## ABSTRACT

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

**DRAFT - NIST Special Publication 1500-8**

---

# NIST Big Data Interoperability Framework: Volume 8, Reference Architecture Interface

---

NIST Big Data Public Working Group
Reference Architecture Subgroup

Version 0.1
May 9, 2017

http://dx.doi.org/10.6028/NIST.SP.1500-8

National Institute of Standards and Technology
U.S. Department of Commerce

NIST Special Publication 1500-6
Information Technology Laboratory

# NIST Big Data Interoperability Framework: Volume 8, Reference Architecture Interface

**Version 0.1**

NIST Big Data Public Working Group (NBD-PWG)
Reference Architecture Subgroup
National Institute of Standards and Technology
Gaithersburg, MD 20899

http://dx.doi.org/10.6028/NIST.SP.1500-8

May 9, 2017

U. S. Department of Commerce
*Penny Pritzker, Secretary*

National Institute of Standards and Technology
*Willie May, Under Secretary of Commerce for Standards and Technology and Director*

**National Institute of Standards and Technology (NIST) Special Publication 1500-8**
20 pages (May 9, 2017)

NIST Special Publication series 1500 is intended to capture external perspectives related to NIST standards, measurement, and testing-related efforts. These external perspectives can come from industry, academia, government, and others. These reports are intended to document external perspectives and do not represent official NIST positions.



**Comments on this publication may be submitted to Wo Chang**

National Institute of Standards and Technology
Attn: Wo Chang, Information Technology Laboratory
100 Bureau Drive (Mail Stop 8900) Gaithersburg, MD 20899-8930
Email: SP1500comments@nist.gov

# REPORTS ON COMPUTER SYSTEMS TECHNOLOGY

The Information Technology Laboratory (ITL) at NIST promotes the U.S. economy and public welfare by providing technical leadership for the Nation's measurement and standards infrastructure. ITL develops tests, test methods, reference data, proof of concept implementations, and technical analyses to advance the development and productive use of information technology (IT). ITL's responsibilities include the development of management, administrative, technical, and physical standards and guidelines for the cost-effective security and privacy of other than national security-related information in federal information systems. This document reports on ITL's research, guidance, and outreach efforts in IT and its collaborative activities with industry, government, and academic organizations.

# NIST Big Data Interoperability Framework: Volume 8, Reference Architecture Interface

**Gregor von Laszewski**[1,*], **Fugang Wang**[1], **Badi Abdhul-Wahid**[1], **and Wo L. Chang**[2]

[1] *School of Informatics and Computing, Bloomington, IN 47408, U.S.A.*
[2] *National INstitute of Standards, 100 Bureau Drive, Gaithersburg, MD 20899*
[*] *Corresponding author: laszewski@gmal.com*

*Draft v0.0.1, May 9, 2017*

---



**Keywords:** Cloudmesh, REST, NIST

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

---

## CONTENTS

# 1. INTRODUCTION

# 2. NBDRA INTERFACE REQUIREMENTS

The Volume 6 Reference Architecture document provides a list of comprehensive high-level reference architecture requirements and introduces the NIST Big Data Reference Architecture (NBDRA) (see Figure **??**). To enable interoperability between the NBDRA components, a list of well-defined NBDRA interface is needed. To introduce them, we will follow the NBDRA and focus on interfaces that allow us to bootstrap the NBDRA. Each section will introduce an Interface while documenting the requirement as well as a simple specification addressing the immediate interface needs. We expect that this document will grow with the help of contributions from the community to achieve a comprehensive set of interfaces that will be usable for the implementation of Big Data Architectures. Validation of this approach can be achieved while applying it to the use cases that have been gathered in Volume 3. These use cases have considerably contributed towards the design of the NBDRA. Hence our expectation is that (a) the interfaces can be used to help implementing a big data architecture for a specific use case, and (b) the proper implementation can validate the NBDRA. Through this approach, we can facilitate subsequent analysis and comparison of the use cases.

## 2.1. High Level Requirements of the Interface Approach

Next, we focus on the high-level requirements of the interface approach as depicted in **??**

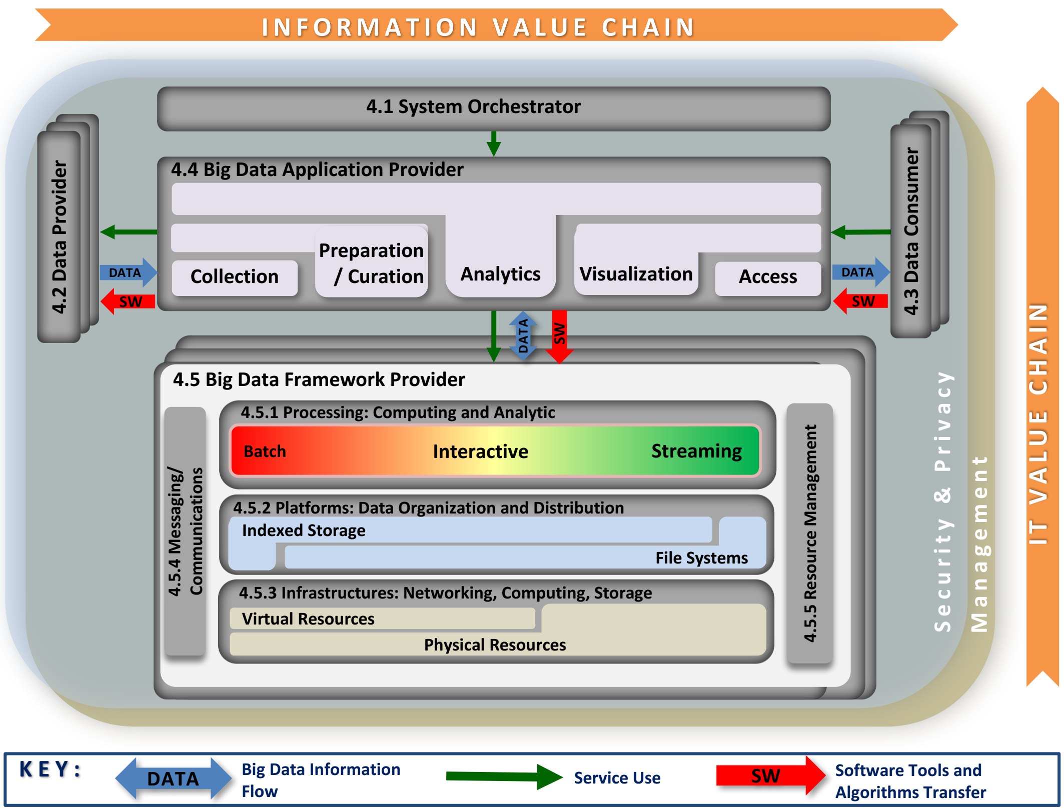


**Fig. 1.** NIST Big Data Reference Architecture (NBDRA)

### *2.1.1. Technology and Vendor Agnostic*

Due to the many different tools, services, and infrastructures available in the general area of big data an interface ought to be as vendor independent as possible, while at the same time be able to leverage best practices. As such we need to provide a methodology that allows extension of interfaces to adapt and leverage existing approaches, but also allows the interfaces to provide merit in easy specifications that assist the formulation and definition of the NBDRA.

### *2.1.2. Support of Plug-In Compute Infrastructure*

As big data is not just about hosting data, but about analyzing data the interfaces we provide must encapsulate a rich infrastructure environment that is used by data scientists. This includes the ability to integrate (or plug-in) various compute resources and services to provide the necessary compute power to analyze the data. This includes (a) access to hierarchy of compute resources, from the laptop/desktop, servers, data clusters, and clouds, (b) he ability to integrate special purpose hardware such as GPUs and FPGAs that are used in accelerated analysis of data, and (c) the integration of services including micro services that allow the analysis of the data by delegating them to hosted or dynamically deployed services on the infrastructure of choice.

#### 2.1.3. Orchestration of Infrastructure and Services

As part of the use case collection we present in Volume 3, it is obvious that we need to address the mechanism of preparing the preparation of infrastructures suitable for various use cases. As such we are not attempting to deliver a single deployed BDRA, but allow the setup of an infrastructure that satisfies the particular uses case. To achieve this task, we need to provision software tacks and services on infrastructures and orchestrate their deployment, It is not focus of this document to replace existing orchestration software and services, but provide an interface to them to leverage them as part of defining and creating the infrastructure. Various orchestration frameworks and services could therefore be leveraged and work in orchestrated fashion to achieve the goal of preparing an infrastructure suitable for one or more applications.

