

Cloudmesh REST Interface for Virtual Clusters

GREGOR VON LASZEWSKI^{1,*}, FUGANG WANG¹, AND BADI ABDHUL-WAHID¹

¹School of Informatics and Computing, Bloomington, IN 47408, U.S.A.

*Corresponding authors: laszewski@gmail.com

Draft v0.0.1, May 2, 2017

This document summarizes a number of objects that are instrumental for the interaction with Clouds, Containers, and HPC systems to manage virtual clusters. TBD

© 2017 <https://creativecommons.org/licenses/>. The authors verify that the text is not plagiarized.

Keywords: Cloudmesh, REST, NIST

<https://github.com/cloudmesh/rest/tree/master/docs>

1	CONTENTS	33	4.2.3 Azure Vm	7
2	1 Introduction	2	5 HPC	7
3	1.1 Lessons Learned	2	5.1 Batch Job	7
4	1.2 Hybrid Cloud	2	6 Virtual Cluster	7
5	1.3 Design by Example	2	6.1 Cluster	7
6	1.4 Tools to Create the Specifications	2	6.2 Compute Resource	8
7	1.5 Installation of the Tools	2	6.3 Computer	8
8	1.6 Document Creation	3	6.4 Compute Node	8
9	1.7 Conversion to Word	3	6.5 Virtual Cluster	9
10	1.8 Interface Compliancy	3	6.6 Virtual Compute node	9
11	2 User and Profile	3	6.7 Virtual Machine	9
12	2.1 Profile	3	6.8 Mesos	9
13	2.2 User	3	7 Containers	9
14	2.3 Organization	3	7.1 Container	9
15	2.4 Group/Role	4	7.2 Kubernetes	10
16	3 Data	4	8 Deployment	10
17	3.1 Var	4	8.1 Deployment	10
18	3.2 Default	4	9 Mapreduce	10
19	3.3 File	4	9.1 Mapreduce	10
20	3.4 File Alias	5	9.2 Hadoop	11
21	3.5 Replica	5	10 Security	12
22	3.6 Virtual Directory	5	10.1 Key	12
23	3.7 Database	5	11 Microservice	12
24	3.8 Stream	5	11.1 Microservice	12
25	4 IaaS	6	11.2 Reservation	12
26	4.1 Openstack	6	12 Network	12
27	4.1.1 Openstack Flavor	6	A Schema Command	13
28	4.1.2 Openstack Image	6		
29	4.1.3 Openstack Vm	6		
30	4.2 Azure	6		
31	4.2.1 Azure Size	6		
32	4.2.2 Azure Image	7		

60	B Schema	13
61	C Contributing	13
62	D Using the Cloudmesh REST Service	13
63	D.1 Element Definition	13
64	D.2 Yaml	13
65	D.3 Cerberus	13
66	E Mongoengine	13
67	F Cloudmesh Notation	13
68	F.1 Defining Elements for the REST Service	13
69	F.2 DOIT	14
70	F.3 Generating service	14
71	G ABC	14
72	H Cloudmesh Rest	14
73	H.1 Prerequistis	14
74	H.1.1 Install Mongo on OSX	14
75	H.1.2 Install Mongo on OSX	14
76	H.2 Introduction	14
77	H.3 Yaml Specification	14
78	H.4 Json Specification	14
79	H.5 Conversion to Eve Settings	14
80	H.5.1 Managing Mongo	14
81	H.5.2 Manageing Eve	14
82	I REST with Eve	15
83	I.1 Overview of REST	15
84	I.2 REST and eve	15
85	I.2.1 Installation	15
86	I.2.2 Starting the service	15
87	I.3 Creating your own objects	15
88	I.4 Towards cmd5 extensions to manage eve and	
89	mongo	16
90	J CMD5	16
91	J.1 Resources	16
92	J.2 Execution	16
93	J.3 Create your own Extension	17
94	J.4 Excercise	17

95 1. INTRODUCTION

96 In this document we summarize elementary objects that are
97 important to for the NBDRA.

98 1.1. Lessons Learned

99 TBD

100 1.2. Hybrid Cloud

101 TBD

102 1.3. Design by Example

103 To accelerate discussion among the team we use an approach
104 to define objects and its interfaces by example. These exam-
105 ples are than taken in a later version of the document and a
106 schema is generated from it. The schema will be added in
107 its complete form to the appendix B. While focusing first on
108 examples it allows us to speed up our design and simplifies

109 discussions of the objects and interfaces eliminating getting
110 lost in complex syntactical specifications. The process and
111 specifications used in this document will also allow us to
112 automatically create a implementation of the objects that can
113 be integrated into a reference architecture as provided by for
114 example the cloudmesh client and rest project [?].

115 An example object will demonstrate our approach. The
116 following object defines a JSON object representing a user.

Listing 1.1: User profile

```

1 {
2   "profile": {
3     "description": "The Profile of a user",
4     "uuid": "jshdjkdh...",
5     "context": "resource",
6     "email": "laszewski@gmail.com",
7     "firstname": "Gregor",
8     "lastname": "von Laszewski",
9     "username": "gregor"
10  }
11 }
```

117 Such an object can be transformed to a schema specifica-
118 tion while introspecting the types of the original example.
119 The resulting schema object follows the Cerberus [?] speci-
120 fication and looks for our object as follows:

```

profile = {
  'description': { 'type': 'string'},
  'email': { 'type': 'email' },
  'firstname': { 'type': 'string'},
  'lastname': { 'type': 'string' },
  'username': { 'type': 'string' }
}
```

122 As mentioned before, the AppendixB will list the schema
123 that is automatically created from the definitions.

124 1.4. Tools to Create the Specifications

125 The tools to create the schema and object are all available
126 opensource and are hosted on github. It includes the follow-
127 ing repositories:

128 **cloudmesh.common**

129 <https://github.com/cloudmesh/cloudmesh.common>

130 **cloudmesh.cmd5**

131 <https://github.com/cloudmesh/cloudmesh.cmd5>

132 **cloudmesh.rest**

133 <https://github.com/cloudmesh/cloudmesh.rest>

134 **cloudmesh/evegenie**

135 <https://github.com/cloudmesh/evegenie>

136 1.5. Installation of the Tools

137 The current best way to install the tools is from source. A
138 convenient shell script conducting the install is located at:

139 TBD

140 Once we have stabilized the code the package will be
141 available from pypi and can be installed as follows:

```

pip install cloudmesh.rest
pip install cloudmesh.evegenie
```

1.6. Document Creation

It is assumed that you have installed all the tools. TO create the document you can simply do

```
git clone https://github.com/cloudmesh/cloudmesh.rest
cd cloudmesh.rest/docs
make
```

This will produce in that directory a file called object.pdf containing this document.

