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

This Draft document is available at
https://laszewski.github.io/papers/NIST.SP.1500-8-draft.pdf

May 29, 2017

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

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

NIST Special Publication 1500-6
Information Technology Laboratory

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

Version 0.1

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

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

May 29, 2017

U. S. Department of Commerce
Penny Pritzker, Secretary

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

National Institute of Standards and Technology (NIST) Special Publication 1500-8
87 pages (May 29, 2017)

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

Comments on this publication may be submitted to Wo Chang

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

REPORTS ON COMPUTER SYSTEMS TECHNOLOGY

The Information Technology Laboratory (ITL) at NIST promotes the U.S. economy and public welfare by providing technical leadership for the Nations 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). ITLs 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 ITLs 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 Interface Subgroup, led by Gregor von Laszewski (Indiana University).

NIST SP1500-8, Version 1 has been collaboratively authored by the NBD-PWG. As of the date of this publication, there are over six hundred NBD-PWG participants from industry, academia, and government. Federal agency participants include the National Archives and Records Administration (NARA), National Aeronautics and Space Administration (NASA), National Science Foundation (NSF), and the U.S. Departments of Agriculture, Commerce, Defense, Energy, Health and Human Services, Homeland Security, Transportation, Treasury, and Veterans Affairs. NIST acknowledges the specific contributions to this volume by the following NBD-PWG members.

Gregor von Laszewski
Indiana University

Wo Chang
National Institute of Standard

Fugang Wang
Indiana University

Badi Abdhul Wahid
Indiana University

Geoffrey C. Fox
Indiana University

Pratik Thakkar
Philips

Alicia Mara Zuniga-Alvarado
Consultant

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

ABSTRACT

This document summarizes interfaces that are instrumental for the interaction with Clouds, Containers, and HPC systems to manage virtual clusters to support the Big Data Reference Architecture. The REST paradigm is used to define these interfaces allowing easy integration and adoption by a wide variety of frameworks.

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 8, summarizes the work performed by the NBD-PWG to identify objects instrumental for the Big Data Reference Architecture (NBDRA) which is introduced in Volume 6.

KEYWORDS

NIST Big Data Reference Architecture; Interfaces, REST

EXECUTIVE SUMMARY

The NIST Big Data Interoperability Framework: Volume 8 document [6] was prepared by the NIST Big Data Public Working Group (NBD-PWG) Interface Subgroup to identify interfaces in support of the NIST Big Data Reference Architecture (NBDRA) The interfaces contain two different aspects:

- the definition of resources that are part of the NBDRA. These resources are formulated in Json format and can be integrated into a REST framework or an object based framework easily.
- the definition of simple interface use cases that allow us to illustrate the usefulness of the resources defined.

We categorized the resources in groups that are identified by the NBDRA set forward in Volume 6. While Volume 3 provides *application* oriented high level use cases the use cases defined in this document are subsets of them and focus on *interface* use cases. The interface use cases are not meant to be complete examples, but showcase why the resource has been defined. Hence, the interfaces use cases are, of course, only representative, and do not represent the entire spectrum of Big Data usage. All of the interfaces were openly discussed in the working group. Additions are welcome and we like to discuss your contributions in the group.
The NIST Big Data Interoperability Framework consists of nine volumes, each of which addresses a specific key topic, resulting from the work of the NBD-PWG. The eight volumes are:

- Volume 1: Definitions
- Volume 2: Taxonomies
- Volume 3: Use Cases and General Requirements
- Volume 4: Security and Privacy
- Volume 5: Architectures White Paper Survey
- Volume 6: Reference Architecture
- Volume 7: Standards Roadmap
- Volume 8: Interfaces
- Volume 9: Big Data Adoption and Modernization

The NIST Big Data Interoperability Framework will be released in three versions, which correspond to the three development stages of the NBD-PWG work. The three stages aim to achieve the following with respect to the NIST Big Data Reference Architecture (NBDRA).

Stage 1: Identify the high-level Big Data reference architecture key components, which are technology-, infrastructure-, and vendor-agnostic.

Stage 2: Define general interfaces between the NBDRA components.

Stage 3: Validate the NBDRA by building Big Data general applications through the general interfaces.

This document is targeting Stage 2 of the NBDRA. Coordination of the group is conducted on its Web page [7].

REFERENCES

[1] Cerberus. URL: http://docs.python-cerberus.org/.

[2] Eve Rest Service. Web Page. URL: http://python-eve.org/.

[3] Cloudmesh enhanced Eveengine. Github. URL: https://github.com/cloudmesh/cloudmesh.evegenie.

[4] Geoffrey C. Fox and Wo Chang. NIST Big Data Interoperability Framework: Volume 3, Use Cases and General Requirements. Special Publication (NIST SP) - 1500-3 1500-3, National INstitute of STandards, 100 Bureau Drive, Gaithersburg, MD 20899, October 2015. URL: http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.1500-3.pdf, doi:NIST.SP.1500-3.

[5] Internet2. eduPerson Object Class Specification (201602). Internet2 Middleware Architecture Committee for Education, Directory Working Group internet2-mace-dir-eduperson-201602, Internet2, March 2016. URL: http://software.internet2.edu/eduperson/internet2-mace-dir-eduperson-201602.html.

[6] Orit Levin, David Boyd, and Wo Chang. NIST Big Data Interoperability Framework: Volume 6, Reference Architecture. Special Publication (NIST SP) - 1500-6 1500-6, National Institute of Standards, 100 Bureau Drive, Gaithersburg, MD 20899, October 2015. URL: http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.1500-6.pdf, doi:NIST.SP.1500-6.

[7] NIST. Big Data Public Working Group (NBD-PWG). Web Page. URL: https://bigdatawg.nist.gov/.

[8] Arnab Roy, Mark Underwood, and Wo Chang. NIST Big Data Interoperability Framework: Volume 4, Security and Privacy. Special Publication (NIST SP) - 1500-4 1500-4, National Institute of Standards, 100 Bureau Drive, Gaithersburg, MD 20899, October 2015. URL: http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.1500-4.pdf, doi:NIST.SP.1500-4.

[9] Gregor von Laszewski. Cloudmesh client. github. URL: https://github.com/cloudmesh/client.

[10] Gregor von Laszewski and Wo Chang. NIST Big Data Interoperability Framework: Volume 8, Interfaces. Special Publication (NIST SP) - 1500-8 1500-8, National Institute of Standards, 100 Bureau Drive, Gaithersburg, MD 20899, October 2015. URL: https://raw.githubusercontent.com/cloudmesh/cloudmesh.rest/master/docs/NIST.SP.1500-8-draft.pdf, doi:NIST.SP.1500-8.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF OBJECTS

1. INTRODUCTION

The Volume 6 Reference Architecture document [6] provides a list of high-level reference architecture requirements and introduces the NIST Big Data Reference Architecture (NBDRA). Figure 1 depicts the high-level overview of the NBDRA.

To enable interoperability between the NBDRA components, a list of well-defined NBDRA interface is needed. These interfaces are documented in this Volume 8 [10]. To introduce them, we will follow the NBDRA and focus on interfaces that allow us to bootstrap the NBDRA. We will start the document with a summary of requirements that we will integrate into our specifications. Subsequently, each section will introduce a number of objects that build the core of the interface addressing a specific aspect of the NBDRA. We will showcase a selected number of *interface use cases* to outline how the specific interface can be used in a reference implementation of the NBDRA. Validation of this approach can be achieved while applying it to the application use cases that have been gathered in Volume 3 [4]. These application 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. Through this approach, we can facilitate subsequent analysis and comparison of the use cases. 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.

2. NBDRA INTERFACE REQUIREMENTS

Before we start outlining the specific interfaces, we introduce general requirements and explain how we define the interfaces while encouraging discussions.

2.1. High Level Requirements of the Interface Approach

First, we focus on the high-level requirements of the interface approach that we need to implement the reference architecture depicted in Figure 1.

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. Hence, we need to provide a methodology that allows extension of interfaces to adapt and leverage existing approaches, but also allows the interfaces to provide merit in easy specifications that assist the formulation and definition of the NBDRA.