#### 2.1.4. Orchestration of Big Data Applications and Experiments

The creation of the infrastructure suitable for big data applications provides the basic infrastructure. However big data applications may require the creation of sophisticated applications as part of interactive experiments to analyze and probe the data. For this purpose, we need to be able to orchestrate and interact with experiments conducted on the data while assuring reproducibility and correctness of the data. For this purpose, a System Orchestrator (either the Data Scientists or a service acting in behalf of the scientist) uses the BD Application Provider as the command center to orchestrate dataflow from Data Provider, carryout the BD application lifecycle with the help of the BD Framework Provider, and enable Data Consumer to consume Big Data processing results. An interface is needed to describe the interactions and to allow leveraging of experiment management frameworks in scripted fashion. We require a customization of parameters on several levels. On the highest level, we require high level- application motivated parameters to drive the orchestration of the experiment. On lower levels these high-level parameters may drive and create service level agreement augmented specifications and parameters that could even lead to the orchestration of infrastructure and services to satisfy experiment needs.

#### 2.1.5. Reusability

The interfaces provided must encourage reusability of the infrastructure, services and experiments described by them. This includes (a) reusability of available analytics packages and services for adoption (b) deployment of customizable analytics tools and services, and (c) operational adjustments that allow the services and infrastructure to be adapted while at the same time allowing for reproducible experiment execution

#### 2.1.6. Execution Workloads

One of the important aspects of distributed big data services can be that the data served is simply to big to be moved to a different location. Instead we are in the need of an interface allowing us to describe and package analytics algorithms and potentially also tools as a payload to a data service. This can be best achieved not by sending the detailed execution, but sending an interface description that describes how such an algorithm or tool can be created on the server end and be executed under security considerations integrated with authentication and authorization in mind.

#### 2.1.7. Security and Privacy Fabric Requirements

Subsection Scope: Discussion of high-level requirements of the interface approach for the Security and Privacy Fabric.

#### 2.1.8. System Orchestration Requirement

Subsection Scope: Discussion of high-level requirements of the interface approach for the System Orchestrator.

#### 2.1.9. Application Providers Requirements

Subsection Scope: Discussion of high-level requirements of the interface approach for the Application Provider.

### 2.2. Component Specific Interface Requirements

In this section, we summarize a set of requirements for the interface of a particular component in the NBDRA. The components are listed in Figure 1 and addressed in each of the subsections as part of Section 2.2 of this document. The five main functional components of the NBDRA represent the different technical roles within a Big Data system. The functional components are listed below and discussed in subsequent subsections. System Orchestrator: Defines and integrates the required data application activities into an operational vertical system; Big Data Application Provider: Executes a data life cycle to meet security and privacy requirements as well as System Orchestrator-defined requirements; Data Provider: Introduces new data or information feeds into the Big Data system; Big Data Framework Provider: Establishes a computing framework in which to execute certain transformation applications while protecting the privacy and integrity of data; and Data Consumer: Includes end users or other systems that use the results of the Big Data Application Provider.

### 2.2.1. System Orchestrator Interface Requirement

The System Orchestrator role includes defining and integrating the required data application activities into an operational vertical system. Typically, the System Orchestrator involves a collection of more specific roles, performed by one or more actors, which manage and orchestrate the operation of the Big Data system. These actors may be human components, software components, or some combination of the two. The function of the System Orchestrator is to configure and manage the other components of the Big Data architecture to implement one or more workloads that the architecture is designed to execute. The workloads managed by the System Orchestrator may be assigning/provisioning framework components to individual physical or virtual nodes at the lower level, or providing a graphical user interface that supports the specification of workflows linking together multiple applications and components at the higher level. The System Orchestrator may also, through the Management Fabric, monitor the workloads and system to confirm that specific quality of service requirements are met for each workload, and may actually elastically assign and provision additional physical or virtual resources to meet workload requirements resulting from changes/surges in the data or number of users/transactions. The interface to the system orchestrator must be capable of specifying the task of orchestration the deployment, configuration, and the execution of applications within the NBDRA. A simple vendor neutral specification to coordinate the various parts either as simple parallel language tasks or as a workflow specification is needed to facilitate the overall coordination. Integration of existing tools and services into the orchestrator as extensible interface is desirable.

### 2.2.2. Data Provider Interface Requirement

The Data Provider role introduces new data or information feeds into the Big Data system for discovery, access, and transformation by the Big Data system. New data feeds are distinct from the data already in use by the system and residing in the various system repositories. Similar technologies can be used to access both new data feeds and existing data. The Data Provider actors can be anything from a sensor, to a human inputting data manually, to another Big Data system. Interfaces for data providers must be able to specify a data provider so it can be located by a data consumer. It also must include enough details to identify the services offered so they can be pragmatically reused by consumers. Interfaces to describe pipes and filters must be addressed.

### 2.2.3. Data Consumer Interface Requirement

Similar to the Data Provider, the role of Data Consumer within the NBDRA can be an actual end user or another system. In many ways, this role is the mirror image of the Data Provider, with the entire Big Data framework appearing like a Data Provider to the Data Consumer. The activities associated with the Data Consumer role include (a) Search and Retrieve (b) Download (c) Analyze Locally (d) Reporting (d) Visualization (e) Data to Use for Their Own Processes. The interface for the data consumer must be able to describe the consuming services and how they retrieve information or leverage data consumers.

### 2.2.4. Big Data Application Interface Provide

The Big Data Application Provider role executes a specific set of operations along the data life cycle to meet the requirements established by the System Orchestrator, as well as meeting security and privacy requirements. The Big Data Application Provider is the architecture component that encapsulates the business logic and functionality to be executed by the architecture. The interfaces to describe big data applications include interfaces for the various subcomponents including collections, preparation/curation, analytics, visualization, and access. Some if the interfaces used in these components can be reused from other interfaces introduced in other sections of this document. Where appropriate we will identify application specific interfaces and provide examples of them while focusing on a use case as identified in Volume 3 of this series.

#### 2.2.4.1

In general, the collection activity of the Big Data Application Provider handles the interface with the Data Provider. This may be a general service, such as a file server or web server configured by the System Orchestrator to accept or perform specific collections of data, or it may be an application-specific service designed to pull data or receive pushes of data from the Data Provider. Since this activity is receiving data at a minimum, it must store/buffer the received data until it is persisted through the Big Data Framework Provider. This persistence need not be to physical media but may simply be to an in-memory queue or other service provided by the processing frameworks of the Big Data Framework Provider. The collection activity is likely where the extraction portion of the Extract, Transform, Load (ETL)/Extract, Load, Transform (ELT) cycle is performed. At the initial collection stage, sets of data (e.g., data records) of similar structure are collected (and combined), resulting in uniform security, policy, and other considerations. Initial metadata is created (e.g., subjects with keys are identified) to facilitate subsequent aggregation or look-up methods.

#### 2.2.4.2

The preparation activity is where the transformation portion of the ETL/ELT cycle is likely performed, although analytics activity will also likely perform advanced parts of the transformation. Tasks performed by this activity could include data validation (e.g., checksums/hashes, format checks), cleansing (e.g., eliminating bad records/-fields), outlier removal, standardization, reformatting, or encapsulating. This activity is also where source data will frequently be persisted to archive storage in the Big Data Framework Provider and provenance data will be verified or attached/associated. Verification or attachment may include optimization of data through manipulations (e.g., deduplication) and indexing to optimize the analytics process. This activity may also aggregate data from different Data Providers, leveraging metadata keys to create an expanded and enhanced data set.

#### 2.2.4.3

The analytics activity of the Big Data Application Provider includes the encoding of the low-level business logic of the Big Data system (with higher-level business process logic being encoded by the System Orchestrator). The activity implements the techniques to extract knowledge from the data based on the requirements of the vertical application. The requirements specify the data processing algorithms for processing the data to produce new insights that will address the technical goal. The analytics activity will leverage the processing frameworks to implement the associated logic. This typically involves the activity providing software that implements the analytic logic to the batch and/or streaming elements of the processing framework for execution. The messaging/communication framework of the Big Data Framework Provider may be used to pass data or control functions to the application logic running in the processing frameworks. The analytic logic may be broken up into multiple modules to be executed by the processing frameworks which communicate, through the messaging/communication framework, with each other and other functions instantiated by the Big Data Application Provider.