1.7. Conversion to Word

We found that it is inconvenient for the developers to maintain this document in Microsoft Word as typically is done for other documents. This is because the majority of the information contains specifications that are directly integrated in a reference implementation, as well as that the current majority of contributors are developers. We would hope that editorial staff provides direct help to improve this document, which even can be done through the github Web interface and does not require any access either to the tools mentioned above or the availability of L^AT_EX.

The files are located at:

- <https://github.com/cloudmesh/cloudmesh.rest/tree/master/docs>

1.8. Interface Compliancy

Due to the extensibility of our interfaces it is important to introduce a terminology that allows us to define interface compliancy. We define it as follows

Full Compliance: These are reference implementations that provide full compliance to the objects defined in this document. A version number will be added to assure the snapshot in time of the objects is associated with the version. This reference implementation will implement all objects.

Partially Compliance: These are reference implementations that provide partial compliance to the objects defined in this document. A version number will be added to assure the snapshot in time of the objects is associated with the version. This reference implementation will implement a partial list of the objects. A document is accompanied that lists all objects defined, but also lists the objects that are not defined by the reference architecture.

Full and extended Compliance: These are interfaces that in addition to the full compliance also introduce additional interfaces and extend them.

2. USER AND PROFILE

In a multiuser environment we need a simple mechanism of associating objects and data to a particular person or group. While we do not want to replace with our efforts more elaborate solutions such as proposed by eduPerson (<http://software.internet2.edu/eduperson/internet2-mace-dir-eduperson-201602.html>) or others [?], we need a very simple way of distinguishing users. Therefore we have introduced a number of simple objects including a profile and a user.

2.1. Profile

A profile is simple the most elementary information to distinguish a user profile. It contains name and e-mail information. It may have an optional uuid and/or use a unique e-mail to distinguish a user.

Listing 2.1: User profile

```
{
  "profile": {
    "description": "The Profile of a user",
    "uuid": "jshdjkdh...",
    "context": "resource",
    "email": "laszewski@gmail.com",
    "firstname": "Gregor",
    "lastname": "von Laszewski",
    "username": "gregor"
  }
}
```

2.2. User

In contrast to the profile a user contains additional attributes that define the role of the user within the system.

Listing 2.2: user

```
{
  "user": {
    "uuid": "jshdjkdh...",
    "context": "resource",
    "email": "laszewski@gmail.com",
    "firstname": "Gregor",
    "lastname": "von Laszewski",
    "username": "gregor",
    "roles": ["admin", "user"]
  }
}
```

2.3. Organization

An important concept in many applications is the management of a group of users in a virtual organization. This can be achieved through two concepts. First, it can be achieved while using the profile and user resources itself as they contain the ability to manage multiple users as part of the REST interface. The second concept is to create a virtual organization that lists all users of this virtual organization. The third concept is to introduce groups and roles either as part of the user definition or as part of a simple list similar to the organization

Listing 2.3: user

```
{
  "organization": {
    "users": [
      "objectid:user"
    ]
  }
}
```

2.4. Group/Role

A group contains a number of users. It is used to manage authorized services.

Listing 2.4: group

```
{
  "group": {
    "name": "users",
    "description": "This group contains all
      ↪ users",
    "users": [
      "objectid:user"
    ]
  }
}
```

A role is a further refinement of a group. Group members can have specific roles. A good example is that ability to formulate a group of users that have access to a repository. However the role defines more specifically read and write privileges to the data within the repository.

Listing 2.5: role

```
{
  "role": {
    "name": "editor",
    "description": "This role contains all
      ↪ editors",
    "users": [
      "objectid:user"
    ]
  }
}
```

3. DATA

Data for Big Data applications are delivered through data providers. They can be either local providers contributed by a user or distributed data providers that refer to data on the internet. At this time we focus on an elementary set of abstractions related to data providers that offer us to utilize variables, files, virtual data directories, data streams, and data filters.

Variables are used to hold specific contents that is associated in programming language as a variable. A variable has a name, value and type.

Default is a special type of variable that allows adding of a context. defaults can than created for different contexts.

Files are used to represent information collected within the context of classical files in an operating system.

Streams are services that offer the consumer a stream of data. Streams may allow the initiation of filters to reduce the amount of data requested by the consumer Stream Filters operate in streams or on files converting them to streams

Batch Filters operate on streams and on files while working in the background and delivering as output Files

Virtual directories and non-virtual directories are collection of files that organize them. For our initial purpose the distinction between virtual and non-virtual directories is non-essential and we will focus on abstracting all directories to be virtual. This could mean that the files are physically hosted on different disks. However, it is important to note that virtual data directories can hold more than files, they can also contain data streams and data filters.

3.1. Var

variables are used to store a simple values. Each variable can have a type. The variable value format is defined as string to allow maximal probability. The type of the value is also provided.

Listing 3.1: var

```
{
  "var": {
    "name": "name of the variable",
    "value": "the value of the variable as
      ↪ string",
    "type": "the datatype of the variable such
      ↪ as int, str, float, ..."
  }
}
```

3.2. Default

A default is a special variable that has a context associated with it. This allow su to define values that can be easily retrieved based on its context. A good example for a default would be the image name for a cloud where the context is defined by the cloud name.

Listing 3.2: default

```
{
  "default": {
    "value": "string",
    "name": "string",
    "context": "string - defines the context
      ↪ of the default (user, cloud, ...)"
  }
}
```

3.3. File

A file is a computer resource allowing to store data that is being processed. The interface to a file provides the mechanism to appropriately locate a file in a distributed system. Identification include the name, and endpoint, the checksum and the size. Additional parameters such as the lasst access time could be stored also. As such the Interface only describes the location of the file

The **file** object has *name*, *endpoint* (location), *size* in GB, MB, Byte, *checksum* for integrity check, and last *accessed* timestamp.

Listing 3.3: file

```
{
  "file": {
    "name": "report.dat",
```

```

4      "endpoint":
5      ↪ "file://gregor@machine.edu:/data/report.dat",
6      "checksum":
7      ↪ {"md5": "8c324f12047dc2254b74031b8f029ad0"},
8      "accessed": "1.1.2017:05:00:00:EST",
9      "created": "1.1.2017:05:00:00:EST",
10     "modified": "1.1.2017:05:00:00:EST",
11     "size": ["GB", "Byte"]
12 }
13 }

```

3.4. File Alias

A file could have one alias or even multiple ones.