2.1.2. Support of Plug-In Compute Infrastructure

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

2.1.3. Orchestration of Infrastructure and Services

As part of the use case collection we present in Volume 3 [4], it is obvious that we need to address the mechanism of preparing a suitable infrastructures for various use cases. As not every infrastructure is suited for every use case a custom infrastructure may be needed. As such we are not attempting to deliver a single deployed BDRA, but allow the setup of an infrastructure that satisfies the particular uses case. To achieve this task, we need to provision software stacks and services while orchestrate their deployment and leveraging infrastructures. 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

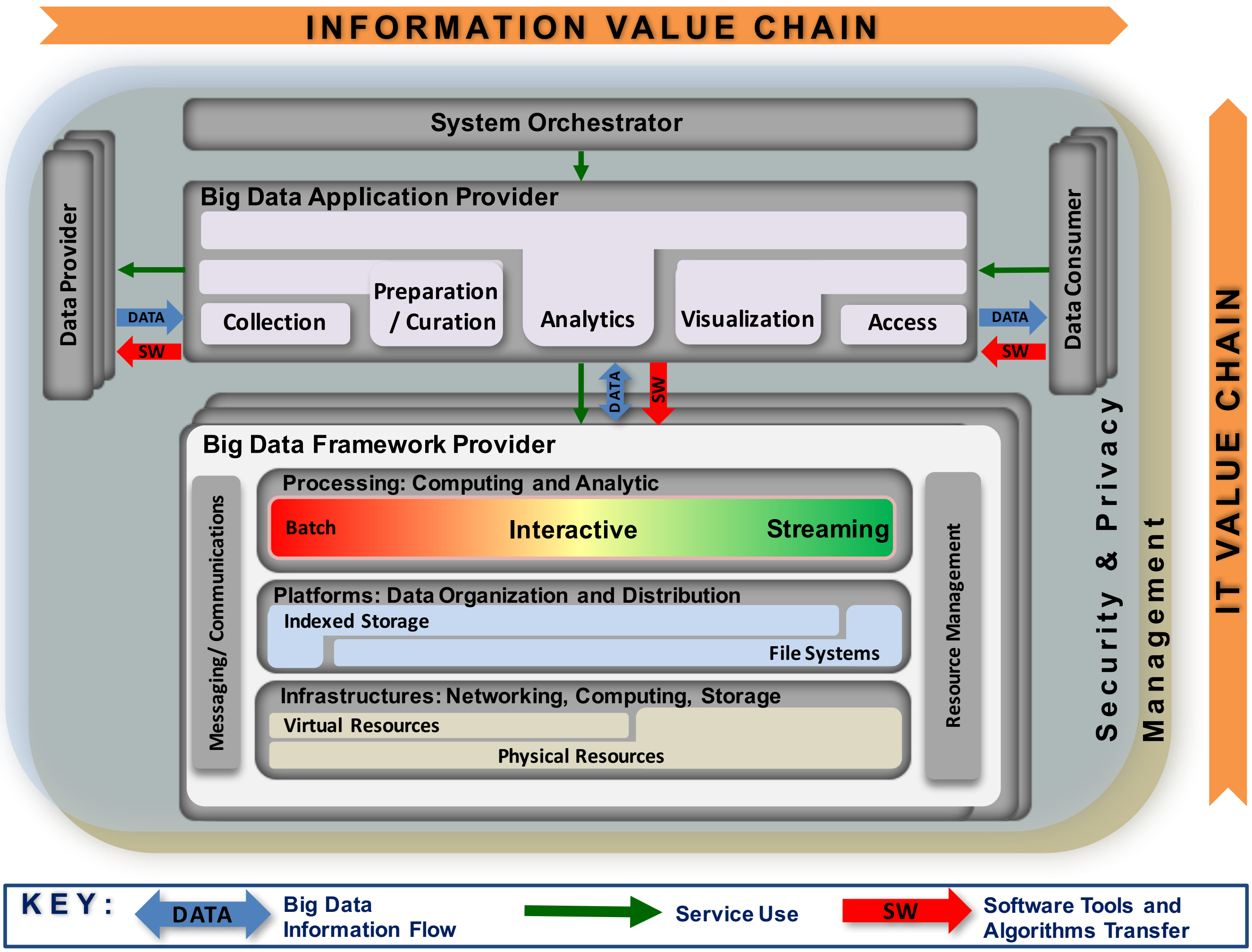

Figure 1: NIST Big Data Reference Architecture (NBDRA)

orchestration frameworks and services could therefore be leveraged even as part of the same framework 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) is used as the command center to interact in behalf of the BD Application Provider to orchestrate dataflow from Data Provider, carryout the BD application lifecycle with the help of the BD Framework Provider, and enable Data Consumer to consume Big Data processing results. An interface is needed to describe the interactions and to allow leveraging of experiment management frameworks in scripted fashion. We require a customization of parameters on several levels. On the highest level, we require high level- application motivated parameters to drive the orchestration of the experiment. On lower levels these high-level parameters may drive and create service level agreement augmented specifications and parameters that could even lead to the orchestration of infrastructure and services to satisfy experiment needs.

2.1.5. Reusability

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

2.1.6. Execution Workloads

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

2.1.7. Security and Privacy Fabric Requirements

Although the focus of this document is not security and privacy, which are documented in Volume 4 [8] of the NBDRA, we must make sure that the interfaces we define can be integrated into a secure reference architecture that supports secure execution, secure data transfer and privacy. Consequently, the interfaces that we define here can be augmented with frameworks and solutions that provide such mechanisms. Thus, we need to distinguish diverse requirement needs stemming from different use cases addressing security. To contrast that the security requirements between applications can drastically vary we use the following example. Although many of the interfaces and its objects to support physics big data application are similar to those in health care, they distinguish themselves from the integration of security interfaces and policies. While in physics the protection of the data is less of an issue, it is s stringent requirement in healthcare. Thus deriving architectural frameworks for both may use largely similar components, but while addressing security they are expected to be very different. In future versions of this document we intend to specifically address interfaces and their security. In the meanwhile we consider them as an advanced use case showcasing that the validity of the specifications introduced here is preserved even if security and privacy requirements vastly differ among application use cases.

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 2.2.1–2.2.6 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 (see Section 2.2.1);
- **Data Provider:** Introduces new data or information feeds into the Big Data system (see Section 2.2.2);
- **Data Consumer:** Includes end users or other systems that use the results of the Big Data Application Provider (see Section 2.2.3).
- **Big Data Application Provider:** Executes a data life cycle to meet security and privacy requirements as well as System Orchestrator-defined requirements (see Section 2.2.4);
- **Big Data Framework Provider:** Establishes a computing framework in which to execute certain transformation applications while protecting the privacy and integrity of data (see Section 2.2.5); and

Big Data Application Provider to Framework Provider Interface: Defines an interface between the application specification and the provider (see Section 2.2.6).

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 Provider Requirements

The Big Data Application Provider role executes a specific set of operations along the data life cycle to meet the requirements established by the System Orchestrator, as well as meeting security and privacy requirements. The Big Data Application Provider is the architecture component that encapsulates the business logic and functionality to be executed by the architecture. The interfaces to describe big data applications include interfaces for the various subcomponents including collections, preparation/curation, analytics, visualization, and access. Some if the interfaces used in these components can be reused from other interfaces introduced in other sections of this document. Where appropriate we will identify application specific interfaces and provide examples of them while focusing on a use case as identified in Volume 3 [4] 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.

2.2.5. 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.

2.2.5.1 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.

2.2.5.2 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.

2.2.5.3 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.

2.2.5.4 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 form Volume 3 [4] of this series.

2.2.5.5 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

2.2.5.6 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.

2.2.6. 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

Originally we used a full REST specification for defining the objets related to the BDRA. However, we found quickly that at this stage of the document it would introduce too complex of a notation framework. This would result in (a) a considerable increase in length of this document (b) a more complex framework reducing participation and (c) a more complex framework for developing a reference implementation. Thus we have decided in this version of the document to introduce a design concept by example that is used to automatically create a schema as well as a reference implementation.

3.2. Hybrid and Multiple Frameworks

It is obvious that we must be able to deal with hybrid and multiple frameworks to avoid vendor lock in. This is not only true for Clouds, containers, DevOps, but also other components of the NBDRA.

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 can than be taken and a schema can generated from them automatically. The schema is added to the Appendix A.1 of the document.

While focusing first on examples it allows us to speed up our design process and simplify discussions about the objects and interfaces Hence, we eliminate getting lost in complex 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 [9].

An example object will demonstrate our approach. The following object defines a JSON object representing a user (see 3.1).

Object 3.1: Example object specification

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

```
}
```

Such an object can be translated to a schema specification while introspecting the types of the original example.

All examples are managed in Github and links to them are automatically generated to be included into this document. A hyperlink is introduced in the Object specification and when clicking on the `</>` icon you will be redirected to the specification in github. The resulting schema object follows the Cerberus [1] specification and looks for our specific object we introduced earlier as follows:

```
profile = {
  'schema': {
    'username':    {'type': 'string'},
    'context:':    {'type': 'string'},
    'description': {'type': 'string'},
    'firstname':   {'type': 'string'},
    'lastname':    {'type': 'string'},
    'publickey':   {'type': 'string'},
    'email':       {'type': 'string'},
    'uuid':        {'type': 'string'}
  }
}
```

Defined objects can alse be embedded into other objects by using the *objectid* tag. This is later demonstrated between the profile and the user objects (see Objects 4.1 and 4.2).

references to objects

As mentioned before, the Appendix A.1 lists the schema that is automatically created from the definitions. More information about the creation can be found in Appendix B.