#### 2.2.4.4

The visualization activity of the Big Data Application Provider prepares elements of the processed data and the output of the analytic activity for presentation to the Data Consumer. The objective of this activity is to format and present data in such a way as to optimally communicate meaning and knowledge. The visualization preparation may involve producing a text-based report or rendering the analytic results as some form of graphic. The resulting output may be a static visualization and may simply be stored through the Big Data Framework Provider for later access. However, the visualization activity frequently interacts with the access activity, the analytics activity, and the Big Data Framework Provider (processing and platform) to provide interactive visualization of the data to the Data Consumer based on parameters provided to the access activity by the Data Consumer. The visualization activity may be completely application-implemented, leverage one or more application libraries, or may use specialized visualization processing frameworks within the Big Data Framework Provider.

#### 2.2.4.5

The access activity within the Big Data Application Provider is focused on the communication/interaction with the Data Consumer. Similar to the collection activity, the access activity may be a generic service such as a web server or application server that is configured by the System Orchestrator to handle specific requests from the Data Consumer. This activity would interface with the visualization and analytic activities to respond to requests from the Data Consumer (who may be a person) and uses the processing and platform frameworks to retrieve data to respond to Data Consumer requests. In addition, the access activity confirms that descriptive and administrative metadata and metadata schemes are captured and maintained for access by the Data Consumer and as data is transferred to the Data Consumer. The interface with the Data Consumer may be synchronous or asynchronous in nature and may use a pull or push paradigm for data transfer.

#### 2.2.4.6

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. We must be able to provide the following functionality (1) interfaces to files (2) interfaces ti virtual data directories (3) interfaces ti data streams (4) and interfaces to data filters.

#### 2.2.4.7

This Big Data Framework Provider element provides all of the resources necessary to host/run the activities of the other components of the Big Data system. Typically, these resources consist of some combination of physical resources, which may host/support similar virtual resources. As part of the NBDRA we need interfaces that can be used to deal with the underlying infrastructure to address networking, computing, and storage

#### 2.2.4.8

As part of the NBDRA platforms we need interfaces that can address platform needs and services for data organization, data distribution, indexed storage, and file systems.

#### 2.2.4.9

The processing frameworks for Big Data provide the necessary infrastructure software to support implementation of applications that can deal with the volume, velocity, variety, and variability of data. Processing frameworks define how the computation and processing of the data is organized. Big Data applications rely on various platforms and technologies to meet the challenges of scalable data analytics and operation. We need to be able to interface easily with computing services that offer specific analytics services, batch processing capabilities, interactive analysis, and data streaming.

#### 2.2.4.10

A number of crosscutting interface requirements within the NBDRA provider frameworks include messaging, communication, and resource management. Often these eservices may actually be hidden from explicit interface use as they are part of larger systems that expose higher level functionality through their interfaces. However, it may be needed to expose such interfaces also on a lower level in case finer grained control is needed. We will identify the need for such crosscutting interface requirements form Volume 3 of this series.

**2.2.4.10.1 Messaging/Communications Frameworks** Messaging and communications frameworks have their roots in the High Performance Computing (HPC) environments long popular in the scientific and research communities. Messaging/Communications Frameworks were developed to provide APIs for the reliable queuing, transmission, and receipt of data

**2.2.4.10.2 Resource Management Framework** As Big Data systems have evolved and become more complex, and as businesses work to leverage limited computation and storage resources to address a broader range of applications and business challenges, the requirement to effectively manage those resources has grown significantly. While tools for resource management and "elastic computing" have expanded and matured in response to the needs of cloud providers and virtualization technologies, Big Data introduces unique requirements for these tools. However, Big Data frameworks tend to fall more into a distributed computing paradigm, which presents additional challenges.

### *2.2.5. BD Application Provider to Framework Provider Interface*

The Big Data Framework Provider typically consists of one or more hierarchically organized instances of the components in the NBDRA IT value chain (Figure 2). There is no requirement that all instances at a given level in the hierarchy be of the same technology. In fact, most Big Data implementations are hybrids that combine multiple technology approaches in order to provide flexibility or meet the complete range of requirements, which are driven from the Big Data Application Provider.

# 3. SPECIFICATION PARADIGM

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

## 3.1. Lessons Learned

TBD

## 3.2. Hybrid Cloud

TBD

## 3.3. Design by Example

To accelerate discussion among the team we use an approach to define objects and its interfaces by example. These examples are than taken in a later version of the document and a schema is generated from it. The schema will be added in its complete form to the appendix **??**. While focusing first on examples it allows us to speed up our design and simplifies discussions of the objects and interfaces eliminating getting lost in complex syntactical specifications. The process and specifications used in this document will also allow us to automatically create a implementation of the objects that can be integrated into a reference architecture as provided by for example the cloudmesh client and rest project [**?** ].

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

Listing 3.1: User profile

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

Such an object can be transformed to a schema specification while introspecting the types of the original example. The resulting schema object follows the Cerberus [**?** ] specification and looks for our object as follows:

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

As mentioned before, the Appendix**??** will list the schema that is automatically created from the definitions.

## 3.4. Tools to Create the Specifications

The tools to create the schema and object are all available opensource and are hosted on github. It includes the following repositories:

**cloudmesh.common**
https://github.com/cloudmesh/cloudmesh.common

**cloudmesh.cmd5**
https://github.com/cloudmesh/cloudmesh.cmd5

**cloudmesh.rest**
https://github.com/cloudmesh/cloudmesh.rest

**cloudmesh/evegenie**
https://github.com/cloudmesh/evegenie

### 3.5. Installation of the Tools

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

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

```
pip install cloudmesh.rest
pip install cloudmesh.evengine
```

### 3.6. 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.

### 3.7. 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 LaTeX.

The files are located at:

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

### 3.8. 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.

## 4. SPECIFICATION

### 4.1. 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.

#### 4.1.1. 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 4.1: User profile


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


#### 4.1.2. User

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

Listing 4.2: user


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


#### 4.1.3. Organization

An important concept in many applications is the management of a roup of users in a virtual organization. This can be achieved through two concepts. First, it can be achieved while useing 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 4.3: user


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


#### 4.1.4. Group/Role

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

Listing 4.4: 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 4.5: role

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

## 4.2. 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.

### *4.2.1. 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 4.6: 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, ..."
  }
}
```

#### 4.2.2. 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 4.7: default

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

#### 4.2.3. 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 andpoint, 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 4.8: file

```
{
    "file": {
        "name": "report.dat",
        "endpoint": "file://gregor@machine.edu:/data/report.dat",
        "checksum": {"md5":"8c324f12047dc2254b74031b8f029ad0"},
        "accessed": "1.1.2017:05:00:00:EST",
        "created": "1.1.2017:05:00:00:EST",
        "modified": "1.1.2017:05:00:00:EST",
        "size": ["GB", "Byte"]
  }
}
```

#### 4.2.4. File Alias

A file could have one alias or even multiple ones.

Listing 4.9: file alias

```
{
    "file_alias": {
        "alias": "report-alias.dat",
        "name": "report.dat"
  }
```

```
}
```

#### 4.2.5. 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 4.10: replica

```
{
  "replica": {
    "name": "replica_report.dat",
    "replica": "report.dat",
    "endpoint": "file://gregor@machine.edu:/data/replica_report.dat",
    "checksum": {
        "md5": "8c324f12047dc2254b74031b8f029ad0"
    },
    "accessed": "1.1.2017:05:00:00:EST",
    "size": [
      "GB",
      "Byte"
    ]
  }
}
```

#### 4.2.6. Virtual Directory

A collection of files or replicas. A virtual directory can contain an number of entities cincluding 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 4.11: virtual directory

```
{
  "virtual_directory": {
    "name": "data",
    "endpoint": "http://.../data/",
    "protocol": "http",
    "collection": [
      "report.dat",
      "file2"
    ]
  }
}
```

#### 4.2.7. Database

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

Listing 4.12: database

```
{
  "database": {
    "name": "data",
    "endpoint": "http://.../data/",
    "protocol": "mongo"
  }
}
```

#### 4.2.8. Stream

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

Listing 4.13: stream

```
{
  "stream": {
    "name": "name of the variable",
    "format": "the format of the data exchanged in the stream",
    "attributes": {
      "rate": 10,
      "limit": 1000
    }
  }
}
```

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 4.14: filter

```
{
  "filter": {
    "name": "name of the filter",
    "function": "the function  of the data exchanged in the stream"
  }
}
```

Filter needs to be refined

### 4.3. IaaS

In this subsection 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.

#### 4.3.1. Openstack

##### 4.3.1.1

Listing 4.15: 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"
  }
}
```

#### 4.3.1.2

Listing 4.16: 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"
    }
  }
}
```

#### 4.3.1.3

Listing 4.17: 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"
    }
  }
```

