluo-4
Thread: updating the status of node
{'_id': ObjectId('5c09c65f56c5a939942a9911'),
 'id': 'i-01ca62f33728f4931',
 'name': 'base-cloudmesh-yuluo-4',
 'state': 'running',
 'public_ips': ['52.39.13.229'],
 ...}
$ cms vm status --vms=base-cloudmesh-yuluo-4
stopped
```

When user wants to start the stopped virtual machine, he has to type the command of below sample.

```
$ cms vm start --vms=base-cloudmesh-yuluo-4
Thread: updating the status of node
{'_id': ObjectId('5c09c65f56c5a939942a9911'),
'id': 'i-01ca62f33728f4931',
'name': 'base-cloudmesh-yuluo-4',
'state': 'stopped',
'public_ips': [],
...}
PING 54.191.109.54 (54.191.109.54): 56 data bytes

--- 54.191.109.54 ping statistics ---
1 packets transmitted, 0 packets received, 100.0% packet loss

$ cms vm status --vms=base-cloudmesh-yuluo-4
running
```

There is a way for users to get the public ip of a virtual machine.

```
$ cms vm publicip --vms=base-cloudmesh-yuluo-4
{'base-cloudmesh-yuluo-4': ['54.191.109.54']}
```

Also, if user wants to know the information of virtual machines under his AWS account, he could do this.

```
$ cms vm list
<Node: uuid=9b46e75095f586471e2cfe8ebc6b102lead0e86b, name=a-b-luoyu-0, state=STOPPED,
↪ public_ips=[],
private_ips=['172.31.28.147'], provider=Amazon EC2 ...>,
<Node: uuid=da309c8acbbc7bc1f21295600323d073afffb04a, name=base-cloudmesh-yuluo-1,
↪ state=TERMINATED,
public_ips=[], private_ips=[], provider=Amazon EC2 ...>
<Node: uuid=cb62a083081350da9e6f229aace4b697358a987b, name=base-cloudmesh-yuluo-4,
↪ state=RUNNING,
public_ips=['54.191.109.54'], private_ips=['172.31.41.197'], provider=Amazon EC2 ...>,
```

Finally, if user wants to delete the virtual machine, he could do this.

```
$ cms vm destroy --vms=base-cloudmesh-yuluo-4
True
```

Azure VM Operation (David)

Chameleon VM Operation (Rui and Kimball)

Same as above, before using the VM Openstack functionalities, user has to update their Openstack information into the `cloudmesh4.yaml` file (`~/cloudmesh/cloudmesh.yaml` by macOS convention). It is also important to notice that openstack has various providers. And it is important to specify each of them with correspondent log-in credentials.

Many of the funtions are supported by the *Libcloud* library. By specifying the config parameters to `openstack` and `chameleaon`, VM Provider will automatically attach futher operations to openstack primitives.

In order to overcome some issues with service provider (mostly delays in operations like spawning, ip-assignments, refactoring and etc), we implement timeout mechanism to synchronize status between our local machines and remote providers. Blocking strategy is used to prevent un-deterministic running result.

Since providers of openstack like Chameleon and Jetstream allow users to associate customized float ip to their instances, we also develop such functions to support tasks like this and give more power to users when runing their jobs.

Please refer to AWS VM Operation for examples. Chameleon Openstack expose same operations as AWS to users. Notice that before running your command, you need to make sure the global default cloud parameter has been set to 'Chameleon' by:

```
$ cms vm set cloud chameleon
Setting env parameter cloud to: chameleon
Writing updata to cloudmesh.yaml
Config has been updated.
```

VM Refactor (Rui)

In addition, in order to offer more flexibilities to our users, we also developed vmrefactor (cloudmesh-cloud/vm/VmRefactor.py) to allow users to customize the flavors of their running instances and services in different providers.

```
1. resize(vm_name, size) : resize the virtual machine with specified size object
2. confirm_resize(vm_name) : some providers requires confirmation message to complete_
   ↳resize() operation
3. revert(vm_name) : revert a resize operation. Revert the virtual machine to_
   ↳previous status
4. rename(vm_name, newname) : rename the virtual machine
5. rebuild(vm_name, image) : rebuild the virtual machine to another image/OS with_
   ↳image object.
```

Currently, major providers usually charge users according to their usage. It might be financially wise sometimes to shift between different service size to reduce unnecessary cost. VmRefactor is designed based on this idea to help users to achieve higher cost efficiency. VmRefactor can also help users navigate their management tasks especially when they have many different tasks on the run=.

23.4.3 Flask Rest API (Sachith)

The cloudmesh REST Api is built using flask and provides the cloud information retrieval functionality through HTTP calls.

Pre-requisites

Use pip install to install the following packages.

- Flask
- Flask-PyMongo

How to run the REST API

```
$ cms admin rest status
$ cms admin rest start
$ cms admin rest stop
```

- Navigate to the cm directory. example:

```
cd ~/git/cloudmesh/cm
```

- Configure cloudmesh

```
pip install .
```

- Add the MongoDB information in the cloudmesh configuration file

```
vi ~/.cloudmesh/cloudmesh4.yaml
```

- Run the REST API

```
python cm4/flask_rest_api/rest_api.py
```

API

- `/vms/` : Provides information on all the VMs.
- `/vms/stopped` : Provides information on all the stopped VMs.
- `/vms/<id>` : Provides information on the VM identified by the

Examples

- Retrieve information about a VM

```
curl localhost:5000/vms/i-0fad7e92ffea8b345
```

Dev - restricting certain ips for certain rest calls

```
from flask import abort, request
from cm4.flask_rest_api.app import app

@app.before_request
def limit_remote_addr():
    if request.remote_addr != '10.20.30.40':
        abort(403) # Forbidden
```

23.4.4 Extra: Run Command/Script in AWS

The `cloudmesh-cloud/aws/CommandAWS.py` contains some methods to run commands/scripts from local to remote AWS virtual machines. Any command/script operations executed by `CommandAWS.py` would be saved into MongoDB.

In `cloudmesh-cloud`, we use python running `ssh` client to connect the AWS virtual machines. Before running the commands or scripts remotely, the `CommandAWS.py` would create the job document in MongoDB for saving the experiment information. This job document contains the information of virtual machine name, the running command or script, and job status, input, output and description. Meanwhile, the job document_id would be added into status document of the `status` collection for describing the job history of each virtual machine.

For example, user wants to run `pwd` command to `base-cloudmesh-yulou-5` machine in AWS.

