

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

NIST Big Data Public Working Group
Standards Roadmap Subgroup

Draft Version 0.1, Revision 1
2017/08/02, 14:58:07

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



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

Draft Version 0.1, Revision 1

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

https://bigdatawg.nist.gov/V2_output_docs.php

Current working drafts are available from
<https://raw.githubusercontent.com/cloudmesh/cloudmesh.rest/master/docs/NIST.SP.1500-8-draft.pdf>

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

2017/08/02, 14:58:07



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

96 pages (2017/08/02)

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

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

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

Comments on this publication may be submitted to Wo Chang

National Institute of Standards and Technology

Attn: Wo Chang, Information Technology Laboratory

100 Bureau Drive (Mail Stop 8900) Gaithersburg, MD 20899-8930

Email: SP1500comments@nist.gov

REQUEST FOR CONTRIBUTION

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 Government Interoperability), David Boyd (InCadence Strategic Solutions), Carl Buffington (Vistrionix), 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 contributions¹ 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.

EXECUTIVE SUMMARY

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

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

We categorized the resources in groups that are identified by the NBDRA set forward in Volume 6. While Volume 3 provides *application* oriented high level use cases the use cases defined in this document are subsets of them and focus on *interface* use cases. The interface use cases are not meant to be complete examples, but showcase why the resource has been defined. Hence, the interfaces use cases are, of course, only representative, and do not represent the entire spectrum of Big Data usage. All of the interfaces were openly discussed in the working group. Additions are welcome and we like to discuss your contributions in the group.

The NIST Big Data Interoperability Framework consists of nine volumes, each of which addresses a specific key topic, resulting from the work of the NBD-PWG. The eight volumes are:

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

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

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

Stage 2: Define general interfaces between the NBDRA components.

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

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

REFERENCES

- [1] Cerberus. URL: <http://docs.python-cerberus.org/>.
- [2] Eve Rest Service. Web Page. URL: <http://python-eve.org/>.
- [3] Cloudmesh enhanced Eveengine. Github. URL: <https://github.com/cloudmesh/cloudmesh.evegenie>.
- [4] Geoffrey C. Fox and Wo Chang. NIST Big Data Interoperability Framework: Volume 3, Use Cases and General Requirements. Special Publication (NIST SP) - 1500-3 1500-3, National INstitute of STANDARDS, 100 Bureau Drive, Gaithersburg, MD 20899, October 2015. URL: <http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.1500-3.pdf>, doi:NIST.SP.1500-3.
- [5] Internet2. eduPerson Object Class Specification (201602). Internet2 Middleware Architecture Committee for Education, Directory Working Group internet2-mace-dir-eduperson-201602, Internet2, March 2016. URL: <http://software.internet2.edu/eduperson/internet2-mace-dir-eduperson-201602.html>.
- [6] Orit Levin, David Boyd, and Wo Chang. NIST Big Data Interoperability Framework: Volume 6, Reference Architecture. Special Publication (NIST SP) - 1500-6 1500-6, National Institute of Standards, 100 Bureau Drive, Gaithersburg, MD 20899, October 2015. URL: <http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.1500-6.pdf>, doi:NIST.SP.1500-6.
- [7] NIST. Big Data Public Working Group (NBD-PWG). Web Page. URL: <https://bigdatawg.nist.gov/>.
- [8] Arnab Roy, Mark Underwood, and Wo Chang. NIST Big Data Interoperability Framework: Volume 4, Security and Privacy. Special Publication (NIST SP) - 1500-4 1500-4, National Institute of Standards, 100 Bureau Drive, Gaithersburg, MD 20899, October 2015. URL: <http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.1500-4.pdf>, doi:NIST.SP.1500-4.
- [9] Gregor von Laszewski. Cloudmesh client. github. URL: <https://github.com/cloudmesh/client>.
- [10] Gregor von Laszewski, 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, Bloomington IN 47408, USA, April 2017. URL: <https://laszewski.github.io/papers/vonLaszewski-nist.pdf>.

TABLE OF CONTENTS

1	Section 1.1	13
2	Introduction	16
2.1	Background	16
2.2	Scope and Objectives of the Reference Architecture Subgroup	16
2.3	Report Production	16
2.4	Report Structure	16
2.5	Future Work on this Volume	16
3	NBDRA Interface Requirements	16
3.1	High Level Requirements of the Interface Approach	16
3.1.1	Technology and Vendor Agnostic	16
3.1.2	Support of Plug-In Compute Infrastructure	16
3.1.3	Orchestration of Infrastructure and Services	17
3.1.4	Orchestration of Big Data Applications and Experiments	18
3.1.5	Reusability	18
3.1.6	Execution Workloads	18
3.1.7	Security and Privacy Fabric Requirements	18
3.2	Component Specific Interface Requirements	19
3.2.1	System Orchestrator Interface Requirement	19
3.2.2	Data Provider Interface Requirement	19
3.2.3	Data Consumer Interface Requirement	20
3.2.4	Big Data Application Interface Provider Requirements	20
3.2.4.1	Collection	20
3.2.4.2	Preparation	20
3.2.4.3	Analytics	20
3.2.4.4	Visualization	21
3.2.4.5	Access	21
3.2.5	Big Data Provider Framework Interface Requirements	21
3.2.5.1	Infrastructures Interface Requirements	21
3.2.5.2	Platforms Interface Requirements	22
3.2.5.3	Processing Interface Requirements	22
3.2.5.4	Crosscutting Interface Requirements	22