### *4.3.2. Azure*

#### 4.3.2.1

The size description of an azure vm

Listing 4.18: 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"
  }
}
```

#### 4.3.2.2

Listing 4.19: azure-image

```
{
  "azure_image": {
    "_uuid": "None",
    "driver": "libcloud",
    "extra": {
      "affinity_group": "",
```

```
      "category": "Public",
      "description": "Linux VM image with coreclr-x64-beta5-11624 installed to /opt/dnx. This image is based on Ubuntu 14.04 LTS, with prerequisites of CoreCLR installed. It also contains PartsUnlimited demo app which runs on the installed coreclr. The demo app is installed to /opt/demo. To run the demo, please type the command /opt/demo/Kestrel in a terminal window. The website is listening on port 5004. Please enable or map a endpoint of HTTP port 5004 for your azure VM.",
      "location": "East Asia;Southeast Asia;Australia East;Australia Southeast;Brazil South;North Europe;West Europe;Japan East;Japan West;Central US;East US;East US 2;North Central US;South Central US;West US",
      "media_link": "",
      "os": "Linux",
      "vm_image": "False"
    },
    "id": "03f55de797f546a1b29d1....",
    "name": "CoreCLR x64 Beta5 (11624) with PartsUnlimited Demo App on Ubuntu Server 14.04 LTS"
  }
}
```

#### 4.3.2.3

An Azure virtual machine

Listing 4.20: azure-vm

```
{
  "azure-vm": {
    "username": "string",
    "status": "string",
    "deployment_slot": "string",
    "cloud_service": "string",
    "image": "string",
    "floating_ip": "string",
    "image_name": "string",
    "key": "string",
    "flavor": "string",
    "resource_location": "string",
    "disk_name": "string",
    "private_ips": "string",
    "group": "string",
    "uuid": "string",
    "dns_name": "string",
    "instance_size": "string",
    "instance_name": "string",
    "public_ips": "string",
    "media_link": "string"
  }
}
```

## 4.4. HPC

### 4.4.1. Batch Job

Listing 4.21: batchjob

```
{
  "batchjob": {
    "output_file": "string",
    "group": "string",
    "job_id": "string",
    "script": "string, the batch job script",
    "cmd": "string, executes the cmd, if None path is used",
    "queue": "string",
    "cluster": "string",
    "time": "string",
    "path": "string, path of the batchjob, if non cmd is used",
    "nodes": "string",
    "dir": "string"
  }
}
```

## 4.5. Virtual Cluster

### 4.5.1. 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 4.22: cluster

```
{
  "cluster": {
    "label": "c0",
    "endpoint": {
      "passwd": "secret",
      "url": "https"
    },
    "name": "myCLuster",
    "provider": [
      "openstack",
      "aws",
      "azure",
      "eucalyptus"
    ]
  }
}
```

### 4.5.2. New Cluster

Listing 4.23: cluster

```
{
  "virtual_cluster": {
    "name": "myvc",
      "frontend": 0,
      "nodes": [
          { "count": 3,
            "node": "objectid:virtual_machine"
          }
```

```
      ]
    },
    "virtual_machine" :{
      "name": "vm1",
      "ncpu": 2,
      "RAM": "4G",
      "disk": "40G",
      "nics": ["objectid:nic"
      ],
      "OS": "Ubuntu-16.04",
      "loginuser": "ubuntu",
      "status": "active",
      "metadata":{
      },
      "authorized_keys": [
        "objectid:sshkey"
      ]
    },
    "sshkey": {
      "comment": "string",
      "source": "string",
      "uri": "string",
      "value": "ssh-rsa AAA......",
      "fingerprint": "string, unique"
    },
    "nic": {
      "name": "eth0",
      "type": "ethernet",
      "mac": "00:00:00:11:22:33",
      "ip": "123.123.1.2",
      "mask": "255.255.255.0",
      "broadcast": "123.123.1.255",
      "gateway": "123.123.1.1",
      "mtu": 1500,
      "bandwidth": "10Gbps"
    }
  }
```

#### 4.5.3. Compute Resource

An important concept for big data analysis it 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 4.24: compute resource

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

#### 4.5.4. Computer

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

Listing 4.25: computer

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

#### 4.5.5. 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 4.26: 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",
        "password": {
          "decrypt": "gpg",
          "from": "yaml",
          "file": "secrets.yml.gpg",
          "key": "users.hello.ssh[1]"
        },
        "bits": 4098,
        "method": "rsa",
        "ssh_keygen": true
      },
      {
        "decrypt": "gpg",
        "from": "secrets/ssh/hello/copied.gpg",
        "ssh_keygen": false,
        "to": ".ssh/copied"
      }
    ]
  }
}
```

#### 4.5.6. 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 4.27: virtual cluster

```
{
  "virtual_cluster": {
    "name": "myvc",
    "frontend": "objectid:virtual_machine",
    "nodes": [
      "objectid:virtual_machine"
    ]
  }
}
```

#### 4.5.7. Virtual Compute node

Listing 4.28: virtual compute node

```
{
  "virtual_compute_node": {
    "name": "data",
    "endpoint": "http://.../cluster/",
    "metadata": {
      "experiment": "exp-001"
    },
    "image": "Ubuntu-16.04",
    "ip": [
      "TBD"
    ],
    "flavor": "TBD",
    "status": "TBD"
  }
}
```

#### 4.5.8. 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 4.29: virtual machine

```
{
  "virtual_machine" :{
    "name": "vm1",
    "ncpu": 2,
    "RAM": "4G",
    "disk": "40G",
    "nics": ["objectid:nic"
    ],
    "OS": "Ubuntu-16.04",
    "loginuser": "ubuntu",
    "status": "active",
    "metadata":{
    },
    "authorized_keys": [
      "objectid:sshkey"
    ]
  }
}
```

#### 4.5.9. Mesos

Refine

Listing 4.30: mesos


```
{
  "mesos-docker": {
    "instances": 1,
    "container": {
      "docker": {
        "credential": {
          "secret": "my-secret",
          "principal": "my-principal"
        },
        "image": "mesosphere/inky"
      },
      "type": "MESOS"
    },
    "mem": 16.0,
    "args": [
      "argument"
    ],
    "cpus": 0.2,
    "id": "mesos-docker"
  }
}
```


### 4.6. Containers

#### 4.6.1. Container

This defines **container** object.

Listing 4.31: container


```
{
    "container": {
        "name": "container1",
        "endpoint": "http://.../container/",
        "ip": "127.0.0.1",
        "label": "server-001",
        "memoryGB": 16
    }
}
```


#### 4.6.2. Kubernetes

REFINE

Listing 4.32: kubernetes


```
{
  "kubernetes": {
    "kind": "List",
    "items": [
      {
        "kind": "None",
        "metadata": {
          "name": "127.0.0.1"
```

```
        },
        "status": {
          "capacity": {
            "cpu": "4"
          },
          "addresses": [
            {
              "type": "LegacyHostIP",
              "address": "127.0.0.1"
            }
          ]
        }
      },
      {
        "kind": "None",
        "metadata": {
          "name": "127.0.0.2"
        },
        "status": {
          "capacity": {
            "cpu": "8"
          },
          "addresses": [
            {
              "type": "LegacyHostIP",
              "address": "127.0.0.2"
            },
            {
              "type": "another",
              "address": "127.0.0.3"
            }
          ]
        }
      }
    ],
    "users": [
      {
        "name": "myself",
        "user": "gregor"
      },
      {
        "name": "e2e",
        "user": {
          "username": "admin",
          "password": "secret"
        }
      }
    ]
  }
}
```

## 4.7. Deployment

### 4.7.1. 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 4.33: deployment

```
{
    "deployment": {
        "cluster": [{ "name": "myCluster"},
                    { "id" : "cm-0001"}
                   ],
        "stack": {
            "layers": [
                "zookeeper",
                "hadoop",
                "spark",
                "postgresql"
            ],
            "parameters": {
                "hadoop": { "zookeeper.quorum": [ "IP", "IP", "IP"]
                          }
            }
        }
    }
}
```

## 4.8. Mapreduce

### 4.8.1. 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 4.34: mapreduce

```
{
  "mapreduce": {
      "function": {
          "source": "file://.",
          "args": {}
      },
      "data": {
          "source": "ftp:///...",
          "dest": "/data"
      },
      "fault_tolerant": true,
      "backend": {"type": "hadoop"}
  }
}
```