```
$ cms aws run command pwd --vm=base-cloudmesh-yuluo-5
Running command pwdin Instance base-cloudmesh-yuluo-5:
/home/ubuntu
```


The job collection and status collection in MongoDB will have below document:

```
> db.job.find()
{ "_id" : ObjectId("5c09ff9b284fa4515ee9e204"), "name" : "base-cloudmesh-yuluo-5",
  ↪ "status" : "done",
  "input" : "Null", "output" : "/home/ubuntu\n", "description" : "single job", "commands
  ↪" : "pwd" }
> db.status.find()
{ "_id" : ObjectId("5c09ff9b284fa4515ee9e205"), "id" : "base-cloudmesh-yuluo-5",
  ↪ "status" : "No Job",
  "currentJob" : "Null", "history" : [ "5c09ff9b284fa4515ee9e204" ] }
```

If user run script file containing ‘#!/bin/sh\npwd’ in base-cloudmesh-yulou-5 machine:

```
$ cms aws run script /Users/yuluo/Desktop/cm.sh --vm=base-cloudmesh-yuluo-5
Running command /Users/yuluo/Desktop/cm.sh Instance base-cloudmesh-yuluo-5:
/home/ubuntu
```

The job collection and status collection in MongoDB will have below document:

```
> db.job.find()
{ "_id" : ObjectId("5c09ff9b284fa4515ee9e204"), "name" : "base-cloudmesh-yuluo-5",
  ↪ "status" : "done",
  "input" : "Null", "output" : "/home/ubuntu\n", "description" : "single job", "commands
  ↪" : "pwd" }
{ "_id" : ObjectId("5c0a00a8284fa451d0ab427d"), "name" : "base-cloudmesh-yuluo-5",
  ↪ "status" : "done",
  "input" : "#!/bin/sh\npwd", "output" : "/home/ubuntu\n", "description" : "single job",
  ↪ "commands" : "Null" }
```

```
> db.status.find()
{ "_id" : ObjectId("5c09ff9b284fa4515ee9e205"), "id" : "base-cloudmesh-yuluo-5",
  ↪ "status" : "No Job",
  "currentJob" : "Null", "history" : [ "5c09ff9b284fa4515ee9e204",
  ↪ "5c0a00a8284fa451d0ab427d" ] }
```

23.5 AWS cm (outdated)

The code is designed for using awscm.py to access the aws instance and run scripts in it.

In the code, we provide these commands for achieving the goal of conducting benchmarks on remote machines.

23.5.1 Code Description

In the awscm folder, there are several basic python files:

cloudmesh.yaml

This file contains the property of each instance, especially the AWS instance. In AWS cm, we only concern about the block of information in “cloud” part.

In the properties of one aws instance, we need users to specify the “name” and “label” of the instance. In the “credentials” part, we need users to fill in the “KEY” and “ID”. Make sure there are no duplicated names and labels in the “aws” list.

:o: Suggestion for Redesign

I propose to redesign and use the old cloudmesh interface in this new implementation

```
cms aws vm list
cms cloud=aws
# all subsequent commands are done on aws without the need to specify the cloud
cms group=cluster1
# all subsequent commands are added to the group. The last group is set to group1, a
↪group can have arbitrary resources in it vms, files, ...
# commands applied to last vm
cms vm start [--cloud=CLOUD]
cms vm stop [--cloud=CLOUD]
cms vm info [--cloud=CLOUD]
cms vm delete [--cloud=CLOUD]
cms vm suspend [--cloud=CLOUD]
#
cms group list [--group=GROUP]
cms group delete [--group=GROUP]

MongoDB is used to manage the data

cms save [--file=FILE]
cms load [--file=FILE]

makes a backup of the data in mongo

cms system status

looks at the system status of mongo and other cms stuff
```

here are some additional thoughts, that may influence what we do:

- <http://cloudmesh.github.io/cmd3/man/man.html#vm>
- There is also a newer version of cloudmesh, that we have not implemented all of this logic but it uses cmd5

awscm.py

The [awscm.py] is the main runnable python class to start the aws cm. It used the “docopt” to build the usage of commands. Here are the version 1 commands that could be used:

Add resources

```
awscm.py resource add <yaml_file>
```

add extra instance information into the default yaml file. Please follow the schema of the asw instance. For example:

```
aws_a:
  credentials: {EC2_ACCESS_ID: "id", EC2_SECRET_KEY: "key"}
  label: aws_a
  name: aws_a
```

List Resources

```
awscm.py resource list [--debug]
```

list all instances from the default yaml file

Remove Resources

```
awscm.py resource remove <label_name>
```

remove the named or labeled instance from yaml file. Please fill in the correct name or label. For example:

```
python awscm.py resource remove aws_a
```

View Resources

```
awscm.py resource view <label_name>
```

view named or labeled instance from the default yaml file. Please fill in the correct name or label. For example:

```
python awscm.py view aws_a
```

Copy Instances from File

```
awscm.py copy instance <label_name> file <file> to <where>
```

copy the file from local to the directory of instance. For example:

```
python awscm.py copy instance aws_a file test.txt to /test/
```

Copy Instances from Folder

```
awscm.py copy instance <label_name> folder <folder> to <where>
```

copy the folder from local to the directory of instance. For example:

```
python awscm.py copy instance aws_a folder /test/ to /test/
```

Copy Instances

```
awscm.py list instance <label_name> from <where>
```

list the files/folders in the directory of instance. For example:

```
python awscm.py instance aws_a from /test/
```

Delete Instances from file

```
awscm.py delete instance <label_name> file <file> from <where>
```

delete the file from the directory of instance. For example:

```
python awscm.py delete instance aws_a file test.txt from /test/
```

Delete Instances from Folder

```
awscm.py delete instance <label_name> folder <folder> from <where>
```

delete the folder from the directory of instance. For example:

```
python awscm.py delete instance aws_a folder test from /test/
```

Create instances from folder

```
awscm.py create instance <label_name> folder <folder> in <where>
```

create a new folder in the directory of instance. For example:

```
python awscm.py create instance aws_a folder test in /test/
```

Read Instances from Folder

```
awscm.py read instance <label_name> file <file> from <where>
```

read the file in the directory of instance. For example:

```
python awscm.py read instance aws_a file test.txt from /test/
```

Download INstances from file

```
awscm.py download instance <label_name> file <file> from <where> to <local>
```

download the file from the directory of instance to local. For example:

```
python awscm.py download instance aws_a file test.txt from /test/ to /test/
```

Download instances from folder

```
awscm.py download instance <label_name> folder <folder> from <where> to <local>
```

download the folder from the directory of instance to local. For example:

```
python awscm.py download instance aws_a folder test from /test/ to /test
```

Check instances

```
awscm.py check instance <label_name> process_name <process>
```

check the running process in the instance. For example:

```
python awscm.py check instance aws_a process_name test
```

Run instances locally

```
awscm.py run instance <label_name> local <scripts>
```

run the scripts from local into the instance. For example:

```
python awscm.py instance aws_a local test.sh,test.sh,test.sh
```

Run instances remotely

```
awscm.py run instance <label_name> remote <scripts>
```

run the scripts from remote instance. For example:

```
python awscm.py run instance aws_a remote test.sh,test.sh,test.sh
```

Run local

```
awscm.py run local <scripts>
```

run the scripts from local into the random parallel instance. For example:

```
python awscm.py run local test.sh,test.sh,test.sh
```

Run local

```
awscm.py run remote <scripts>
```

run the scripts from the remote parallel instances. Make sure all instances have the required scripts. For example:

```
python awscm.py run remote test.sh,test.sh,test.sh
```

Run advanced

```
awscm.py run advanced <string>
```

this command is running the advanced algorithm. Developing a string based formulation of the tasks while providing the task in a def and using the chars | for parallel, ; for sequential and + for adding results. In cloudmesh, we only develop simples string to be executed via ssh on a remote machines. The default setting is running the local scripts into remote parallel instances.

For example, we define the function in `[advanced.py]`:

```
def a() :  
def b() :  
def c() :  
def d() ;
```

then we run the command to get the result:

