NIST Special Publication 1500-8

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

NIST Big Data Public Working Group
Standards Roadmap Subgroup

Draft Version 1, Revision 1
2017/08/10, 13:13:32
https://bigdatawg.nist.gov/V2_output_docs.php

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

NIST Special Publication 1500-6
Information Technology Laboratory

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

Draft Version 1

Revision 1

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

This draft publication is available free of charge from:
`https://bigdatawg.nist.gov/V2_output_docs.php`
The current unreleased working draft is available in github

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

2017/08/10, 13:13:32

U. S. Department of Commerce
Wilbur L. Ross, Jr., Secretary

National Institute of Standards and Technology
Dr. Kent Rochford, Acting Under Secretary of Commerce for Standards and Technology and Acting NIST Director

National Institute of Standards and Technology (NIST) Special Publication 1500-8

99 pages (2017/08/10)

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

REQUEST FOR CONTRIBUTIONS

The NIST Big Data Public Working Group (NBD-PWG) requests contributions to this draft Version 2 of the NIST Big Data Interoperability Framework (NBDIF): Volume 6, Reference Architecture. All contributions are welcome, especially comments or additional content for the current draft.

The NBD-PWG is actively working to complete Version 2 of the set of NBDIF documents. The goals of Version 2 are to enhance the Version 1 content, define general interfaces between the NIST Big Data Reference Architecture (NBDRA) components by aggregating low-level interactions into high-level general interfaces, and demonstrate how the NBDRA can be used. To contribute to this document, please follow the steps below as soon as possible but no later than September 21, 2017.

1. Obtain your user ID by registering as a user of the NBD-PWG Portal (`https://bigdatawg.nist.gov/newuser.php`)
2. Record comments and/or additional content in one of the following methods:
   (a) **TRACK CHANGES:** make edits to and comments on the text directly into this Word document using track changes
   (b) **COMMENT TEMPLATE:** capture specific edits using the Comment Template (`http://bigdatawg.nist.gov/_uploadfiles/SP1500-1-to-7_comment_template.docx`), which includes space for section number, page number, comment, and text edits
3. Submit the edited file from either method above by uploading the document to the NBD-PWG portal (`https://bigdatawg.nist.gov/upload.php`). Use the User ID (obtained in step 1) to upload documents. Alternatively, the edited file (from step 2) can be emailed to SP1500comments@nist.gov with the volume number in the subject line (e.g., Edits for Volume 1).
4. Attend the weekly virtual meetings on Tuesdays for possible presentation and discussion of your submission. Virtual meeting logistics can be found at `https://bigdatawg.nist.gov/program.php`

Please be as specific as possible in any comments or edits to the text. Specific edits include, but are not limited to, changes in the current text, additional text further explaining a topic or explaining a new topic, additional references, or comments about the text, topics, or document organization.

The comments and additional content will be reviewed by the subgroup co-chair responsible for the volume in question. Comments and additional content may be presented and discussed by the NBD-PWG during the weekly virtual meetings on Tuesday.

Three versions are planned for the NBDIF set of documents, with Versions 2 and 3 building on the first. Further explanation of the three planned versions, and the information contained therein, is included in Section 1 of each NBDIF document.

Please contact Wo Chang (wchang@nist.gov) with any questions about the feedback submission process.

Big Data professionals are always welcome to join the NBD-PWG to help craft the work contained in the volumes of the NBDIF. Additional information about the NBD-PWG can be found at `http://bigdatawg.nist.gov`. Information about the weekly virtual meetings on Tuesday can be found at `https://bigdatawg.nist.gov/program.php`.

REPORTS ON COMPUTER SYSTEMS TECHNOLOGY

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

ABSTRACT

This document summarizes interfaces that are instrumental for the interaction with Clouds, Containers, and HPC systems to manage virtual clusters to support the NIST Big Data Reference Architecture (NBDRA). The Representational State Transfer (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 extensive datasets, primarily in the characteristics of volume, variety, velocity, and/or variability. While opportunities exist with Big Data, the data characteristics 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 (NBDIF)* series of volumes. This volume, Volume 8, uses the work performed by the NBD-PWG to identify objects instrumental for the NIST Big Data Reference Architecture (NBDRA) which is introduced in the *NBDIF: Volume 6, Reference Architecture*.

KEYWORDS

Adoption, barriers, market maturity, project maturity, organizational maturity, implementation, system modernization, interfaces

ACKNOWLEDGEMENTS

This document reflects the contributions and discussions by the membership of the NBD-PWG, co-chaired by Wo Chang (NIST ITL), Bob Marcus (ET-Strategies), and Chaitan Baru (San Diego Supercomputer Center; National Science Foundation). For all versions, the Subgroups were led by the following people: Nancy Grady (SAIC), Natasha Balac (SDSC), and Eugene Luster (R2AD) for the Definitions and Taxonomies Subgroup; Geoffrey Fox (Indiana University) and Tsegereda Beyene (Cisco Systems) for the Use Cases and Requirements Subgroup; Arnab Roy (Fujitsu), Mark Underwood (Krypton Brothers; Synchrony Financial), and Akhil Manchanda (GE) for the Security and Privacy Subgroup; David Boyd (InCadence Strategic Solutions), Orit Levin (Microsoft), Don Krapohl (Augmented Intelligence), and James Ketner (AT&T) for the Reference Architecture Subgroup; and Russell Reinsch (Center for Governmentt Interoperability), David Boyd (InCadence Strategic Solutions), Carl Buffington (Vistronix), and Dan McClary (Oracle), for the Standards Roadmap Subgroup.

The editors for this document were the following:

- **Version 1:** This volume resulted from Stage 2 work and was not part of the Version 1 scope.
- **Version 2:** Gregor von Laszewski (Indiana University) and Wo Chang (NIST)

Laurie Aldape (Energetics Incorporated) provided editorial assistance across all NBDIF volumes.

NIST SP1500-1, Version 2 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, Census, Health and Human Services, Homeland Security, Transportation, Treasury, and Veterans Affairs. NIST would like to acknowledge the specific contributions[1] to this volume, during Version 1 and/or 2 activities, 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 Maria Zuniga-Alvarado
Consultant

Robert C. Whetsel
DISA/NBIS

[1] "Contributors" are members of the NIST Big Data Public Working Group who dedicated great effort to prepare and gave substantial time on a regular basis to research and development in support of this document.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF OBJECTS

EXECUTIVE SUMMARY

The *NIST Big Data Interoperability Framework (NBDIF): Volume 8, Reference Architecture Interfaces* document [10] 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.

The resources were categorized in groups that are identified by the NBDRA set forward in the *NBDIF: Volume 6, Reference Architecture* document. While the *NBDIF: Volume 3, Use Cases and Requirements* document 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 NBDIF 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 NBDIF 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 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].

1. INTRODUCTION

1.1. Background

There is broad agreement among commercial, academic, and government leaders about the remarkable potential of Big Data to spark innovation, fuel commerce, and drive progress. Big Data is the common term used to describe the deluge of data in today's networked, digitized, sensor-laden, and information-driven world. The availability of vast data resources carries the potential to answer questions previously out of reach, including the following:

- How can a potential pandemic reliably be detected early enough to intervene?
- Can new materials with advanced properties be predicted before these materials have ever been synthesized?
- How can the current advantage of the attacker over the defender in guarding against cyber-security threats be reversed?

There is also broad agreement on the ability of Big Data to overwhelm traditional approaches. The growth rates for data volumes, speeds, and complexity are outpacing scientific and technological advances in data analytics, management, transport, and data user spheres. Despite widespread agreement on the inherent opportunities and current limitations of Big Data, a lack of consensus on some important fundamental questions continues to confuse potential users and stymie progress. These questions include the following:

- How is Big Data defined?
- What attributes define Big Data solutions?
- What is new in Big Data?
- What is the difference between Big Data and bigger data that has been collected for years?
- How is Big Data different from traditional data environments and related applications?
- What are the essential characteristics of Big Data environments?
- How do these environments integrate with currently deployed architectures?
- What are the central scientific, technological, and standardization challenges that need to be addressed to accelerate the deployment of robust, secure Big Data solutions?

Within this context, on March 29, 2012, the White House announced the Big Data Research and Development Initiative. The initiative's goals include helping to accelerate the pace of discovery in science and engineering, strengthening national security, and transforming teaching and learning by improving analysts' ability to extract knowledge and insights from large and complex collections of digital data.