Listing 3.4: file alias

```

1 {
2   "file_alias": {
3     "alias": "report-alias.dat",
4     "name": "report.dat"
5   }
6 }

```

3.5. Replica

In many distributed systems, it is of importance that a file can be replicated among different systems in order to provide faster access. It is important to provide a mechanism that allows to trace the pedigree of the file while pointing to its original source

Listing 3.5: replica

```

1 {
2   "replica": {
3     "name": "replica_report.dat",
4     "replica": "report.dat",
5     "endpoint":
6     ↪ "file://gregor@machine.edu:/data/replica_report.dat",
7     "checksum": {
8       "md5":
9       ↪ "8c324f12047dc2254b74031b8f029ad0"
10    },
11    "accessed": "1.1.2017:05:00:00:EST",
12    "size": [
13      "GB",
14      "Byte"
15    ]
16 }

```

3.6. Virtual Directory

A collection of files or replicas. A virtual directory can contain an number of entities including files, streams, and other virtual directories as part of a collection. The element in the collection can either be defined by uuid or by name.

Listing 3.6: virtual directory

```

1 {
2   "virtual_directory": {
3     "name": "data",

```

```

4     "endpoint": "http://.../data/",
5     "protocol": "http",
6     "collection": [
7       "report.dat",
8       "file2"
9     ]
10  }
11 }

```

3.7. Database

A **database** could have a name, an *endpoint* (e.g., host:port), and protocol used (e.g., SQL, mongo, etc.).

Listing 3.7: database

```

1 {
2   "database": {
3     "name": "data",
4     "endpoint": "http://.../data/",
5     "protocol": "mongo"
6   }
7 }

```

3.8. Stream

A stream provides a stream of data while providing information about rate and number of items exchanged while issuing requests to the stream. A stream may return data items in a specific format that is defined by the stream.

Listing 3.8: stream

```

1 {
2   "stream": {
3     "name": "name of the variable",
4     "format": "the format of the data
5     ↪ exchanged in the stream",
6     "attributes": {
7       "rate": 10,
8       "limit": 1000
9     }
10  }

```

Examples for streams could be a stream of random numbers but could also include more complex formats such as the retrieval of data records.

Services can subscribe, unsubscribe from a stream, while also applying filters to the subscribed stream.

Listing 3.9: filter

```

1 {
2   "filter": {
3     "name": "name of the filter",
4     "function": "the function of the data
5     ↪ exchanged in the stream"
6   }

```

Filter needs to be refined

4. IAAS

In this section we are defining resources related to Infrastructure as a Service frameworks. This includes specific objects useful for OpenStack, Azure, and AWS, as well as others.

4.1. Openstack

4.1.1. Openstack Flavor

Listing 4.1: openstack flavor

```
{
  "openstack_flavor": {
    "os_flv_disabled": "string",
    "uuid": "string",
    "os_flv_ext_data": "string",
    "ram": "string",
    "os_flavor_acces": "string",
    "vcpus": "string",
    "swap": "string",
    "rxtx_factor": "string",
    "disk": "string"
  }
}
```

4.1.2. Openstack Image

Listing 4.2: openstack image

```
{
  "openstack_image": {
    "status": "string",
    "username": "string",
    "updated": "string",
    "uuid": "string",
    "created": "string",
    "minDisk": "string",
    "progress": "string",
    "minRam": "string",
    "os_image_size": "string",
    "metadata": {
      "image_location": "string",
      "image_state": "string",
      "description": "string",
      "kernel_id": "string",
      "instance_type_id": "string",
      "ramdisk_id": "string",
      "instance_type_name": "string",
      "instance_type_rxtx_factor": "string",
      "instance_type_vcpus": "string",
      "user_id": "string",
      "base_image_ref": "string",
      "instance_uuid": "string",
      "instance_type_memory_mb": "string",
      "instance_type_swap": "string",
      "image_type": "string",
      "instance_type_ephemeral_gb": "string",
      "instance_type_root_gb": "string",
      "network_allocated": "string",
      "instance_type_flavorid": "string",
      "owner_id": "string"
    }
  }
}
```

```
}
```

4.1.3. Openstack Vm

Listing 4.3: openstack vm

```
{
  "openstack_vm": {
    "username": "string",
    "vm_state": "string",
    "updated": "string",
    "hostId": "string",
    "availability_zone": "string",
    "terminated_at": "string",
    "image": "string",
    "floating_ip": "string",
    "diskConfig": "string",
    "key": "string",
    "flavor__id": "string",
    "user_id": "string",
    "flavor": "string",
    "static_ip": "string",
    "security_groups": "string",
    "volumes_attached": "string",
    "task_state": "string",
    "group": "string",
    "uuid": "string",
    "created": "string",
    "tenant_id": "string",
    "accessIPv4": "string",
    "accessIPv6": "string",
    "status": "string",
    "power_state": "string",
    "progress": "string",
    "image_id": "string",
    "launched_at": "string",
    "config_drive": "string"
  }
}
```

4.2. Azure

4.2.1. Azure Size

The size description of an azure vm

Listing 4.4: azure-size

```
{
  "azure-size": {
    "_uuid": "None",
    "name": "D14 Faster Compute Instance",
    "extra": {
      "cores": 16,
      "max_data_disks": 32
    },
    "price": 1.6261,
    "ram": 114688,
    "driver": "libcloud",
    "bandwidth": "None",
    "disk": 127,
    "id": "Standard_D14"
  }
}
```



```
16 }
329
```