```
python awscm.py advanced a|b|c;d;a+b+c+d
```

config.py

This python class is reading the configuration of instances. In the yaml file, we set three types of instances: cloud, cluster and default, and the `[config.py]` could return relative block information of them.

resource.py

`[resource.py]` is used to read and manage the default yaml file. In the class, we provides the read, update, add, remove and review functionalities for yaml file. And `[awscm.py]` would call these functions to run the commands.

utility.py

The `[utility.py]` file contains the functions to do preparation before running scripts in remote instance. In this python class, we implement the functions: copy file to instance, copy folder into instance, list files from the instance, delete file from instance, delete folder from instance, create folder instance, read file from instance, download file from instance, download folder from instance and check whether the process is running or not.

run.py

The `[run.py]` file contains the functions to call the scripts in remote instance. In this class, we provides three functions: run the scripts locally to the instance, run the remote scripts in the instance and run the scripts in parallel instances.

advanced.py

This class is used for the advanced approach to run a string based formulation of the tasks. Need to be updated later.

23.5.2 TODO - Spark

- [] update more functionalities
- [] try the Spark in AWS instances
- [] try Spark by using awscm python code
- [] develop the test code

23.6 REST Service (outdated)

The REST Api is built using flask and provides the cloud information retrieval functionality through HTTP calls.

23.6.1 Pre-requisites

Use pip install to install the following packages.

- Flask
- Flask-PyMongo

23.6.2 How to run the REST API

- Navigate to the cm directory. example:

```
cd ~/git/cloudmesh/cm
```

- Configure cloudmesh

```
pip install .
```

- Add the MongoDB information in the configuration file

```
vi ~/.cloudmesh/cloudmesh4.yaml
```

- Run the REST API

```
python cm4/flask_rest_api/rest_api.py
```

23.6.3 API

- /vms/ : Provides information on all the VMs.
- /vms/stopped : Provides information on all the stopped VMs.
- /vms/<id> : Provides information on the VM identified by the

23.6.4 Examples

- Retrieve information about a VM

```
curl localhost:5000/vms/i-0fad7e92ffea8b345
```

23.6.5 Dev - restricting certain ips for certain rest calls

```
from flask import abort, request
from cm4.flask_rest_api.app import app

@app.before_request
```

(continues on next page)

(continued from previous page)

```
def limit_remote_addr():
    if request.remote_addr != '10.20.30.40':
        abort(403) # Forbidden
```

23.7 Virtual Cluster (in progress)

This is a tool used to submit jobs to remote hosts in parallel and contains the following subcommands:

```
cms vcluster create virtual-cluster VIRTUALCLUSTER_NAME --clusters=CLUSTERS_LIST [--
↪computers=COMPUTERS_LIST] [--debug]
cms vcluster destroy virtual-cluster VIRTUALCLUSTER_NAME
cms vcluster create runtime-config CONFIG_NAME PROCESS_NUM in:params out:stdout [--
↪fetch-proc-num=FETCH_PROCESS_NUM [default=1]] [--download-now [default=True]] [--
↪debug]
cms vcluster create runtime-config CONFIG_NAME PROCESS_NUM in:params out:file [--
↪fetch-proc-num=FETCH_PROCESS_NUM [default=1]] [--download-now [default=True]] [--
↪debug]
cms vcluster create runtime-config CONFIG_NAME PROCESS_NUM in:params+file out:stdout ↪
↪[--fetch-proc-num=FETCH_PROCESS_NUM [default=1]] [--download-now [default=True]] ↪
↪[--debug]
cms vcluster create runtime-config CONFIG_NAME PROCESS_NUM in:params+file out:file [--
↪fetch-proc-num=FETCH_PROCESS_NUM [default=1]] [--download-now [default=True]] [--
↪debug]
cms vcluster create runtime-config CONFIG_NAME PROCESS_NUM in:params+file ↪
↪out:stdout+file [--fetch-proc-num=FETCH_PROCESS_NUM [default=1]] [--download-now ↪
↪[default=True]] [--debug]
cms vcluster set-param runtime-config CONFIG_NAME PARAMETER VALUE
cms vcluster destroy runtime-config CONFIG_NAME
cms vcluster list virtual-clusters [DEPTH [default:1]]
cms vcluster list runtime-configs [DEPTH [default:1]]
cms vcluster run-script --script-path=SCRIPT_PATH --job-name=JOB_NAME --vcluster-
↪name=VIRTUALCLUSTER_NAME --config-name=CONFIG_NAME --arguments=SET_OF_PARAMS --
↪remote-path=REMOTE_PATH> --local-path=LOCAL_PATH [--argfile-path=ARGUMENT_FILE_
↪PATH] [--outfile-name=OUTPUT_FILE_NAME] [--suffix=SUFFIX] [--overwrite]
cms vcluster fetch JOB_NAME
cms vcluster clean-remote JOB_NAME PROCESS_NUM
cms vcluster test-connection VIRTUALCLUSTER_NAME PROCESS_NUM
```

As can be seen, the command `vcluster` can be called with XX possible options:

- create
 - virtual-cluster
 - runtime-config
- destroy
 - virtual-cluster
 - runtime-config
- list
 - virtual-clusters
 - runtime-configs

- set-param
 - virtual-cluster
 - runtime-config
- run-script
- fetch
- clean-remote
- test-connection

The information needed to create a virtual cluster, are extracted from the `yaml` file of the cloudmesh v4, aka `cms`, however, it does not modify that file. Instead, it will create a new configuration file in a folder called `vcluster_workspace`. This newly generate configuration file contains all the information about the virtual clusters, runtime configurations as well as submitted jobs and therefore the file is crucial for fetching the result of the previous runs. Although possible, it is highly recommended not to modify the file directly but instead use the `set-param` command to modify the file.

When you are creating a virtual cluster, you can *pick* your nodes of interest from the cloudmesh configuration and just pass it as an argument to `create virtual-cluster` and you will have your *Virtual Cluster* created this way. When you are done with a Virtual Cluster, aka `vcluster`, you can simply destroy it.

23.7.1 Creating a Virtual Cluster and testing connections

Consider the following two dummy clusters in the `cloudmesh4.yaml` file:

```
cloudmesh:
  ...
  vcluster_test1:
    computer_a:
      name: machine1
      label: one
      address: localhost
      credentials:
        sshconfigpath: ~/vms/ubuntu14/sshconfig1
    computer_b:
      name: computer_a
      label: one
      address: localhost
      credentials:
        username: TBD
        pulickey: ~/.ssh/id_rsa.pub
  vcluster_test2:
    c2:
      name: machine2
      label: two
      address: localhost
      credentials:
        sshconfigpath: ~/vms/ubuntu14/sshconfig2
  ...
```

Suppose you want to create a virtual cluster called `new_vcluster` using `computer_a` from `vcluster_test1` and `c2` from `vcluster_test2`. This can be achieved using the following command:

```
$ cms vcluster create virtual-cluster vcluster1 --clusters=vcluster_test1,vcluster_
↪test2 --computers=computer_a,c2
Virtual cluster created/replaced successfully.
```

This command will create the `vcluster.yaml` file in the `vcluster_workspace` folder and will keep the information about the virtual cluster in there. Now, we can get the information about the virtual cluster that we just created:

```
$ cms vcluster list virtual-clusters
vcluster1:
  computer_a
  c2
```

By passing a depth higher than one as an extra argument, you can get more information about the virtual clusters:

```
$ cms vcluster list virtual-clusters 2
vcluster1:
  computer_a:
    name:      machine1
    label:     one
    address:   localhost
    credentials: sshconfigpath
  c2:
    name:      machine2
    label:     two
    address:   localhost
    credentials: sshconfigpath
```

Now that the virtual cluster is created, we can test the connection to the remote nodes. We will try that using 2 processes in parallel:

```
$ cms vcluster test-connection vcluster1 2
Node computer_a is accessible.
Node c2 is accessible.
```

The output indicates that both nodes in the `vcluster1` are accessible. In case you did not need the `vcluster1` anymore, you can easily remove it using:

```
$ cms vcluster destroy virtual-cluster vcluster1
Virtual-cluster vcluster1 destroyed successfully.
```

23.7.2 Creating a runtime-configuration

Next, we have to create a `runtime-configuration` which defines the type of input and output for possibly a set of jobs that are going to be submitted later. In the next example we will create a runtime configuration for jobs that we want to run remotely using 5 processes, fetch their results using 3 processes and the script that we want to run remotely takes just some parameter (which could be left empty for no parameters), and the output of the script is

going to be printed on the standard output, and suppose we want to just submit the jobs for running on remote nodes and download them later (hence the `--download-later` flag):

```
$ cms vcluster create runtime-config ParamInStdOut 5 in:params out:stdout --fetch-
↪proc-num=3 --download-later
Runtime-configuration created/replaced successfully.
```

Let's get the list of runtime configurations to make sure our configuration is created as we expected:

```
$ cms vcluster list runtime-configs 2
ParamInStdOut:
  proc_num:
    5
  download_proc_num:
    1
  download-later:
    False
  input-type:
    params
  output-type:
    stdout
```

Similar to the virtual cluster, you can remove a runtime-configuration using the `destroy` sub-command:

```
$ cms vcluster destroy runtime-config ParamInStdOut
Runtime-configuration ParamInStdOut destroyed successfully.
```

23.7.3 Running Parallel Remote Jobs

Now that we have both the virtual cluster and runtime configuration ready, we can try to submit a batch job to our virtual cluster using `cms vcluster run-script`. This is by far the most complicated sub-command of the `vcluster`, however, the name of the arguments are pretty clear and looking at the names you would be able to pretty much find your way. In the next example, we submit the `inf_script_stdin_stdout.sh` file to the nodes of `vcluster1` and using the `ParamInStdOut` configuration we run 10 instance of that script on the virtual cluster. This script will be copied and run on the home directory of the remote nodes (`~/`). Note that even though the remote path is set to home directory, for each job a folder with a unique suffix will be created to avoid conflicts. Also, note that this script does not take any argument, but we indicated 10 `_` separated by commas as a meaningless argument. This will notify the tool that you need 10 instances of this script to be executed:

```
$ cms vcluster run-script --script-path=./cm4/vcluster/sample_scripts/inf_script_
↪stdin_stdout.sh --job-name=TestJob1 --vcluster-name=vcluster1 --config-
↪name=ParamInStdOut --arguments=_,_,_,_,_,_,_,_,_,_,_ --remote-path=~/ --local-path=./
↪cm4/vcluster/sample_output --overwrite
Remote Pid on c2: 10104
Remote Pid on c2: 10109
Remote Pid on c2: 10402
Remote Pid on computer_a: 8973
Remote Pid on computer_a: 8979
Remote Pid on computer_a: 8983
Remote Pid on computer_a: 9464
Remote Pid on c2: 10884
Remote Pid on c2: 10993
Remote Pid on computer_a: 9592
collecting results
waiting for other results if any...
```

(continues on next page)

(continued from previous page)

```
Results collected from c2.
Results collected from c2.
Results collected from c2.
Results collected from computer_a.
Results collected from computer_a.
Results collected from computer_a.
Results collected from computer_a.
Results collected from c2.
Results collected from c2.
Results collected from computer_a.
waiting for other results if any...
All of the remote results collected.
```

As you can see all of the jobs were submitted (using 5 processes) and results were collected afterwards (using 3 processes). We can check the existence of the results:

```
$ ll ./cloudmesh-cloud/vcluster/sample_output/
total 48
drwxr-xr-x 2 corriel 4096 Oct 31 22:12 ./
drwxr-xr-x 8 corriel 4096 Oct 31 22:12 ../
-rw-r--r-- 1 corriel 255 Oct 31 22:12 outputfile_0_20181031_22123465
-rw-r--r-- 1 corriel 255 Oct 31 22:12 outputfile_1_20181031_22123465
-rw-r--r-- 1 corriel 255 Oct 31 22:12 outputfile_2_20181031_22123465
-rw-r--r-- 1 corriel 255 Oct 31 22:12 outputfile_3_20181031_22123465
-rw-r--r-- 1 corriel 255 Oct 31 22:12 outputfile_4_20181031_22123465
-rw-r--r-- 1 corriel 255 Oct 31 22:12 outputfile_5_20181031_22123465
-rw-r--r-- 1 corriel 255 Oct 31 22:12 outputfile_6_20181031_22123465
-rw-r--r-- 1 corriel 255 Oct 31 22:12 outputfile_7_20181031_22123465
-rw-r--r-- 1 corriel 255 Oct 31 22:12 outputfile_8_20181031_22123465
-rw-r--r-- 1 corriel 255 Oct 31 22:12 outputfile_9_20181031_22123465
```

Now, suppose the jobs were going to take so long that we could not wait for the results and we had to download them later. To prepare this scenario, we can set the `download-later` attribute of the runtime configuration to `true`:

```
$ cms vcluster set-param runtime-config ParamInStdOut download-later true
Runtime-configuration parameter download-later set to true successfully.
```

Now that we set this parameter, we can submit the jobs and this time the tool will not wait for the results:

```
$ cms vcluster run-script --script-path=./cloudmesh-cloud/vcluster/sample_scripts/inf_
↪script_stdin_stdout.sh --job-name=TestJob1 --vcluster-name=vcluster1 --config-
↪name=ParamInStdOut --arguments=_,_,_,_,_,_,_,_,_ --remote-path=~ --local-path=./
↪cloudmesh-cloud/vcluster/sample_output --overwrite
Remote Pid on c2: 12981
Remote Pid on c2: 12987
Remote Pid on c2: 13280
Remote Pid on computer_a: 11858
Remote Pid on computer_a: 11942
Remote Pid on computer_a: 11945
Remote Pid on computer_a: 12300
Remote Pid on c2: 13795
Remote Pid on computer_a: 12427
Remote Pid on c2: 13871
```

As you can see, the jobs are submitted and the script is finished. Note that since a job with that exact job name exists, you cannot submit the job unless you use the `--overwrite` flag. Now that we have submitted the jobs and their results are ready, we can fetch their produced results using the `fetch` command and all results will be collected using

the same number of processes that were indicated in the runtime-configuration using which the job was submitted in the first place:

```
$ cms vcluster fetch TestJob1
collecting results
Results collected from c2.
Results collected from c2.
Results collected from c2.
Results collected from computer_a.
Results collected from computer_a.
Results collected from computer_a.
Results collected from c2.
Results collected from computer_a.
Results collected from computer_a.
Results collected from c2.
waiting for other results if any...
All of the remote results collected.
```

23.7.4 Cleaning the remote

By default the Virtual Cluster tool does not clean the remotes automatically and this task is left to be performed manually since important results might be lost due to mistakes. To clean the remotes, the user has to explicitly use the `clean-remote` command for a specific job and this way only the results of that particular job will be removed from **ALL** remotes using 2 parallel processes:

```
$ cms vcluster clean-remote TestJob1 4
Node c2 cleaned successfully.
Node computer_a cleaned successfully.
```


CHAPTER 24

Indices and tables

- `genindex`
- `modindex`
- `search`

C

cloudmesh, 136

cloudmesh.abstractclass.ComputeNodeABC, 145

cloudmesh.abstractclass.ProcessManagerABC, 145

cloudmesh.abstractclass.State, 145

cloudmesh.abstractclass.StorageABC, 145

cloudmesh.admin.command.admin, 139

cloudmesh.common.console, 125

cloudmesh.common.dotdict, 121

cloudmesh.common.error, 127

cloudmesh.common.FlatDict, 122

cloudmesh.common.locations, 122

cloudmesh.common.logger, 126

cloudmesh.common.parameter, 122

cloudmesh.common.Printer, 123

cloudmesh.common.run.background, 130

cloudmesh.common.run.file, 130

cloudmesh.common.Shell, 127

cloudmesh.common.ssh.authorized_keys, 130

cloudmesh.common.ssh.encrypt, 131

cloudmesh.common.ssh.ssh_config, 131

cloudmesh.common.StopWatch, 125

cloudmesh.common.util, 119

cloudmesh.compute.azure.AzProvider, 144

cloudmesh.compute.azure.AzureVm, 145

cloudmesh.compute.docker.Provider, 144

cloudmesh.compute.libcloud.Provider, 145

cloudmesh.compute.virtualbox.Provider, 145

cloudmesh.compute.vm.Provider, 145

cloudmesh.config.command.config, 141

cloudmesh.container.command.container, 144

cloudmesh.data.api.CloudFile, 148

cloudmesh.data.api.db.DBProviderABC, 147

cloudmesh.data.api.Driver, 148

cloudmesh.data.api.File, 147

cloudmesh.data.api.storage.StorageProviderABC, 147

cloudmesh.data.command.data, 148

cloudmesh.db.strdb, 130

cloudmesh.DEBUG, 119

cloudmesh.default.command.default, 131

cloudmesh.display, 144

cloudmesh.emr.api.manager, 148

cloudmesh.emr.command.emr, 148

cloudmesh.flavor.command.flavor, 147

cloudmesh.iaas.flavor, 141

cloudmesh.iaas.image, 141

cloudmesh.image.api.manager, 144

cloudmesh.image.command.image, 144

cloudmesh.image.Image, 144

cloudmesh.inventory.command.inventory, 139

cloudmesh.inventory.inventory, 139

cloudmesh.key.api.key, 144

cloudmesh.key.api.manager, 144

cloudmesh.key.command.key, 144

cloudmesh.login.api.manager, 148

cloudmesh.login.command.login, 148

cloudmesh.man.command.man, 131

cloudmesh.management.configuration.arguments, 137

cloudmesh.management.configuration.config, 136

cloudmesh.management.configuration.counter, 136

cloudmesh.management.configuration.generic_config, 136

cloudmesh.management.configuration.name, 137

cloudmesh.management.configuration.operatingsystem, 137

cloudmesh.management.configuration.SSHkey, 137

cloudmesh.management.debug, 139

- cloudmesh.management.printer.Printer,
137
- cloudmesh.management.script, 138
- cloudmesh.mongo.CmDatabase, 145
- cloudmesh.mongo.DataBaseDecorator, 145
- cloudmesh.mongo.MongoDBController, 147
- cloudmesh.network.api.manager, 142
- cloudmesh.network.command.network, 142
- cloudmesh.open.command.open, 148
- cloudmesh.secgrouop.api.manager, 144
- cloudmesh.secgrouop.command.secgrouop, 144
- cloudmesh.security.authorized_keys, 141
- cloudmesh.security.encrypt, 141
- cloudmesh.security.ssh_config, 141
- cloudmesh.set.command.set, 131
- cloudmesh.shell.command, 131
- cloudmesh.shell.plugin, 135
- cloudmesh.shell.shell, 132
- cloudmesh.shell.variables, 132
- cloudmesh.source.api.manager, 142
- cloudmesh.source.command.source, 142
- cloudmesh.ssh.api.manager, 144
- cloudmesh.ssh.command.ssh, 144
- cloudmesh.storage.command.storage, 140
- cloudmesh.storage.Provider, 139
- cloudmesh.storage.provider.awss3.Provider,
140
- cloudmesh.storage.provider.azureblob.Provider,
139
- cloudmesh.storage.provider.box.Provider,
139
- cloudmesh.storage.provider.gdrive.Provider,
139
- cloudmesh.storage.provider.local.Provider,
139
- cloudmesh.storage.provider.storage, 140
- cloudmesh.storage.spec.cloudmesh.storage,
139
- cloudmesh.storage.spec.tests.test_openapi_storage,
139
- cloudmesh.storage.StorageABC, 140
- cloudmesh.terminal.command.terminal, 131
- cloudmesh.url.command.url, 144
- cloudmesh.var.command.var, 131
- cloudmesh.vbox.api.provider, 144
- cloudmesh.vbox.command.vbox, 144
- cloudmesh.vcluster.api.VirtualCluster,
144
- cloudmesh.vdir.api.manager, 140
- cloudmesh.vdir.command.vdir, 140
- cloudmesh.vm.api.manager, 144
- cloudmesh.vm.command.vm, 144
- cloudmesh.workflow.api.manager, 145
- cloudmesh.workflow.command.workflow, 145

A

`add()` (*cloudmesh.common.ssh.authorized_keys.AuthorizedKeys* class method), 128
 method), 130
`add()` (*cloudmesh.data.api.db.DBProviderABC.DBProviderABC* class method), 147
 method), 147
`add()` (*cloudmesh.management.script.SystemPath* static method), 138, 143
`add()` (*cloudmesh.security.authorized_keys.AuthorizedKeys* class method), 141
`attribute()` (*cloudmesh.common.Printer.Printer* class method), 123
`AuthorizedKeys` (class in *cloudmesh.common.ssh.authorized_keys*), 130
`AuthorizedKeys` (class in *cloudmesh.security.authorized_keys*), 141
`auto_create_requirements()` (in module *cloudmesh.common.util*), 119
`auto_create_version()` (in module *cloudmesh.common.util*), 119
`check_output()` (*cloudmesh.common.Shell.Shell* class method), 128
`check_passphrase()` (*cloudmesh.security.encrypt.EncryptFile* class method), 141
`check_python()` (*cloudmesh.common.Shell.Shell* class method), 128
`class_name()` (*cloudmesh.shell.shell.Plugin* class method), 135
`classes()` (*cloudmesh.shell.shell.Plugin* class method), 135
`clear()` (*cloudmesh.common.StopWatch.StopWatch* class method), 125
`clear()` (*cloudmesh.db.strdb.YamlDB* method), 130
`close()` (*cloudmesh.db.strdb.YamlDB* method), 130
cloudmesh (module), 136
cloudmesh.abstractclass.ComputeNodeABC (module), 145
cloudmesh.abstractclass.ProcessManagerABC (module), 145
cloudmesh.abstractclass.State (module), 145
cloudmesh.abstractclass.StorageABC (module), 145
cloudmesh.admin.command.admin (module), 139, 142
cloudmesh.common.console (module), 125
cloudmesh.common.dotdict (module), 121
cloudmesh.common.error (module), 127
cloudmesh.common.FlatDict (module), 122
cloudmesh.common.locations (module), 122
cloudmesh.common.logger (module), 126
cloudmesh.common.parameter (module), 122
cloudmesh.common.Printer (module), 123
cloudmesh.common.run.background (module), 130
cloudmesh.common.run.file (module), 130
