

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>

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

CONTENTS

28	1 Introduction	8
29		
30	2 NBDRA Interface Requirements	8
31	2.1 High Level Requirements of the Interface Approach	8
32	2.1.1 Technology and Vendor Agnostic	8
33	2.1.2 Support of Plug-In Compute Infrastructure	9
34	2.1.3 Orchestration of Infrastructure and Services	9
35	2.1.4 Orchestration of Big Data Applications and Experiments	9
36	2.1.5 Reusability	9
37	2.1.6 Execution Workloads	9
38	2.1.7 Security and Privacy Fabric Requirements	9
39	2.1.8 System Orchestration Requirement	9
40	2.1.9 Application Providers Requirements	10
41	2.2 Component Specific Interface Requirements	10
42	2.2.1 System Orchestrator Interface Requirement	10
43	2.2.2 Data Provider Interface Requirement	10
44	2.2.3 Data Consumer Interface Requirement	10
45	2.2.4 Big Data Application Interface Provide	10
46	2.2.4.1 Collection	11
47	2.2.4.2 Preparation	11
48	2.2.4.3 Analytics	11
49	2.2.4.4 Visualization	11
50	2.2.4.5 Access	11
51	2.2.4.6 Big Data Provider Framework Interface Requirements	12
52	2.2.4.7 Infrastructures Interface Requirements	12
53	2.2.4.8 Platforms Interface Requirements	12
54	2.2.4.9 Processing Interface Requirements	12
55	2.2.4.10 Crosscutting Interface Requirements	12
56	2.2.4.11 Messaging/Communications Frameworks	12
57	2.2.4.12 Resource Management Framework	12
58	2.2.5 BD Application Provider to Framework Provider Interface	12
59	3 Specification Paradigm	13
60	3.1 Lessons Learned	13
61	3.2 Hybrid Cloud	13
62	3.3 Design by Example	13
63	3.4 Tools to Create the Specifications	13
64	3.5 Installation of the Tools	14
65	3.6 Document Creation	14
66	3.7 Conversion to Word	14
67	3.8 Interface Compliancy	14
68	4 Specification	14
69	4.1 User and Profile	14
70	4.1.1 Profile	15
71	4.1.2 User	15
72	4.1.3 Organization	15
73	4.1.4 Group/Role	16
74	4.2 Data	16
75	4.2.1 Var	17
76	4.2.2 Default	17
77	4.2.3 File	17
78	4.2.4 File Alias	18
79	4.2.5 Replica	18
80	4.2.6 Virtual Directory	18

81	4.2.7	Database	19
82	4.2.8	Stream	19
83	4.3	IaaS	19
84	4.3.1	Openstack	19
85	4.3.1.1	Openstack Flavor	19
86	4.3.1.2	Openstack Image	20
87	4.3.1.3	Openstack Vm	20
88	4.3.2	Azure	21
89	4.3.2.1	Azure Size	21
90	4.3.2.2	Azure Image	22
91	4.3.2.3	Azure Vm	22
92	4.4	HPC	23
93	4.4.1	Batch Job	23
94	4.5	Virtual Cluster	23
95	4.5.1	Cluster	23
96	4.5.2	New Cluster	23
97	4.5.3	Compute Resource	24
98	4.5.4	Computer	25
99	4.5.5	Compute Node	25
100	4.5.6	Virtual Cluster	27
101	4.5.7	Virtual Compute node	27
102	4.5.8	Virtual Machine	27
103	4.5.9	Mesos	28
104	4.6	Containers	28
105	4.6.1	Container	28
106	4.6.2	Kubernetes	28
107	4.7	Deployment	30
108	4.7.1	Deployment	30
109	4.8	Mapreduce	30
110	4.8.1	Mapreduce	30
111	4.8.2	Hadoop	31
112	4.9	Security	32
113	4.9.1	Key	32
114	4.10	Microservice	32
115	4.10.1	Microservice	32
116	4.10.2	Reservation	33
117	4.11	Network	33
118	A	Appendix	34
119	A.1	Schema Command	34
120	A.2	Schema	34
121	A.3	Contributing	62
122	A.4	Using the Cloudmesh REST Service	62
123	A.4.1	Element Definition	63
124	A.4.2	Yaml	63
125	A.4.3	Cerberus	63
126	A.5	Mongoengine	63
127	A.6	Cloudmesh Notation	63
128	A.6.1	Defining Elements for the REST Service	64
129	A.6.2	DOIT	64
130	A.6.3	Generating service	64
131	A.7	ABC	64

132	B Cloudmesh Rest	64
133	B.1 Prerequistis	64
134	B.1.1 Install Mongo on OSX	64
135	B.1.2 Install Mongo on OSX	64
136	B.2 Introduction	64
137	B.3 Yaml Specification	65
138	B.4 Json Specification	65
139	B.5 Conversion to Eve Settings	65
140	B.5.1 Managing Mongo	65
141	B.5.2 Manageing Eve	66
142	C REST with Eve	66
143	C.1 Overview of REST	66
144	C.2 REST and eve	66
145	C.2.1 Installation	67
146	C.2.2 Starting the service	67
147	C.3 Creating your own objects	67
148	C.4 Towards cmd5 extensions to manage eve and mongo	68
149	D CMD5	68
150	D.1 Resources	68
151	D.2 Execution	69
152	D.3 Create your own Extension	70
153	D.4 Excercise	71
154	D.5 Acronyms	71

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.

Certain commercial entities, equipment, or materials may be identified in this document in order to describe an experimental procedure or concept adequately. Such identification is not intended to imply recommendation or endorsement by NIST, nor is it intended to imply that the entities, materials, or equipment are necessarily the best available for the purpose.

There may be references in this publication to other publications currently under development by NIST in accordance with its assigned statutory responsibilities. The information in this publication, including concepts and methodologies, may be used by federal agencies even before the completion of such companion publications. Thus, until each publication is completed, current requirements, guidelines, and procedures, where they exist, remain operative. For planning and transition purposes, federal agencies may wish to closely follow the development of these new publications by NIST.

Organizations are encouraged to review all draft publications during public comment periods and provide feedback to NIST. All NIST publications are available at <http://www.nist.gov/publication-portal.cfm>.

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.

ACKNOWLEDGEMENTS

This document reflects the contributions and discussions by the membership of the NBD-PWG, co-chaired by Wo Chang of the NIST ITL, Robert Marcus of ET-Strategies, and Chaitanya Baru, University of California San Diego Supercomputer Center. The document contains input from members of the NBD-PWG:

<TBD>

NIST SP1500-8, Version 1 has been collaboratively authored by the NBD-PWG. As of the date of publication, there are over

<TBD>

NIST acknowledges the specific contributions to this volume by the following NBD-PWG members:

<TBD>

The editors for this document were Gregor von Laszewski and Wo Chang.

ABSTRACT

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

Big Data is a term used to describe the large amount of data in the networked, digitized, sensor-laden, information-driven world. While opportunities exist with Big Data, the data can overwhelm traditional technical approaches, and the growth of data is outpacing scientific and technological advances in data analytics. To advance progress in Big Data, the NIST Big Data Public Working Group (NBD-PWG) is working to develop consensus on important fundamental concepts related to Big Data. The results are reported in the NIST Big Data Interoperability Framework series of volumes. This volume, Volume 6, summarizes the work performed by the NBD-PWG to characterize Big Data from an architecture perspective, presents the NIST Big Data Reference Architecture (NBDRA) conceptual model, and discusses the components and fabrics of the NBDRA.

KEYWORDS

Application Provider; Big Data; Big Data characteristics; Data Consumer; Data Provider; Framework Provider; Management Fabric; reference architecture; Security and Privacy Fabric; System Orchestrator; use cases.