Six federal departments and their agencies announced more than $200 million in commitments spread across more than 80 projects, which aim to significantly improve the tools and techniques needed to access, organize, and draw conclusions from huge volumes of digital data. The initiative also challenged industry, research universities, and nonprofits to join with the federal government to make the most of the opportunities created by Big Data.

Motivated by the White House initiative and public suggestions, the National Institute of Standards and Technology (NIST) has accepted the challenge to stimulate collaboration among industry professionals to further the secure and effective adoption of Big Data. As one result of NIST's Cloud and Big Data Forum

held on January 15âĂŞ17, 2013, there was strong encouragement for NIST to create a public working group for the development of a Big Data Standards Roadmap. Forum participants noted that this roadmap should define and prioritize Big Data requirements, including interoperability, portability, reusability, extensibility, data usage, analytics, and technology infrastructure. In doing so, the roadmap would accelerate the adoption of the most secure and effective Big Data techniques and technology.

On June 19, 2013, the NIST Big Data Public Working Group (NBD-PWG) was launched with extensive participation by industry, academia, and government from across the nation. The scope of the NBD-PWG involves forming a community of interests from all sectorsâĂŤincluding industry, academia, and governmentâĂŤwith the goal of developing consensus on definitions, taxonomies, secure reference architectures, security and privacy, andâĂŤfrom theseâĂŤa standards roadmap. Such a consensus would create a vendor-neutral, technology- and infrastructure-independent framework that would enable Big Data stakeholders to identify and use the best analytics tools for their processing and visualization requirements on the most suitable computing platform and cluster, while also allowing added value from Big Data service providers.

The NIST Big Data Interoperability Framework (NBDIF) will be released in three versions, which correspond to the three 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.

On September 16, 2015, seven NBDIF Version 1 volumes were published (`http://bigdatawg.nist.gov/V1_output_docs.php`), each of which addresses a specific key topic, resulting from the work of the NBD-PWG. The seven volumes are as follows:

- 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

Currently, the NBD-PWG is working on Stage 2 with the goals to enhance the Version 1 content, define general interfaces between the NBDRA components by aggregating low-level interactions into high-level general interfaces, and demonstrate how the NBDRA can be used. As a result of the Stage 2 work, the following two additional NBDIF volumes have been identified.

- Volume 8, Reference Architecture Interfaces
- Volume 9, Adoption and Modernization

Version 2 of the NBDIF volumes, resulting from Stage 2 work, can be downloaded from the NBD-PWG website (`https://bigdatawg.nist.gov/V2_output_docs.php`). Potential areas of future work for each volume during Stage 3 are highlighted in Section 1.5 of each volume. The current effort documented in this volume reflects concepts developed within the rapidly evolving field of Big Data.

2. INTRODUCTION - GREGOR

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 2 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.

2.1. Scope and Objectives of the Reference Architecture Subgroup

Reference architectures provide "an authoritative source of information about a specific subject area that guides and constrains the instantiations of multiple architectures and solutions." Reference architectures generally serve as a foundation for solution architectures and may also be used for comparison and alignment of instantiations of architectures and solutions.

The goal of the NBD-PWG Reference Architecture Subgroup is to develop an open reference architecture for Big Data that achieves the following objectives:

- Provides a common language for the various stakeholders
- Encourages adherence to common standards, specifications, and patterns
- Provides consistent methods for implementation of technology to solve similar problem sets
- Illustrates and improves understanding of the various Big Data components, processes, and systems, in the context of a vendor- and technology-agnostic Big Data conceptual model
- Provides a technical reference for U.S. government departments, agencies, and other consumers to understand, discuss, categorize, and compare Big Data solutions
- Facilitates analysis of candidate standards for interoperability, portability, reusability, and extendibility

The NBDRA is a high-level conceptual model crafted to serve as a tool to facilitate open discussion of the requirements, design structures, and operations inherent in Big Data. The NBDRA is intended to facilitate the understanding of the operational intricacies in Big Data. It does not represent the system architecture of a specific Big Data system, but rather is a tool for describing, discussing, and developing system-specific architectures using a common framework of reference. The model is not tied to any specific vendor products, services, or reference implementation, nor does it define prescriptive solutions that inhibit innovation.

The NBDRA does not address the following:

- Detailed specifications for any organizationâĂŹs operational systems

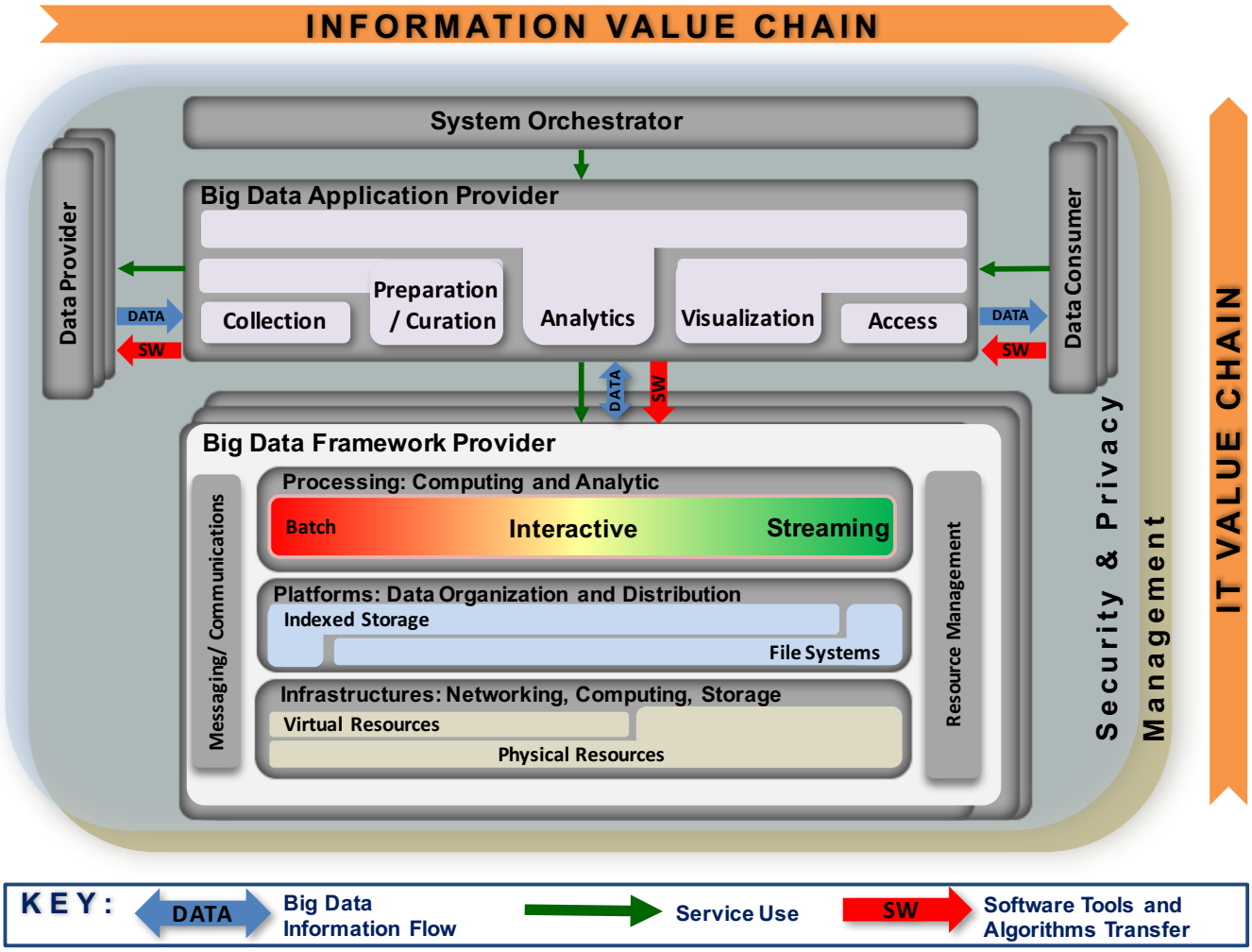

Figure 1: NIST Big Data Reference Architecture (NBDRA)

- Detailed specifications of information exchanges or services
- Recommendations or standards for integration of infrastructure products

The goals of the Subgroup will be realized throughout the three planned phases of the NBD-PWG work, as outlined in Section 1.1 **??**.