cloudmesh.common.Shell (module), 127
cloudmesh.common.ssh.authorized_keys (module), 130

B

`backup_name()` (in module *cloudmesh.common.util*), 120
`banner()` (in module *cloudmesh.common.util*), 120
`basecommand()` (in module *cloudmesh.shell.command*), 131
`bash()` (*cloudmesh.common.Shell.Shell* class method), 127
`benchmark()` (*cloudmesh.common.StopWatch.StopWatch* class method), 125
`blockdiag()` (*cloudmesh.common.Shell.Shell* class method), 127
`brew()` (*cloudmesh.common.Shell.Shell* class method), 127

C

`cat()` (*cloudmesh.common.Shell.Shell* class method), 127

cloudmesh.common.ssh.encrypt (*module*), 131
cloudmesh.common.ssh.ssh_config (*module*), 131
cloudmesh.common.StopWatch (*module*), 125
cloudmesh.common.util (*module*), 119
cloudmesh.compute.azure.AzProvider (*module*), 144
cloudmesh.compute.azure.AzureVm (*module*), 145
cloudmesh.compute.docker.Provider (*module*), 144
cloudmesh.compute.libcloud.Provider (*module*), 145
cloudmesh.compute.virtualbox.Provider (*module*), 145
cloudmesh.compute.vm.Provider (*module*), 145
cloudmesh.config.command.config (*module*), 141
cloudmesh.container.command.container (*module*), 144
cloudmesh.data.api.CloudFile (*module*), 148
cloudmesh.data.api.db.DBProviderABC (*module*), 147
cloudmesh.data.api.Driver (*module*), 148
cloudmesh.data.api.File (*module*), 147
cloudmesh.data.api.storage.StorageProviderABC (*module*), 147
cloudmesh.data.command.data (*module*), 148
cloudmesh.db.strdb (*module*), 130
cloudmesh.DEBUG (*module*), 119
cloudmesh.default.command.default (*module*), 131
cloudmesh.display (*module*), 144
cloudmesh.emr.api.manager (*module*), 148
cloudmesh.emr.command.emr (*module*), 148
cloudmesh.flavor.command.flavor (*module*), 147
cloudmesh.iaas.flavor (*module*), 141
cloudmesh.iaas.image (*module*), 141
cloudmesh.image.api.manager (*module*), 144
cloudmesh.image.command.image (*module*), 144
cloudmesh.image.Image (*module*), 144
cloudmesh.inventory.command.inventory (*module*), 139
cloudmesh.inventory.inventory (*module*), 139
cloudmesh.key.api.key (*module*), 144
cloudmesh.key.api.manager (*module*), 144
cloudmesh.key.command.key (*module*), 144
cloudmesh.login.api.manager (*module*), 148
cloudmesh.login.command.login (*module*), 148
cloudmesh.man.command.man (*module*), 131
cloudmesh.management.configuration.arguments (*module*), 137, 142
cloudmesh.management.configuration.config (*module*), 136, 142
cloudmesh.management.configuration.counter (*module*), 136, 143
cloudmesh.management.configuration.generic_config (*module*), 136, 142
cloudmesh.management.configuration.name (*module*), 136, 137, 142
cloudmesh.management.configuration.operatingsystem (*module*), 137, 142
cloudmesh.management.configuration.SSHkey (*module*), 137, 142
cloudmesh.management.debug (*module*), 139, 143
cloudmesh.management.printer.Printer (*module*), 137, 143
cloudmesh.management.script (*module*), 138, 143
cloudmesh.mongo.CmDatabase (*module*), 145
cloudmesh.mongo.DataBaseDecorator (*module*), 145
cloudmesh.mongo.MongoDBController (*module*), 147
cloudmesh.network.api.manager (*module*), 142
cloudmesh.network.command.network (*module*), 142
cloudmesh.open.command.open (*module*), 148
cloudmesh.secgroup.api.manager (*module*), 144
cloudmesh.secgroup.command.secgroup (*module*), 144
cloudmesh.security.authorized_keys (*module*), 141
cloudmesh.security.encrypt (*module*), 141
cloudmesh.security.ssh_config (*module*), 141
cloudmesh.set.command.set (*module*), 131
cloudmesh.shell.command (*module*), 131
cloudmesh.shell.plugin (*module*), 135
cloudmesh.shell.shell (*module*), 132
cloudmesh.shell.variables (*module*), 132
cloudmesh.source.api.manager (*module*), 142
cloudmesh.source.command.source (*module*), 142
cloudmesh.ssh.api.manager (*module*), 144
cloudmesh.ssh.command.ssh (*module*), 144
cloudmesh.storage.command.storage (*module*), 140
cloudmesh.storage.Provider (*module*), 139
cloudmesh.storage.provider.awss3.Provider

(module), 140
 cloudmesh.storage.provider.azureblob.Provider (module), 136, 143
 (module), 139
 cloudmesh.storage.provider.box.Provider (module), 139
 cloudmesh.storage.provider.gdrive.Provider (module), 139
 cloudmesh.storage.provider.local.Provider (module), 139
 cloudmesh.storage.provider.storage (module), 140
 cloudmesh.storage.spec.cloudmesh.storagecreation_date() (in module cloudmesh.storage.provider.local.Provider), (module), 139
 cloudmesh.storage.spec.tests.test_openapi_storage (module), 140
 (module), 139
 cloudmesh.storage.StorageABC (module), 140
 cloudmesh.terminal.command.terminal (module), 131
 cloudmesh.url.command.url (module), 144
 cloudmesh.var.command.var (module), 131
 cloudmesh.vbox.api.provider (module), 144
 cloudmesh.vbox.command.vbox (module), 144
 cloudmesh.vcluster.api.VirtualCluster (module), 144
 cloudmesh.vcluster.command.vcluster (module), 144
 cloudmesh.vdir.api.manager (module), 140
 cloudmesh.vdir.command.vdir (module), 140
 cloudmesh.vm.api.manager (module), 144
 cloudmesh.vm.command.vm (module), 144
 cloudmesh.workflow.api.manager (module), 145
 cloudmesh.workflow.command.workflow (module), 145
 cm() (cloudmesh.common.Shell.Shell class method), 128
 CMShell (class in cloudmesh.shell.shell), 132
 command (cloudmesh.common.Shell.Shell attribute), 128
 command() (in module cloudmesh.shell.command), 132
 command_exists() (cloudmesh.common.Shell.Shell class method), 128
 config_dir_setup() (in module cloudmesh.common.locations), 122
 config_file() (in module cloudmesh.common.locations), 122
 config_file_prefix() (in module cloudmesh.common.locations), 122
 config_file_raw() (in module cloudmesh.common.locations), 122
 Console (class in cloudmesh.common.console), 125
 convert_from_unicode() (in module cloudmesh.common.util), 120
 copy_files() (in module cloudmesh.common.util), 120
 Counter (class in cloudmesh.management.configuration.counter),
 cprint() (cloudmesh.common.console.Console static method), 126
 create() (cloudmesh.compute.azure.AzProvider.Provider method), 144
 create_dir() (cloudmesh.storage.provider.local.Provider.Provider method), 139
 create_dir_from_filename() (cloudmesh.storage.provider.local.Provider.Provider method), 139