[0]

REFERENCES

- [1] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Trans. Comput. Syst.*, vol. 26, no. 2, pp. 4:1–4:26, Jun. 2008. [Online]. Available: <http://doi.acm.org/10.1145/1365815.1365816>
- [2] B. Smith, T. Malyuta, W. S. Mandrick, C. Fu, and K. Parent, "Horizontal integration of warfighter intelligence data."
- [3] S. Yoakum-Stover and T. Malyuta, "Unified data integration for situation management," in *MILCOM 2008 - 2008 IEEE Military Communications Conference*, Nov 2008, pp. 1–7.
- [4] P. Colella, "Defining software requirements for scientific computing," Slide of 2004 presentation included in David Patterson's 2005 talk, 2005. [Online]. Available: <http://www.lanl.gov/conferences/salishan/salishan2005/davidpatterson.pdf>
- [5] D. Patterson and K. Yelick, "Dwarf mine," Online. [Online]. Available: http://view.eecs.berkeley.edu/wiki/Dwarf_Mine
- [6] T. W. H. O. of Science and T. Policy, "Big data is a big deal," Online, 2012. [Online]. Available: <http://www.whitehouse.gov/blog/2012/03/29/big-data-big-deal>
- [7] O. of The Assistant Secretary of Defense, "Reference architecture description," Online, 2010. [Online]. Available: http://dodcio.defense.gov/Portals/0/Documents/DIEA/Ref_Archi_Description_Final_v1_18Jun10.pdf
- [8] Internet2, "eduperson." [Online]. Available: <http://software.internet2.edu/eduperson/internet2-mace-dir-eduperson-201602.html>

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 1). 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 1

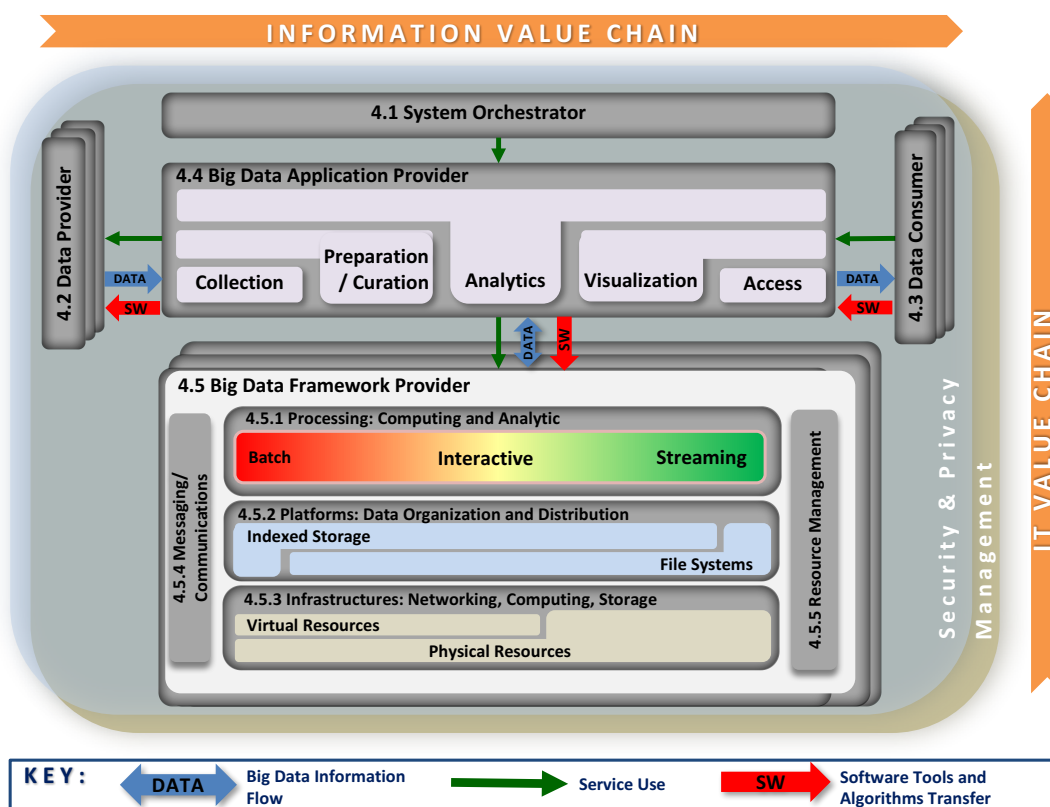


Fig. 1. NIST Big Data Reference Architecture (NBDRA)

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) the 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 use case. To achieve this task, we need to provision software stacks 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 too 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 of 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 Collection

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 Preparation

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 Analytics

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 Visualization

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 Access

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 Big Data Provider Framework Interface Requirements

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 to virtual data directories (3) interfaces to data streams (4) and interfaces to data filters.

2.2.4.7 Infrastructures Interface Requirements

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 Platforms Interface Requirements

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 Processing Interface Requirements

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 Crosscutting Interface Requirements

A number of crosscutting interface requirements within the NBDRA provider frameworks include messaging, communication, and resource management. Often these services 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 from Volume 3 of this series.

2.2.4.11 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

2.2.4.12 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 A.2. 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

```

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

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 AppendixA.2 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 L^AT_EX.

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 [8] 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.

what does the "context" represent? What are possible values? How do those values alter the interpretation of a profile?

Listing 4.1: User profile

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

4.1.2. User

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

There's redundancy in the definition of Profile and User, namely everything except "roles". I don't think the current definitions clearly illustrate what each is supposed to represent and how they fit together in the system.

Listing 4.2: user

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

Listing 4.3: user

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


4.1.4. Group/Role

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

The examples objects for Organization, Group, and Role should clearly illustrate the differences. Right now it is a bit unclear.

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 then be created for different contexts.

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

I don't think this is very clear. Elaborate with examples?

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.

What are the semantics of streams?

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

Whats the difference between Batch Filters and Stream Filters mentioned in Streams?

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.

Do we have examples of what this would look like?

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

```

1 {
2   "var": {
3     "name": "name of the variable",
4     "value": "the value of the variable as string",
5     "type": "the datatype of the variable such as int, str, float, ..."
6   }
7 }

```

4.2.2. Default

A default is a special variable that has a context associated with it. This allows one 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

```

1 {
2   "default": {
3     "value": "string",
4     "name": "string",
5     "context": "string - defines the context of the default (user, cloud, ...)"
6   }
7 }

```

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 endpoint, the checksum and the size. Additional parameters such as the last 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

```

1 {
2   "file": {
3     "name": "report.dat",
4     "endpoint": "file://gregor@machine.edu:/data/report.dat",
5     "checksum": {"sha256": "c01b39c7a35ccc ..... ebf45c69f08e17dfe3ef375a7b"},
6     "accessed": "1.1.2017:05:00:00:EST",
7     "created": "1.1.2017:05:00:00:EST",
8     "modified": "1.1.2017:05:00:00:EST",
9     "size": ["GB", "Byte"]
10  }
11 }

```

564 4.2.4. File Alias

565 A file could have one alias or even multiple ones. The reason for an alias is that a file may have a complex name but
566 a user may want to refer to that file in a name space that is suitable for the users application.

Listing 4.9: file alias

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

568 4.2.5. Replica

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

We need to describe why a Replica is different from a File object.

Listing 4.10: replica

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

574 4.2.6. Virtual Directory

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

Listing 4.11: virtual directory

```
1 {  
2   "virtual_directory": {  
3     "name": "data",  
4     "endpoint": "http://.../data/",  
5     "protocol": "http",  
6     "collection": [  
7       "report.dat",  
8       "file2"  
9     ]  
10  }  
11 }
```

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

Listing 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 Openstack Flavor

Listing 4.15: openstack flavor

```
{
  "openstack_flavor": {
    "os_flv_disabled": "string",
    "uuid": "string",
    "os_flv_ext_data": "string",
  }
}
```