2.2. Report Production

The NBDIF: Volume 8, References Architecture Implementation is one of nine volumes, whose overall aims are to define and prioritize Big Data requirements, including interoperability, portability, reusability, extensibility, data usage, analytic techniques, and technology infrastructure in order to support secure and effective adoption of Big Data. The overall goals of this volume are to define and specify interfaces to implement the Big Data Reference Architecture. This volume arose from discussions during the weekly NBD-PWG conference calls. Topics included in this volume began to take form in Phase 2 of the NBD-PWG work. This

volume represents the groundwork for additional content planned for Phase 3. During the discussions, the NBD-PWG identified the need to specify a variety of interfaces including:

TBD

To enable interoperability between the NBDRA components, a list of well-defined NBDRA interfaces is needed. These interfaces are documented in this volume [10]. To introduce them, the NBDRA structure will be followed, focusing on interfaces that allow bootstrapping of the NBDRA. The document begins with a summary of requirements that will be integrated 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. A selected number of *interface use cases* will be showcased 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 the *NBDIF: Volume 3, USe Cases and Requirements* [4] document. These application use cases have considerably contributed towards the design of the NBDRA. Hence the expectation is that: (a) the interfaces can be used to help implement a Big Data architecture for a specific use case; and (b) the proper implementation . This approach can facilitate subsequent analysis and comparison of the use cases.

This document is expected to 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. To achieve technical and high quality document content, this document will go through public comments period along with NIST internal review.

NBDIF: Volume 8, Interfaces is one of nine volumes, whose overall aims are to define and specify interfaces to implement the Big Data Reference Architecture.

173 The NBDIF: Volume 8, interafces from discussions during the weekly NBD-PWG 174 conference calls. Topics included in this volume began to take form in Phase 2 of the NBD-PWG work and this 175 volume represents the groundwork for additional content planned for Phase 3.

176 During the discussions, the NBD-PWG identified the need to specify a variety of interfaces including:

TBD

include the list here. The Standards Roadmap Subgroup will continue to develop these and possibly other topics during Phase 3. The current version reflects the breadth of knowledge of the Subgroup members. The publicâĂŹs participation in Phase 3 of the NBD-PWG work is encouraged. To achieve technical and high quality document content, this document will go through public comments period along with NIST internal review.

2.3. Report Structure

TBD

2.4. Future Work on this Volume

A number of topics have not been discussed and clarified sufficiently to be included in Version 2. Topics that remain to be addressed in Version 3 of this document include the following:

TBD

3. NBDRA INTERFACE REQUIREMENTS

The development of a Big Data reference architecture requires a thorough understanding of current techniques, issues, and concerns. To this end, the NBD-PWG collected use cases to gain an understanding of current applications of Big Data, conducted a survey of reference architectures to understand commonalities within Big Data architectures in use, developed a taxonomy to understand and organize the information collected, and reviewed existing technologies and trends relevant to Big Data. The results of these NBD-PWG activities were used in the development of the NBDRA (Figure 2) and the interfaces presented herein. Detailed descriptions of these activities can be found in the other volumes of the *NBDIF*.

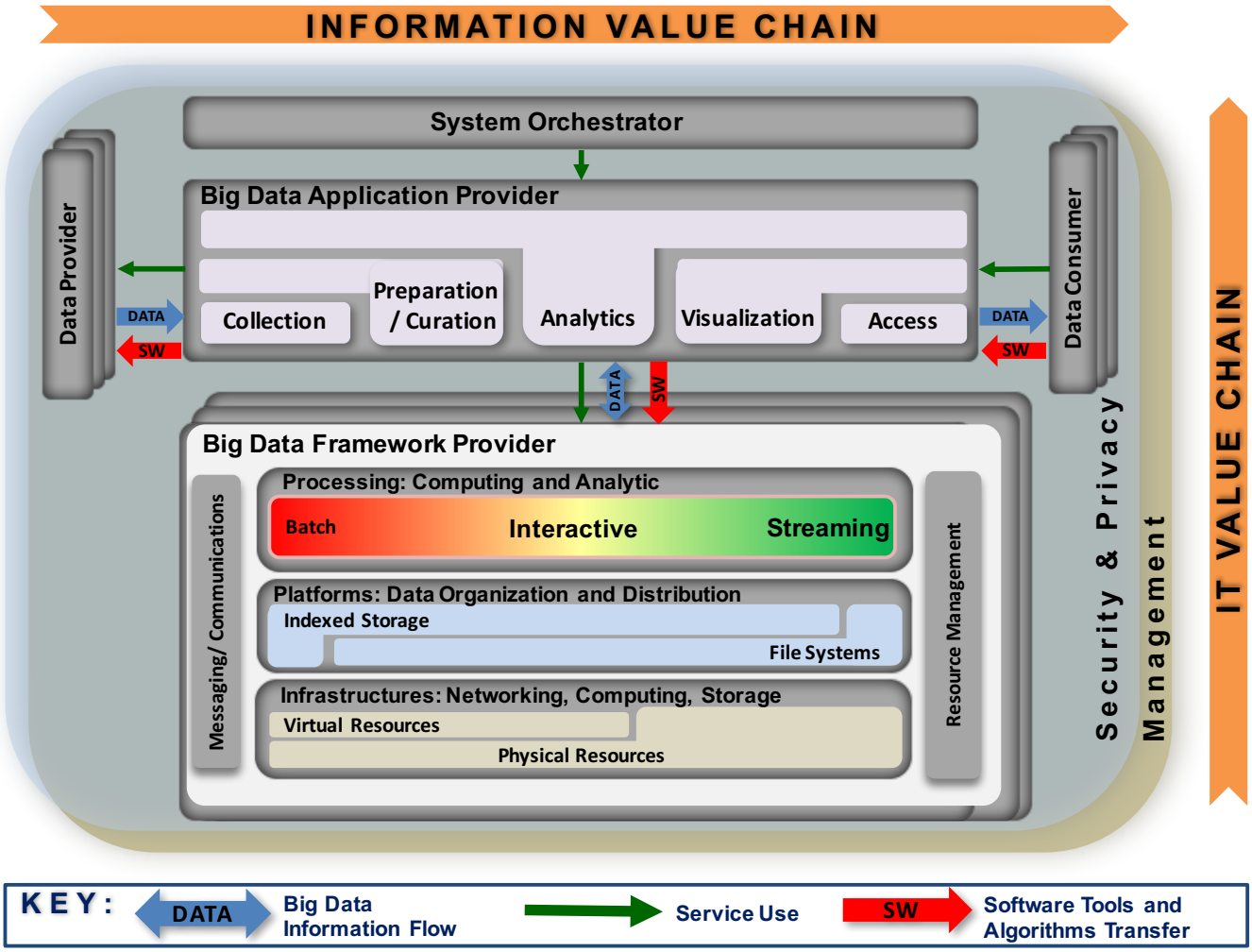

Figure 2: NIST Big Data Reference Architecture (NBDRA)

This vendor-neutral, technology- and infrastructure-agnostic conceptual model, the NBDRA, is shown in Figure 2 and represents a Big Data system comprised of five logical functional components connected by interoperability interfaces (i.e., services). Two fabrics envelop the components, representing the interwoven nature of management and security and privacy with all five of the components. These two fabrics provide services and functionality to the five main roles in the areas specific to Big Data and are crucial to any Big Data solution. Note: None of the terminology or diagrams in these documents is intended to be normative or

to imply any business or deployment model. The terms *provider* and *consumer* as used are descriptive of general roles and are meant to be informative in nature.

The NBDRA is organized around five major roles and multiple sub-roles aligned along two axes representing the two Big Data value chains: the Information Value (horizontal axis) and the Information Technology (IT; vertical axis). Along the Information Value axis, the value is created by data collection, integration, analysis, and applying the results following the value chain. Along the IT axis, the value is created by providing networking, infrastructure, platforms, application tools, and other IT services for hosting of and operating the Big Data in support of required data applications. At the intersection of both axes is the Big Data Application Provider role, indicating that data analytics and its implementation provide the value to Big Data stakeholders in both value chains. The term provider as part of the Big Data Application Provider and Bid Data Framework Provider is there to indicate that those roles provide or implement specific activities and functions within the system. It does not designate a service model or business entity.