When using the objets we assume one can implement the typical CRUD actions using HTTP methods mapped as follows:

GET	profile	Retrieves a list of profile
GET	profile12	Retrieves a specific profile
POST	profile	Creates a new profile
PUT	profile12	Updates profile #12
PATCH	profile12	Partially updates profile #12
DELETE	profile12	Deletes profile #12

In our reference implementation these methods are provided automatically.

3.4. Interface Compliancy

Due to the easy extensibility of our objects and their implicit 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. A document will outline which objects and interfaces have been implemented.

Full and extended Compliance: These are interfaces that in addition to the full compliance also introduce additional interfaces and extend them. A document will be provided that lists the differences to the document defined here.

Such documents can than be forwarded to the subgroup for further discussion and for possible future modifications based on additional practical user feedback.

4. SPECIFICATION

As several objects are used across the NBDRA we have not organized them by component as introduced in Figure 1. Instead we have grouped the objects by functional use as depicted summarized in Figure 2.

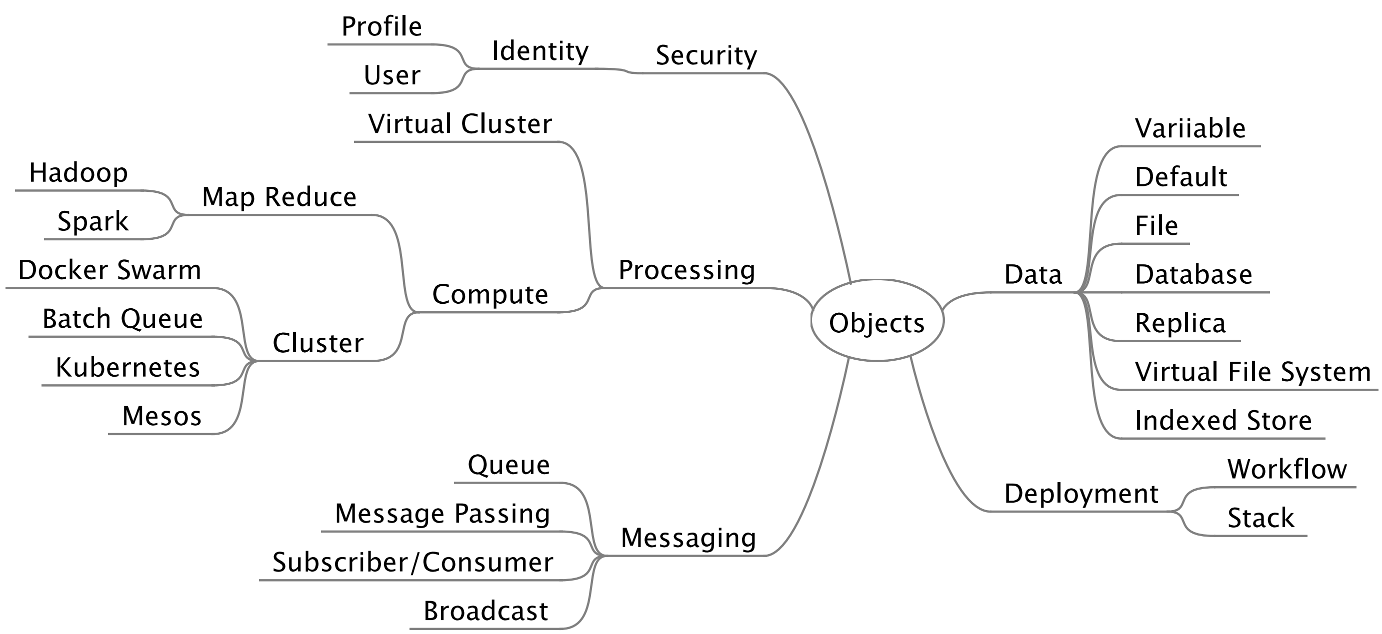

Figure 2: NIST Big Data Reference Architecture Interfaces

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 [5] 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 defines the identity of an individual. It contains name and e-mail information. It may have an optional uuid and/or use a unique e-mail to distinguish a user. Profiles are used to identify diffrent users.

Object 4.1: Profile

```
{
  "profile": {
    "description": "The Profile of a user",
    "uuid": "jshdjkdh...",
```

```
    "context:": "resource",
    "email": "laszewski@gmail.com",
    "firstname": "Gregor",
    "lastname": "von Laszewski",
    "username": "gregor",
    "publickey": "ssh ...."
  }
}
```

4.1.2. User

In contrast to the profile a user contains additional attributs that define the role of the user within the multi-user system. This associates different roles to individuals, these roles potentially have gradations of responsibility and privilege.

> 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.

Object 4.2: User

```
{
    "user": {
        "profile": "objectid:profile",
        "roles": ["admin"]
  }
}
```

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

Object 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.

Object 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.

Object 4.5: Role

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

4.2. Data

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

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

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

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

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.

Object 4.6: Var

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

4.2.2. Default

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

Object 4.7: Default

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

4.2.3. File

A file is a computer resource allowing to store data that is being processed. The interface to a file provides the mechanism to appropriately locate a file in a distributed system. Identification include the name, and endpoint, the checksum and the size. Additional parameters such as the lasst access time could be stored also. As such the Interface only describes the location of the file
The *file* object has *name*, *endpoint* (location), *size* in GB, MB, Byte, *checksum* for integrity check, and last *accessed* timestamp.

Object 4.8: File

```
{
    "file": {
        "name": "report.dat",
        "endpoint": "file://gregor@machine.edu:/data/report.dat",
        "checksum": {"sha256":"c01b39c7a35ccc ....... ebfeb45c69f08e17dfe3ef375a7b"},
        "accessed": "1.1.2017:05:00:00:EST",
        "created": "1.1.2017:05:00:00:EST",
        "modified": "1.1.2017:05:00:00:EST",
        "size": ["GB", "Byte"]
```

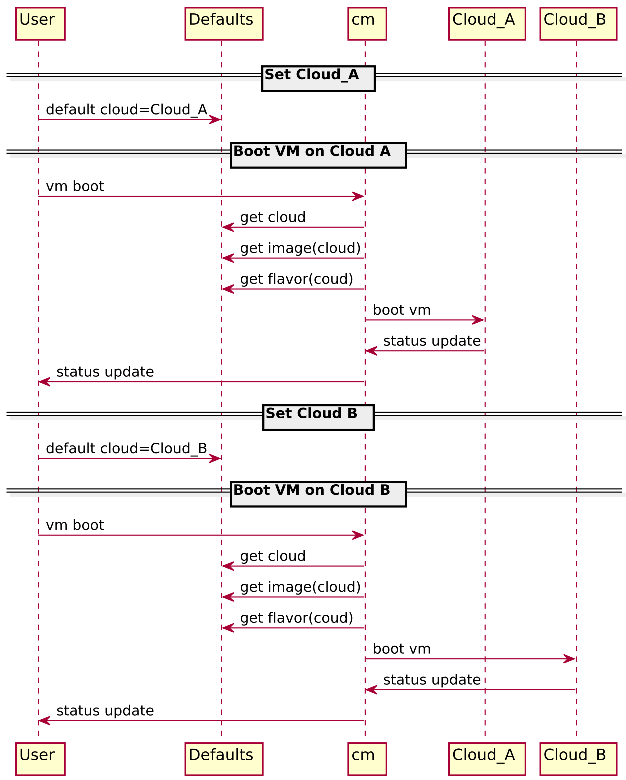

Figure 3: Booting a virtual machine from defaults

```
  }
}
```

4.2.4. File Alias

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 a user may want to refer to that file in a name space that is suitable for the users application.

Object 4.9: File alias

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

4.2.5. Replica

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

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

Object 4.10: Replica

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

4.2.6. Virtual Directory

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

Object 4.11: Virtual directory

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

4.2.7. Database

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

Object 4.12: Database

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

4.2.8. Stream

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

Object 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.