```
6     "ram": "string",
7     "os_flavor_acces": "string",
8     "vcpus": "string",
9     "swap": "string",
10    "rxtx_factor": "string",
11    "disk": "string"
12  }
13 }
```

597

598 4.3.1.2 Openstack Image

Listing 4.16: openstack image



```
1 {
2   "openstack_image": {
3     "status": "string",
4     "username": "string",
5     "updated": "string",
6     "uuid": "string",
7     "created": "string",
8     "minDisk": "string",
9     "progress": "string",
10    "minRam": "string",
11    "os_image_size": "string",
12    "metadata": {
13      "image_location": "string",
14      "image_state": "string",
15      "description": "string",
16      "kernel_id": "string",
17      "instance_type_id": "string",
18      "ramdisk_id": "string",
19      "instance_type_name": "string",
20      "instance_type_rxtx_factor": "string",
21      "instance_type_vcpus": "string",
22      "user_id": "string",
23      "base_image_ref": "string",
24      "instance_uuid": "string",
25      "instance_type_memory_mb": "string",
26      "instance_type_swap": "string",
27      "image_type": "string",
28      "instance_type_ephemeral_gb": "string",
29      "instance_type_root_gb": "string",
30      "network_allocated": "string",
31      "instance_type_flavorid": "string",
32      "owner_id": "string"
33    }
34  }
35 }
```

599

600 4.3.1.3 Openstack Vm

Listing 4.17: openstack vm



```
1 {
2   "openstack_vm": {
3     "username": "string",
```

601

```
4     "vm_state": "string",
5     "updated": "string",
6     "hostId": "string",
7     "availability_zone": "string",
8     "terminated_at": "string",
9     "image": "string",
10    "floating_ip": "string",
11    "diskConfig": "string",
12    "key": "string",
13    "flavor__id": "string",
14    "user_id": "string",
15    "flavor": "string",
16    "static_ip": "string",
17    "security_groups": "string",
18    "volumes_attached": "string",
19    "task_state": "string",
20    "group": "string",
21    "uuid": "string",
22    "created": "string",
23    "tenant_id": "string",
24    "accessIPv4": "string",
25    "accessIPv6": "string",
26    "status": "string",
27    "power_state": "string",
28    "progress": "string",
29    "image__id": "string",
30    "launched_at": "string",
31    "config_drive": "string"
32  }
33 }
```

4.3.2. Azure

4.3.2.1 Azure Size

The size description of an azure vm

Listing 4.18: azure-size

```
1 {
2   "azure-size": {
3     "_uuid": "None",
4     "name": "D14 Faster Compute Instance",
5     "extra": {
6       "cores": 16,
7       "max_data_disks": 32
8     },
9     "price": 1.6261,
10    "ram": 114688,
11    "driver": "libcloud",
12    "bandwidth": "None",
13    "disk": 127,
14    "id": "Standard_D14"
15  }
16 }
```

4.3.2.2 Azure Image

Listing 4.19: azure-image

```

1 {
2   "azure_image": {
3     "_uuid": "None",
4     "driver": "libcloud",
5     "extra": {
6       "affinity_group": "",
7       "category": "Public",
8       "description": "Linux VM image with 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.",
9       "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",
10      "media_link": "",
11      "os": "Linux",
12      "vm_image": "False"
13    },
14    "id": "03f55de797f546a1b29d1....",
15    "name": "CoreCLR x64 Beta5 (11624) with PartsUnlimited Demo App on Ubuntu Server 14.04
→ LTS"
16  }
17 }

```

4.3.2.3 Azure Vm

An Azure virtual machine

Listing 4.20: azure-vm

```

1 {
2   "azure-vm": {
3     "username": "string",
4     "status": "string",
5     "deployment_slot": "string",
6     "cloud_service": "string",
7     "image": "string",
8     "floating_ip": "string",
9     "image_name": "string",
10    "key": "string",
11    "flavor": "string",
12    "resource_location": "string",
13    "disk_name": "string",
14    "private_ips": "string",
15    "group": "string",
16    "uuid": "string",
17    "dns_name": "string",
18    "instance_size": "string",
19    "instance_name": "string",
20    "public_ips": "string",
21    "media_link": "string"
22  }

```

}

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": [
```

```

6       { "count": 3,
7         "node": "objectid:virtual_machine"
8       }
9     ]
10  },
11  "virtual_machine" :{
12    "name": "vm1",
13    "ncpu": 2,
14    "RAM": "4G",
15    "disk": "40G",
16    "nics": ["objectid:nic"
17  ],
18    "OS": "Ubuntu-16.04",
19    "loginuser": "ubuntu",
20    "status": "active",
21    "metadata":{
22    },
23    "authorized_keys": [
24      "objectid:sshkey"
25    ]
26  },
27  "sshkey": {
28    "comment": "string",
29    "source": "string",
30    "uri": "string",
31    "value": "ssh-rsa AAA.....",
32    "fingerprint": "string, unique"
33  },
34  "nic": {
35    "name": "eth0",
36    "type": "ethernet",
37    "mac": "00:00:00:11:22:33",
38    "ip": "123.123.1.2",
39    "mask": "255.255.255.0",
40    "broadcast": "123.123.1.255",
41    "gateway": "123.123.1.1",
42    "mtu": 1500,
43    "bandwidth": "10Gbps"
44  }
45  }

```

4.5.3. Compute Resource

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

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

Listing 4.24: compute resource

```

1  {
2    "compute_resource": {
3      "name": "Compute1",
4      "endpoint": "http://.../cluster/",
5      "kind": "baremetal"

```



```

6   }
7   }

```

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

```

1  {
2    "computer": {
3      "ip": "127.0.0.1",
4      "name": "myComputer",
5      "memoryGB": 16,
6      "label": "server-001"
7    }
8  }

```

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

```

1  {
2    "node_new": {
3      "authorized_keys": [
4        "ssh-rsa AAAA...",
5        "ssh-ed25519 AAAA...",
6        "...etc"
7      ],
8      "name": "example-001",
9      "external_ip": "",
10     "loginuser": "root",
11     "create_external_ip": true,
12     "internal_ip": "",
13     "memory": 2048,
14     "owner": "",
15     "cores": 2,
16     "users": {

```

```
17     "name": "hello",
18     "groups": [
19         "wheel"
20     ]
21 },
22 "disk": 80,
23 "security_groups": [
24     {
25         "ingress": "0.0.0.0/32",
26         "egress": "0.0.0.0/32",
27         "ports": [
28             22,
29             443,
30             80,
31             8080
32         ],
33         "protocols": [
34             "tcp",
35             "udp"
36         ]
37     }
38 ],
39 "ssh_keys": [
40     {
41         "to": ".ssh/id_rsa",
42         "password": {
43             "decrypt": "gpg",
44             "from": "yaml",
45             "file": "secrets.yml.gpg",
46             "key": "users.hello.ssh[0]"
47         },
48         "method": "ed25519",
49         "ssh_keygen": true
50     },
51     {
52         "to": ".ssh/testing",
53         "password": {
54             "decrypt": "gpg",
55             "from": "yaml",
56             "file": "secrets.yml.gpg",
57             "key": "users.hello.ssh[1]"
58         },
59         "bits": 4098,
60         "method": "rsa",
61         "ssh_keygen": true
62     },
63     {
64         "decrypt": "gpg",
65         "from": "secrets/ssh/hello/copied.gpg",
66         "ssh_keygen": false,
67         "to": ".ssh/copied"
68     }
69 ]
70 }
71 }
```

657 4.5.6. Virtual Cluster

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

Listing 4.27: virtual cluster

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

661 4.5.7. Virtual Compute node

Listing 4.28: virtual compute node

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

663 4.5.8. Virtual Machine

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

Listing 4.29: virtual machine

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

```
16     ]
17   }
18 }
```

668

4.5.9. Mesos

Refine

670

Listing 4.30: mesos



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

671

4.6. Containers

4.6.1. Container

This defines **container** object.