4.2.2. Azure Image

Listing 4.5: azure-image

```
1 {
2   "azure_image": {
3     "_uuid": "None",
4     "driver": "libcloud",
5     "extra": {
6       "affinity_group": "",
7       "category": "Public",
8       "description": "Linux VM image with
9         ↳ coreclr-x64-beta5-11624 installed to
10        ↳ /opt/dnx. This image is based on
11        ↳ Ubuntu 14.04 LTS, with prerequisites
12        ↳ of CoreCLR installed. It also
13        ↳ contains PartsUnlimited demo app
14        ↳ which runs on the installed coreclr.
15        ↳ The demo app is installed to
16        ↳ /opt/demo. To run the demo, please
17        ↳ type the command /opt/demo/Kestrel
18        ↳ in a terminal window. The website is
19        ↳ listening on port 5004. Please
20        ↳ enable or map a endpoint of HTTP
21        ↳ port 5004 for your azure VM.",
22       "location": "East Asia;Southeast
23        ↳ Asia;Australia East;Australia
24        ↳ Southeast;Brazil South;North
25        ↳ Europe;West Europe;Japan East;Japan
26        ↳ West;Central US;East US;East US 2;
27        ↳ North Central US;South Central
28        ↳ US;West US",
29       "media_link": "",
30       "os": "Linux",
31       "vm_image": "False"
32     },
33     "id": "03f55de797f546a1b29d1...",
34     "name": "CoreCLR x64 Beta5 (11624) with
35       ↳ PartsUnlimited Demo App on Ubuntu
36       ↳ Server 14.04 LTS"
37   }
38 }
```

4.2.3. Azure Vm

An Azure virtual machine

Listing 4.6: azure-vm

```
1 {
2   "azure-vm": {
3     "username": "string",
4     "status": "string",
5     "deployment_slot": "string",
6     "cloud_service": "string",
7     "image": "string",
8     "floating_ip": "string",
9     "image_name": "string",
10    "key": "string",
11    "flavor": "string",
12    "resource_location": "string",
```

```
13 "disk_name": "string",
14 "private_ips": "string",
15 "group": "string",
16 "uuid": "string",
17 "dns_name": "string",
18 "instance_size": "string",
19 "instance_name": "string",
20 "public_ips": "string",
21 "media_link": "string"
22 }
23 }
```

5. HPC

5.1. Batch Job

Listing 5.1: batchjob

```
1 {
2   "batchjob": {
3     "output_file": "string",
4     "group": "string",
5     "job_id": "string",
6     "script": "string, the batch job script",
7     "cmd": "string, executes the cmd, if None
8       ↳ path is used",
9     "queue": "string",
10    "cluster": "string",
11    "time": "string",
12    "path": "string, path of the batchjob, if
13      ↳ non cmd is used",
14    "nodes": "string",
15    "dir": "string"
16  }
17 }
```

6. VIRTUAL CLUSTER

6.1. Cluster

The cluster object has name, label, endpoint and provider. The *endpoint* defines.... The *provider* defines the nature of the cluster, e.g., its a virtual cluster on an openstack cloud, or from AWS, or a bare-metal cluster.

Listing 6.1: cluster

```
1 {
2   "cluster": {
3     "label": "c0",
4     "endpoint": {
5       "passwd": "secret",
6       "url": "https"
7     },
8     "name": "myCLuster",
9     "provider": [
10      "openstack",
11      "aws",
12      "azure",
13      "eucalyptus"
14    ]
15  }
16 }
```

6.2. Compute Resource

An important concept for big data analysis is the representation of a compute resource on which we execute the analysis. We define a compute resource by name and by endpoint. A compute resource is an abstract concept and can be instantiated through virtual machines, containers, or bare metal resources. This is defined by the “kind” of the compute resource.

compute_resource object has attribute *endpoint* which specifies ... The *kind* could be *baremetal* or *VC*.

Listing 6.2: compute resource

```
{
  "compute_resource": {
    "name": "Compute1",
    "endpoint": "http://.../cluster/",
    "kind": "baremetal"
  }
}
```

6.3. Computer

This defines a **computer** object. A computer has name, label, IP address. It also lists the relevant specs such as memory, disk size, etc.

Listing 6.3: computer

```
{
  "computer": {
    "ip": "127.0.0.1",
    "name": "myComputer",
    "memoryGB": 16,
    "label": "server-001"
  }
}
```

6.4. Compute Node

A node is composed of multiple components:

1. Metadata such as the name or owner.
2. Physical properties such as cores or memory.
3. Configuration guidance such as `create_external_ip`, `security_groups`, or `users`.

The metadata is associated with the node on the provider end (if supported) as well as in the database. Certain parts of the metadata (such as owner) can be used to implement access control. Physical properties are relevant for the initial allocation of the node. Other configuration parameters control and further provisioning.

In the above, after allocation, the node is configured with a user called `hello` who is part of the `wheel` group whose account can be accessed with several SSH identities whose public keys are provided (in `authorized_keys`).

Additionally, three ssh keys are generated on the node for the `hello` user. The first uses the `ed25519` cryptographic method with a password read in from a GPG-encrypted file on the Command and Control node. The second is a 4098-bit

RSA key also password-protected from the GPG-encrypted file. The third key is copied to the remote node from an encrypted file on the Command and Control node.

This definition also provides a security group to control access to the node from the wide-area-network. In this case all ingress and egress TCP and UDP traffic is allowed provided they are to ports 22 (SSH), 443 (SSL), and 80 and 8080 (web).

Listing 6.4: node

```
{
  "node_new": {
    "authorized_keys": [
      "ssh-rsa AAAA...",
      "ssh-ed25519 AAAA...",
      "...etc"
    ],
    "name": "example-001",
    "external_ip": "",
    "loginuser": "root",
    "create_external_ip": true,
    "internal_ip": "",
    "memory": 2048,
    "owner": "",
    "cores": 2,
    "users": {
      "name": "hello",
      "groups": [
        "wheel"
      ]
    },
    "disk": 80,
    "security_groups": [
      {
        "ingress": "0.0.0.0/32",
        "egress": "0.0.0.0/32",
        "ports": [
          22,
          443,
          80,
          8080
        ],
        "protocols": [
          "tcp",
          "udp"
        ]
      }
    ],
    "ssh_keys": [
      {
        "to": ".ssh/id_rsa",
        "password": {
          "decrypt": "gpg",
          "from": "yaml",
          "file": "secrets.yml.gpg",
          "key": "users.hello.ssh[0]"
        },
        "method": "ed25519",
        "ssh_keygen": true
      },
      {
        "to": ".ssh/testing",

```



```

53     "password": {
54         "decrypt": "gpg",
55         "from": "yaml",
56         "file": "secrets.yml.gpg",
57         "key": "users.hello.ssh[1]"
58     },
59     "bits": 4098,
60     "method": "rsa",
61     "ssh_keygen": true
62 },
63 {
64     "decrypt": "gpg",
65     "from":
66         ↪ "secrets/ssh/hello/copied.gpg",
67     "ssh_keygen": false,
68     "to": ".ssh/copied"
69 }
70 }
71 }

```

6.5. Virtual Cluster

A virtual cluster is an agglomeration of virtual compute nodes that constitute the cluster. Nodes can be assembled to be baremetal, virtual machines, and containers. A virtual cluster contains a number of virtual compute nodes.