Object 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

Object 4.15: Openstack flavor

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

4.3.1.2 Openstack Image

Object 4.16: Openstack image

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

4.3.1.3 Openstack Vm

Object 4.17: Openstack vm

```
{
  "openstack_vm": {
    "username": "string",
    "vm_state": "string",
    "updated": "string",
    "hostId": "string",
    "availability_zone": "string",
    "terminated_at": "string",
```

```
    "image": "string",
    "floating_ip": "string",
    "diskConfig": "string",
    "key": "string",
    "flavor__id": "string",
    "user_id": "string",
    "flavor": "string",
    "static_ip": "string",
    "security_groups": "string",
    "volumes_attached": "string",
    "task_state": "string",
    "group": "string",
    "uuid": "string",
    "created": "string",
    "tenant_id": "string",
    "accessIPv4": "string",
    "accessIPv6": "string",
    "status": "string",
    "power_state": "string",
    "progress": "string",
    "image__id": "string",
    "launched_at": "string",
    "config_drive": "string"
  }
}
```

4.3.2. Azure

4.3.2.1 Azure Size

The size description of an azure vm

Object 4.18: Azure-size

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

4.3.2.2 Azure Image

Object 4.19: Azure-image

```json
{
  "azure_image": {
    "_uuid": "None",
    "driver": "libcloud",
    "extra": {
      "affinity_group": "",
      "category": "Public",
      "description": "Linux VM image with coreclr-x64-beta5-11624 installed to /opt/dnx. This image is based on Ubuntu 14.04 LTS, with prerequisites of CoreCLR installed. It also contains PartsUnlimited demo app which runs on the installed coreclr. The demo app is installed to /opt/demo. To run the demo, please type the command /opt/demo/Kestrel in a terminal window. The website is listening on port 5004. Please enable or map a endpoint of HTTP port 5004 for your azure VM.",
      "location": "East Asia;Southeast Asia;Australia East;Australia Southeast;Brazil South;North Europe;West Europe;Japan East;Japan West;Central US;East US;East US 2; North Central US;South Central US;West US",
      "media_link": "",
      "os": "Linux",
      "vm_image": "False"
    },
    "id": "03f55de797f546a1b29d1....",
    "name": "CoreCLR x64 Beta5 (11624) with PartsUnlimited Demo App on Ubuntu Server 14.04 LTS"
  }
}
```

4.3.2.3 Azure Vm

An Azure virtual machine

Object 4.20: Azure-vm

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

```
    "instance_size": "string",
    "instance_name": "string",
    "public_ips": "string",
    "media_link": "string"
  }
}
```

4.4. HPC

4.4.1. Batch Job

Object 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.

Figure 4 illustrates the process for allocating and provisioning a virtual cluster. The user defines the desired physical properties of the cluster such CPU, memory, disk and the intended configuration (such as software, users, etc). After requesting the stack to be deployed, cloudmesh allocates the machines as desired by matching the desired properties with the available images and booting. The stack definition is then parsed then evaluated to provision the cluster.

Object 4.22: Cluster

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

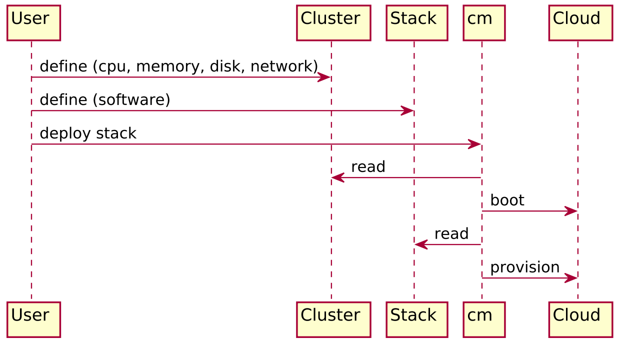

Figure 4: Allocating and provisioning a virtual cluster

```
      "aws",
      "azure",
      "eucalyptus"
    ]
  }
}
```

4.5.2. New Cluster

Object 4.23: Cluster

```
{
  "virtual_cluster": {
    "name": "myvc",
      "frontend": 0,
      "nodes": [
          { "count": 3,
            "node": "objectid:virtual_machine"
          }
      ]
  },
  "virtual_machine" :{
    "name": "vm1",
    "ncpu": 2,
    "RAM": "4G",
    "disk": "40G",
    "nics": ["objectid:nic"
    ],
    "OS": "Ubuntu-16.04",
    "loginuser": "ubuntu",
    "status": "active",
    "metadata":{
    },
    "authorized_keys": [
      "objectid:sshkey"
    ]
  },
```

```
    "sshkey": {
      "comment": "string",
      "source": "string",
      "uri": "string",
      "value": "ssh-rsa AAA......",
      "fingerprint": "string, unique"
    },
    "nic": {
      "name": "eth0",
      "type": "ethernet",
      "mac": "00:00:00:11:22:33",
      "ip": "123.123.1.2",
      "mask": "255.255.255.0",
      "broadcast": "123.123.1.255",
      "gateway": "123.123.1.1",
      "mtu": 1500,
      "bandwidth": "10Gbps"
    }
  }
```

4.5.3. Compute Resource

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

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

Object 4.24: Compute resource

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

4.5.4. Computer

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

Object 4.25: Computer

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

4.5.5. Compute Node

A node is composed of multiple components:

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

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

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

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

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

Object 4.26: Node

```
{
  "node_new": {
    "authorized_keys": [
      "ssh-rsa AAAA...",
      "ssh-ed25519 AAAA...",
      "...etc"
    ],
    "name": "example-001",
    "external_ip": "",
    "loginuser": "root",
    "create_external_ip": true,
    "internal_ip": "",
    "memory": 2048,
    "owner": "",
    "cores": 2,
    "users": {
      "name": "hello",
      "groups": [
        "wheel"
      ]
    },
    "disk": 80,
    "security_groups": [
      {
        "ingress": "0.0.0.0/32",
        "egress": "0.0.0.0/32",
        "ports": [
          22,
```

```
          443,
          80,
          8080
        ],
        "protocols": [
          "tcp",
          "udp"
        ]
      }
    ],
    "ssh_keys": [
      {
        "to": ".ssh/id_rsa",
        "password": {
          "decrypt": "gpg",
          "from": "yaml",
          "file": "secrets.yml.gpg",
          "key": "users.hello.ssh[0]"
        },
        "method": "ed25519",
        "ssh_keygen": true
      },
      {
        "to": ".ssh/testing",
        "password": {
          "decrypt": "gpg",
          "from": "yaml",
          "file": "secrets.yml.gpg",
          "key": "users.hello.ssh[1]"
        },
        "bits": 4098,
        "method": "rsa",
        "ssh_keygen": true
      },
      {
        "decrypt": "gpg",
        "from": "secrets/ssh/hello/copied.gpg",
        "ssh_keygen": false,
        "to": ".ssh/copied"
      }
    ]
  }
}
```

4.5.6. Virtual Cluster

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

Object 4.27: Virtual cluster

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

4.5.7. Virtual Compute node

Object 4.28: Virtual compute node

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

4.5.8. Virtual Machine

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

Object 4.29: Virtual machine

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

```
      "objectid:sshkey"
    ]
  }
}
```

4.5.9. Mesos

Refine

Object 4.30: Mesos

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

4.6. Containers

4.6.1. Container

This defines *container* object.

Object 4.31: Container

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

4.6.2. Kubernetes

REFINE

Object 4.32: Kubernetes

```json
{
  "kubernetes": {
    "kind": "List",
    "items": [
      {
        "kind": "None",
        "metadata": {
          "name": "127.0.0.1"
        },
        "status": {
          "capacity": {
            "cpu": "4"
          },
          "addresses": [
            {
              "type": "LegacyHostIP",
              "address": "127.0.0.1"
            }
          ]
        }
      },
      {
        "kind": "None",
        "metadata": {
          "name": "127.0.0.2"
        },
        "status": {
          "capacity": {
            "cpu": "8"
          },
          "addresses": [
            {
              "type": "LegacyHostIP",
              "address": "127.0.0.2"
            },
            {
              "type": "another",
              "address": "127.0.0.3"
            }
          ]
        }
      }
    ],
    "users": [
      {
        "name": "myself",
        "user": "gregor"
      },
      {
```

```
        "name": "e2e",
        "user": {
          "username": "admin",
          "password": "secret"
        }
      }
    ]
  }
}
```

4.7. Deployment

4.7.1. Deployment

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

Object 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.

Object 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.

Object 4.35: Mapreduce function

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

```
    }
  }
}
```

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.