Listing 4.31: container



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

675

4.6.2. Kubernetes

REFINE

677

Listing 4.32: kubernetes



```
1 {
2   "kubernetes": {
3     "kind": "List",
4     "items": [
```

678

```
5      {
6        "kind": "None",
7        "metadata": {
8          "name": "127.0.0.1"
9        },
10       "status": {
11         "capacity": {
12           "cpu": "4"
13         },
14         "addresses": [
15           {
16             "type": "LegacyHostIP",
17             "address": "127.0.0.1"
18           }
19         ]
20       },
21     },
22     {
23       "kind": "None",
24       "metadata": {
25         "name": "127.0.0.2"
26       },
27       "status": {
28         "capacity": {
29           "cpu": "8"
30         },
31         "addresses": [
32           {
33             "type": "LegacyHostIP",
34             "address": "127.0.0.2"
35           },
36           {
37             "type": "another",
38             "address": "127.0.0.3"
39           }
40         ]
41       },
42     },
43   ],
44   "users": [
45     {
46       "name": "myself",
47       "user": "gregor"
48     },
49     {
50       "name": "e2e",
51       "user": {
52         "username": "admin",
53         "password": "secret"
54       }
55     }
56   ]
57 }
58 }
```

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 4.35. 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

```

1  {"name": "name of this function",
2   "description": "These should be self-describing",
3   "source": "a URI to obtain the resource",
4   "install": {
5     "description": "instructions to install the source if needed",
6     "script": "source://install.sh"
7   },
8   "eval": {
9     "description": "How to evaluate this function",
10    "script": "source://run.sh",
11  },
12  "args": [],
13  "buildInputs": [
14    "list of",
15    "objects this function",
16    "depends on"
17  ],
18  "systemBuildInputs": [
19    "list of",
20    "packages required",
21    "to install"
22  ],
23  "outputs": {
24    "key1": {},
25    "key2": {}
26  }
27 }
```

Some example functions include the “NoOp” function shown in Listing 4.36. 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

```

1  { "name": "noop",
2    "description": "A function with no effect"
3  }
```

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

```

1  {
2    "hadoop": {
3      "deployers": {
4        "ansible": "git://github.com/cloudmesh_roles/hadoop"
```

```

5     },
6     "requires": {
7         "java": {
8             "implementation": "OpenJDK",
9             "version": "1.8",
10            "zookeeper": "TBD",
11            "supervisord": "TBD"
12        }
13    },
14    "parameters": {
15        "num_resourcemanager": 1,
16        "num_namenodes": 1,
17        "use_yarn": false,
18        "use_hdfs": true,
19        "num_datanodes": 1,
20        "num_historyservers": 1,
21        "num_journalnodes": 1
22    }
23 }
24 }

```

4.9. Security

4.9.1. Key

Listing 4.38: key

```

1 {
2   "sshkey": {
3     "comment": "string",
4     "source": "string",
5     "uri": "string",
6     "value": "ssh-rsa",
7     "fingerprint": "string, unique"
8   }
9 }

```

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

limitation of resource capacity such as networking.

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

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

Listing 4.39: microservice

```

1 {
2   "microservice" :{
3     "name": "ms1",
4     "endpoint": "http://.../ms/",
5     "function": "microservice spec"
6   }
7 }

```


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.

[0]

REFERENCES

- [1] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Trans. Comput. Syst.*, vol. 26, no. 2, pp. 4:1–4:26, Jun. 2008. [Online]. Available: <http://doi.acm.org/10.1145/1365815.1365816>
- [2] B. Smith, T. Malyuta, W. S. Mandrick, C. Fu, and K. Parent, "Horizontal integration of warfighter intelligence data."
- [3] S. Yoakum-Stover and T. Malyuta, "Unified data integration for situation management," in *MILCOM 2008 - 2008 IEEE Military Communications Conference*, Nov 2008, pp. 1–7.
- [4] P. Colella, "Defining software requirements for scientific computing," Slide of 2004 presentation included in David Patterson's 2005 talk, 2005. [Online]. Available: <http://www.lanl.gov/conferences/salishan/salishan2005/davidpatterson.pdf>
- [5] D. Patterson and K. Yelick, "Dwarf mine," Online. [Online]. Available: http://view.eecs.berkeley.edu/wiki/Dwarf_Mine
- [6] T. W. H. O. of Science and T. Policy, "Big data is a big deal," Online, 2012. [Online]. Available: <http://www.whitehouse.gov/blog/2012/03/29/big-data-big-deal>
- [7] O. of The Assistant Secretary of Defense, "Reference architecture description," Online, 2010. [Online]. Available: http://dodcio.defense.gov/Portals/0/Documents/DIEA/Ref_Archi_Description_Final_v1_18Jun10.pdf
- [8] Internet2, "eduperson." [Online]. Available: <http://software.internet2.edu/eduperson/internet2-mace-dir-eduperson-201602.html>

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|yaml] DIRECTORY FILENAME
  concatenates all files with ending yaml
  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|yaml] DIRECTORY FILENAME
  Concatinates all files with the given
  ending (either json, or yaml) 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'
    },
  },
}
```

```
15     'lastname': {
16         'type': 'string'
17     },
18     'email': {
19         'type': 'string'
20     },
21     'uuid': {
22         'type': 'string'
23     }
24 }
25 }
26
27 virtual_machine = {
28     'schema': {
29         'status': {
30             'type': 'string'
31         },
32         'authorized_keys': {
33             'type': 'list',
34             'schema': {
35                 'type': 'objectid',
36                 'data_relation': {
37                     'resource': 'sshkey',
38                     'field': '_id',
39                     'embeddable': True
40                 }
41             }
42         },
43         'name': {
44             'type': 'string'
45         },
46         'nics': {
47             'type': 'list',
48             'schema': {
49                 'type': 'objectid',
50                 'data_relation': {
51                     'resource': 'nic',
52                     'field': '_id',
53                     'embeddable': True
54                 }
55             }
56         },
57         'RAM': {
58             'type': 'string'
59         },
60         'ncpu': {
61             'type': 'integer'
62         },
63         'loginuser': {
64             'type': 'string'
65         },
66         'disk': {
67             'type': 'string'
68         },
69         'OS': {
```

```

70         'type': 'string'
71     },
72     'metadata': {
73         'type': 'dict',
74         'schema': {}
75     }
76 }
77 }
78
79 kubernetes = {
80     'schema': {
81         'items': {
82             'type': 'list',
83             'schema': {
84                 'type': 'dict',
85                 'schema': {
86                     'status': {
87                         'type': 'dict',
88                         'schema': {
89                             'capacity': {
90                                 'type': 'dict',
91                                 'schema': {
92                                     'cpu': {
93                                         'type': 'string'
94                                     }
95                                 }
96                             },
97                             'addresses': {
98                                 'type': 'list',
99                                 'schema': {
100                                     'type': 'dict',
101                                     'schema': {
102                                         'type': {
103                                             'type': 'string'
104                                         },
105                                         'address': {
106                                             'type': 'string'
107                                         }
108                                     }
109                                 }
110                             }
111                         }
112                     },
113                     'kind': {
114                         'type': 'string'
115                     },
116                     'metadata': {
117                         'type': 'dict',
118                         'schema': {
119                             'name': {
120                                 'type': 'string'
121                             }
122                         }
123                     }
124                 }
125             }
126         }
127     }
128 }

```