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 **??**. 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 4.35: mapreduce function

```
{"name": "name of this function",
 "description": "These should be self-describing",
 "source": "a URI to obtain the resource",
 "install": {
     "description": "instructions to install the source if needed",
     "script": "source://install.sh"
 },
 "eval": {
     "description": "How to evaluate this function",
     "script": "source://run.sh",
 },
 "args": [],
 "buildInputs": [
     "list of",
     "objects this function",
     "depends on"
 ],
 "systemBuildInputs": [
     "list of",
     "packages required",
     "to install"
 ],
 "outputs": {
     "key1": {},
     "key2": {}
 }
}
```

Some example functions include the "NoOp" function shown in Listing **??**. 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 4.36: mapreduce noop

```
{ "name": "noop",
  "description": "A function with no effect"
}
```

### 4.8.2. 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 4.37: hadoop

```
{
  "hadoop": {
    "deployers": {
      "ansible": "git://github.com/cloudmesh_roles/hadoop"
    },
    "requires": {
      "java": {
        "implementation": "OpenJDK",
        "version": "1.8",
```

```
        "zookeeper": "TBD",
        "supervisord": "TBD"
      }
    },
    "parameters": {
      "num_resourcemanagers": 1,
      "num_namenodes": 1,
      "use_yarn": false,
      "use_hdfs": true,
      "num_datanodes": 1,
      "num_historyservers": 1,
      "num_journalnodes": 1
    }
  }
}
```


## 4.9. Security

### 4.9.1. Key

Listing 4.38: key


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


## 4.10. Microservice

### 4.10.1. 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 Kubernetis, 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 4.39: microservice


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

#### 4.10.2. Reservation

Listing 4.40: reservation


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


### 4.11. Network

We are looking for volunteers to contribute here.

# A. APPENDIX

## A.1. Schema Command

Listing A.1: man page

```
Could not execute the command.
Usage:
    schema cat DIRECTORY FILENAME
    schema convert INFILE [OUTFILE]

::

  Usage:
    schema cat DIRECTORY FILENAME
    schema convert INFILE [OUTFILE]

  Arguments:
      FILENAME   a filename
      DIRECTORY  the derectory where the schma
                 objects are defined

  Options:
      -h     help

  Description:
     schema eve [json|yml] DIRECTORY FILENAME
        concatenates all files with ending yml
        or json in the directory and combines
        them. Using evegenie on the combined
        file a eve settings file is generated
        and written into FILENAME

     schema cat [json|yml] DIRECTORY FILENAME
        Concatinates all files with the given
        ending (either json, or yml) into the
        file called FILENAME
```

## A.2. Schema

TBD

Listing A.2: schema

```
profile = {
    'schema': {
        'username': {
            'type': 'string'
        },
        'context:': {
            'type': 'string'
        },
        'description': {
            'type': 'string'
        },
        'firstname': {
            'type': 'string'
        },
```

```
        'lastname': {
            'type': 'string'
        },
        'email': {
            'type': 'string'
        },
        'uuid': {
            'type': 'string'
        }
    }
}

virtual_machine = {
    'schema': {
        'status': {
            'type': 'string'
        },
        'authorized_keys': {
            'type': 'list',
            'schema': {
                'type': 'objectid',
                'data_relation': {
                    'resource': 'sshkey',
                    'field': '_id',
                    'embeddable': True
                }
            }
        },
        'name': {
            'type': 'string'
        },
        'nics': {
            'type': 'list',
            'schema': {
                'type': 'objectid',
                'data_relation': {
                    'resource': 'nic',
                    'field': '_id',
                    'embeddable': True
                }
            }
        },
        'RAM': {
            'type': 'string'
        },
        'ncpu': {
            'type': 'integer'
        },
        'loginuser': {
            'type': 'string'
        },
        'disk': {
            'type': 'string'
        },
        'OS': {
```

705

```
            'type': 'string'
        },
        'metadata': {
            'type': 'dict',
            'schema': {}
        }
    }
}

kubernetes = {
    'schema': {
        'items': {
            'type': 'list',
            'schema': {
                'type': 'dict',
                'schema': {
                    'status': {
                        'type': 'dict',
                        'schema': {
                            'capacity': {
                                'type': 'dict',
                                'schema': {
                                    'cpu': {
                                        'type': 'string'
                                    }
                                }
                            },
                            'addresses': {
                                'type': 'list',
                                'schema': {
                                    'type': 'dict',
                                    'schema': {
                                        'type': {
                                            'type': 'string'
                                        },
                                        'address': {
                                            'type': 'string'
                                        }
                                    }
                                }
                            }
                        }
                    },
                    'kind': {
                        'type': 'string'
                    },
                    'metadata': {
                        'type': 'dict',
                        'schema': {
                            'name': {
                                'type': 'string'
                            }
                        }
                    }
                }
```

706

```
            }
        },
        'kind': {
            'type': 'string'
        },
        'users': {
            'type': 'list',
            'schema': {
                'type': 'dict',
                'schema': {
                    'name': {
                        'type': 'string'
                    },
                    'user': {
                        'type': 'dict',
                        'schema': {
                            'username': {
                                'type': 'string'
                            },
                            'password': {
                                'type': 'string'
                            }
                        }
                    }
                }
            }
        }
    }
}

nic = {
    'schema': {
        'name': {
            'type': 'string'
        },
        'ip': {
            'type': 'string'
        },
        'mask': {
            'type': 'string'
        },
        'bandwidth': {
            'type': 'string'
        },
        'mtu': {
            'type': 'integer'
        },
        'broadcast': {
            'type': 'string'
        },
        'mac': {
            'type': 'string'
        },
        'type': {
            'type': 'string'
```