The DATA arrows in Figure 2 show the flow of data between the systemâĂŹs main roles. Data flows between the roles either physically (i.e., by value) or by providing its location and the means to access it (i.e., by reference). The SW arrows show transfer of software tools for processing of Big Data *in situ*. The Service Use arrows represent software programmable interfaces. While the main focus of the NBDRA is to represent the run-time environment, all three types of communications or transactions can happen in the configuration phase as well. Manual agreements (e.g., service-level agreements) and human interactions that may exist throughout the system are not shown in the NBDRA.

Detailed information on the NBDRA conceptual model is presented in the *NBDIF: Volume 6, Reference Architecture* document.

Prior to outlining the specific interfaces, general requirements are introduced and the interfaces are defined.

3.1. 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 2.

3.1.1. 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, a methodology is needed 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.

3.1.2. 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.

3.1.3. 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 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.

3.1.4. Orchestration of Big Data Applications and Experiments

The creation of the infrastructure suitable for Big Data applications provides the basic computing environment. 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, the applications must 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 on behalf of the data scientist) is used as the command center to interact on behalf of the Big Data Application Provider to orchestrate dataflow from Data Provider, carryout the Big Data application lifecycle with the help of the Big Data Framework Provider, and enable the Data Consumer to consume Big Data processing results. An interface is needed to describe these interactions and to allow leveraging of experiment management frameworks in scripted fashion. A customization of parameters is needed on several levels. On the highest level, high- level, application-motivated parameters are needed to drive the orchestration of the experiment. On lower levels, these high-level parameters may drive and create service level agreements, augmented specifications, and parameters that could even lead to the orchestration of infrastructure and services to satisfy experiment needs.

3.1.5. 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

3.1.6. Execution Workloads

One of the important aspects of distributed Big Data services can be that the data served is simply too big to be moved to a different location. Instead, an interface could allow the description and packaging of 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 by 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).

3.1.7. Security and Privacy Fabric Requirements

Although the focus of this document is not security and privacy, which are documented in the *NBDIF: Volume 4, Security and Privacy* [8], the interfaces defined herein must be capable of integration into a secure reference architecture that supports secure execution, secure data transfer, and privacy. Consequently, the interfaces defined herein can be augmented with frameworks and solutions that provide such mechanisms. Thus, diverse requirement needs stemming from different use cases addressing security need to be distinguished. To contrast that the security requirements between applications can vary drastically, the following example is provided. Although many of the interfaces and their objects to support Big Data applications in physics are similar to those in healthcare, they differ in the integration of security interfaces and policies. While in physics the protection of data is less of an issue, it is a stringent requirement in healthcare. Thus, deriving architectural

frameworks for both may use largely similar components, but addressing security issues will be very different. In future versions of this document, the security of interfaces may be addressed. In the meanwhile, they are considered an advanced use case showcasing that the validity of the specifications introduced here is preserved, even if security and privacy requirements differ vastly among application use cases.

3.2. 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 2 and addressed in each of the subsections as part of Section 3.2.1–3.2.6 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 3.2.1);

Data Provider: Introduces new data or information feeds into the Big Data system (see Section 3.2.2);

Data Consumer: Includes end users or other systems that use the results of the Big Data Application Provider (see Section 3.2.3).

Big Data Application Provider: Executes a data life cycle to meet security and privacy requirements as well as System Orchestrator-defined requirements (see Section 3.2.4);

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 3.2.5); and

Big Data Application Provider to Framework Provider Interface: Defines an interface between the application specification and the provider (see Section 3.2.6).

3.2.1. System Orchestrator Interface Requirements

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 System Orchestrator as extensible interfaces is desirable.

3.2.2. Data Provider Interface Requirements

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.

3.2.3. Data Consumer Interface Requirements

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 the following:

- Search and Retrieve,
- Download,
- Analyze Locally,
- Reporting,
- Visualization, and
- 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.

3.2.4. 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 of the interfaces used in these subcomponents can be reused from other interfaces, which are introduced in other sections of this document. Where appropriate, application specific interfaces will be identified and examples provided with a focus on use cases as identified in the it NBDIF: Volume 3 Use Cases and Requirements [4].

3.2.4.1 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.

3.2.4.2 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.

3.2.4.3 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.

3.2.4.4 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.

3.2.4.5 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.

3.2.5. 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. This interface must be able to provide the following functionality:

- Interfaces to files,
- Interfaces to virtual data directories,
- Interfaces to data streams, and
- Interfaces to data filters.

3.2.5.1 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. The NBDRA needs interfaces that can be used to deal with the underlying infrastructure to address networking, computing, and storage.

3.2.5.2 Platforms Interface Requirements

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

3.2.5.3 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. A requirement is the ability to interface easily with computing services that offer specific analytics services, batch processing capabilities, interactive analysis, and data streaming.

3.2.5.4 Crosscutting Interface Requirements

Several crosscutting interface requirements within the Big Data Framework Provider 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, such interfaces may also be exposed on a lower level in case finer grained control is needed. The need for such crosscutting interface requirements will be extracted from the it NBDIF: Volume 3, Use Cases and Requirements [4] document.

3.2.5.5 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 application programming interfaces (APIs) for the reliable queuing, transmission, and receipt of data

3.2.5.6 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.

3.2.6. Big Data Application Provider to Big Data 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.

4. SPECIFICATION PARADIGM

This section summarizes the elementary objects that are important to the NBDRA.

4.1. Lessons Learned

Originally, a full REpresentational State Transfer (REST) specification was used for defining the objects related to the NBDRA [11]. However, at this stage of the document, it would introduce too complex of a notation framework. This would result in (1) a considerable increase in length of this document, (2) a more complex framework reducing participation in the project, and (3) a more complex framework for developing a reference implementation. Thus, in this version of the document, a design concept by example will be introduced, which is used to automatically create a schema as well as a reference implementation.

4.2. Hybrid and Multiple Frameworks

To avoid vendor lock in, Big Data systems must be able to deal with hybrid and multiple frameworks. This is not only true for Clouds, containers, DevOps, but also components of the NBDRA.

4.3. Design by Research Oriented Architecture

A resource-oriented architecture represents a software architecture and programming paradigm for designing and developing software in the form of resources. It is often associated with *RESTful* interfaces. The resources are software components which can be reused in concrete reference implementations.

4.4. Design by Example

To accelerate discussion among the NBD-PWG members, an approach by example is used to define objects and their interfaces. These examples can then be used to automatically generate a schema. The schema is added to the Appendix A.1 of the document. Appendix A.1 lists the schema that is automatically created from the definitions. More information about the creation can be found in Appendix B.

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][11].

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

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

Figure 3: Object Example object specification

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 **??** and **??**).

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.

4.5. Interface Compliancy

Due to the easy extensibility of the objects in this document and their implicit interfaces, it is important to introduce a terminology that allows definition of interface compliancy. The Subgroup defines three levels of interface compliance 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.

Partial 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 will be generated during the reference implementation that lists all objects defined, but also lists the objects that are not defined by the reference architecture. The 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 generated during the reference implementation that lists the differences to the document defined here.

The documents generated during the reference implementation can then be forwarded to the Reference Architecture Subgroup for further discussion and for possible future modifications based on additional practical user feedback.

5. SPECIFICATION

As several objects are used across the NBDRA we have not organized them by component as introduced in Figure 2. Instead we have grouped the objects by functional use as depicted summarized in Figure 4.

5.1. Identity

In a multiuser environment, a simple mechanism is used in this document for associating objects and data to a particular person or group. While these efforts do not intend to replace more elaborate solutions such as proposed by eduPerson [5] or others, a very simple way was chosen to distinguish users. Therefore, the following sections introduce a number of simple objects including a profile and a user.

5.1.1. Profile

A profile defines the identity of an individual. It contains name and e-mail information. It may have an optional unique user ID (uuid) and/or use a unique e-mail to distinguish a user. Profiles are used to identify different users.

```
{
  "profile": {
    "description": "The Profile of a user",
    "uuid": "jshdjkdh...",
    "context:": "resource",
    "email": "laszewski@gmail.com",
```

Figure 4: NIST Big Data Reference Architecture Interfaces

```
    "firstname": "Gregor",
    "lastname": "von Laszewski",
    "username": "gregor",
    "publickey": "ssh ...."
  }
}
```

Figure 5: Object Profile

5.1.2. User

In contrast to the profile, a user contains additional attributes that define the role of the user within the multiuser system. The user associates different roles to individuals. These roles potentially have gradations of responsibility and privilege.