Object 4.36: Mapreduce noop

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

4.8.2. Hadoop

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

Object 4.37: Hadoop

```
{
  "hadoop": {
    "deployers": {
      "ansible": "git://github.com/cloudmesh_roles/hadoop"
    },
    "requires": {
      "java": {
        "implementation": "OpenJDK",
        "version": "1.8",
        "zookeeper": "TBD",
        "supervisord": "TBD"
      }
    },
    "parameters": {
      "num_resourcemanagers": 1,
      "num_namenodes": 1,
      "use_yarn": false,
      "use_hdfs": true,
      "num_datanodes": 1,
      "num_historyservers": 1,
      "num_journalnodes": 1
    }
  }
}
```

4.9. Security

4.9.1. Key

Object 4.38: Key

```
{
  "hadoop": {
    "deployers": {
      "ansible": "git://github.com/cloudmesh_roles/hadoop"
    },
    "requires": {
      "java": {
        "implementation": "OpenJDK",
        "version": "1.8",
        "zookeeper": "TBD",
        "supervisord": "TBD"
      }
    },
    "parameters": {
      "num_resourcemanagers": 1,
      "num_namenodes": 1,
      "use_yarn": false,
      "use_hdfs": true,
      "num_datanodes": 1,
      "num_historyservers": 1,
      "num_journalnodes": 1
    }
  }
}
```

4.10. Microservice

4.10.1. Microservice

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

necessity of local direct attached storage. Mimd model to storage Kubernetis, mesos can not spin up ?

Takes time to spin them up and coordinate them. While setting up environment takes more than 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.

Object 4.39: Microservice

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

4.10.2. Reservation

Object 4.40: Reservation

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

4.10.3. Accounting

As in big data applications and systems considerable amount of resources are used an accounting system must be present either on the server side or on the application and user side to allow checking of balances. Due to the potential heterogeneous nature of the services used existing accounting frameworks may not be present to dela with this issue. E.g. we see potentially the use of multiple accounting systems with different scales of accuracy information feedback rates. For example, if the existing accounting system informs the user only hours after she has started a job this could pose a significant risk because charging is started immediately. While making access to big data infrastructure and services more simple, the user or application may underestimate the overall cost projected by the implementation of the big data reference architecture.

Object 4.41: Accounting

```
{
  "accounting_resource": {
    "description": "The Description of a resource that we apply accounting to",
    "uuid": "unique uuid for this resource",
    "name": "the name of the resource",
    "charge": "1.1 * parameter1 + 3.1 * parameter2",
    "parameters": {"parameter1": 1.0,
                   "parameter2": 1.0},
    "unites": {"parameter1": "GB",
               "parameter2": "cores"},
    "user": "username",
      "group": "groupname",
    "account": "accountname"
  }
}
```

Object 4.42: Account

```
{
  "account": {
      "description": "The Description of the account",
```

```
        "uuid": "unique uuid for this resource",
        "name": "the name of the account",
        "startDate": "10/10/2017:00:00:00",
        "endDate": "10/10/2017:00:00:00",
        "status": "one of active, suspended, closed",
        "balance": 1.0,
        "user": ["username"],
        "group": ["groupname"]
  }
}
```

4.10.3.1 Usecase: Accounting Service

Figure ?? depicts a possible accounting service that allows an administrator to register a variety of resources to an account for a user. The services that are than invoked by the user can than consume the resource and are charged accordingly.

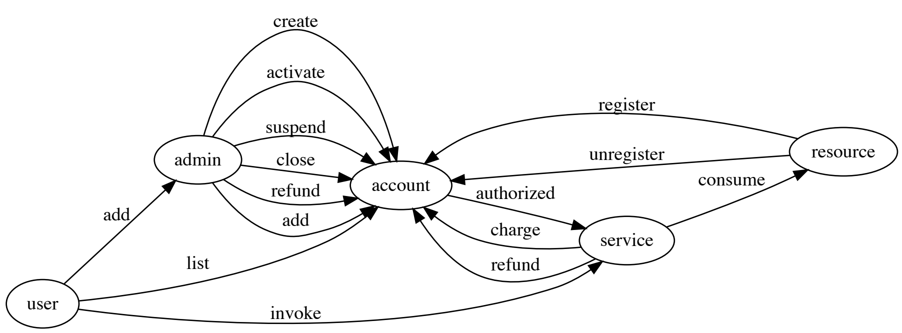

Figure 5: Create Resource

4.11. Network

We are looking for volunteers to contribute here.

5. STATUS CODES AND ERROR RESPONSES

In case of an error or a successful response, the response header contains a HTTP code. The response body usually contains

- the HTTP response code
- an accompanying message for the HTTP response code
- a field or object where the error occurred

http://www.restapitutorial.com/httpstatuscodes.html https://en.wikipedia.org/wiki/List_of_HTTP_status_codes http://www.ietf.org/assignments/http-status-code http-status-codes.xml https://tools.ietf.org/html/rfc7231

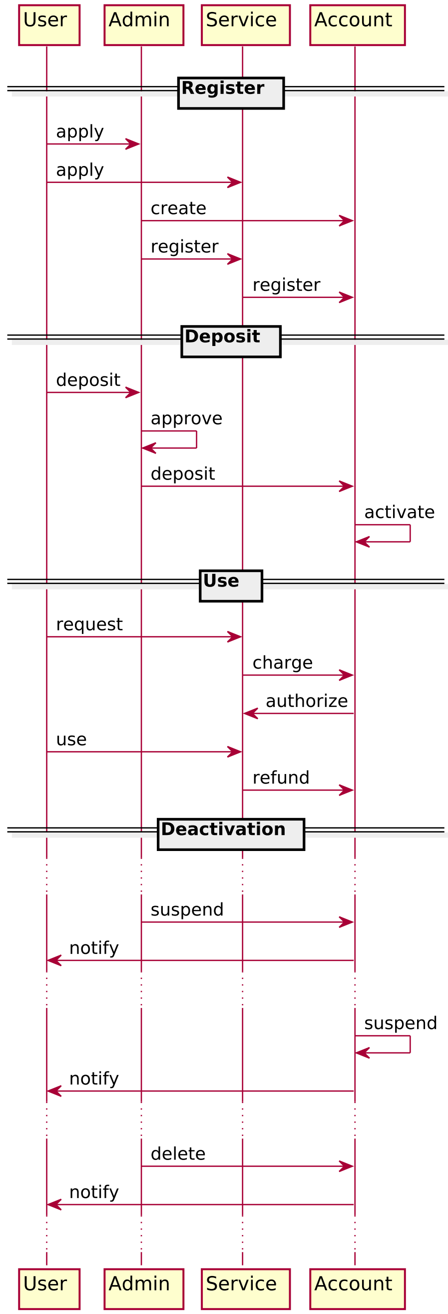

Figure 6: Accounting

5.1. Acronyms and Terms

The following acronyms and terms are used in the paper

ACID Atomicity, Consistency, Isolation, Durability

API Application Programming Interface

ASCII American Standard Code for Information Interchange

Table 1: HTTP response codes

HTTP	response Description code
200	*OK* success code, for GET or HEAD request.
201	*Created* success code, for POST request.
204	*No Content* success code, for DELETE request.
300	The value returned when an external ID exists in more than one record.
304	The request content has not changed since a specified date and time.
400	The request could not be understood.
401	The session ID or OAuth token used has expired or is invalid.
403	The request has been refused.
404	The requested resource could not be found.
405	The method specified in the Request-Line isnt allowed for the resource specified in the URI.
415	The entity in the request is in a format thats not supported by the specified method.

BASE Basically Available, Soft state, Eventual consistency

Container see `http://csrc.nist.gov/publications/drafts/800-180/sp800-180_draft.pdf`

Cloud Computing
the practice of using a network of remote servers hosted on the Internet to store, manage, and process data, rather than a local server or a personal computer. See `http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf`

DevOps A clipped compound of *software DEVelopment* and *information technology OPerationS*

Deployment The action of installing software on resources.

HTTP HyperText Transfer Protocol HTTPS HTTP Secure

Hybrid Cloud See `http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf`

IaaS Infrastructure as a Service SaaS Software as a Service

ITL Information Technology Laboratory

Microservice Architecture
Is an approach to build applications based on many smaller modular services. Each module supports a specific goal and uses a simple, well-defined interface to communicate with other sets of services.

NBD-PWG NIST Big Data Public Working Group

NBDRA NIST Big Data Reference Architecture

NBDRAI NIST Big Data Reference Architecture Interface

NIST National Institute of Standards

OS Operating System

REST REpresentational State Transfer

Replica A duplicate of a file on another resource in order to avoid costly transfer costs in case of frequent access.

Serverless Computing
Serverless computing specifies the paragdigm of function as a service (FaaS). It is a cloud computing code execution model in which a cloud provider manages the function deployment and utilization while clients can utilize them. The charge model is based on execution of the function rather than the cost to manage and host the virtual machine or container.

Software Stack A set of programs and services that are installed on a resource in order to support applications.

Virtual Filesysyem
An abstraction layer on top of a a distributed physical file system to allow easy access to the files by the user or application.

Virtual Machine
A virtual machine is a software computer that, like a physical computer, runs an operating system and applications. The virtual machine is comprised of a set of specification and configuration files and is backed by the physical resources of a host.