707

```
        },
        'gateway': {
            'type': 'string'
        }
    }
}

virtual_compute_node = {
    'schema': {
        'status': {
            'type': 'string'
        },
        'endpoint': {
            'type': 'string'
        },
        'name': {
            'type': 'string'
        },
        'ip': {
            'type': 'list',
            'schema': {
                'type': 'string'
            }
        },
        'image': {
            'type': 'string'
        },
        'flavor': {
            'type': 'string'
        },
        'metadata': {
            'type': 'dict',
            'schema': {
                'experiment': {
                    'type': 'string'
                }
            }
        }
    }
}

openstack_flavor = {
    'schema': {
        'os_flv_disabled': {
            'type': 'string'
        },
        'uuid': {
            'type': 'string'
        },
        'os_flv_ext_data': {
            'type': 'string'
        },
        'ram': {
            'type': 'string'
        },
```

```
        'os_flavor_acces': {
            'type': 'string'
        },
        'vcpus': {
            'type': 'string'
        },
        'swap': {
            'type': 'string'
        },
        'rxtx_factor': {
            'type': 'string'
        },
        'disk': {
            'type': 'string'
        }
    }
}

azure_vm = {
    'schema': {
        'username': {
            'type': 'string'
        },
        'status': {
            'type': 'string'
        },
        'deployment_slot': {
            'type': 'string'
        },
        'group': {
            'type': 'string'
        },
        'private_ips': {
            'type': 'string'
        },
        'cloud_service': {
            'type': 'string'
        },
        'dns_name': {
            'type': 'string'
        },
        'image': {
            'type': 'string'
        },
        'floating_ip': {
            'type': 'string'
        },
        'image_name': {
            'type': 'string'
        },
        'instance_name': {
            'type': 'string'
        },
        'public_ips': {
            'type': 'string'
```

```
        },
        'media_link': {
            'type': 'string'
        },
        'key': {
            'type': 'string'
        },
        'flavor': {
            'type': 'string'
        },
        'resource_location': {
            'type': 'string'
        },
        'instance_size': {
            'type': 'string'
        },
        'disk_name': {
            'type': 'string'
        },
        'uuid': {
            'type': 'string'
        }
    }
}

azure_size = {
    'schema': {
        'ram': {
            'type': 'integer'
        },
        'name': {
            'type': 'string'
        },
        'extra': {
            'type': 'dict',
            'schema': {
                'cores': {
                    'type': 'integer'
                },
                'max_data_disks': {
                    'type': 'integer'
                }
            }
        },
        'price': {
            'type': 'float'
        },
        '_uuid': {
            'type': 'string'
        },
        'driver': {
            'type': 'string'
        },
        'bandwidth': {
            'type': 'string'
```

```
        },
        'disk': {
            'type': 'integer'
        },
        'id': {
            'type': 'string'
        }
    }
}

openstack_vm = {
    'schema': {
        'vm_state': {
            'type': 'string'
        },
        'availability_zone': {
            'type': 'string'
        },
        'terminated_at': {
            'type': 'string'
        },
        'image': {
            'type': 'string'
        },
        'diskConfig': {
            'type': 'string'
        },
        'flavor': {
            'type': 'string'
        },
        'security_groups': {
            'type': 'string'
        },
        'volumes_attached': {
            'type': 'string'
        },
        'user_id': {
            'type': 'string'
        },
        'uuid': {
            'type': 'string'
        },
        'accessIPv4': {
            'type': 'string'
        },
        'accessIPv6': {
            'type': 'string'
        },
        'power_state': {
            'type': 'string'
        },
        'progress': {
            'type': 'string'
        },
        'image__id': {
```

```
            'type': 'string'
        },
        'launched_at': {
            'type': 'string'
        },
        'config_drive': {
            'type': 'string'
        },
        'username': {
            'type': 'string'
        },
        'updated': {
            'type': 'string'
        },
        'hostId': {
            'type': 'string'
        },
        'floating_ip': {
            'type': 'string'
        },
        'static_ip': {
            'type': 'string'
        },
        'key': {
            'type': 'string'
        },
        'flavor__id': {
            'type': 'string'
        },
        'group': {
            'type': 'string'
        },
        'task_state': {
            'type': 'string'
        },
        'created': {
            'type': 'string'
        },
        'tenant_id': {
            'type': 'string'
        },
        'status': {
            'type': 'string'
        }
    }
}

cluster = {
    'schema': {
        'provider': {
            'type': 'list',
            'schema': {
                'type': 'string'
            }
        },
```

```
        'endpoint': {
            'type': 'dict',
            'schema': {
                'passwd': {
                    'type': 'string'
                },
                'url': {
                    'type': 'string'
                }
            }
        },
        'name': {
            'type': 'string'
        },
        'label': {
            'type': 'string'
        }
    }
}

computer = {
    'schema': {
        'ip': {
            'type': 'string'
        },
        'name': {
            'type': 'string'
        },
        'memoryGB': {
            'type': 'integer'
        },
        'label': {
            'type': 'string'
        }
    }
}

libcloud_image = {
    'schema': {
        'username': {
            'type': 'string'
        },
        'status': {
            'type': 'string'
        },
        'updated': {
            'type': 'string'
        },
        'description': {
            'type': 'string'
        },
        'owner_alias': {
            'type': 'string'
        },
        'kernel_id': {
```

```
                'type': 'string'
            },
            'hypervisor': {
                'type': 'string'
            },
            'ramdisk_id': {
                'type': 'string'
            },
            'state': {
                'type': 'string'
            },
            'created': {
                'type': 'string'
            },
            'image_id': {
                'type': 'string'
            },
            'image_location': {
                'type': 'string'
            },
            'platform': {
                'type': 'string'
            },
            'image_type': {
                'type': 'string'
            },
            'is_public': {
                'type': 'string'
            },
            'owner_id': {
                'type': 'string'
            },
            'architecture': {
                'type': 'string'
            },
            'virtualization_type': {
                'type': 'string'
            },
            'uuid': {
                'type': 'string'
            }
        }
}

user = {
    'schema': {
        'username': {
            'type': 'string'
        },
        'context:': {
            'type': 'string'
        },
        'uuid': {
            'type': 'string'
        },
```

```
        'firstname': {
            'type': 'string'
        },
        'lastname': {
            'type': 'string'
        },
        'roles': {
            'type': 'list',
            'schema': {
                'type': 'string'
            }
        },
        'email': {
            'type': 'string'
        }
    }
}

file = {
    'schema': {
        'endpoint': {
            'type': 'string'
        },
        'name': {
            'type': 'string'
        },
        'created': {
            'type': 'string'
        },
        'checksum': {
            'type': 'dict',
            'schema': {
                'md5': {
                    'type': 'string'
                }
            }
        },
        'modified': {
            'type': 'string'
        },
        'accessed': {
            'type': 'string'
        },
        'size': {
            'type': 'list',
            'schema': {
                'type': 'string'
            }
        }
    }
}

deployment = {
    'schema': {
        'cluster': {
```

```
            'type': 'list',
            'schema': {
                'type': 'dict',
                'schema': {
                    'id': {
                        'type': 'string'
                    }
                }
            }
        },
        'stack': {
            'type': 'dict',
            'schema': {
                'layers': {
                    'type': 'list',
                    'schema': {
                        'type': 'string'
                    }
                },
                'parameters': {
                    'type': 'dict',
                    'schema': {
                        'hadoop': {
                            'type': 'dict',
                            'schema': {
                                'zookeeper.quorum': {
                                    'type': 'list',
                                    'schema': {
                                        'type': 'string'
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}

mapreduce = {
    'schema': {
        'layers': {
            'type': 'list',
            'schema': {
                'type': 'string'
            }
        },
        'hdfs_datanode': {
            'type': 'string'
        },
        'java': {
            'type': 'dict',
            'schema': {
                'platform': {
```

```
                    ’type’: ’string’
                },
                ’version’: {
                    ’type’: ’string’
                }
            }
        },
        ’supervisord’: {
            ’type’: ’string’
        },
        ’yarn_historyserver’: {
            ’type’: ’string’
        },
        ’zookeeper’: {
            ’type’: ’string’
        },
        ’hdfs_namenode’: {
            ’type’: ’string’
        },
        ’hdfs_journalnode’: {
            ’type’: ’string’
        },
        ’yarn_resourcemanager’: {
            ’type’: ’string’
        }
    }
}

group = {
    ’schema’: {
        ’users’: {
            ’type’: ’list’,
            ’schema’: {
                ’type’: ’objectid’,
                ’data_relation’: {
                    ’resource’: ’user’,
                    ’field’: ’_id’,
                    ’embeddable’: True
                }
            }
        },
        ’name’: {
            ’type’: ’string’
        },
        ’description’: {
            ’type’: ’string’
        }
    }
}

role = {
    ’schema’: {
        ’users’: {
            ’type’: ’list’,
            ’schema’: {
```

```
                'type': 'objectid',
                'data_relation': {
                    'resource': 'user',
                    'field': '_id',
                    'embeddable': True
                }
            }
        },
        'name': {
            'type': 'string'
        },
        'description': {
            'type': 'string'
        }
    }
}

virtual_directory = {
    'schema': {
        'endpoint': {
            'type': 'string'
        },
        'protocol': {
            'type': 'string'
        },
        'name': {
            'type': 'string'
        },
        'collection': {
            'type': 'list',
            'schema': {
                'type': 'string'
            }
        }
    }
}

file_alias = {
    'schema': {
        'alias': {
            'type': 'string'
        },
        'name': {
            'type': 'string'
        }
    }
}

virtual_cluster = {
    'schema': {
        'nodes': {
            'type': 'list',
            'schema': {
                'type': 'objectid',
                'data_relation': {
```