Listing 6.5: virtual cluster

```

1 {
2     "virtual_cluster": {
3         "name": "myvc",
4         "frontend": "objectid:virtual_machine",
5         "nodes": [
6             "objectid:virtual_machine"
7         ]
8     }
9 }

```

6.6. Virtual Compute node

Listing 6.6: virtual compute node

```

1 {
2     "virtual_compute_node": {
3         "name": "data",
4         "endpoint": "http://.../cluster/",
5         "metadata": {
6             "experiment": "exp-001"
7         },
8         "image": "Ubuntu-16.04",
9         "ip": [
10             "TBD"
11         ],
12         "flavor": "TBD",
13         "status": "TBD"
14     }
15 }

```

6.7. Virtual Machine

Virtual Machine Virtual machines are an emulation of a computer system. We are maintaining a very basic set of information. It is expected that through the endpoint the virtual machine can be introspected and more detailed information can be retrieved.

Listing 6.7: virtual machine

```

1 {
2     "virtual_machine": {
3         "name": "vm1",
4         "ncpu": 2,
5         "RAM": "4G",
6         "disk": "40G",
7         "nics": ["objectid:nic"
8         ],
9         "OS": "Ubuntu-16.04",
10        "loginuser": "ubuntu",
11        "status": "active",
12        "metadata": {
13        },
14        "authorized_keys": [
15            "objectid:sshkey"
16        ]
17    }
18 }

```

6.8. Mesos

Refine

Listing 6.8: mesos

```

1 {
2     "mesos-docker": {
3         "instances": 1,
4         "container": {
5             "docker": {
6                 "credential": {
7                     "secret": "my-secret",
8                     "principal": "my-principal"
9                 },
10                "image": "mesosphere/inky"
11            },
12            "type": "MESOS"
13        },
14        "mem": 16.0,
15        "args": [
16            "argument"
17        ],
18        "cpus": 0.2,
19        "id": "mesos-docker"
20    }
21 }

```

7. CONTAINERS

7.1. Container

This defines **container** object.

Listing 7.1: container

```

1 {
2   "container": {
3     "name": "container1",
4     "endpoint": "http://.../container/",
5     "ip": "127.0.0.1",
6     "label": "server-001",
7     "memoryGB": 16
8   }
9 }

```

7.2. Kubernetes

REFINE

Listing 7.2: kubernetes

```

1 {
2   "kubernetes": {
3     "kind": "List",
4     "items": [
5       {
6         "kind": "None",
7         "metadata": {
8           "name": "127.0.0.1"
9         },
10        "status": {
11          "capacity": {
12            "cpu": "4"
13          },
14          "addresses": [
15            {
16              "type": "LegacyHostIP",
17              "address": "127.0.0.1"
18            }
19          ]
20        },
21      },
22      {
23        "kind": "None",
24        "metadata": {
25          "name": "127.0.0.2"
26        },
27        "status": {
28          "capacity": {
29            "cpu": "8"
30          },
31          "addresses": [
32            {
33              "type": "LegacyHostIP",
34              "address": "127.0.0.2"
35            },
36            {
37              "type": "another",
38              "address": "127.0.0.3"
39            }
40          ]
41        },
42      },
43    ],
44    "users": [

```

```

45 {
46   "name": "myself",
47   "user": "gregor"
48 },
49 {
50   "name": "e2e",
51   "user": {
52     "username": "admin",
53     "password": "secret"
54   }
55 }
56 ]
57 }
58 }

```

8. DEPLOYMENT

8.1. Deployment

A **deployment** consists of the resource *cluster*, the location *provider*, e.g., AWS, OpenStack, etc., and software *stack* to be deployed (e.g., hadoop, spark).

Listing 8.1: deployment

```

1 {
2   "deployment": {
3     "cluster": [{ "name": "myCluster"},
4                 { "id" : "cm-0001"}
5               ],
6     "stack": {
7       "layers": [
8         "zookeeper",
9         "hadoop",
10        "spark",
11        "postgresql"
12      ],
13      "parameters": {
14        "hadoop": {
15          ↪ "zookeeper.quorum": [
16            ↪ "IP", "IP", "IP"
17          ]
18        }
19      }
20    }
21  }
22 }

```

9. MAPREDUCE

9.1. Mapreduce

The **mapreduce** deployment has as inputs parameters defining the applied function and the input data. Both function and data objects define a “source” parameter, which specify the location it is retrieved from. For instance, the “file:///” URI indicates sending a directory structure from the local file system where the “ftp:///” indicates that the data should be fetched from a FTP resource. It is the framework’s responsibility to materialize and instantiation of the desired environment along with the function and data.

Listing 9.1: mapreduce

```

1 {
2   "mapreduce": {
3     "function": {
4       "source": "file://.",
5       "args": {}
6     },
7     "data": {
8       "source": "ftp:///...",
9       "dest": "/data"
10    },
11    "fault_tolerant": true,
12    "backend": {"type": "hadoop"}
13  }
14 }

```

Additional parameters include the “fault_tolerant” and “backend” parameters. The former flag indicates if the **mapreduce** deployment should operate in a fault tolerant mode. For instance, in the case of Hadoop, this may mean configuring automatic failover of name nodes using Zookeeper. The “backend” parameter accepts an object describing the system providing the **mapreduce** workflow. This may be a native deployment of Hadoop, or a special instantiation using other frameworks such as Mesos.

A function prototype is defined in Listing 9.2. Key properties are that functions describe their input parameters and generated results. For the former, the “buildInputs” and “systemBuildInputs” respectively describe the objects which should be evaluated and system packages which should be present before this function can be installed. The “eval” attribute describes how to apply this function to its input data. Parameters affecting the evaluation of the function may be passed in as the “args” attribute. The results of the function application can be accessed via the “outputs” object, which is a mapping from arbitrary keys (e.g. “data”, “processed”, “model”) to an object representing the result.