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

Figure 6: Object Organization

5.1.3. Organization

An important concept in many applications is the management of a group of users in an organization that manages a Big Data application or infrastructure. User group management can be achieved through three

concepts. First, it can be achieved by 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 within the 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.

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

Figure 7: Object User

The profile, user, and organization concepts allow for the clear definition of various roles such as data provider, data consumer, data curator, and others. These concepts also allow for the creation of services that restrict data access by role, or organizational affiliation.

5.1.4. Group/Role

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

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

Figure 8: Object Group

A role is a further refinement of a group. Group members can have specific roles. For example, a group of users can be assigned a role that allows access to a repository. More specifically, the role would define a userâĂŹs read and write privileges to the data within the repository.

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

Figure 9: Object Role

5.2. 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.

Defaults are special type of variables that allow adding of a context. Defaults can created for different contexts.

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

Directories are locations for storing and organizing multiple files on a compute resource.

Virtual Directories are collection of endpoints to files. Files in a virtual directory may be located on different resources. 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.

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

Batch Filters operate on streams and on files while working in the background and delivering as output Files. In contrast to Streams Batch filters process on the data set and return after all operations have been applied.

Indexed Stores are storage systems that store objects and can be accessed by an index for each objects. Search and Filter functions are integrated to allow identifying objects from it.

Databases are traditional but also NoSQL databases.

Collections are agglomeration of any type of data.

Replicas are duplication of data objects in order to avoid overhead due to network or other physical restrictions on a remote resource.

5.2.1. TimeStamp

Often data needs to be time stamped to indicate when it has been accessed, created, or modified. All objects defined in this document will have, in their final version, a time stamp.

```
{
    "timestamp": {
        "accessed": "1.1.2017:05:00:00:EST",
        "created": "1.1.2017:05:00:00:EST",
        "modified": "1.1.2017:05:00:00:EST"
  }
}
```

Figure 10: Object Timestamp

5.2.2. Variables

Variables are used to store simple values. Each variable can have a type, which is also provided as demonstrated in the object below. The variable value format is defined as string to allow maximal probability.

```
{
  "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, ..."
  }
}
```

Figure 11: Object Var

5.2.3. 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.

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

Figure 12: Object Default

5.2.4. File

A file is a computer resource allowing storage of data that is being processed. The interface to a file provides the mechanism to appropriately locate a file in a distributed system. File identification includes the name, endpoint, checksum, and size. Additional parameters, such as the last access time, could also be stored. 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.

```
{
    "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"]
  }
}
```

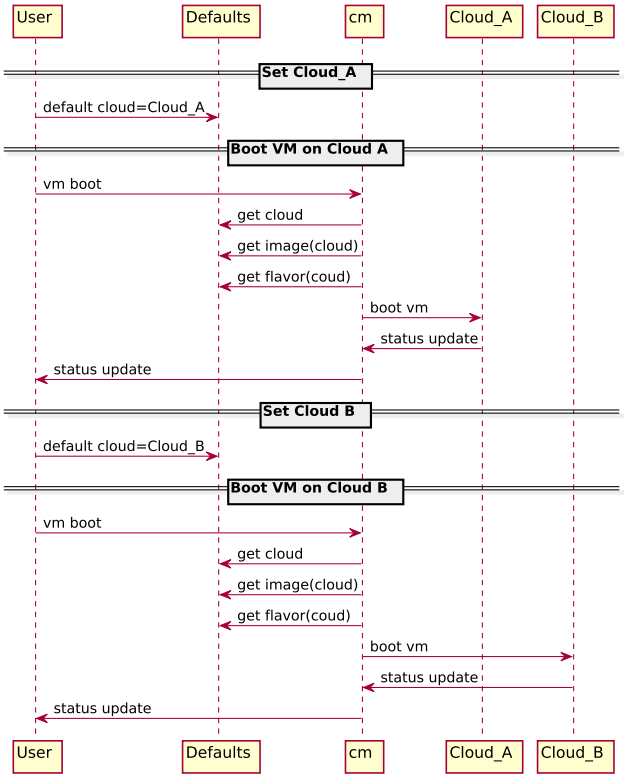

Figure 13: Booting a VM from defaults

Figure 14: Object File

5.2.5. Alias

A data object 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 user's application.

```
{
  "alias": {
    "name": "a better name for the object",
    "origin": "the original object name"
  }
}
```

Figure 15: Object File alias

5.2.6. 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. A replica can be applied to all data types introduced in this document.

```
{
```

```
  "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"
    ]
  }
}
```

Figure 16: Object Replica

5.2.7. 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.

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

Figure 17: Object Virtual directory

5.2.8. Database

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

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

Figure 18: Object Database

5.2.9. Stream

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

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

Figure 19: Object Stream

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 and unsubscribe from a stream, while also applying filters to the subscribed stream.

5.2.10. Filter

Filters can operate on a variety of objects and reduce the information received based on a search criterion.

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

Figure 20: Object Filter

5.3. Virtual Cluster

One of the essential features for Big Data is the creation of a Big Data analysis cluster. A virtual cluster combines resources that generally are used to serve the Big Data application and can constitute a variety of data analysis nodes that together build the virtual cluster. Instead of focusing only on the deployment of a physical cluster, the creation of a virtual cluster can be instantiated on a number of different platforms. Such platforms include clouds, containers, physical hardware, or a mix thereof to support different aspects of the Big Data application.

Figure 21 illustrates the process for allocating and provisioning a virtual cluster. The user defines the desired physical properties of the cluster (e.g., CPU, memory, disk) and the intended configuration (e.g., software, users). After requesting the stack to be deployed, cloudmesh allocates the machines by matching the desired properties with the available images and booting. The stack definition is then parsed then evaluated to provision the cluster.

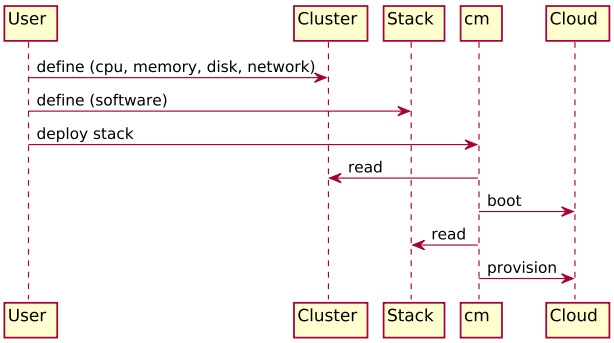

Figure 21: Allocating and provisioning a virtual cluster

5.3.1. Virtual Cluster

A virtual cluster is an agglomeration of virtual compute nodes that

constitute the cluster. Nodes can be assembled to be baremetal, VMs, and containers. A virtual cluster contains a number of virtual compute nodes.

The virtual cluster object has name, label, endpoint, and provider. The *endpoint* defines a mechanism to connect to it. The *provider* defines the nature of the cluster (e.g., it is a virtual cluster on an OpenStack cloud, or from AWS, or a bare-metal cluster).

To manage the cluster it can have a frontend node that is used to manage other nodes. Authorized keys within the definition of the cluster allow administrative functions, while authorized keys on a compute node allow login and use functionality of the virtual nodes.

```
{
  "virtual_cluster": {
      "name": "myvirtualcluster",
      "label": "C0",
      "uuid": "sgdlsjlaj....",
      "endpoint": {
          "passwd": "secret",
          "url": "https:..."
      },
      "provider": "virtual_cluster_provider:openstack",
      "frontend": "objectid:virtual_machine",
      "authorized_keys": ["objectid:sshkey"],
      "nodes": [
          "objectid:virtual_machine"
      ]
  }
}
```

Figure 22: Object Virtual cluster

```
"virtual_cluster_provider":  "aws" | "azure" | "google" | "comet" | "openstack"
```

Figure 23: Object Virtual cluster provider

5.3.2. Compute Node

Compute nodes are used to conduct compute and data functions. They are of a specific *kind*. For example,compute nodes could be a virtual machine (VM), bare metal, or part of a predefined virtual cluster framework.