718

```
                    'resource': 'virtual_machine',
                    'field': '_id',
                    'embeddable': True
                }
            }
        },
        'frontend': {
            'type': 'objectid',
            'data_relation': {
                'resource': 'virtual_machine',
                'field': '_id',
                'embeddable': True
            }
        },
        'name': {
            'type': 'string'
        }
    }
}

libcloud_flavor = {
    'schema': {
        'uuid': {
            'type': 'string'
        },
        'price': {
            'type': 'string'
        },
        'ram': {
            'type': 'string'
        },
        'bandwidth': {
            'type': 'string'
        },
        'flavor_id': {
            'type': 'string'
        },
        'disk': {
            'type': 'string'
        },
        'cpu': {
            'type': 'string'
        }
    }
}

batchjob = {
    'schema': {
        'output_file': {
            'type': 'string'
        },
        'group': {
            'type': 'string'
        },
        'job_id': {
```

```
                'type': 'string'
            },
            'script': {
                'type': 'string'
            },
            'cmd': {
                'type': 'string'
            },
            'queue': {
                'type': 'string'
            },
            'cluster': {
                'type': 'string'
            },
            'time': {
                'type': 'string'
            },
            'path': {
                'type': 'string'
            },
            'nodes': {
                'type': 'string'
            },
            'dir': {
                'type': 'string'
            }
        }
}

organization = {
    'schema': {
        'users': {
            'type': 'list',
            'schema': {
                'type': 'objectid',
                'data_relation': {
                    'resource': 'user',
                    'field': '_id',
                    'embeddable': True
                }
            }
        }
    }
}

container = {
    'schema': {
        'ip': {
            'type': 'string'
        },
        'endpoint': {
            'type': 'string'
        },
        'name': {
            'type': 'string'
```

```
        },
        ’memoryGB’: {
            ’type’: ’integer’
        },
        ’label’: {
            ’type’: ’string’
        }
    }
}

sshkey = {
    ’schema’: {
        ’comment’: {
            ’type’: ’string’
        },
        ’source’: {
            ’type’: ’string’
        },
        ’uri’: {
            ’type’: ’string’
        },
        ’value’: {
            ’type’: ’string’
        },
        ’fingerprint’: {
            ’type’: ’string’
        }
    }
}

stream = {
    ’schema’: {
        ’attributes’: {
            ’type’: ’dict’,
            ’schema’: {
                ’rate’: {
                    ’type’: ’integer’
                },
                ’limit’: {
                    ’type’: ’integer’
                }
            }
        },
        ’name’: {
            ’type’: ’string’
        },
        ’format’: {
            ’type’: ’string’
        }
    }
}

database = {
    ’schema’: {
        ’endpoint’: {
```

```
            'type': 'string'
        },
        'protocol': {
            'type': 'string'
        },
        'name': {
            'type': 'string'
        }
    }
}

default = {
    'schema': {
        'context': {
            'type': 'string'
        },
        'name': {
            'type': 'string'
        },
        'value': {
            'type': 'string'
        }
    }
}

openstack_image = {
    'schema': {
        'status': {
            'type': 'string'
        },
        'username': {
            'type': 'string'
        },
        'updated': {
            'type': 'string'
        },
        'uuid': {
            'type': 'string'
        },
        'created': {
            'type': 'string'
        },
        'minDisk': {
            'type': 'string'
        },
        'progress': {
            'type': 'string'
        },
        'minRam': {
            'type': 'string'
        },
        'os_image_size': {
            'type': 'string'
        },
        'metadata': {
```

```
            'type': 'dict',
            'schema': {
                'instance_uuid': {
                    'type': 'string'
                },
                'image_location': {
                    'type': 'string'
                },
                'image_state': {
                    'type': 'string'
                },
                'instance_type_memory_mb': {
                    'type': 'string'
                },
                'user_id': {
                    'type': 'string'
                },
                'description': {
                    'type': 'string'
                },
                'kernel_id': {
                    'type': 'string'
                },
                'instance_type_name': {
                    'type': 'string'
                },
                'ramdisk_id': {
                    'type': 'string'
                },
                'instance_type_id': {
                    'type': 'string'
                },
                'instance_type_ephemeral_gb': {
                    'type': 'string'
                },
                'instance_type_rxtx_factor': {
                    'type': 'string'
                },
                'image_type': {
                    'type': 'string'
                },
                'network_allocated': {
                    'type': 'string'
                },
                'instance_type_flavorid': {
                    'type': 'string'
                },
                'instance_type_vcpus': {
                    'type': 'string'
                },
                'instance_type_root_gb': {
                    'type': 'string'
                },
                'base_image_ref': {
                    'type': 'string'
```

```
                    },
                    'instance_type_swap': {
                        'type': 'string'
                    },
                    'owner_id': {
                        'type': 'string'
                    }
                }
            }
        }
}

azure_image = {
    'schema': {
        '_uuid': {
            'type': 'string'
        },
        'driver': {
            'type': 'string'
        },
        'id': {
            'type': 'string'
        },
        'name': {
            'type': 'string'
        },
        'extra': {
            'type': 'dict',
            'schema': {
                'category': {
                    'type': 'string'
                },
                'description': {
                    'type': 'string'
                },
                'vm_image': {
                    'type': 'string'
                },
                'location': {
                    'type': 'string'
                },
                'affinity_group': {
                    'type': 'string'
                },
                'os': {
                    'type': 'string'
                },
                'media_link': {
                    'type': 'string'
                }
            }
        }
    }
}
```

724

```
hadoop = {
    'schema': {
        'deployers': {
            'type': 'dict',
            'schema': {
                'ansible': {
                    'type': 'string'
                }
            }
        },
        'requires': {
            'type': 'dict',
            'schema': {
                'java': {
                    'type': 'dict',
                    'schema': {
                        'implementation': {
                            'type': 'string'
                        },
                        'version': {
                            'type': 'string'
                        },
                        'zookeeper': {
                            'type': 'string'
                        },
                        'supervisord': {
                            'type': 'string'
                        }
                    }
                }
            }
        },
        'parameters': {
            'type': 'dict',
            'schema': {
                'num_resourcemanagers': {
                    'type': 'integer'
                },
                'num_namenodes': {
                    'type': 'integer'
                },
                'use_yarn': {
                    'type': 'boolean'
                },
                'num_datanodes': {
                    'type': 'integer'
                },
                'use_hdfs': {
                    'type': 'boolean'
                },
                'num_historyservers': {
                    'type': 'integer'
                },
                'num_journalnodes': {
                    'type': 'integer'
```

725

```
                }
            }
        }
    }
}

compute_resource = {
    'schema': {
        'kind': {
            'type': 'string'
        },
        'endpoint': {
            'type': 'string'
        },
        'name': {
            'type': 'string'
        }
    }
}

node_new = {
    'schema': {
        'authorized_keys': {
            'type': 'list',
            'schema': {
                'type': 'string'
            }
        },
        'name': {
            'type': 'string'
        },
        'external_ip': {
            'type': 'string'
        },
        'memory': {
            'type': 'integer'
        },
        'create_external_ip': {
            'type': 'boolean'
        },
        'internal_ip': {
            'type': 'string'
        },
        'loginuser': {
            'type': 'string'
        },
        'owner': {
            'type': 'string'
        },
        'cores': {
            'type': 'integer'
        },
        'disk': {
            'type': 'integer'
        },
```

```
        'ssh_keys': {
            'type': 'list',
            'schema': {
                'type': 'dict',
                'schema': {
                    'from': {
                        'type': 'string'
                    },
                    'decrypt': {
                        'type': 'string'
                    },
                    'ssh_keygen': {
                        'type': 'boolean'
                    },
                    'to': {
                        'type': 'string'
                    }
                }
            }
        },
        'security_groups': {
            'type': 'list',
            'schema': {
                'type': 'dict',
                'schema': {
                    'ingress': {
                        'type': 'string'
                    },
                    'egress': {
                        'type': 'string'
                    },
                    'ports': {
                        'type': 'list',
                        'schema': {
                            'type': 'integer'
                        }
                    },
                    'protocols': {
                        'type': 'list',
                        'schema': {
                            'type': 'string'
                        }
                    }
                }
            }
        },
        'users': {
            'type': 'dict',
            'schema': {
                'name': {
                    'type': 'string'
                },
                'groups': {
                    'type': 'list',
                    'schema': {
```

```
                        'type': 'string'
                    }
                }
            }
        }
    }
}

filter = {
    'schema': {
        'function': {
            'type': 'string'
        },
        'name': {
            'type': 'string'
        }
    }
}

reservation = {
    'schema': {
        'start_time': {
            'type': 'list',
            'schema': {
                'type': 'string'
            }
        },
        'hosts': {
            'type': 'string'
        },
        'description': {
            'type': 'string'
        },
        'end_time': {
            'type': 'list',
            'schema': {
                'type': 'string'
            }
        }
    }
}

replica = {
    'schema': {
        'endpoint': {
            'type': 'string'
        },
        'name': {
            'type': 'string'
        },
        'checksum': {
            'type': 'dict',
            'schema': {
                'md5': {
                    'type': 'string'
```

```
                }
            }
        },
        'replica': {
            'type': 'string'
        },
        'accessed': {
            'type': 'string'
        },
        'size': {
            'type': 'list',
            'schema': {
                'type': 'string'
            }
        }
    }
}

microservice = {
    'schema': {
        'function': {
            'type': 'string'
        },
        'endpoint': {
            'type': 'string'
        },
        'name': {
            'type': 'string'
        }
    }
}

var = {
    'schema': {
        'type': {
            'type': 'string'
        },
        'name': {
            'type': 'string'
        },
        'value': {
            'type': 'string'
        }
    }
}

mesos_docker = {
    'schema': {
        'container': {
            'type': 'dict',
            'schema': {
                'docker': {
                    'type': 'dict',
                    'schema': {
                        'credential': {
```

```
                                    'type': 'dict',
                                    'schema': {
                                        'secret': {
                                            'type': 'string'
                                        },
                                        'principal': {
                                            'type': 'string'
                                        }
                                    }
                                },
                                'image': {
                                    'type': 'string'
                                }
                            }
                        },
                        'type': {
                            'type': 'string'
                        }
                    }
                },
                'mem': {
                    'type': 'float'
                },
                'args': {
                    'type': 'list',
                    'schema': {
                        'type': 'string'
                    }
                },
                'cpus': {
                    'type': 'float'
                },
                'instances': {
                    'type': 'integer'
                },
                'id': {
                    'type': 'string'
                }
            }
        }

        libcloud_vm = {
            'schema': {
                'username': {
                    'type': 'string'
                },
                'status': {
                    'type': 'string'
                },
                'root_device_type': {
                    'type': 'string'
                },
                'private_ips': {
                    'type': 'string'
                },
```

```
        'instance_type': {
            'type': 'string'
        },
        'image': {
            'type': 'string'
        },
        'private_dns': {
            'type': 'string'
        },
        'image_name': {
            'type': 'string'
        },
        'instance_id': {
            'type': 'string'
        },
        'image_id': {
            'type': 'string'
        },
        'public_ips': {
            'type': 'string'
        },
        'state': {
            'type': 'string'
        },
        'root_device_name': {
            'type': 'string'
        },
        'key': {
            'type': 'string'
        },
        'group': {
            'type': 'string'
        },
        'flavor': {
            'type': 'string'
        },
        'availability': {
            'type': 'string'
        },
        'uuid': {
            'type': 'string'
        }
    }
}


eve_settings = {
    'MONGO_HOST': 'localhost',
    'MONGO_DBNAME': 'testing',
    'RESOURCE_METHODS': ['GET', 'POST', 'DELETE'],
    'BANDWIDTH_SAVER': False,
    'DOMAIN': {
        'profile': profile,
        'virtual_machine': virtual_machine,
        'kubernetes': kubernetes,
```

```
        'nic': nic,
        'virtual_compute_node': virtual_compute_node,
        'openstack_flavor': openstack_flavor,
        'azure-vm': azure_vm,
        'azure-size': azure_size,
        'openstack_vm': openstack_vm,
        'cluster': cluster,
        'computer': computer,
        'libcloud_image': libcloud_image,
        'user': user,
        'file': file,
        'deployment': deployment,
        'mapreduce': mapreduce,
        'group': group,
        'role': role,
        'virtual_directory': virtual_directory,
        'file_alias': file_alias,
        'virtual_cluster': virtual_cluster,
        'libcloud_flavor': libcloud_flavor,
        'batchjob': batchjob,
        'organization': organization,
        'container': container,
        'sshkey': sshkey,
        'stream': stream,
        'database': database,
        'default': default,
        'openstack_image': openstack_image,
        'azure_image': azure_image,
        'hadoop': hadoop,
        'compute_resource': compute_resource,
        'node_new': node_new,
        'filter': filter,
        'reservation': reservation,
        'replica': replica,
        'microservice': microservice,
        'var': var,
        'mesos-docker': mesos_docker,
        'libcloud_vm': libcloud_vm,
    },
}
```