Listing 9.2: mapreduce function

```

1 {"name": "name of this function",
2  "description": "These should be
3    ↪ self-describing",
4  "source": "a URI to obtain the resource",
5  "install": {
6    "description": "instructions to install
7      ↪ the source if needed",
8    "script": "source://install.sh"
9  },
10 "eval": {
11   "description": "How to evaluate this
12     ↪ function",
13   "script": "source://run.sh",
14 },
15 "args": [],
16 "buildInputs": [
17   "list of",
18   "objects this function",
19   "depends on"
20 ],
21 "systemBuildInputs": [

```

```

19   "list of",
20   "packages required",
21   "to install"
22 ],
23 "outputs": {
24   "key1": {},
25   "key2": {}
26 }
27 }

```

Some example functions include the “NoOp” or identity function show in Listing 9.3. In the case of undefined arguments, the parameters default to an identity element. In the case of mappings this is the empty mapping while for lists this is the empty list.

Listing 9.3: mapreduce noop

```

1 { "name": "noop",
2   "description": "A NoOp (identity) function"
3 }

```

9.2. Hadoop

A **hadoop** definition defines which *deployer* to be used, the *parameters* of the deployment, and the system packages as *requires*. For each requirement, it could have attributes such as the library origin, version, etc.

Listing 9.4: hadoop

```

1 {
2   "hadoop": {
3     "deployers": {
4       "ansible":
5         ↪ "git://github.com/cloudmesh_roles/hadoop"
6     },
7     "requires": {
8       "java": {
9         "implementation": "OpenJDK",
10        "version": "1.8",
11        "zookeeper": "TBD",
12        "supervisord": "TBD"
13      }
14    },
15    "parameters": {
16      "num_resourcemanager": 1,
17      "num_namenodes": 1,
18      "use_yarn": false,
19      "use_hdfs": true,
20      "num_datanodes": 1,
21      "num_historyservers": 1,
22      "num_journalnodes": 1
23    }
24  }
25 }

```

10. SECURITY

10.1. Key

Listing 10.1: key

```
{
  "sshkey": {
    "comment": "string",
    "source": "string",
    "uri": "string",
    "value": "ssh-rsa",
    "fingerprint": "string, unique"
  }
}
```

11. MICROSERVICE

11.1. Microservice

introduce registry we can register many things to it latency provide example on how to use each of them, not just the object definition example

necessity of local direct attached storage. Mimd model to storage Kubernetes, mesos can not spin up ? Takes time to spin them up and coordinate them. While setting up environment takes more thsn using the microservice, so we must make sure that the micorservices are used sufficiently to offset spinup cost.

limitation of resource capacity such as networking.

Benchmarking to find out thing about service level agreement to access the

A system could be composed of from various microservices, and this defines each of them.

Listing 11.1: microservice

```
{
  "microservice" :{
    "name": "ms1",
    "endpoint": "http://.../ms/",
    "function": "microservice spec"
  }
}
```

11.2. Reservation

Listing 11.2: reservation

```
{
  "reservation": {
    "hosts": "string",
    "description": "string",
    "start_time": [
      "date",
      "time"
    ],
    "end_time": [
      "date",
      "time"
    ]
  }
}
```

12. NETWORK

We are looking for volunteers to contribute here.

A. SCHEMA COMMAND

B. SCHEMA

TBD

Listing B.1: schema

```

1 {
2   "reservation": {
3     "hosts": "string",
4     "description": "string",
5     "start_time": [
6       "date",
7       "time"
8     ],
9     "end_time": [
10      "date",
11      "time"
12    ]
13  }
14 }
```

C. CONTRIBUTING

We invite you to contribute to this paper and its discussion to improve it. Improvements can be done with pull requests. We suggest you do *small* individual changes to a single section and object rather than large changes as this allows us to integrate the changes individually and comment on your contribution via github.

Once contributed we will appropriately acknowledge you either as contributor or author. Please discuss with us how we best acknowledge you.

D. USING THE CLOUDMESH REST SERVICE

Components are written as YAML markup in files in the `resources/samples` directory.

For example:

Listing D.1: profile

```

1 {
2   "profile": {
3     "description": "The Profile of a user",
4     "uuid": "jshdjkd...",
5     "context": "resource",
6     "email": "laszewski@gmail.com",
7     "firstname": "Gregor",
8     "lastname": "von Laszewski",
9     "username": "gregor"
10  }
11 }
```

D.1. Element Definition

Each resource should have a `description` entry to act as documentation. The documentation should be formatted as `reStructuredText`. For example:

D.2. Yaml

```
entry = yaml.read('')
profile:
```

```

description: |
  A user profile that specifies general information
  about the user
email: laszewski@gmail.com, required
firstname: Gregor, required
lastname: von Laszewski, required
height: 180
'''
```

D.3. Cerberus

```

schema = {
  'profile': {
    'description': {'type': 'string'}
    'email': {'type': 'string', 'required': True}
    'firstname': {'type': 'string', 'required': True}
    'lastname': {'type': 'string', 'required': True}
    'height': {'type': 'float'}
  }
}
```

E. MONGOENGINE

```

class profile(Document):
    description = StringField()
    email = EmailField(required=True)
    firstname = StringField(required=True)
    lastname = StringField(required=True)
    height = FloatField(max_length=50)
```

F. CLOUDMESH NOTATION

```

profile:
  description: string
  email: email, required
  firstname: string, required
  lastname: string, required
  height: flat, max=10
```

proposed command

```

cms schema FILENAME --format=mongo -o OUTPUT
cms schema FILENAME --format=cerberus -o OUTPUT
cms schema FILENAME --format=yaml -o OUTPUT
```

reads `FILENAME` in cloudmesh notation and returns format

```

cms schema FILENAME --input=evegenie -o OUTPUT
reads eavegene example and create settings for eve
```

F.1. Defining Elements for the REST Service

To manage a large number of elements defined in our REST service easily, we manage them through definitions in yaml files. To generate the appropriate settings file for the rest service, we can use the following command:

```
cms admin elements <directory> <out.json>
```

where

- `<directory>`: directory where the YAML definitions reside
- `<out.json>`: path to the combined definition

For example, to generate a file called `all.json` that integrates all `yaml` objects defined in the directory `resources/samples` you can use the following command:

```
cms elements resources/samples all.json
```

F.2. DOIT

```
cms schema spec2tex resources/specification resources/tex
```

F.3. Generating service

With `evegenie` installed, the generated JSON file from the above step is processed to create the stub REST service definitions.