Virtual Cluster
A virtual cluster is a software cluster that integrate either virtual machines, containers or physical resources into an agglomeration of compute resources. A virtual cluster allows user sto authenticate and authorize to the virtual compute nodes tu utilize them for calculations. Optional high level services that can be deployed on a virtual cluster may simplify interaction with the virtual cluster or provide higher level services.

Workflow the sequence of processes or tasks

WWW World Wide Web

A. APPENDIX

A.1. Schema

Listing A.1 showcases the schema generated from the objects defined in this document.

Object A.1: schema

```
profile = {
    'schema': {
        'username': {
            'type': 'string'
        },
        'context:': {
            'type': 'string'
        },
        'description': {
            'type': 'string'
        },
        'firstname': {
            'type': 'string'
        },
        'lastname': {
            'type': 'string'
        },
        'publickey': {
            'type': 'string'
        },
        'email': {
            'type': 'string'
        },
        'uuid': {
            'type': 'string'
        }
    }
}

stream = {
    'schema': {
        'attributes': {
            'type': 'dict',
            'schema': {
                'rate': {
                    'type': 'integer'
                },
                'limit': {
                    'type': 'integer'
                }
            }
        },
        'name': {
            'type': 'string'
        },
```

```
        'format': {
            'type': 'string'
        }
    }
}

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

virtual_compute_node = {
```

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

deployment = {
    'schema': {
        'cluster': {
            'type': 'list',
            'schema': {
                'type': 'dict',
                'schema': {
                    'id': {
                        'type': 'string'
                    }
                }
            }
        },
        'stack': {
            'type': 'dict',
            'schema': {
                'layers': {
```

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

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

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

cluster = {
    'schema': {
        'provider': {
            'type': 'list',
            'schema': {
                'type': 'string'
            }
        },
        'endpoint': {
            'type': 'dict',
            'schema': {
                'passwd': {
                    'type': 'string'
                },
                'url': {
                    'type': 'string'
                }
            }
        },
        'name': {
            'type': 'string'
        },
        'label': {
            'type': 'string'
        }
    }
}

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

662

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

mesos_docker = {
    'schema': {
        'container': {
            'type': 'dict',
            'schema': {
                'docker': {
                    'type': 'dict',
                    'schema': {
                        'credential': {
                            'type': 'dict',
                            'schema': {
                                'secret': {
                                    'type': 'string'
                                },
                                'principal': {
                                    'type': 'string'
                                }
                            }
                        },
                        'image': {
                            'type': 'string'
                        }
                    }
                },
                'type': {
                    'type': 'string'
                }
            }
        },
        'mem': {
            'type': 'float'
        },
        'args': {
            'type': 'list',
            'schema': {
                'type': 'string'
            }
        },
        'cpus': {
```

```
                'type': 'float'
            },
            'instances': {
                'type': 'integer'
            },
            'id': {
                'type': 'string'
            }
        }
}

file = {
    'schema': {
        'endpoint': {
            'type': 'string'
        },
        'name': {
            'type': 'string'
        },
        'created': {
            'type': 'string'
        },
        'checksum': {
            'type': 'dict',
            'schema': {
                'sha256': {
                    'type': 'string'
                }
            }
        },
        'modified': {
            'type': 'string'
        },
        'accessed': {
            'type': 'string'
        },
        'size': {
            'type': 'list',
            'schema': {
                'type': 'string'
            }
        }
    }
}

reservation = {
    'schema': {
        'start_time': {
            'type': 'list',
            'schema': {
```

```
                'type': 'string'
            }
        },
        'hosts': {
            'type': 'string'
        },
        'description': {
            'type': 'string'
        },
        'end_time': {
            'type': 'list',
            'schema': {
                'type': 'string'
            }
        }
    }
}

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

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

```
}

mapreduce_function = {
    'schema': {
        'name': {
            'type': 'string'
        },
        'outputs': {
            'type': 'dict',
            'schema': {
                'key': {
                    'type': 'string'
                }
            }
        },
        'args': {
            'type': 'list',
            'schema': {
                'type': 'dict',
                'schema': {
                    'argument': {
                        'type': 'string'
                    }
                }
            }
        },
        'systemBuildInputs': {
            'type': 'list',
            'schema': {
                'type': 'string'
            }
        },
        'source': {
            'type': 'string'
        },
        'install': {
            'type': 'dict',
            'schema': {
                'description': {
                    'type': 'string'
                },
                'script': {
                    'type': 'string'
                }
            }
        },
        'eval': {
            'type': 'dict',
            'schema': {
                'description': {
```

```
                    'type': 'string'
                },
                'script': {
                    'type': 'string'
                }
            }
        },
        'buildInputs': {
            'type': 'list',
            'schema': {
                'type': 'string'
            }
        },
        'description': {
            'type': 'string'
        }
    }
}

virtual_cluster = {
    'schema': {
        'nodes': {
            'type': 'list',
            'schema': {
                'type': 'objectid',
                'data_relation': {
                    'resource': 'virtual_machine',
                    'field': '_id',
                    'embeddable': True
                }
            }
        },
        'frontend': {
            'type': 'objectid',
            'data_relation': {
                'resource': 'virtual_machine',
                'field': '_id',
                'embeddable': True
            }
        },
        'name': {
            'type': 'string'
        }
    }
}

libcloud_flavor = {
    'schema': {
        'uuid': {
            'type': 'string'
```

```
        },
        'price': {
            'type': 'string'
        },
        'ram': {
            'type': 'string'
        },
        'bandwidth': {
            'type': 'string'
        },
        'flavor_id': {
            'type': 'string'
        },
        'disk': {
            'type': 'string'
        },
        'cpu': {
            'type': 'string'
        }
    }
}

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

sshkey = {
    'schema': {
        'comment': {
            'type': 'string'
        },
        'source': {
```

```
                'type': 'string'
            },
            'uri': {
                'type': 'string'
            },
            'value': {
                'type': 'string'
            },
            'fingerprint': {
                'type': 'string'
            }
    }
}

mapreduce_noop = {
    'schema': {
        'name': {
            'type': 'string'
        },
        'description': {
            'type': 'string'
        }
    }
}

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

AzureNodeExtra = {
    'schema': {
        'ssh_port': {
```

```
            'type': 'string'
        },
        'instance_size': {
            'type': 'string'
        },
        'remote_desktop_port': {
            'type': 'string'
        },
        'ex_cloud_service_name': {
            'type': 'string'
        },
        'power_state': {
            'type': 'string'
        },
        'instance_endpoints': {
            'type': 'string'
        }
    }
}

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

node = {
    'schema': {
        'private_ips': {
            'type': 'list',
            'schema': {
                'type': 'string'
            }
        },
        'extra': {
            'type': 'dict',
            'schema': {}
        },
        'created_at': {
            'type': 'string'
        },
        'driver': {
```

```
                'type': 'string'
            },
            'state': {
                'type': 'string'
            },
            'public_ips': {
                'type': 'list',
                'schema': {
                    'type': 'string'
                }
            },
            'id': {
                'type': 'string'
            },
            'name': {
                'type': 'string'
            }
        }
    }

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

```
        'RAM': {
            'type': 'string'
        },
        'ncpu': {
            'type': 'integer'
        },
        'loginuser': {
            'type': 'string'
        },
        'disk': {
            'type': 'string'
        },
        'OS': {
            'type': 'string'
        },
        'metadata': {
            'type': 'dict',
            'schema': {}
        }
    }
}

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

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

nic = {
    'schema': {
```

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

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

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

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

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

ec2NodeExtra = {
    'schema': {
        'product_codes': {
            'type': 'string'
        },
        'tags': {
            'type': 'list',
            'schema': {
                'type': 'string'
            }
        },
        'network_interfaces': {
            'type': 'list',
            'schema': {
                'type': 'string'
            }
        },
        'groups': {
            'type': 'list',
            'schema': {
                'type': 'string'
            }
        },
        'block_device_mapping': {
            'type': 'string'
        }
```

```
    }
}

libcloud_image = {
    'schema': {
        'username': {
            'type': 'string'
        },
        'status': {
            'type': 'string'
        },
        'updated': {
            'type': 'string'
        },
        'description': {
            'type': 'string'
        },
        'owner_alias': {
            'type': 'string'
        },
        'kernel_id': {
            'type': 'string'
        },
        'hypervisor': {
            'type': 'string'
        },
        'ramdisk_id': {
            'type': 'string'
        },
        'state': {
            'type': 'string'
        },
        'created': {
            'type': 'string'
        },
        'image_id': {
            'type': 'string'
        },
        'image_location': {
            'type': 'string'
        },
        'platform': {
            'type': 'string'
        },
        'image_type': {
            'type': 'string'
        },
        'is_public': {
            'type': 'string'
        },
```