```

125     }
126 },
127 'kind': {
128     'type': 'string'
129 },
130 'users': {
131     'type': 'list',
132     'schema': {
133         'type': 'dict',
134         'schema': {
135             'name': {
136                 'type': 'string'
137             },
138             'user': {
139                 'type': 'dict',
140                 'schema': {
141                     'username': {
142                         'type': 'string'
143                     },
144                     'password': {
145                         'type': 'string'
146                     }
147                 }
148             }
149         }
150     }
151 }
152 }
153 }
154
155 nic = {
156     'schema': {
157         'name': {
158             'type': 'string'
159         },
160         'ip': {
161             'type': 'string'
162         },
163         'mask': {
164             'type': 'string'
165         },
166         'bandwidth': {
167             'type': 'string'
168         },
169         'mtu': {
170             'type': 'integer'
171         },
172         'broadcast': {
173             'type': 'string'
174         },
175         'mac': {
176             'type': 'string'
177         },
178         'type': {
179             'type': 'string'

```

```
180     },
181     'gateway': {
182         'type': 'string'
183     }
184 }
185 }
186
187 virtual_compute_node = {
188     'schema': {
189         'status': {
190             'type': 'string'
191         },
192         'endpoint': {
193             'type': 'string'
194         },
195         'name': {
196             'type': 'string'
197         },
198         'ip': {
199             'type': 'list',
200             'schema': {
201                 'type': 'string'
202             }
203         },
204         'image': {
205             'type': 'string'
206         },
207         'flavor': {
208             'type': 'string'
209         },
210         'metadata': {
211             'type': 'dict',
212             'schema': {
213                 'experiment': {
214                     'type': 'string'
215                 }
216             }
217         }
218     }
219 }
220
221 openstack_flavor = {
222     'schema': {
223         'os_flv_disabled': {
224             'type': 'string'
225         },
226         'uuid': {
227             'type': 'string'
228         },
229         'os_flv_ext_data': {
230             'type': 'string'
231         },
232         'ram': {
233             'type': 'string'
234         },

```

```
235     'os_flavor_acces': {
236         'type': 'string'
237     },
238     'vcpus': {
239         'type': 'string'
240     },
241     'swap': {
242         'type': 'string'
243     },
244     'rxtx_factor': {
245         'type': 'string'
246     },
247     'disk': {
248         'type': 'string'
249     }
250 }
251 }
252
253 azure_vm = {
254     'schema': {
255         'username': {
256             'type': 'string'
257         },
258         'status': {
259             'type': 'string'
260         },
261         'deployment_slot': {
262             'type': 'string'
263         },
264         'group': {
265             'type': 'string'
266         },
267         'private_ips': {
268             'type': 'string'
269         },
270         'cloud_service': {
271             'type': 'string'
272         },
273         'dns_name': {
274             'type': 'string'
275         },
276         'image': {
277             'type': 'string'
278         },
279         'floating_ip': {
280             'type': 'string'
281         },
282         'image_name': {
283             'type': 'string'
284         },
285         'instance_name': {
286             'type': 'string'
287         },
288         'public_ips': {
289             'type': 'string'
```

```
290     },
291     'media_link': {
292         'type': 'string'
293     },
294     'key': {
295         'type': 'string'
296     },
297     'flavor': {
298         'type': 'string'
299     },
300     'resource_location': {
301         'type': 'string'
302     },
303     'instance_size': {
304         'type': 'string'
305     },
306     'disk_name': {
307         'type': 'string'
308     },
309     'uuid': {
310         'type': 'string'
311     }
312 }
313 }
314
315 azure_size = {
316     'schema': {
317         'ram': {
318             'type': 'integer'
319         },
320         'name': {
321             'type': 'string'
322         },
323         'extra': {
324             'type': 'dict',
325             'schema': {
326                 'cores': {
327                     'type': 'integer'
328                 },
329                 'max_data_disks': {
330                     'type': 'integer'
331                 }
332             }
333         },
334         'price': {
335             'type': 'float'
336         },
337         '_uuid': {
338             'type': 'string'
339         },
340         'driver': {
341             'type': 'string'
342         },
343         'bandwidth': {
344             'type': 'string'
345         }
346     }
347 }
```



```
345     },
346     'disk': {
347         'type': 'integer'
348     },
349     'id': {
350         'type': 'string'
351     }
352 }
353 }
354
355 openstack_vm = {
356     'schema': {
357         'vm_state': {
358             'type': 'string'
359         },
360         'availability_zone': {
361             'type': 'string'
362         },
363         'terminated_at': {
364             'type': 'string'
365         },
366         'image': {
367             'type': 'string'
368         },
369         'diskConfig': {
370             'type': 'string'
371         },
372         'flavor': {
373             'type': 'string'
374         },
375         'security_groups': {
376             'type': 'string'
377         },
378         'volumes_attached': {
379             'type': 'string'
380         },
381         'user_id': {
382             'type': 'string'
383         },
384         'uuid': {
385             'type': 'string'
386         },
387         'accessIPv4': {
388             'type': 'string'
389         },
390         'accessIPv6': {
391             'type': 'string'
392         },
393         'power_state': {
394             'type': 'string'
395         },
396         'progress': {
397             'type': 'string'
398         },
399         'image__id': {
```

```
400         'type': 'string'
401     },
402     'launched_at': {
403         'type': 'string'
404     },
405     'config_drive': {
406         'type': 'string'
407     },
408     'username': {
409         'type': 'string'
410     },
411     'updated': {
412         'type': 'string'
413     },
414     'hostId': {
415         'type': 'string'
416     },
417     'floating_ip': {
418         'type': 'string'
419     },
420     'static_ip': {
421         'type': 'string'
422     },
423     'key': {
424         'type': 'string'
425     },
426     'flavor__id': {
427         'type': 'string'
428     },
429     'group': {
430         'type': 'string'
431     },
432     'task_state': {
433         'type': 'string'
434     },
435     'created': {
436         'type': 'string'
437     },
438     'tenant_id': {
439         'type': 'string'
440     },
441     'status': {
442         'type': 'string'
443     }
444 }
445 }
446
447 cluster = {
448     'schema': {
449         'provider': {
450             'type': 'list',
451             'schema': {
452                 'type': 'string'
453             }
454         },
455     },
456 }
```

```
455     'endpoint': {
456         'type': 'dict',
457         'schema': {
458             'passwd': {
459                 'type': 'string'
460             },
461             'url': {
462                 'type': 'string'
463             }
464         }
465     },
466     'name': {
467         'type': 'string'
468     },
469     'label': {
470         'type': 'string'
471     }
472 }
473 }
```

```
475 computer = {
476     'schema': {
477         'ip': {
478             'type': 'string'
479         },
480         'name': {
481             'type': 'string'
482         },
483         'memoryGB': {
484             'type': 'integer'
485         },
486         'label': {
487             'type': 'string'
488         }
489     }
490 }
```

```
492 libcloud_image = {
493     'schema': {
494         'username': {
495             'type': 'string'
496         },
497         'status': {
498             'type': 'string'
499         },
500         'updated': {
501             'type': 'string'
502         },
503         'description': {
504             'type': 'string'
505         },
506         'owner_alias': {
507             'type': 'string'
508         },
509         'kernel_id': {
```

```
510         'type': 'string'
511     },
512     'hypervisor': {
513         'type': 'string'
514     },
515     'ramdisk_id': {
516         'type': 'string'
517     },
518     'state': {
519         'type': 'string'
520     },
521     'created': {
522         'type': 'string'
523     },
524     'image_id': {
525         'type': 'string'
526     },
527     'image_location': {
528         'type': 'string'
529     },
530     'platform': {
531         'type': 'string'
532     },
533     'image_type': {
534         'type': 'string'
535     },
536     'is_public': {
537         'type': 'string'
538     },
539     'owner_id': {
540         'type': 'string'
541     },
542     'architecture': {
543         'type': 'string'
544     },
545     'virtualization_type': {
546         'type': 'string'
547     },
548     'uuid': {
549         'type': 'string'
550     }
551 }
552 }
553
554 user = {
555     'schema': {
556         'username': {
557             'type': 'string'
558         },
559         'context': {
560             'type': 'string'
561         },
562         'uuid': {
563             'type': 'string'
564         },
565     },
```