G. ABC

`README.rst`

H. CLOUDMESH REST

H.1. Prerequisite

- mongo instalation
- eve instalation
- cloudmesh cmd5
- cloudmesh rest

H.1.1. Install Mongo on OSX

```
brew update
brew install mongodb
# brew install mongodb --with-openssl
```

H.1.2. Install Mongo on OSX

ASSIGNMET TO STUDENTS, PROVIDE PULL REQUEST WITH INSTRUCTIONS

H.2. Introduction

With the cloudmesh REST framework it is easy to create REST services while defining the resources in the service easily with examples. The service is based on `eve` and the examples are defined in `yaml` to be converted to `json` and from `json` with `evegenie` into a valid `eve` settings file.

Thus you can either write your examples in `yaml` or in `json`. The resources are individually specified in a directory. The directory can contain multiple resource files. We recommend that for each resource you define your own file. Conversion of the specifications can be achieved with the `schema` command.

H.3. Yaml Specification

Let us first introduce you to a `yaml` specification. Let us assume that your `yaml` file is called `profile.yaml` and located in a directory called 'example':

```
profile:
  description: The Profile of a user
  email: laszewski@gmail.com
  firstname: Gregor
  lastname: von Laszewski
  username: gregor
```

As `eve` takes `json` objects as default we need to convert it first to `json`. This is achieved with:

```
cd example
cms schema convert profile.yaml profile.json
```

This will provide the `json` file `profile.json` as Listed in the next section

H.4. Json Specification

A valid `json` resource specification looks like this:

```
{
  "profile": {
    "description": "The Profile of a user",
    "email": "laszewski@gmail.com",
    "firstname": "Gregor",
    "lastname": "von Laszewski",
    "username": "gregor"
  }
}
```

H.5. Conversion to Eve Settings

The `json` files in the `~/sample` directory need now to be converted to a valid `eve` schema. This is achieved with two commands. First, we must concatenate all `json` specified resource examples into a single `json` file. We do this with:

```
cms schema cat . all.json
```

As we assume you are in the `samples` directory, we use a `.` for the current location of the directory that contains the samples. Next, we need to convert it to the settings file. This can be achieved with the `convert` program when you specify a `json` file:

```
cms schema convert all.json
```

The result will be a `eve` configuration file that you can use to start an `eve` service. The file is called `all.settings.py`

H.5.1. Managing Mongo

Next you need to start the `mongo` service with

```
cms admin mongo start
```

You can look at the status and information about the service with :

```
cms admin mongo info
cms admin mongo status
```

If you need to stop the service you can use:

```
cms admin mongo stop
```

H.5.2. Managing Eve

Now it is time to start the REST service. This is done in a separate window with the following commands:

```
cms admin settings all.settings.json
cms admin rest start
```

The first command copies the settings file to

```
~/cloudmesh/eve/settings.py
```


This file is than used by the start action to start the eve service. Please make sure that you execute this command in a separate window, as for debugging purposes you will be able to monitor this way interactions with this service

Testing - OLD ^^^^^:

```
make setup      # install mongo and eve
make install    # installs the code and integrates it into ends
make deploy
make test
classes lessons rest.rst
```

I. REST WITH EVE

I.1. Overview of REST

REST stands for REpresentational State Transfer. REST is an architecture style for designing networked applications. It is based on stateless, client-server, cacheable communications protocol. Although not based on http, in most cases, the HTTP protocol is used. In contrast to what some others write or say, REST is not a *standard*.

RESTful applications use HTTP requests to:

- post data: while creating and/or updating it,
- read data: while making queries, and
- delete data.

Hence REST uses HTTP for the four CRUD operations:

- Create
- Read
- Update
- Delete

As part of the HTTP protocol we have methods such as GET, PUT, POST, and DELETE. These methods can than be used to implement a REST service. As REST introduces collections and items we need to implement the CRUD functions for them. The semantics is explained in the Table illustrating how to implement them with HTTP methods.

Source: https://en.wikipedia.org/wiki/Representational_state_transfer

I.2. REST and eve

Now that we have outlined the basic functionality that we need, we lke to introduce you to Eve that makes this process rather trivial. We will provide you with an implementation example that showcases that we can create REST services without writing a single line of code. The code for this is located at <https://github.com/cloudmesh/rest>

This code will have a master branch but will also have a dev branch in which we will add gradually more objects. Objects in the dev branch will include:

- virtual directories
- virtual clusters
- job sequences
- inventories

;You may want to check our active development work in the dev branch. However for the purpose of this class the master branch will be sufficient.

I.2.1. Installation

First we havt to install mongodb. The instalation will depend on your operating system. For the use of the rest service it is not important to integrate mongodb into the system upon reboot, which is focus of many online documents. However, for us it is better if we can start and stop the services explicitly for now.

On ubuntu, you need to do the following steps:

TO BE CONTRIBUTED BY THE STUDENTS OF THE CLASS as homework

On windows 10, you need to do the following steps:

TO BE CONTRIBUTED BY THE STUDENTS OF THE CLASS as homework, if elect Windows 10. YOu could be using the online documentation provided by starting it on Windows, or rinning it in a docker c

On OSX you can use homebrew and install it with:

```
brew update
brew install mongodb
```

In future we may want to add ssl authentication in which case you may need to install it as follows:

```
brew install mongodb --with-openssl
```

I.2.2. Starting the service

We have provided a convenient Makefile that currently only works for OSX. It will be easy for you to adapt it to Linux. Certainly you can look at the targers in the makefile and replicate them one by one. Improtahnt targest are deploy and test.

When using the makefile you can start the services with:

```
make deploy
```

IT will start two terminals. IN one you will see the mongo service, in the other you will see the eve service. The eve service will take a file called sample.settings.py that is base on sample.json for the start of the eve service. The mongo servide is configured in suc a wahy that it only accepts in-cimming connections from the local host which will be sufficient fpr our case. The mongo data is written into the \$USER/.cloudmesh directory, so make sure it exists.

To test the services you can say:

```
make test
```

YOu will se a number of json text been written to the screen.

I.3. Creating your own objects

The example demonstrated how easy it is to create a mongodb and an eve rest service. Now lets use this example to creat your own. FOr this we have modified a tool called evegenie to install it onto your system.