Compute nodes are a representation of a computer system (physical or virtual). A very basic set of information about the compute node is maintained in this document. It is expected that, through the endpoint, the VM can be introspected and more detailed information can be retrieved. A compute node has name, label, a flavor, network interface cards (NICs) and other relevant information.

```
{
  "compute_node": {
    "name": "vm1",
    "label": "gregor-vm001",
    "uuid": "sgklfgslakj....",
    "kind": "vm",
    "flavor": ["objectid:flavor"],
    "image": "Ubuntu-16.04",
    "secgroups": ["objectid:secgroup"],
    "nics": ["objectid:nic"],
    "status": "",
    "loginuser": "ubuntu",
    "status": "active",
    "authorized_keys": ["objectid:sshkey"],
    "metadata": {
       "owner":"gregor",
       "experiment": "exp-001"
    }
  }
}
```

Figure 24: Object Compute node of a virtual cluster

5.3.3. Flavor

The flavor specifies elementary information about the compute node, such as memory and number of cores, as well as other attributes that can be added. Flavors are essential to size a virtual cluster appropriately.

```
{
    "flavor": {
      "name": "flavor1",
      "label": "2-4G-40G",
      "uuid": "sgklfgslakj....",
      "ncpu": 2,
      "ram": "4G",
      "disk": "40G"
    }
}
```

Figure 25: Object Flavor

5.3.4. Network Interface Card

To interact between the nodes, a network interface is needed. Such a network interface, specified on a virtual machine with a NIC object, is showcased in Object **??**.

```
{
  "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"
  }
}
```

Figure 26: Object Network interface card

5.3.5. Key

Many services and frameworks use Secure Shell (SSH) keys to authenticate. To allow the convenient storage of the public key, the sshkey object can be used (see Object **??**).

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

Figure 27: Object Key

5.3.6. Security Groups

To allow secure communication between the nodes, security groups are introduced. They define the typical security groups that will be deployed once a compute node is specified. The security group object is depicted in Object **??**.

```
{
  "secgroup":
      {
        "ingress": "0.0.0.0/32",
        "egress": "0.0.0.0/32",
```

```
        "ports": 22,
        "protocols": "tcp"
      }
}
```

Figure 28: Object Security Groups

5.4. Infrastructure as a Service

Although Section 5.3 defines a general virtual cluster useful for Big Data, sometimes the need exists to specifically utilize Infrastructure as a Service (IaaS) frameworks, such as Openstack, Azure, and others. To do so, it is beneficial to be able to define virtual clusters using these frameworks. Hence, this subsection defines interfaces related to IaaS frameworks. This includes specific objects useful for OpenStack, Azure, and AWS, as well as others. The definition of the objects used between the clouds to manage them, are different and not standardized. In this case the objects support functions such as

starting, stoping, suspending resuming, migration, network configuration, assigning of resources, assigning of operating systems for and others for the VMs.

Inspecting other examples, such as *LibCloud*, shows the definition of generalized objects are discovered, which are augmented with extra fields to specifically integrate with the various frameworks. When working with cloudmesh, it is sufficient to be able to specify a cloud based on a cloud specific action. Actions include boot, terminate, suspend, resume, assign network intrusion prevention system, and add users.

To support such actions, objects can be selected based on the IaaS type in use when invoked. The following subsections list these objects as used in LibCloud, OpenStack, and Azure.

5.4.1. LibCloud

Libcloud is a Python library for interacting with different cloud service providers. It uses a unified API that exposes similar access to a variety of clouds. Internally, it uses objects that can interface with different IaaS frameworks. However, as these frameworks are different from each other, specific adaptations are done for each IaaS, mostly reflected in the LibCloud Node (see Section 5.4.1.5)

5.4.1.1 Challenges

For time considerations, LibCloud was used for some time practically in various versions of cloudmesh. However, it became apparent that at times the representation and functionality provided by LibCloud, for reference implementations, did not support some advanced aspects provided by the native cloud objects. Depending on the application, libraries for interfacing with different frameworks, direct utilization of the native objects, and interfaces provided by a particular IaaS framework could all be viable options. Additional interfaces have been introduced in Sections 5.4.2 and 5.4.3. Additional sections addressing other IaaS frameworks may be integrated in the future.

5.4.1.2 LibCloud Flavor

The object referring to flavors is listed in Object **??**.

```
{
  "libcloud_flavor": {
    "bandwidth": "string",
    "disk": "string",
    "uuid": "string",
    "price": "string",
```

```
    "ram": "string",
    "cpu": "string",
    "flavor_id": "string"
  }
}
```

Figure 29: Object Libcloud flavor

5.4.1.3 LibCloud Image

The object referring to images is listed in Object **??**.

```
{
    "libcloud_image": {
        "username": "string",
        "status": "string",
        "updated": "string",
        "description": "string",
        "owner_alias": "string",
        "kernel_id": "string",
        "ramdisk_id": "string",
        "image_id": "string",
        "is_public": "string",
        "image_location": "string",
        "uuid": "string",
        "created": "string",
        "image_type": "string",
        "hypervisor": "string",
        "platform": "string",
        "state": "string",
        "architecture": "string",
        "virtualization_type": "string",
        "owner_id": "string"
    }
}
```

Figure 30: Object Libcloud image

5.4.1.4 LibCloud VM

The object referring to virtual machines is listed in

```
{
  "libcloud_vm": {
    "username": "string",
    "status": "string",
    "root_device_type": "string",
    "image": "string",
    "image_name": "string",
    "image_id": "string",
    "key": "string",
    "flavor": "string",
```

```
    "availability": "string",
    "private_ips": "string",
    "group": "string",
    "uuid": "string",
    "public_ips": "string",
    "instance_id": "string",
    "instance_type": "string",
    "state": "string",
    "root_device_name": "string",
    "private_dns": "string"
  }
}
```

Figure 31: Object LibCloud VM

5.4.1.5 LibCloud Node

Virtual machines for the various clouds have additional attributes that are summarized in Object **??**. These attributes will be integrated into the VM object in the future.

```
{
  "LibCLoudNode": {
    "id": "instance_id",
    "name": "name",
    "state": "state",
    "public_ips": ["111.222.111.1"],
    "private_ips": ["192.168.1.101"],
    "driver": "connection.driver",
    "created_at": "created_timestamp",
    "extra": {
    }
  },
  "ec2NodeExtra": {
    "block_device_mapping": "deviceMapping",
    "groups": ["security_group1", "security_group2"],
    "network_interfaces": ["nic1", "nic2"],
    "product_codes": "product_codes",
    "tags": ["tag1", "tag2"]
  },
  "OpenStackNodeExtra": {
    "addresses": ["addresses"],
    "hostId": "hostId",
    "access_ip": "accessIPv4",
    "access_ipv6": "accessIPv6",
    "tenantId": "tenant_id",
    "userId": "user_id",
    "imageId": "image_id",
    "flavorId": "flavor_id",
    "uri": "",
    "service_name": "",
    "metadata": ["metadata"],
    "password": "adminPass",
```

```
    "created": "created",
    "updated": "updated",
    "key_name": "key_name",
    "disk_config": "diskConfig",
    "config_drive": "config_drive",
    "availability_zone": "availability_zone",
    "volumes_attached": "volumes_attached",
    "task_state": "task_state",
    "vm_state": "vm_state",
    "power_state": "power_state",
    "progress": "progress",
    "fault": "fault"
  },
  "AzureNodeExtra": {
    "instance_endpoints": "instance_endpoints",
    "remote_desktop_port": "remote_desktop_port",
    "ssh_port": "ssh_port",
    "power_state": "power_state",
    "instance_size": "instance_size",
    "ex_cloud_service_name": "ex_cloud_service_name"
  },
  "GCENodeExtra": {
    "status": "status",
    "statusMessage": "statusMessage",
    "description": "description",
    "zone": "zone",
    "image": "image",
    "machineType": "machineType",
    "disks": "disks",
    "networkInterfaces": "networkInterfaces",
    "id": "node_id",
    "selfLink": "selfLink",
    "kind": "kind",
    "creationTimestamp": "creationTimestamp",
    "name": "name",
    "metadata": "metadata",
    "tags_fingerprint": "fingerprint",
    "scheduling": "scheduling",
    "deprecated": "True or False",
    "canIpForward": "canIpForward",
    "serviceAccounts": "serviceAccounts",
    "boot_disk":  "disk"
  }
}
```

Figure 32: Object LibCloud Node

5.4.2. OpenStack

Objects related to OpenStack VMs are summarized in this section.