 csv() (cloudmesh.common.Printer.Printer class method), 123

D

DatabaseAlter (class in cloudmesh.mongo.DataBaseDecorator), 145
 DatabaseUpdate (class in cloudmesh.mongo.DataBaseDecorator), 146
 DBProviderABC (class in cloudmesh.data.api.db.DBProviderABC), 147
 debug() (cloudmesh.common.error.Error class method), 127
 debug_msg() (cloudmesh.common.console.Console static method), 126
 decr() (cloudmesh.management.configuration.counter.Counter method), 136, 143
 defaults() (in module cloudmesh.vbox.command.vbox), 144
 delete() (cloudmesh.data.api.db.DBProviderABC.DBProviderABC method), 147
 delete() (cloudmesh.data.api.storage.StorageProviderABC.StorageProviderABC method), 147
 delete() (cloudmesh.storage.provider.local.Provider.Provider method), 140
 destroy() (cloudmesh.compute.azure.AzProvider.Provider method), 144
 dialog() (cloudmesh.common.Shell.Shell class method), 128
 dict() (cloudmesh.common.Printer.Printer class method), 123
 dict_table() (cloudmesh.common.Printer.Printer class method), 123
 Display (class in cloudmesh.display), 144
 do_EOF() (cloudmesh.shell.shell.CMShell method), 132
 do_help() (cloudmesh.shell.shell.CMShell method), 132

`do_info()` (*cloudmesh.shell.shell.CMShell method*), 133
`do_plugin()` (*cloudmesh.shell.shell.CMShell method*), 133
`do_q()` (*cloudmesh.shell.shell.CMShell method*), 133
`do_quit()` (*cloudmesh.shell.shell.CMShell method*), 133
`do_shell()` (*cloudmesh.shell.shell.CMShell method*), 133
`do_version()` (*cloudmesh.shell.shell.CMShell method*), 134
`dotdict` (*class in cloudmesh.common.dotdict*), 122

E

`emptyline()` (*cloudmesh.shell.shell.CMShell method*), 134
`EncryptFile` (*class in cloudmesh.security.encrypt*), 141
`Error` (*class in cloudmesh.common.error*), 127
`error()` (*cloudmesh.common.console.Console class method*), 126
`execute()` (*cloudmesh.common.Shell.Shell class method*), 128
`execute()` (*cloudmesh.common.ssh.ssh_config.ssh_config method*), 131
`execute()` (*cloudmesh.security.ssh_config.ssh_config method*), 141
`exists()` (*cloudmesh.data.api.storage.StorageProviderABC.StorageProviderABC method*), 147
`exit()` (*cloudmesh.common.error.Error class method*), 127
`exponential_backoff()` (*in module cloudmesh.common.util*), 120

F

`fgrep()` (*cloudmesh.common.Shell.Shell class method*), 128
`find_cygwin_executables()` (*cloudmesh.common.Shell.Shell class method*), 128
`find_lines_with()` (*cloudmesh.common.Shell.Shell class method*), 128
`find_process()` (*in module cloudmesh.management.script*), 138, 144
`FlatDict` (*class in cloudmesh.common.FlatDict*), 122
`flatten()` (*in module cloudmesh.common.FlatDict*), 122
`flatwrite()` (*cloudmesh.common.Printer.Printer class method*), 124

G

`generate()` (*cloudmesh.common.ssh.ssh_config.ssh_config method*), 131

`generate()` (*cloudmesh.security.ssh_config.ssh_config method*), 141
`generate_password()` (*in module cloudmesh.common.util*), 120
`get()` (*cloudmesh.common.console.Console static method*), 126
`get()` (*cloudmesh.common.StopWatch.StopWatch class method*), 125
`get()` (*cloudmesh.data.api.storage.StorageProviderABC.StorageProviderABC method*), 147
`get()` (*cloudmesh.management.configuration.counter.Counter method*), 136, 143
`get()` (*cloudmesh.storage.provider.local.Provider.Provider method*), 140
`get_fingerprint_from_public_key()` (*in module cloudmesh.common.ssh.authorized_keys*), 131
`get_fingerprint_from_public_key()` (*in module cloudmesh.security.authorized_keys*), 142
`get_from_git()` (*cloudmesh.management.configuration.SSHkey.SSHkey method*), 137, 142
`get_python()` (*cloudmesh.common.Shell.Shell class method*), 128
`git()` (*cloudmesh.common.Shell.Shell class method*), 129
`grep()` (*cloudmesh.common.Shell.Shell class method*), 129
`grep()` (*in module cloudmesh.common.util*), 120

H

`head()` (*cloudmesh.common.Shell.Shell class method*), 129
`HEADING()` (*in module cloudmesh.common.util*), 119
`help_help()` (*cloudmesh.shell.shell.CMShell method*), 134

I

`incr()` (*cloudmesh.management.configuration.counter.Counter method*), 136, 143
`indent()` (*in module cloudmesh.common.console*), 126
`info()` (*cloudmesh.common.console.Console static method*), 126
`info()` (*cloudmesh.common.error.Error class method*), 127
`info()` (*cloudmesh.compute.azure.AzProvider.Provider method*), 144

K

`key_prefix_replace()` (*in module cloudmesh.common.FlatDict*), 122
`keys()` (*cloudmesh.common.FlatDict.FlatDict method*), 122

- keys() (cloudmesh.common.StopWatch.StopWatch class method), 125
- keystone() (cloudmesh.common.Shell.Shell class method), 129
- kill() (cloudmesh.common.Shell.Shell class method), 129
- ## L
- list() (cloudmesh.common.Printer.Printer class method), 124
- list() (cloudmesh.common.ssh.ssh_config.ssh_config method), 131
- list() (cloudmesh.compute.azure.AzProvider.Provider method), 144
- list() (cloudmesh.security.ssh_config.ssh_config method), 141
- list_files() (cloudmesh.data.api.db.DBProviderABC.DBProviderABC class method), 147
- load() (cloudmesh.common.ssh.authorized_keys.AuthorizedKeys class method), 130
- load() (cloudmesh.common.ssh.ssh_config.ssh_config method), 131
- load() (cloudmesh.security.authorized_keys.AuthorizedKeys class method), 141
- load() (cloudmesh.security.ssh_config.ssh_config method), 141
- load() (cloudmesh.shell.shell.Plugin class method), 135
- local() (cloudmesh.common.ssh.ssh_config.ssh_config method), 131
- local() (cloudmesh.security.ssh_config.ssh_config method), 141
- LOGGER() (in module cloudmesh.common.logger), 126
- LOGGING_OFF() (in module cloudmesh.common.logger), 127
- LOGGING_ON() (in module cloudmesh.common.logger), 127
- login() (cloudmesh.common.ssh.ssh_config.ssh_config method), 131
- login() (cloudmesh.security.ssh_config.ssh_config method), 141
- ls() (cloudmesh.common.Shell.Shell class method), 129
- ## M
- main() (in module cloudmesh.common.Shell), 130
- main() (in module cloudmesh.shell.shell), 135
- map_parameters() (in module cloudmesh.shell.command), 132
- mkdir() (cloudmesh.common.Shell.Shell class method), 129
- modules() (cloudmesh.shell.shell.Plugin class method), 135
- mongod() (cloudmesh.common.Shell.Shell class method), 129