### A.3. 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 subsection 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 acknoledge you either as contributor or author. Please discuss with us how we best acknowledge you.

### A.4. Using the Cloudmesh REST Service

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

For example:

Listing A.3: profile

```
{
  "profile": {
```

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

### A.4.1. Element Definition

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

### A.4.2. Yaml

```
entry = yaml.read('''
profile:
  description: |
    A user profile that specifies general information
    about the user
  email: laszewski@gmail.com, required
  firtsname: Gregor, required
  lastname: von Laszewski, required
  height: 180
'''}
```

### A.4.3. Cerberus

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

## A.5. Mongoengine

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

## A.6. 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
```

#### A.6.1. Defining Elements for the REST Service

To manage a large number of elements defined in our REST service easily, we manage them trhough definitions in yaml files. To generate the appropriate settings file for the rest service, we can use teh 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 yml objects defined in the directory `resources/samples` you can use the following command:

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

#### A.6.2. DOIT

cms schema spec2tex resources/specification resources/tex

#### A.6.3. Generating service

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

### A.7. ABC

**README.rst**

## B. CLOUDMESH REST

### B.1. Prerequistis

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

#### B.1.1. Install Mongo on OSX

```
brew update
brew install mongodb

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

#### B.1.2. Install Mongo on OSX

ASSIGNMET TO STUDENTS, PROVIDE PULL REQUEST WITH INSTRUCTIONS

### B.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 yml to be converted to json and from json with evegenie into a valid eve settings file.

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

### B.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 wih:

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

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

### B.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"
  }
}
```

### B.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 tow 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 containes 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

#### *B.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
```

#### *B.5.2. Manageing 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 coppies 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 purpposes 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 cmd5
make deploy
make test
```

classes lessons rest.rst

# C. REST WITH EVE

## C.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 illustrationg how to implement them with HTTP methods.

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

## C.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.

#### *C.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 you
elect Windows 10. YOu could be using the online documentation
provided by starting it on Windows, or rinning it in a docker container.
```

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

#### *C.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 targes in the makefile and replicate them one by one. Improtaht 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 incimming connections from the local host which will be suffiicent 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.

### C.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:

```
cd ~/github
git clone https://github.com/cloudmesh/evegenie
cd evegenie
python setup.py install
pip install .
```

This shoudl install in your system evegenie. YOu can verify this by typing:

```
which evegenie
```

If you see the path evegenie is installed. With evegenie installed its usaage is simple:

```
$ evegenie

Usage:
    evegenie --help
    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 evegenie 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 evegenie, 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.

### C.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 hobe that students through collaborative work and pull requests complete such an admin command.

The proposed command is located at:

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

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

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

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

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

## D. 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 developping commands by a number of people and to have a dynamic plugin mechnism, we developed cmd5. It is a rewrite on our ealier effords in cloudmesh and cmd3.

### D.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:

```
pyenev virtualenv 2.7.13 ENV2
```

Now you need to get two source directories. We assume yo 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.

### D.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:

```
+-------------------------------------------------------+
|   ____ _                 _                     _      |
|  / ___| | ___  _   _  __| |_ __ ___   ___  ___| |__   |
| | |   | |/ _ \| | | |/ _‘ | ’_ ‘ _ \ / _ \/ __| ’_ \  |
| | |___| | (_) | |_| | (_| | | | | | |  __/\__ \ | | | |
|  \____|_|\___/ \__,_|\__,_|_| |_| |_|\___||___/_| |_| |
+-------------------------------------------------------+
|                  Cloudmesh CMD5 Shell                 |
+-------------------------------------------------------+

cms>
```

To see the list of commands you can say

cms> help

To see the manula page for a specific command, please use:

```
help COMMANDNAME
```

### D.3. Create your own Extension

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

- https://github.com/cloudmesh/extbar.git

Simply copy the code and modify the bar and foo commands to fit yor needs.

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

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

```
cloudmsesh/ext/command/COMMANDNAME.py
```

An example for the bar command is presented at:

- https://github.com/cloudmesh/extbar/blob/master/cloudmesh/ext/command/bar.py

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

```
from __future__ import print_function
from cloudmesh.shell.command import command
from cloudmesh.shell.command import PluginCommand

class BarCommand(PluginCommand):

    @command
    def do_bar(self, args, arguments):
        """
        ::
          Usage:
                command -f FILE
                command FILE
                command list
          This command does some useful things.
          Arguments:
              FILE   a file name
          Options:
              -f      specify the file
        """
        print(arguments)
```

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

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

### D.4. Excersise

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

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

**CMD5.3:** Write a new command and experiment with docopt syntax and argument interpretation of the dict with if conditions.

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

### D.5. Acronyms

The following acronyms are used in the paper

| | |
|---|---|
| **ACID** | Atomicity, Consistency, Isolation, Durability |
| **API** | Application Programming Interface |
| **ASCII** | American Standard Code for Information Interchange |
| **BASE** | Basically Available, Soft state, Eventual consistency |
| **DevOps** | A clipped compound of *software DEVelopment* and *information technology OPerationS* |
| **HTTP** | HyperText Transfer Protocol HTTPS HTTP Secure |
| **IaaS** | Infrastructure as a Service SaaS Software as a Service |
| **ITL** | Information Technology Laboratory |
| **NBD-PWG** | NIST Big Data Public Working Group |
| **NBDRA** | NIST Big Data Reference Architecture |
| **NBDRAI** | NIST Big Data Reference Architecture Interface |
| **NIST** | Big Data Interoperability Framework: Volume 8, Reference Architecture Interface |
| **NIST** | National Institute of Standards |
| **OS** | Operating System |
| **REST** | REpresentational State Transfer |
| **WWW** | World Wide Web |