5.4.2.1 OpenStack Flavor

The object referring to flavors is listed in Object **??**.

```
{
  "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"
  }
}
```

Figure 33: Object Openstack flavor

5.4.2.2 OpenStack Image

The object referring to images is listed in Object **??**.

```
{
  "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"
    }
  }
}
```

Figure 34: Object Openstack image

5.4.2.3 OpenStack VM

The object referring to VMs is listed in Object **??**.

```
{
  "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"
  }
}
```

Figure 35: Object Openstack vm

5.4.3. Azure

Objects related to Azure virtual machines are summarized in this section.

5.4.3.1 Azure Size

The object referring to the image size machines is listed in Object **??**.


```
{
  "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"
  }
}
```


Figure 36: Object Azure-size

5.4.3.2 Azure Image

The object referring to the images machines is listed in Object **??**.


```
{
  "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 b
      "location": "East Asia;Southeast Asia;Australia East;Australia Southeast;Brazil South;North Europe
      "media_link": "",
      "os": "Linux",
      "vm_image": "False"
    },
    "id": "03f55de797f546a1b29d1....",
    "name": "CoreCLR x64 Beta5 (11624) with PartsUnlimited Demo App on Ubuntu Server 14.04 LTS"
  }
}
```


Figure 37: Object Azure-image

5.4.3.3 Azure VM

The object referring to the virtual machines is listed in Object **??**.


```
{
```

```
  "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"
  }
}
```

Figure 38: Object Azure-vm

5.5. Compute Services

5.5.1. Batch Queue

Computing jobs that can run without end user interaction, or are scheduled based on resource permission are called batch jobs. It is used to minimize human interaction and allows the submission and scheduling of many jobs in parallel while attempting to utilize the resources through a resource scheduler more efficiently or simply in sequential order. Batch processing is not to be underestimated even in todays shifting IoT environment towards clouds and containers. This is based on the fact that for some application resources managed by batch queues are highly optimized and in many cases provide significant performance advantages. Disadvantages are the limited and preinstalled software stacks that in some cases do not allow to run the latests applications.

```
{
  "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"
  }
}
```

Figure 39: Object Batchjob

5.5.2. Reservation

Some services may consume a considerable amount of resources, necessitating the reservation of resources. For this purpose, a reservation object (Object **??**) has been introduced.

```
{
  "reservation": {
    "service": "name of the service",
    "description": "what is this reservation for",
    "start_time": ["date", "time"],
    "end_time": ["date", "time"]
  }
}
```

Figure 40: Object Reservation

5.6. Containers

The following defines the *container* object.

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

Figure 41: Object Container

5.7. Deployment

A *deployment* consists of the resource *cluster*, the location *provider* (e.g., OpenStack), and software *stack* to be deployed (e.g., Hadoop, Spark).

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

```
            ],
            "parameters": {
                "hadoop": { "zookeeper.quorum": [ "IP", "IP", "IP"]
                          }
            }
        }
    }
}
```

Figure 42: Object Deployment

5.8. 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.

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

Figure 43: Object Mapreduce

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

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

```
{
  "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"
    }
  }
}
```

Figure 44: Object Mapreduce function

Some example functions include the "NoOp" function shown in Listing **??**. In the case of undefined arguments, the parameters default to an identity element. In the case of mappings this is the empty mapping while for lists this is the empty list.

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

Figure 45: Object Mapreduce noop

5.8.1. 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, and others (see Object **??**)

```
{
  "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
    }
  }
}
```

Figure 46: Object Hadoop

5.9. Microservice

As part of microservices, a function with parameters that can be invoked has been defined. To describe such services, the Object **??** was created. Defining multiple services facilitates the finding of the microservices and the use as part of a microservice based implementation.

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

Figure 47: Object Microservice

5.9.1. 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.

```
{
```

```
  "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"
  }
}
```

Figure 48: Object Accounting

```
{
  "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"]
  }
}
```

Figure 49: Object Account

5.9.1.1 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.

6. STATUS CODES AND ERROR RESPONSES

In case of an error or a successful response, the response header contains a HTTP code (see `https://tools.ietf.org/html/rfc7231`). 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

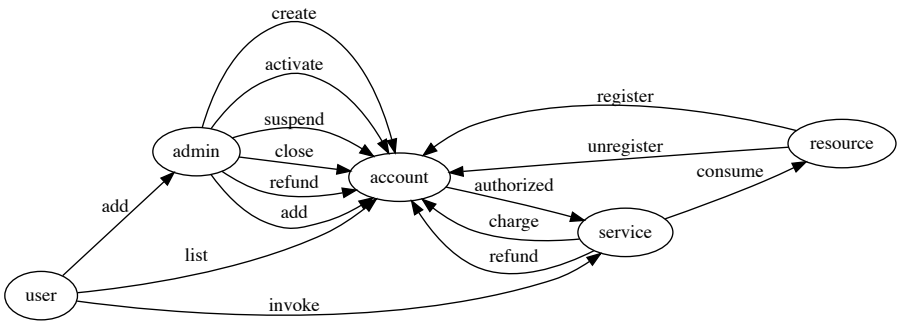

Figure 50: Create Resource

Table 1: HTTP response codes

HTTP response code	Description
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 is not allowed for the resource specified in the URI.
415	The entity in the request is in a format that is not supported by the specified method.

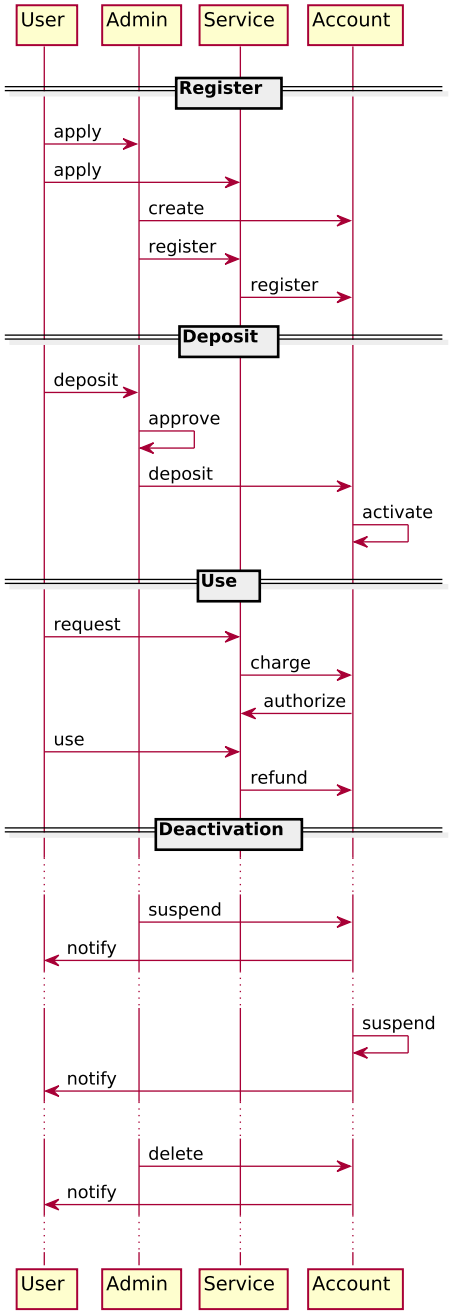

Figure 51: Accounting

7. 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

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 VM 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 VM is a software computer that, like a physical computer, runs an operating system and applications. The VM 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 VMs, 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

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, Wo Chang, Fugang Wang, Badi Abdhul Wahid, , Geoffrey C. Fox, Pratik Thakkar, Alicia Mara Zuniga-Alvarado, and Robert C. Whetsel. 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://laszewski.github.io/papers/NIST.SP.1500-8-draft.pdf, doi:NIST.SP.1500-8.

[11] Gregor von Laszewski, Fugang Wang, Badi Abdul-Wahid, Hyungro Lee, Geoffrey C. Fox, and Wo Chang. Cloudmesh in support of the nist big data architecture framework. Technical report, Indiana University, Bloomingtion IN 47408, USA, April 2017. URL: https://laszewski.github.io/papers/vonLaszewski-nist.pdf.