```
        'owner_id': {
            'type': 'string'
        },
        'architecture': {
            'type': 'string'
        },
        'virtualization_type': {
            'type': 'string'
        },
        'uuid': {
            'type': 'string'
        }
    }
}

user = {
    'schema': {
        'profile': {
            'type': 'objectid',
            'data_relation': {
                'resource': 'profile',
                'field': '_id',
                'embeddable': True
            }
        },
        'roles': {
            'type': 'list',
            'schema': {
                'type': 'string'
            }
        }
    }
}

GCENodeExtra = {
    'schema': {
        'status': {
            'type': 'string'
        },
        'kind': {
            'type': 'string'
        },
        'machineType': {
            'type': 'string'
        },
        'description': {
            'type': 'string'
        },
        'zone': {
            'type': 'string'
```

```
        },
        'deprecated': {
            'type': 'string'
        },
        'image': {
            'type': 'string'
        },
        'disks': {
            'type': 'string'
        },
        'tags_fingerprint': {
            'type': 'string'
        },
        'name': {
            'type': 'string'
        },
        'boot_disk': {
            'type': 'string'
        },
        'selfLink': {
            'type': 'string'
        },
        'scheduling': {
            'type': 'string'
        },
        'canIpForward': {
            'type': 'string'
        },
        'serviceAccounts': {
            'type': 'string'
        },
        'metadata': {
            'type': 'string'
        },
        'creationTimestamp': {
            'type': 'string'
        },
        'id': {
            'type': 'string'
        },
        'statusMessage': {
            'type': 'string'
        },
        'networkInterfaces': {
            'type': 'string'
        }
    }
}

container = {
```

```
    ’schema’: {
        ’ip’: {
            ’type’: ’string’
        },
        ’endpoint’: {
            ’type’: ’string’
        },
        ’name’: {
            ’type’: ’string’
        },
        ’memoryGB’: {
            ’type’: ’integer’
        },
        ’label’: {
            ’type’: ’string’
        }
    }
}

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

node_new = {
    ’schema’: {
        ’authorized_keys’: {
            ’type’: ’list’,
            ’schema’: {
                ’type’: ’string’
            }
        },
        ’name’: {
            ’type’: ’string’
        },
        ’external_ip’: {
            ’type’: ’string’
        },
        ’memory’: {
            ’type’: ’integer’
        },
        ’create_external_ip’: {
            ’type’: ’boolean’
        },
```

```
        'internal_ip': {
            'type': 'string'
        },
        'loginuser': {
            'type': 'string'
        },
        'owner': {
            'type': 'string'
        },
        'cores': {
            'type': 'integer'
        },
        'disk': {
            'type': 'integer'
        },
        'ssh_keys': {
            'type': 'list',
            'schema': {
                'type': 'dict',
                'schema': {
                    'from': {
                        'type': 'string'
                    },
                    'decrypt': {
                        'type': 'string'
                    },
                    'ssh_keygen': {
                        'type': 'boolean'
                    },
                    'to': {
                        'type': 'string'
                    }
                }
            }
        },
        'security_groups': {
            'type': 'list',
            'schema': {
                'type': 'dict',
                'schema': {
                    'ingress': {
                        'type': 'string'
                    },
                    'egress': {
                        'type': 'string'
                    },
                    'ports': {
                        'type': 'list',
                        'schema': {
                            'type': 'integer'
```

```
                        }
                    },
                    'protocols': {
                        'type': 'list',
                        'schema': {
                            'type': 'string'
                        }
                    }
                }
            }
        },
        'users': {
            'type': 'dict',
            'schema': {
                'name': {
                    'type': 'string'
                },
                'groups': {
                    'type': 'list',
                    'schema': {
                        'type': 'string'
                    }
                }
            }
        }
    }
}

batchjob = {
    'schema': {
        'output_file': {
            'type': 'string'
        },
        'group': {
            'type': 'string'
        },
        'job_id': {
            'type': 'string'
        },
        'script': {
            'type': 'string'
        },
        'cmd': {
            'type': 'string'
        },
        'queue': {
            'type': 'string'
        },
        'cluster': {
            'type': 'string'
```

```
        },
        'time': {
            'type': 'string'
        },
        'path': {
            'type': 'string'
        },
        'nodes': {
            'type': 'string'
        },
        'dir': {
            'type': 'string'
        }
    }
}

account = {
    'schema': {
        'status': {
            'type': 'string'
        },
        'startDate': {
            'type': 'string'
        },
        'endDate': {
            'type': 'string'
        },
        'description': {
            'type': 'string'
        },
        'uuid': {
            'type': 'string'
        },
        'user': {
            'type': 'list',
            'schema': {
                'type': 'string'
            }
        },
        'group': {
            'type': 'list',
            'schema': {
                'type': 'string'
            }
        },
        'balance': {
            'type': 'float'
        },
        'name': {
            'type': 'string'
```

```
        }
    }
}

libcloud_vm = {
    'schema': {
        'username': {
            'type': 'string'
        },
        'status': {
            'type': 'string'
        },
        'root_device_type': {
            'type': 'string'
        },
        'private_ips': {
            'type': 'string'
        },
        'instance_type': {
            'type': 'string'
        },
        'image': {
            'type': 'string'
        },
        'private_dns': {
            'type': 'string'
        },
        'image_name': {
            'type': 'string'
        },
        'instance_id': {
            'type': 'string'
        },
        'image_id': {
            'type': 'string'
        },
        'public_ips': {
            'type': 'string'
        },
        'state': {
            'type': 'string'
        },
        'root_device_name': {
            'type': 'string'
        },
        'key': {
            'type': 'string'
        },
        'group': {
            'type': 'string'
```

```
        },
        'flavor': {
            'type': 'string'
        },
        'availability': {
            'type': 'string'
        },
        'uuid': {
            'type': 'string'
        }
    }
}

database = {
    'schema': {
        'endpoint': {
            'type': 'string'
        },
        'protocol': {
            'type': 'string'
        },
        'name': {
            'type': 'string'
        }
    }
}

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

openstack_image = {
    'schema': {
        'status': {
            'type': 'string'
        },
        'username': {
            'type': 'string'
        },
        'updated': {
```

```
                'type': 'string'
            },
            'uuid': {
                'type': 'string'
            },
            'created': {
                'type': 'string'
            },
            'minDisk': {
                'type': 'string'
            },
            'progress': {
                'type': 'string'
            },
            'minRam': {
                'type': 'string'
            },
            'os_image_size': {
                'type': 'string'
            },
            'metadata': {
                'type': 'dict',
                'schema': {
                    'instance_uuid': {
                        'type': 'string'
                    },
                    'image_location': {
                        'type': 'string'
                    },
                    'image_state': {
                        'type': 'string'
                    },
                    'instance_type_memory_mb': {
                        'type': 'string'
                    },
                    'user_id': {
                        'type': 'string'
                    },
                    'description': {
                        'type': 'string'
                    },
                    'kernel_id': {
                        'type': 'string'
                    },
                    'instance_type_name': {
                        'type': 'string'
                    },
                    'ramdisk_id': {
                        'type': 'string'
                    },
```

686

```
                'instance_type_id': {
                    'type': 'string'
                },
                'instance_type_ephemeral_gb': {
                    'type': 'string'
                },
                'instance_type_rxtx_factor': {
                    'type': 'string'
                },
                'image_type': {
                    'type': 'string'
                },
                'network_allocated': {
                    'type': 'string'
                },
                'instance_type_flavorid': {
                    'type': 'string'
                },
                'instance_type_vcpus': {
                    'type': 'string'
                },
                'instance_type_root_gb': {
                    'type': 'string'
                },
                'base_image_ref': {
                    'type': 'string'
                },
                'instance_type_swap': {
                    'type': 'string'
                },
                'owner_id': {
                    'type': 'string'
                }
            }
        }
    }
}

OpenStackNodeExtra = {
    'schema': {
        'vm_state': {
            'type': 'string'
        },
        'addresses': {
            'type': 'list',
            'schema': {
                'type': 'string'
            }
        },
        'availability_zone': {
```

```
                'type': 'string'
            },
            'service_name': {
                'type': 'string'
            },
            'userId': {
                'type': 'string'
            },
            'imageId': {
                'type': 'string'
            },
            'volumes_attached': {
                'type': 'string'
            },
            'task_state': {
                'type': 'string'
            },
            'disk_config': {
                'type': 'string'
            },
            'power_state': {
                'type': 'string'
            },
            'progress': {
                'type': 'string'
            },
            'metadata': {
                'type': 'list',
                'schema': {
                    'type': 'string'
                }
            },
            'updated': {
                'type': 'string'
            },
            'hostId': {
                'type': 'string'
            },
            'key_name': {
                'type': 'string'
            },
            'flavorId': {
                'type': 'string'
            },
            'password': {
                'type': 'string'
            },
            'access_ip': {
                'type': 'string'
            },
```