```
565     'firstname': {
566         'type': 'string'
567     },
568     'lastname': {
569         'type': 'string'
570     },
571     'roles': {
572         'type': 'list',
573         'schema': {
574             'type': 'string'
575         }
576     },
577     'email': {
578         'type': 'string'
579     }
580 }
581 }
582
583 file = {
584     'schema': {
585         'endpoint': {
586             'type': 'string'
587         },
588         'name': {
589             'type': 'string'
590         },
591         'created': {
592             'type': 'string'
593         },
594         'checksum': {
595             'type': 'dict',
596             'schema': {
597                 'md5': {
598                     'type': 'string'
599                 }
600             }
601         },
602         'modified': {
603             'type': 'string'
604         },
605         'accessed': {
606             'type': 'string'
607         },
608         'size': {
609             'type': 'list',
610             'schema': {
611                 'type': 'string'
612             }
613         }
614     }
615 }
616
617 deployment = {
618     'schema': {
619         'cluster': {
```

```

620         'type': 'list',
621         'schema': {
622             'type': 'dict',
623             'schema': {
624                 'id': {
625                     'type': 'string'
626                 }
627             }
628         },
629     },
630     'stack': {
631         'type': 'dict',
632         'schema': {
633             'layers': {
634                 'type': 'list',
635                 'schema': {
636                     'type': 'string'
637                 }
638             },
639             'parameters': {
640                 'type': 'dict',
641                 'schema': {
642                     'hadoop': {
643                         'type': 'dict',
644                         'schema': {
645                             'zookeeper.quorum': {
646                                 'type': 'list',
647                                 'schema': {
648                                     'type': 'string'
649                                 }
650                             }
651                         }
652                     }
653                 }
654             }
655         }
656     }
657 }
658
659
660 mapreduce = {
661     'schema': {
662         'layers': {
663             'type': 'list',
664             'schema': {
665                 'type': 'string'
666             }
667         },
668         'hdfs_datanode': {
669             'type': 'string'
670         },
671         'java': {
672             'type': 'dict',
673             'schema': {
674                 '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': {

```

```
730         'type': 'objectid',
731         'data_relation': {
732             'resource': 'user',
733             'field': '_id',
734             'embeddable': True
735         }
736     },
737 },
738 'name': {
739     'type': 'string'
740 },
741 'description': {
742     'type': 'string'
743 }
744 }
745 }
746
747 virtual_directory = {
748     'schema': {
749         'endpoint': {
750             'type': 'string'
751         },
752         'protocol': {
753             'type': 'string'
754         },
755         'name': {
756             'type': 'string'
757         },
758         'collection': {
759             'type': 'list',
760             'schema': {
761                 'type': 'string'
762             }
763         }
764     }
765 }
766
767 file_alias = {
768     'schema': {
769         'alias': {
770             'type': 'string'
771         },
772         'name': {
773             'type': 'string'
774         }
775     }
776 }
777
778 virtual_cluster = {
779     'schema': {
780         'nodes': {
781             'type': 'list',
782             'schema': {
783                 'type': 'objectid',
784                 'data_relation': {
```



```
785         'resource': 'virtual_machine',
786         'field': '_id',
787         'embeddable': True
788     }
789 }
790 },
791 'frontend': {
792     'type': 'objectid',
793     'data_relation': {
794         'resource': 'virtual_machine',
795         'field': '_id',
796         'embeddable': True
797     }
798 },
799 'name': {
800     'type': 'string'
801 }
802 }
803 }
```

```
804
805 libcloud_flavor = {
806     'schema': {
807         'uuid': {
808             'type': 'string'
809         },
810         'price': {
811             'type': 'string'
812         },
813         'ram': {
814             'type': 'string'
815         },
816         'bandwidth': {
817             'type': 'string'
818         },
819         'flavor_id': {
820             'type': 'string'
821         },
822         'disk': {
823             'type': 'string'
824         },
825         'cpu': {
826             'type': 'string'
827         }
828     }
829 }
```

```
830
831 batchjob = {
832     'schema': {
833         'output_file': {
834             'type': 'string'
835         },
836         'group': {
837             'type': 'string'
838         },
839         'job_id': {
```

```

840         'type': 'string'
841     },
842     'script': {
843         'type': 'string'
844     },
845     'cmd': {
846         'type': 'string'
847     },
848     'queue': {
849         'type': 'string'
850     },
851     'cluster': {
852         'type': 'string'
853     },
854     'time': {
855         'type': 'string'
856     },
857     'path': {
858         'type': 'string'
859     },
860     'nodes': {
861         'type': 'string'
862     },
863     'dir': {
864         'type': 'string'
865     }
866 }
867 }
868
869 organization = {
870     'schema': {
871         'users': {
872             'type': 'list',
873             'schema': {
874                 'type': 'objectid',
875                 'data_relation': {
876                     'resource': 'user',
877                     'field': '_id',
878                     'embeddable': True
879                 }
880             }
881         }
882     }
883 }
884
885 container = {
886     'schema': {
887         'ip': {
888             'type': 'string'
889         },
890         'endpoint': {
891             'type': 'string'
892         },
893         'name': {
894             'type': 'string'

```

```
895     },
896     'memoryGB': {
897         'type': 'integer'
898     },
899     'label': {
900         'type': 'string'
901     }
902 }
903 }
904
905 sshkey = {
906     'schema': {
907         'comment': {
908             'type': 'string'
909         },
910         'source': {
911             'type': 'string'
912         },
913         'uri': {
914             'type': 'string'
915         },
916         'value': {
917             'type': 'string'
918         },
919         'fingerprint': {
920             'type': 'string'
921         }
922     }
923 }
924
925 stream = {
926     'schema': {
927         'attributes': {
928             'type': 'dict',
929             'schema': {
930                 'rate': {
931                     'type': 'integer'
932                 },
933                 'limit': {
934                     'type': 'integer'
935                 }
936             }
937         },
938         'name': {
939             'type': 'string'
940         },
941         'format': {
942             'type': 'string'
943         }
944     }
945 }
946
947 database = {
948     'schema': {
949         'endpoint': {
```

```
950         'type': 'string'
951     },
952     'protocol': {
953         'type': 'string'
954     },
955     'name': {
956         'type': 'string'
957     }
958 }
959 }
960
961 default = {
962     'schema': {
963         'context': {
964             'type': 'string'
965         },
966         'name': {
967             'type': 'string'
968         },
969         'value': {
970             'type': 'string'
971         }
972     }
973 }
974
975 openstack_image = {
976     'schema': {
977         'status': {
978             'type': 'string'
979         },
980         'username': {
981             'type': 'string'
982         },
983         'updated': {
984             'type': 'string'
985         },
986         'uuid': {
987             'type': 'string'
988         },
989         'created': {
990             'type': 'string'
991         },
992         'minDisk': {
993             'type': 'string'
994         },
995         'progress': {
996             'type': 'string'
997         },
998         'minRam': {
999             'type': 'string'
1000         },
1001         'os_image_size': {
1002             'type': 'string'
1003         },
1004         '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'
  }
}
```

```
1060         },
1061         'instance_type_swap': {
1062             'type': 'string'
1063         },
1064         'owner_id': {
1065             'type': 'string'
1066         }
1067     }
1068 }
1069 }
1070 }
1071
1072 azure_image = {
1073     'schema': {
1074         '_uuid': {
1075             'type': 'string'
1076         },
1077         'driver': {
1078             'type': 'string'
1079         },
1080         'id': {
1081             'type': 'string'
1082         },
1083         'name': {
1084             'type': 'string'
1085         },
1086         'extra': {
1087             'type': 'dict',
1088             'schema': {
1089                 'category': {
1090                     'type': 'string'
1091                 },
1092                 'description': {
1093                     'type': 'string'
1094                 },
1095                 'vm_image': {
1096                     'type': 'string'
1097                 },
1098                 'location': {
1099                     'type': 'string'
1100                 },
1101                 'affinity_group': {
1102                     'type': 'string'
1103                 },
1104                 'os': {
1105                     'type': 'string'
1106                 },
1107                 'media_link': {
1108                     'type': 'string'
1109                 }
1110             }
1111         }
1112     }
1113 }
```