A. APPENDIX

A.1. Schema

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

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

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': {
            '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'
            }
        },
        'description': {
            'type': 'string'
        },
        'service': {
            'type': 'string'
        },
        'end_time': {
            'type': 'list',
            'schema': {
                'type': 'string'
            }
        }
    }
}

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

flavor = {
    'schema': {
        'uuid': {
            'type': 'string'
        },
        'ram': {
            'type': 'string'
        },
        'label': {
            'type': 'string'
        },
        'ncpu': {
            'type': 'integer'
        },
        'disk': {
            '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': {
        'authorized_keys': {
            'type': 'list',
            'schema': {
                'type': 'objectid',
                'data_relation': {
                    'resource': 'sshkey',
                    'field': '_id',
                    'embeddable': True
```

```
                }
            }
        },
        'endpoint': {
            'type': 'dict',
            'schema': {
                'passwd': {
                    'type': 'string'
                },
                'url': {
                    'type': 'string'
                }
            }
        },
        'frontend': {
            'type': 'objectid',
            'data_relation': {
                'resource': 'virtual_machine',
                'field': '_id',
                'embeddable': True
            }
        },
        'uuid': {
            'type': 'string'
        },
        'label': {
            'type': 'string'
        },
        'provider': {
            'type': 'string'
        },
        'nodes': {
            'type': 'list',
            'schema': {
                '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'
        }
    }
}

LibCLoudNode = {
    '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'
        }
    }
}

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

timestamp = {
    'schema': {
        'accessed': {
            'type': 'string'
        },
        'modified': {
            'type': 'string'
        },
        'created': {
            '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'
        }
    }
}

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'
        }
    }
}

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'
        }
    }
}

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

secgroup = {
    'schema': {
        'ingress': {
            'type': 'string'
        },
        'egress': {
            'type': 'string'
        },
        'ports': {
            'type': 'integer'
        },
        'protocols': {
            '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'
        }
    }
}

compute_node = {
    'schema': {
        'status': {
            'type': 'string'
        },
        'authorized_keys': {
            'type': 'list',
            'schema': {
                'type': 'objectid',
                'data_relation': {
                    'resource': 'sshkey',
                    'field': '_id',
                    'embeddable': True
                }
            }
        },
        'kind': {
            'type': 'string'
        },
        'uuid': {
            'type': 'string'
        },
        'secgroups': {
            'type': 'list',
            'schema': {
```

```
            'type': 'objectid',
            'data_relation': {
                'resource': 'secgroup',
                'field': '_id',
                'embeddable': True
            }
        }
    },
    'nics': {
        'type': 'list',
        'schema': {
            'type': 'objectid',
            'data_relation': {
                'resource': 'nic',
                'field': '_id',
                'embeddable': True
            }
        }
    },
    'image': {
        'type': 'string'
    },
    'label': {
        'type': 'string'
    },
    'loginuser': {
        'type': 'string'
    },
    'flavor': {
        'type': 'list',
        'schema': {
            'type': 'objectid',
            'data_relation': {
                'resource': 'flavor',
                'field': '_id',
                'embeddable': True
            }
        }
    },
    'metadata': {
        'type': 'dict',
        'schema': {
            'owner': {
                'type': 'string'
            },
            'experiment': {
                'type': 'string'
            }
        }
    },
    'name': {
```

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

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

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

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

```
    'type': 'string'
},
'progress': {
    'type': 'string'
},
'minRam': {
    'type': 'string'
},
'os_image_size': {
    'type': 'string'
},
'metadata': {
    'type': 'dict',
    'schema': {
        'instance_uuid': {
            'type': 'string'
        },
        'image_location': {
            'type': 'string'
        },
        'image_state': {
            'type': 'string'
        },
        'instance_type_memory_mb': {
            'type': 'string'
        },
        'user_id': {
            'type': 'string'
        },
        'description': {
            'type': 'string'
        },
        'kernel_id': {
            'type': 'string'
        },
        'instance_type_name': {
            'type': 'string'
        },
        'ramdisk_id': {
            'type': 'string'
        },
        'instance_type_id': {
            'type': 'string'
        },
        'instance_type_ephemeral_gb': {
            'type': 'string'
        },
        'instance_type_rxtx_factor': {
            'type': 'string'
        },
        'image_type': {
```

```
                    'type': 'string'
                },
                'network_allocated': {
                    'type': 'string'
                },
                'instance_type_flavorid': {
                    'type': 'string'
                },
                'instance_type_vcpus': {
                    'type': 'string'
                },
                'instance_type_root_gb': {
                    'type': 'string'
                },
                'base_image_ref': {
                    'type': 'string'
                },
                'instance_type_swap': {
                    'type': 'string'
                },
                'owner_id': {
                    'type': 'string'
                }
            }
        }
    }
}

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'
                }
            }
        }
    }
}

filter = {
    'schema': {
        'function': {
```

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

alias = {
    'schema': {
        'origin': {
            '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': {
        'container': container,
        'stream': stream,
        'azure_image': azure_image,
        'deployment': deployment,
        'azure-size': azure_size,
        'cluster': cluster,
        'computer': computer,
        'mesos-docker': mesos_docker,
        'file': file,
        'reservation': reservation,
        'microservice': microservice,
        'flavor': flavor,
        'virtual_directory': virtual_directory,
        'mapreduce_function': mapreduce_function,
        'virtual_cluster': virtual_cluster,
        'libcloud_flavor': libcloud_flavor,
        'LibCLoudNode': LibCLoudNode,
        'sshkey': sshkey,
        'timestamp': timestamp,
        'mapreduce_noop': mapreduce_noop,
        'role': role,
        'AzureNodeExtra': AzureNodeExtra,
        'var': var,
        'profile': profile,
        '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,
        'group': group,
        'secgroup': secgroup,
        'node_new': node_new,
        'batchjob': batchjob,
        'account': account,
        'libcloud_vm': libcloud_vm,
        'compute_node': compute_node,
        'database': database,
        'default': default,
        'openstack_image': openstack_image,
        'OpenStackNodeExtra': OpenStackNodeExtra,
        'mapreduce': mapreduce,
        'filter': filter,
        'alias': alias,
        'replica': replica,
        'openstack_vm': openstack_vm,
        'organization': organization,
        'hadoop': hadoop,
        'accounting_resource': accounting_resource,
    },
}
```

Figure 52: Object Schema

B. CLOUDMESH REST

Cloudmesh Rest is a reference implementation for the NBDRA. It allows for automatic definition of a REST service based on the objects specified by the NBDRA. In collaboration with other cloudmesh components it allows easy interaction with hybrid clouds and the creation of user managed Big Data services.

B.1. Prerequisites

The prerequisiits 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 only ubuntu greater 16.04 and OSX Sierra have been tested). However, it would naturally be possible to also port it to Windows. At the time of publication, the installation instructions in this document are not complete. The reader is referred to the cloudmesh manuals, which are under development. The goal will be to make the installation (after the system is set up for developing Python) as simple as the following:

```
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"
  }
}
```

In this example, an object called profile is defined, which contains a number of attributes and values. The type of the values is 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 following commands:

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

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

Once the schema has been defined, cloudmesh specifies defaults for managing a sample database that is coupled with the REST service. MongoDB was used 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. 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`

C.2. Object Specification

All objects are located in

`cloudmesh.rest/cloudmesh/specification/examples`

And can be modified there

C.3. Creation of the PDF document

We assume that you have LaTeX installed. Latex can be trivially installed on Windows, OSX, and Linux. Please refer to the instalation instructions for your OS. If you have Windows and have not make installed, you can obtain it from `http://gnuwin32.sourceforge.net/packages/make.htm` Please google for it and find the version most suitable for you.

Firts you have to obtain the document from github.com. Currently, you can do this with

```
git clone https://github.com/cloudmesh/cloudmesh.rest
```

To compile the document please use

```
cd docs
make
```

This will generate the PDF file

```
NIST.SP.1500-8-draft.pdf
```

On OSX we have also integrated a quick view whit

```
make view
```

The PDF document can be transfered to doc and docx, with the following online tool:

* `http://pdf2docx.com/`

We noticed that some tabs in the object definitions may get lost, but they can be integrated easily. If yo notice any other formatting issues, please file an issue.

We assume that those writeing the document in word use a simple style theme using regular styles. Once the NIST editors have provided a suitable style them we will upload it to the repository so it can be applied easily.

C.4. Code Generation

This section is intended for experts and guidance on using it can be obtained by contacting Gregor von Laszewski. 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
python setup.py install; pip install .
cd cloudmesh.rest
cd docs
make schema
make
```

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