```
        'access_ipv6': {
            'type': 'string'
        },
        'created': {
            'type': 'string'
        },
        'fault': {
            'type': 'string'
        },
        'uri': {
            'type': 'string'
        },
        'tenantId': {
            'type': 'string'
        },
        'config_drive': {
            'type': 'string'
        }
    }
}

mapreduce = {
    'schema': {
        'function': {
            'type': 'dict',
            'schema': {
                'source': {
                    'type': 'string'
                },
                'args': {
                    'type': 'dict',
                    'schema': {}
                }
            }
        },
        'fault_tolerant': {
            'type': 'boolean'
        },
        'data': {
            'type': 'dict',
            'schema': {
                'dest': {
                    'type': 'string'
                },
                'source': {
                    'type': 'string'
                }
            }
        },
        'backend': {
```

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

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

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

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

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

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

```
        },
        'accessIPv6': {
            'type': 'string'
        },
        'power_state': {
            'type': 'string'
        },
        'progress': {
            'type': 'string'
        },
        'image__id': {
            'type': 'string'
        },
        'launched_at': {
            'type': 'string'
        },
        'config_drive': {
            'type': 'string'
        },
        'username': {
            'type': 'string'
        },
        'updated': {
            'type': 'string'
        },
        'hostId': {
            'type': 'string'
        },
        'floating_ip': {
            'type': 'string'
        },
        'static_ip': {
            'type': 'string'
        },
        'key': {
            'type': 'string'
        },
        'flavor__id': {
            'type': 'string'
        },
        'group': {
            'type': 'string'
        },
        'task_state': {
            'type': 'string'
        },
        'created': {
            'type': 'string'
        },
        'tenant_id': {
```

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

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

hadoop = {
    'schema': {
        'deployers': {
            'type': 'dict',
            'schema': {
                'ansible': {
                    'type': 'string'
                }
            }
        },
        'requires': {
            'type': 'dict',
            'schema': {
                'java': {
                    'type': 'dict',
                    'schema': {
                        'implementation': {
                            'type': 'string'
                        },
                        'version': {
                            'type': 'string'
                        },
                        'zookeeper': {
                            'type': 'string'
                        },
                        'supervisord': {
```

```
                            'type': 'string'
                        }
                    }
                }
            }
        },
        'parameters': {
            'type': 'dict',
            'schema': {
                'num_resourcemanagers': {
                    'type': 'integer'
                },
                'num_namenodes': {
                    'type': 'integer'
                },
                'use_yarn': {
                    'type': 'boolean'
                },
                'num_datanodes': {
                    'type': 'integer'
                },
                'use_hdfs': {
                    'type': 'boolean'
                },
                'num_historyservers': {
                    'type': 'integer'
                },
                'num_journalnodes': {
                    'type': 'integer'
                }
            }
        }
    }
}

accounting_resource = {
    'schema': {
        'account': {
            'type': 'string'
        },
        'group': {
            'type': 'string'
        },
        'description': {
            'type': 'string'
        },
        'parameters': {
            'type': 'dict',
            'schema': {
                'parameter1': {
```

```
                    'type': 'float'
                },
                'parameter2': {
                    'type': 'float'
                }
            }
        },
        'uuid': {
            'type': 'string'
        },
        'charge': {
            'type': 'string'
        },
        'unites': {
            'type': 'dict',
            'schema': {
                'parameter1': {
                    'type': 'string'
                },
                'parameter2': {
                    'type': 'string'
                }
            }
        },
        'user': {
            'type': 'string'
        },
        'name': {
            'type': 'string'
        }
    }
}



eve_settings = {
    'MONGO_HOST': 'localhost',
    'MONGO_DBNAME': 'testing',
    'RESOURCE_METHODS': ['GET', 'POST', 'DELETE'],
    'BANDWIDTH_SAVER': False,
    'DOMAIN': {
        'profile': profile,
        'stream': stream,
        'azure_image': azure_image,
        'virtual_compute_node': virtual_compute_node,
        'deployment': deployment,
        'azure-size': azure_size,
        'cluster': cluster,
        'computer': computer,
        'mesos-docker': mesos_docker,
```

```
        'file': file,
        'reservation': reservation,
        'microservice': microservice,
        'virtual_directory': virtual_directory,
        'mapreduce_function': mapreduce_function,
        'virtual_cluster': virtual_cluster,
        'libcloud_flavor': libcloud_flavor,
        'group': group,
        'sshkey': sshkey,
        'mapreduce_noop': mapreduce_noop,
        'role': role,
        'AzureNodeExtra': AzureNodeExtra,
        'var': var,
        'node': node,
        'virtual_machine': virtual_machine,
        'kubernetes': kubernetes,
        'nic': nic,
        'openstack_flavor': openstack_flavor,
        'azure-vm': azure_vm,
        'ec2NodeExtra': ec2NodeExtra,
        'libcloud_image': libcloud_image,
        'user': user,
        'GCENodeExtra': GCENodeExtra,
        'container': container,
        'file_alias': file_alias,
        'node_new': node_new,
        'batchjob': batchjob,
        'account': account,
        'libcloud_vm': libcloud_vm,
        'database': database,
        'default': default,
        'openstack_image': openstack_image,
        'OpenStackNodeExtra': OpenStackNodeExtra,
        'mapreduce': mapreduce,
        'compute_resource': compute_resource,
        'filter': filter,
        'replica': replica,
        'openstack_vm': openstack_vm,
        'organization': organization,
        'hadoop': hadoop,
        'accounting_resource': accounting_resource,
    },
}
```

B. CLOUDMESH REST

Cloudmesh Rest is a refernce implementattion for the NBDRA. It allows to define automatically a REST service based on the objects specified by the NBDRA document. In collaboration with other cloudmesh components it allows easy interaction with hybrid clouds and the creation of user managed big data services.

B.1. Prerequistis

The preriquisits for Cloudmesh REST are Python 2.7.13 or 3.6.1 it can easily be installed on a variety of systems (at this time we have only tried ubuntu greater 16.04 and OSX Sierra. However, it would naturally be possible to also port it to Windows. The instalation instruction in this document are not complete and we recommend to refer to the cloudmesh manuals which are under development. The goal will be to make the instalation (after your system is set up for developing python) as simple as

```
    pip install cloudmesh.rest
```

B.2. REST Service

With the cloudmesh REST framework it is easy to create REST services while defining the resources via example json objects. This is achieved while leveraging the python eve [2] and a modified version of python evengine [3].

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"
  }
}
```

here we define an object called profile, that contains a number of attributes and values. The type of the values are automatically determined. All json specifications are contained in a directory and can easily be converted into a valid schema for the eve rest service by executing the commands

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

This will create a the configuration `all.settings.py` that can be used to start an eve service

Once the schema has defined, cloudmesh specifies defaults for managing a sample data base that is coupled with the REST service. We use mongodb which could be placed on a sharded mongo service.

B.3. Limitations

The current implementation is a demonstration and showcases that it is easy to generate a fully functioning REST service based on the specifications provided in this document. However, it is expected that scalability, distribution of services, and other advanced options need to be addrassed based on application requirements.

C. CONTRIBUTING

We invite you to contribute to this paper and its discussion to improve it. Improvements can be done with pull requests. We suggest you do *small* individual changes to a single subsection and object rather than large changes as this allows us to integrate the changes individually and comment on your contribution via github. Once contributed we will appropriately acknowledge you either as contributor or author. Please discuss with us how we best acknowledge you.

C.1. 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.

C.2. Conversion to Word

We found that it is most convenient to manage the draft document on github. Currently the document is located at:

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

Managing the document in github has provided us with the advantage that a reference implementation can be automatically derived from the specified objects. Also it is easy to contribute as all text is written in ASCII while using LaTeXsyntax to allow for formating in PDF.
Contributions can be mades as follows:

Contributions with git pull requests : You can fork the repository, make modifications and create a pull request that we than review and integrate

Contribution with direct access : Cloudmesh.rest developers have direct access to the repository. If you are a frequent contributor to the document and are familiar with github we can grant you access. However, we do prefer pull requests as this minimizes our administrative overhead to avoid issues with git

Contributing ASCII sections with git issues : You can identify the version of the document, specify the section and line numbers you want to modify and include the new text. We will integrate and address these issues ASAP. Issues can be submitted at `https://github.com/cloudmesh/cloudmesh.rest/issues`