- msg() (cloudmesh.common.console.Console static method), 126
- msg() (cloudmesh.common.error.Error class method), 127
- ## N
- Name (class in cloudmesh.management.configuration.name), 137, 138, 143
- name() (cloudmesh.shell.shell.Plugin class method), 135
- names() (cloudmesh.common.ssh.ssh_config.ssh_config method), 131
- names() (cloudmesh.security.ssh_config.ssh_config method), 141
- nosetests() (cloudmesh.common.Shell.Shell class method), 129
- ## O
- ok() (cloudmesh.common.console.Console static method), 126
- oscmd() (cloudmesh.shell.shell.CMShell method), 134
- operating_system() (cloudmesh.common.Shell.Shell class method), 129
- ## P
- pandoc() (cloudmesh.common.Shell.Shell class method), 129
- path_expand() (in module cloudmesh.common.util), 120
- pem_verify() (cloudmesh.security.encrypt.EncryptFile method), 141
- ping() (cloudmesh.common.Shell.Shell class method), 129
- pip() (cloudmesh.common.Shell.Shell class method), 129
- Plugin (class in cloudmesh.shell.shell), 135
- PluginCommandClasses (in module cloudmesh.shell.shell), 135
- postcmd() (cloudmesh.shell.shell.CMShell method), 134
- precmd() (cloudmesh.shell.shell.CMShell method), 135
- preloop() (cloudmesh.shell.shell.CMShell method), 135
- print() (cloudmesh.common.StopWatch.StopWatch class method), 125
- print_list() (cloudmesh.common.Printer.Printer class method), 124
- print_list() (in module cloudmesh.shell.shell), 135
- Printer (class in cloudmesh.common.Printer), 123

Provider (class in *cloudmesh.compute.azure.AzProvider*), 144
 Provider (class in *cloudmesh.storage.provider.local.Provider*), 139
 ps () (*cloudmesh.common.Shell.Shell* class method), 129
 put () (*cloudmesh.data.api.storage.StorageProviderABC.StorageProviderABC* class method), 147
 put () (*cloudmesh.storage.provider.local.Provider.Provider* class method), 140
 pwd () (*cloudmesh.common.Shell.Shell* class method), 129
R
 rackdiag () (*cloudmesh.common.Shell.Shell* class method), 129
 readfile () (in module *cloudmesh.common.util*), 121
 remove () (*cloudmesh.common.ssh.authorized_keys.AuthorizedKeys* class method), 131
 remove () (*cloudmesh.security.authorized_keys.AuthorizedKeys* class method), 142
 remove_line_with () (*cloudmesh.common.Shell.Shell* class method), 129
 rename () (*cloudmesh.compute.azure.AzProvider.ProviderStopWatch* class method), 144
 resume () (*cloudmesh.compute.azure.AzProvider.ProviderStorageProviderABC* class method), 145
 rm () (*cloudmesh.common.Shell.Shell* class method), 129
 row_table () (*cloudmesh.common.Printer.Printer* class method), 124
 rsync () (*cloudmesh.common.Shell.Shell* class method), 129
 run () (*cloudmesh.management.script.Script* static method), 138, 143
S
 scp () (*cloudmesh.common.Shell.Shell* class method), 129
 Script (class in *cloudmesh.management.script*), 138, 143
 search () (*cloudmesh.storage.provider.local.Provider.Provider* class method), 140
 search () (in module *cloudmesh.common.util*), 121
 set () (*cloudmesh.management.configuration.counter.Counter* class method), 136, 143
 set_debug () (*cloudmesh.common.console.Console* class method), 126
 set_permissions () (*cloudmesh.management.configuration.SSHkey.SSHkey* class method), 137, 142
 set_theme () (*cloudmesh.common.console.Console* static method), 126
 sh () (*cloudmesh.common.Shell.Shell* class method), 129
 Shell (class in *cloudmesh.common.Shell*), 127
 sort () (*cloudmesh.common.Shell.Shell* class method), 130
 ssh_config (class in *cloudmesh.common.ssh.ssh_config*), 131
 ssh_config (class in *cloudmesh.security.ssh_config*), 141
 SSHkey (class in *cloudmesh.management.configuration.SSHkey*), 137, 142
 start () (*cloudmesh.common.StopWatch.StopWatch* class method), 125
 start () (*cloudmesh.compute.azure.AzProvider.Provider* class method), 145
 State (class in *cloudmesh.abstractclass.State*), 145
 status () (*cloudmesh.common.ssh.ssh_config.ssh_config* class method), 141
 status () (*cloudmesh.security.ssh_config.ssh_config* class method), 141
 stop () (*cloudmesh.common.StopWatch.StopWatch* class method), 125
 stop () (*cloudmesh.compute.azure.AzProvider.Provider* class method), 145
 StopWatch (class in *cloudmesh.common.StopWatch*), 125
 StorageProviderABC (class in *cloudmesh.data.api.storage.StorageProviderABC*), 147
 str_banner () (in module *cloudmesh.common.util*), 121
 Subprocess (class in *cloudmesh.common.Shell*), 130
 SubprocessError, 130
 sudo () (*cloudmesh.common.Shell.Shell* class method), 130
 suspend () (*cloudmesh.compute.azure.AzProvider.Provider* class method), 145
 SystemPath (class in *cloudmesh.management.script*), 138, 143
T
 tail () (*cloudmesh.common.Shell.Shell* class method), 130
 tempdir () (in module *cloudmesh.common.util*), 121
 terminal_type () (*cloudmesh.common.Shell.Shell* class method), 130
 Test_cloud_storage (class in *cloudmesh.storage.spec.tests.test_openapi_storage*), 139
 TODO () (*cloudmesh.common.console.Console* static method), 125
 traceback () (*cloudmesh.common.error.Error* class method), 127
 tree () (*cloudmesh.storage.provider.local.Provider.Provider* class method), 140

`txt_msg()` (*cloudmesh.common.console.Console static method*), [126](#)

U

`unzip()` (*cloudmesh.common.Shell.Shell method*), [130](#)
`update()` (*cloudmesh.data.api.db.DBProviderABC.DBProviderABC method*), [147](#)
`username()` (*cloudmesh.common.ssh.ssh_config.ssh_config method*), [131](#)
`username()` (*cloudmesh.security.ssh_config.ssh_config method*), [141](#)

V

`vagrant()` (*cloudmesh.common.Shell.Shell class method*), [130](#)
`values()` (*cloudmesh.common.FlatDict.FlatDict method*), [122](#)
`VBoxManage()` (*cloudmesh.common.Shell.Shell class method*), [127](#)

W

`warning()` (*cloudmesh.common.console.Console static method*), [126](#)
`warning()` (*cloudmesh.common.error.Error class method*), [127](#)
`which()` (*cloudmesh.common.Shell.Shell class method*), [130](#)
`write()` (*cloudmesh.common.Printer.Printer class method*), [124](#)
`writefile()` (*in module cloudmesh.common.util*), [121](#)

Y

`YamlDB` (*class in cloudmesh.db.strdb*), [130](#)
`yn_choice()` (*in module cloudmesh.common.util*), [121](#)