```
1115  hadoop = {
1116    'schema': {
1117      'deployers': {
1118        'type': 'dict',
1119        'schema': {
1120          'ansible': {
1121            'type': 'string'
1122          }
1123        }
1124      },
1125      'requires': {
1126        'type': 'dict',
1127        'schema': {
1128          'java': {
1129            'type': 'dict',
1130            'schema': {
1131              'implementation': {
1132                'type': 'string'
1133              },
1134              'version': {
1135                'type': 'string'
1136              },
1137              'zookeeper': {
1138                'type': 'string'
1139              },
1140              'supervisord': {
1141                'type': 'string'
1142              }
1143            }
1144          }
1145        }
1146      },
1147      'parameters': {
1148        'type': 'dict',
1149        'schema': {
1150          'num_resourcemangers': {
1151            'type': 'integer'
1152          },
1153          'num_namenodes': {
1154            'type': 'integer'
1155          },
1156          'use_yarn': {
1157            'type': 'boolean'
1158          },
1159          'num_datanodes': {
1160            'type': 'integer'
1161          },
1162          'use_hdfs': {
1163            'type': 'boolean'
1164          },
1165          'num_historyservers': {
1166            'type': 'integer'
1167          },
1168          'num_journalnodes': {
1169            'type': 'integer'
```

```
1170     }
1171   }
1172 }
1173 }
1174 }
1175
1176 compute_resource = {
1177   'schema': {
1178     'kind': {
1179       'type': 'string'
1180     },
1181     'endpoint': {
1182       'type': 'string'
1183     },
1184     'name': {
1185       'type': 'string'
1186     }
1187   }
1188 }
1189
1190 node_new = {
1191   'schema': {
1192     'authorized_keys': {
1193       'type': 'list',
1194       'schema': {
1195         'type': 'string'
1196       }
1197     },
1198     'name': {
1199       'type': 'string'
1200     },
1201     'external_ip': {
1202       'type': 'string'
1203     },
1204     'memory': {
1205       'type': 'integer'
1206     },
1207     'create_external_ip': {
1208       'type': 'boolean'
1209     },
1210     'internal_ip': {
1211       'type': 'string'
1212     },
1213     'loginuser': {
1214       'type': 'string'
1215     },
1216     'owner': {
1217       'type': 'string'
1218     },
1219     'cores': {
1220       'type': 'integer'
1221     },
1222     'disk': {
1223       'type': 'integer'
1224     },
1225   },
1226 }
```



```
1225     'ssh_keys': {
1226         'type': 'list',
1227         'schema': {
1228             'type': 'dict',
1229             'schema': {
1230                 'from': {
1231                     'type': 'string'
1232                 },
1233                 'decrypt': {
1234                     'type': 'string'
1235                 },
1236                 'ssh_keygen': {
1237                     'type': 'boolean'
1238                 },
1239                 'to': {
1240                     'type': 'string'
1241                 }
1242             }
1243         }
1244     },
1245     'security_groups': {
1246         'type': 'list',
1247         'schema': {
1248             'type': 'dict',
1249             'schema': {
1250                 'ingress': {
1251                     'type': 'string'
1252                 },
1253                 'egress': {
1254                     'type': 'string'
1255                 },
1256                 'ports': {
1257                     'type': 'list',
1258                     'schema': {
1259                         'type': 'integer'
1260                     }
1261                 },
1262                 'protocols': {
1263                     'type': 'list',
1264                     'schema': {
1265                         'type': 'string'
1266                     }
1267                 }
1268             }
1269         }
1270     },
1271     'users': {
1272         'type': 'dict',
1273         'schema': {
1274             'name': {
1275                 'type': 'string'
1276             },
1277             'groups': {
1278                 'type': 'list',
1279                 'schema': {
```

```
1280         'type': 'string'
1281     }
1282 }
1283 }
1284 }
1285 }
1286 }
1287
1288 filter = {
1289     'schema': {
1290         'function': {
1291             'type': 'string'
1292         },
1293         'name': {
1294             'type': 'string'
1295         }
1296     }
1297 }
1298
1299 reservation = {
1300     'schema': {
1301         'start_time': {
1302             'type': 'list',
1303             'schema': {
1304                 'type': 'string'
1305             }
1306         },
1307         'hosts': {
1308             'type': 'string'
1309         },
1310         'description': {
1311             'type': 'string'
1312         },
1313         'end_time': {
1314             'type': 'list',
1315             'schema': {
1316                 'type': 'string'
1317             }
1318         }
1319     }
1320 }
1321
1322 replica = {
1323     'schema': {
1324         'endpoint': {
1325             'type': 'string'
1326         },
1327         'name': {
1328             'type': 'string'
1329         },
1330         'checksum': {
1331             'type': 'dict',
1332             'schema': {
1333                 'md5': {
1334                     'type': 'string'
```

```

1335         }
1336     },
1337 },
1338     'replica': {
1339         'type': 'string'
1340     },
1341     'accessed': {
1342         'type': 'string'
1343     },
1344     'size': {
1345         'type': 'list',
1346         'schema': {
1347             'type': 'string'
1348         }
1349     }
1350 }
1351 }
1352
1353 microservice = {
1354     'schema': {
1355         'function': {
1356             'type': 'string'
1357         },
1358         'endpoint': {
1359             'type': 'string'
1360         },
1361         'name': {
1362             'type': 'string'
1363         }
1364     }
1365 }
1366
1367 var = {
1368     'schema': {
1369         'type': {
1370             'type': 'string'
1371         },
1372         'name': {
1373             'type': 'string'
1374         },
1375         'value': {
1376             'type': 'string'
1377         }
1378     }
1379 }
1380
1381 mesos_docker = {
1382     'schema': {
1383         'container': {
1384             'type': 'dict',
1385             'schema': {
1386                 'docker': {
1387                     'type': 'dict',
1388                     'schema': {
1389                         'credential': {

```

```

1390         'type': 'dict',
1391         'schema': {
1392             'secret': {
1393                 'type': 'string'
1394             },
1395             'principal': {
1396                 'type': 'string'
1397             }
1398         },
1399     },
1400     'image': {
1401         'type': 'string'
1402     }
1403 },
1404 },
1405 'type': {
1406     'type': 'string'
1407 }
1408 },
1409 },
1410 'mem': {
1411     'type': 'float'
1412 },
1413 'args': {
1414     'type': 'list',
1415     'schema': {
1416         'type': 'string'
1417     }
1418 },
1419 'cpus': {
1420     'type': 'float'
1421 },
1422 'instances': {
1423     'type': 'integer'
1424 },
1425 'id': {
1426     'type': 'string'
1427 }
1428 }
1429 }

```

```

1430
1431 libcloud_vm = {
1432     'schema': {
1433         'username': {
1434             'type': 'string'
1435         },
1436         'status': {
1437             'type': 'string'
1438         },
1439         'root_device_type': {
1440             'type': 'string'
1441         },
1442         'private_ips': {
1443             'type': 'string'
1444         },

```