The original documentation for evegenie is located at:

- <http://evegenie.readthedocs.io/en/latest/>

However, we have improved evegenie while providing a commandline tool based on it. The improved code is located at:

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

You clone it and install on your system as follows:

```
730 cd ~/github
731 git clone https://github.com/cloudmesh/evegenie
732 cd evegenie
733 python setup.py install
734 pip install .
```

735 This shouldl install in your system evegenie. YOu can
736 verify this by typing:

737 which evegenie

738 If you see the path evegenie is installed. With evegenie
739 installed its usage is simple:

740 \$ evegenie

742 Usage:

```

743     evegenie --help
744     evegenie FILENAME

```

It takes a json file as input and writes out a settings file for the use in eve. Lets assume the file is called sample.json, than the settings file will be called sample.settings.py. Having the evejenie programm will allow us to generate the settings files easily. You can include them into your project and leverage the Makefile targets to start the services in your project. In case you generate new objects, make sure you rerun evejenie, kill all previous windows in whcih you run eve and mongo and restart. In case of changes to objects that you have designed and run previously, you need to also delete the mongod database.

756 I.4. Towards cmd5 extensions to manage eve and mongo

Naturally it is of advantage to have in cms administration commands to manage mongo and eve from cmd instead of targets in the Makefile. Hence, we **propose** that the class develops such an extension. We will create in the repository the extension called admin and hope that students through collaborative work and pull requests complete such an admin command.

764 The proposed command is located at:

- <https://github.com/cloudmesh/rest/blob/master/cloudmesh/ext/command/admin.py>

767 It will be up to the class to implement such a command.
768 Please coordinate with each other.

The implementation based on what we provided in the Make file seems straight forward. A great extension is to load the objects definitions or eve e.g. settings.py not from the class, but form a place in .cloudmesh. I propose to place the file at:

```
774 .cloudmesh/db/settings.py
```

the location of this file is used when the Service class is initialized with None. Prior to starting the service the file needs to be copied there. This could be achieved with a set command. classes lesson python cmd5.rst

J. CMD5

CMD is a very useful package in python to create command line shells. However it does not allow the dynamic integration of newly defined commands. Furthermore, addition to cmd need to be done within the same source tree. To simplify developing commands by a number of people and to have a dynamic plugin mechanism, we developed cmd5. It is a rewrite on our earlier efforts in cloudmesh and cmd3.

J.1. Resources

The source code for cmd5 is located in github:

- <https://github.com/cloudmesh/cmd5>

Installation from source

We recommend that you use a virtualenv either with virtualenv or pyenv. This can be either achieved vor virtualenv with:

```
virtualenv ~/ENV2
```

or for pyenv, with:

```
pyenv virtualenv 2.7.13 ENV2
```

Now you need to get two source directories. We assume you place them in `~/github`:

```
mkdir ~/github
cd ~/github
```

```
git clone https://github.com/cloudmesh/common.git
git clone https://github.com/cloudmesh/cmd5.git
git clone https://github.com/cloudmesh/extbar.git
```

```
cd ~/github/common
python setup.py install
pip install .
```

```
cd ~/github/cmd5
python setup.py install
pip install .
```

```
cd ~/github/extbar
python setup.py install
pip install .
```

The `cmd5` repository contains the shell, while the `extbar` directory contains the sample to add the dynamic commands `foo` and `bar`.

J.2. Execution

To run the shell you can activate it with the `cms` command. `cms` stands for cloudmesh shell:

(ENV2) \$ cms

It will print the banner and enter the shell:

[illegible]

```

831 +-----+
832 |           Cloudmesh CMD5 Shell           |
833 +-----+
834
835 cms>
836
837     To see the list of commands you can say
838
839     cms> help
840
841     To see the manula page for a specific command, please
842 use:
843
844 help COMMANDNAME

```

841 J.3. Create your own Extension

842 One of the most important features of CMD5 is its ability to
 843 extend it with new commands. This is done via packaged
 844 name spaces. This is defined in the setup.py file of your
 845 enhancement. The best way to create an enhancement is to
 846 take a look at the code in

- 847 • <https://github.com/cloudmesh/extbar.git>

848 Simply copy the code and modify the bar and foo com-
 849 mands to fit your needs.

850 **make sure you are not copying the .git directory. Thus we**
 851 **recommend that you copy it explicitly file by file or**
 852 **directory by directory**

853 It is important that all objects are defined in the command
 854 itself and that no global variables be use in order to allow each
 855 shell command to stand alone. Naturally you should develop
 856 API libraries outside of the cloudmesh shell command and
 857 reuse them in order to keep the command code as small as
 858 possible. We place the command in:

```
859 cloudmesh/ext/command/COMMANDNAME.py
```

860 An example for the bar command is presented at:

- 861 • [https://github.com/cloudmesh/extbar/blob/master/](https://github.com/cloudmesh/extbar/blob/master/cloudmesh/ext/command/bar.py)
 862 [cloudmesh/ext/command/bar.py](https://github.com/cloudmesh/extbar/blob/master/cloudmesh/ext/command/bar.py)

863 It shows how simple the command definition is (bar.py):

```

864 from __future__ import print_function
865 from cloudmesh.shell.command import command
866 from cloudmesh.shell.command import PluginCommand
867
868 class BarCommand(PluginCommand):
869
870     @command
871     def do_bar(self, args, arguments):
872         """
873         ::
874         Usage:
875             command -f FILE
876             command FILE
877             command list
878         This command does some useful things.
879         Arguments:
880             FILE    a file name
881         Options:
882             -f      specify the file
883         """
884         print(arguments)

```

885 + An important difference to other CMD solutions is that
 886 our commands can leverage (besides the standrad definition),
 887 docopts as a way to define the manual page. This allows
 888 us to use arguments as dict and use simple if conditions to
 889 interpret the command. Using docopts has the advantage
 890 that contributors are forced to think about the command and
 891 its options and document them from the start. Previously we
 892 used not to use docopts and argparse was used. However
 893 we noticed that for some contributions the lead to commands
 894 that were either not properly documented or the developers
 895 delivered ambiguous commands that resulted in confusion
 896 and wrong ussage by the users. Hence, we do recommend
 897 that you use docopts.

898 The transformation is enabled by the @command decora-
 899 tor that takes also the manual page and creates a proper help
 900 message for the shell automatically. Thus there is no need
 901 to introduce a sepaarte help method as would normally be
 902 needed in CMD.

903 J.4. Excercise

904 **CMD5.1:** Install cmd5 on your computer.

905 **CMD5.2:** Write a new command with your firstname as the
 906 command name.

907 **CMD5.3:** Write a new command and experiment with do-
 908 copt syntax and argument interpretation of the dict with
 909 if conditions.

910 **CMD5.4:** If you have useful extensions that you like us to
 911 add by default, please work with us.