```
1445     'instance_type': {
1446         'type': 'string'
1447     },
1448     'image': {
1449         'type': 'string'
1450     },
1451     'private_dns': {
1452         'type': 'string'
1453     },
1454     'image_name': {
1455         'type': 'string'
1456     },
1457     'instance_id': {
1458         'type': 'string'
1459     },
1460     'image_id': {
1461         'type': 'string'
1462     },
1463     'public_ips': {
1464         'type': 'string'
1465     },
1466     'state': {
1467         'type': 'string'
1468     },
1469     'root_device_name': {
1470         'type': 'string'
1471     },
1472     'key': {
1473         'type': 'string'
1474     },
1475     'group': {
1476         'type': 'string'
1477     },
1478     'flavor': {
1479         'type': 'string'
1480     },
1481     'availability': {
1482         'type': 'string'
1483     },
1484     'uuid': {
1485         'type': 'string'
1486     }
1487 }
1488 }
```

```
1491 eve_settings = {
1492     'MONGO_HOST': 'localhost',
1493     'MONGO_DBNAME': 'testing',
1494     'RESOURCE_METHODS': ['GET', 'POST', 'DELETE'],
1495     'BANDWIDTH_SAVER': False,
1496     'DOMAIN': {
1497         'profile': profile,
1498         'virtual_machine': virtual_machine,
1499         'kubernetes': kubernetes,
```

```

1500     'nic': nic,
1501     'virtual_compute_node': virtual_compute_node,
1502     'openstack_flavor': openstack_flavor,
1503     'azure-vm': azure_vm,
1504     'azure-size': azure_size,
1505     'openstack_vm': openstack_vm,
1506     'cluster': cluster,
1507     'computer': computer,
1508     'libcloud_image': libcloud_image,
1509     'user': user,
1510     'file': file,
1511     'deployment': deployment,
1512     'mapreduce': mapreduce,
1513     'group': group,
1514     'role': role,
1515     'virtual_directory': virtual_directory,
1516     'file_alias': file_alias,
1517     'virtual_cluster': virtual_cluster,
1518     'libcloud_flavor': libcloud_flavor,
1519     'batchjob': batchjob,
1520     'organization': organization,
1521     'container': container,
1522     'sshkey': sshkey,
1523     'stream': stream,
1524     'database': database,
1525     'default': default,
1526     'openstack_image': openstack_image,
1527     'azure_image': azure_image,
1528     'hadoop': hadoop,
1529     'compute_resource': compute_resource,
1530     'node_new': node_new,
1531     'filter': filter,
1532     'reservation': reservation,
1533     'replica': replica,
1534     'microservice': microservice,
1535     'var': var,
1536     'mesos-docker': mesos_docker,
1537     'libcloud_vm': libcloud_vm,
1538 },
1539 }

```

783 A.3. Contributing

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

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

789 A.4. Using the Cloudmesh REST Service

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

791 For example:

Listing A.3: profile

```

1 {
2   "profile": {

```

```

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

```

794 A.4.1. Element Definition

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

797 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
''')

```

798 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'}
}
}

```

799 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)

```

800 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

801 **A.6.1. Defining Elements for the REST Service**

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

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

805 where

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

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

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

811 **A.6.2. DOIT**

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

813 **A.6.3. Generating service**

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

816 **A.7. ABC**

817 README.rst

818 **B. CLOUDMESH REST**

819 **B.1. Prerequisites**

- 820 • mongo installation
- 821 • eve installation
- 822 • cloudmesh cmd5
- 823 • cloudmesh rest

824 **B.1.1. Install Mongo on OSX**

```
825 brew update
```

```
826 brew install mongod
```

```
827
```

```
828 # brew install mongod --with-openssl
```

829 **B.1.2. Install Mongo on OSX**

830 ASSIGNMET TO STUDENTS, PROVIDE PULL REQUEST WITH INSTRUCTIONS

831 **B.2. Introduction**

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

835 Thus you can either write your examples in yaml or in json. The resources are individually specified in a
836 directory. The directory can contain multiple resource files. We recommend that for each resource you define your
837 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 with:

```
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 two commands. First, we must concatenate all json specified resource examples into a single json file. We do this with:

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

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

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

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

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
```

879 **B.5.2. Managing Eve**

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

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

883 The first command copies the settings file to

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

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

887 Testing - OLD ~~~~~:

```
888 make setup      # install mongo and eve
889 make install    # installs the code and integrates it into cmd5
890 make deploy
891 make test
```

892 classes lessons rest.rst

893 **C. REST WITH EVE**

894 **C.1. Overview of REST**

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

898 RESTful applications use HTTP requests to:

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

902 Hence REST uses HTTP for the four CRUD operations:

- 903 • Create
- 904 • Read
- 905 • Update
- 906 • Delete

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

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

912 **C.2. REST and eve**

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

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

- 918 • virtual directories
- 919 • 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 targers in the makefile and replicate them one by one. Imprtaht 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 suffiient 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 should 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 usage 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. Let's assume the file is called sample.json, then the settings file will be called sample.settings.py. Having the evegenie program 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 which 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 hope 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 extension is to load the objects definitions or eve e.g. settings.py not from the class, but from a place in .cloudmesh. I propose to place the file at:

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

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

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 needs to be done within the same source tree. To simplify developing commands by a number of people and to have a dynamic plugin mechanism, we developed cmd5. It is a rewrite on our earlier efforts in cloudmesh and cmd3.

D.1. Resources

The source code for cmd5 is located in github:

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

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

or for pyenv, with:

```
pyenv virtualenv 2.7.13 ENV2
```

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

```
mkdir ~/github
cd ~/github

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

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

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

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

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

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

(ENV2) \$ cms

It will print the banner and enter the shell:

[illegible]

```
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

1049 D.3. Create your own Extension

1050 One of the most important features of CMD5 is its ability to extend it with new commands. This is done via
 1051 packaged name spaces. This is defined in the setup.py file of your enhancement. The best way to create an
 1052 enhancement is to take a look at the code in

- 1053 • <https://github.com/cloudmesh/extbar.git>

1054 Simply copy the code and modify the bar and foo commands to fit your needs.

1055 **make sure you are not copying the .git directory. Thus we** recommend that you copy it explicitly file by file or
 1056 directory by directory

1057 It is important that all objects are defined in the command itself and that no global variables be used in order to
 1058 allow each shell command to stand alone. Naturally you should develop API libraries outside of the cloudmesh
 1059 shell command and reuse them in order to keep the command code as small as possible. We place the command in:

1060 cloudmesh/ext/command/COMMANDNAME.py

1061 An example for the bar command is presented at:

- 1062 • <https://github.com/cloudmesh/extbar/blob/master/cloudmesh/ext/command/bar.py>

1063 It shows how simple the command definition is (bar.py):

```
1064 from __future__ import print_function
1065 from cloudmesh.shell.command import command
1066 from cloudmesh.shell.command import PluginCommand
1067
1068 class BarCommand(PluginCommand):
1069
1070     @command
1071     def do_bar(self, args, arguments):
1072         """
1073         ::
1074         Usage:
1075             command -f FILE
1076             command FILE
1077             command list
1078         This command does some useful things.
1079         Arguments:
1080             FILE    a file name
1081         Options:
1082             -f      specify the file
1083         """
1084         print(arguments)
```

1085 An important difference to other CMD solutions is that our commands can leverage (besides the standard
 1086 definition), docopts as a way to define the manual page. This allows us to use arguments as dict and use simple
 1087 if conditions to interpret the command. Using docopts has the advantage that contributors are forced to think
 1088 about the command and its options and document them from the start. Previously we used not to use docopts and
 1089 argparse was used. However we noticed that for some contributions the lead to commands that were either not
 1090 properly documented or the developers delivered ambiguous commands that resulted in confusion and wrong
 1091 usage by the users. Hence, we do recommend that you use docopts.

1092 The transformation is enabled by the @command decorator that takes also the manual page and creates a proper
 1093 help message for the shell automatically. Thus there is no need to introduce a separate help method as would
 1094 normally be needed in CMD.

D.4. Excercise

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 <i>software DEVELOPMENT</i> and <i>information technology OPERATION</i> S
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