3.2.5.5	Messaging/Communications Frameworks	22
3.2.5.6	Resource Management Framework	22
3.2.6	BD Application Provider to Framework Provider Interface	22
4	Specification Paradigm	22
4.1	Lessons Learned	22
4.2	Hybrid and Multiple Frameworks	23
4.3	Design be Research Oriented Architecture	23
4.4	Design by Example	23
4.5	Interface Compliancy	24
5	Specification	25
5.1	Identity	25
5.1.1	Profile	25
5.1.2	User	26
5.1.3	Organization	26
5.1.4	Group/Role	26
5.2	Data	27
5.2.1	TimeStamp	28
5.2.2	Var	28
5.2.3	Default	28
5.2.4	File	29
5.2.5	Alias	30
5.2.6	Replica	30
5.2.7	Virtual Directory	30
5.2.8	Database	31
5.2.9	Stream	31
5.2.10	Filter	31
5.3	Virtual Cluster	32
5.3.1	Virtual Cluster	32
5.3.2	Compute Node	33
5.3.3	Flavor	34
5.3.4	Nic	34
5.3.5	Key	34
5.3.6	Security Groups	35

5.4	IaaS	35
5.4.1	LibCloud	35
5.4.1.1	Disadvantages	35
5.4.1.2	LibCloud Flavor	36
5.4.1.3	LibCloud Image	36
5.4.1.4	LibCloud VM	37
5.4.1.5	LibCloud Node	37
5.4.2	Openstack	39
5.4.2.1	Openstack Flavor	39
5.4.2.2	Openstack Image	39
5.4.2.3	Openstack Vm	40
5.4.3	Azure	41
5.4.3.1	Azure Size	41
5.4.3.2	Azure Image	41
5.4.3.3	Azure Vm	42
5.5	Compute Services	43
5.5.1	Batch Queue	43
5.5.2	Reservation	43
5.6	Containers	43
5.7	Deployment	44
5.8	Mapreduce	44
5.8.1	Hadoop	46
5.9	Microservice	47
5.9.1	Accounting	47
5.9.1.1	Usecase: Accounting Service	48
6	Status Codes and Error Responses	48
6.1	Acronyms and Terms	48
A	Appendix	52
A.1	Schema	52
B	Cloudmesh Rest	93
B.1	Prerequistis	93
B.2	REST Service	93
B.3	Limitations	94

C Contributing	94
C.1 Conversion to Word	94
C.2 Object Specification	95
C.3 Creation of the PDF document	95
C.4 Code Generation	95

LIST OF FIGURES

1 NIST Big Data Reference Architecture (NBDRA)	17
2 NIST Big Data Reference Architecture Interfaces	25
3 Booting a virtual machine from defaults	29
4 Allocating and provisioning a virtual cluster	32
5 Create Resource	48
6 Accounting	49

LIST OF TABLES

1 HTTP response codes	50
---------------------------------	----

LIST OF OBJECTS

Object 4.1: Example object specification	23
Object 5.1: Profile	25
Object 5.2: Organization	26
Object 5.3: User	26
Object 5.4: Group	26
Object 5.5: Role	27
Object 5.6: Timestamp	28
Object 5.7: Var	28
Object 5.8: Default	28
Object 5.9: File	29
Object 5.10: File alias	30
Object 5.11: Replica	30
Object 5.12: Virtual directory	30
Object 5.13: Database	31
Object 5.14: Stream	31
Object 5.15: Filter	31
Object 5.16: Virtual cluster	32
Object 5.17: Virtual cluster provider	33
Object 5.18: Compute node of a virtual cluster	33
Object 5.19: Flavor	34
Object 5.20: Network interface card	34
Object 5.21: Key	34
Object 5.22: Security Groups	35
Object 5.23: Libcloud flavor	36
Object 5.24: Libcloud image	36

Object 5.25: LibCloud VM	37
Object 5.26: LibCloud Node	37
Object 5.27: Openstack flavor	39
Object 5.28: Openstack image	39
Object 5.29: Openstack vm	40
Object 5.30: Azure-size	41
Object 5.31: Azure-image	41
Object 5.32: Azure-vm	42
Object 5.33: Batchjob	43
Object 5.34: Reservation	43
Object 5.35: Container	44
Object 5.36: Deployment	44
Object 5.37: Mapreduce	44
Object 5.38: Mapreduce function	45
Object 5.39: Mapreduce noop	46
Object 5.40: Hadoop	46
Object 5.41: Microservice	47
Object 5.42: Accounting	47
Object 5.43: Account	47
Object A.1: Schema	52

1. SECTION 1.1

File: section1-1.tex

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 15–17, 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 sectors—including industry, academia, and government—with the goal of developing consensus on definitions, taxonomies, secure reference architectures, security and privacy, and—from these—a 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

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

To enable interoperability between the NBDRA components, a list of well-defined NBDRA interface is needed. These interfaces are documented in this Volume 8 [10]. To introduce them, we will follow the NBDRA and focus on interfaces that allow us to bootstrap the NBDRA. We will start the document with a summary of requirements that we will integrate into our specifications. Subsequently, each section will introduce a number of objects that build the core of the interface addressing a specific aspect of the NBDRA. We will showcase a selected number of *interface use cases* to outline how the specific interface can be used in a reference implementation of the NBDRA. Validation of this approach can be achieved while applying it to the application use cases that have been gathered in Volume 3 [4]. These application use cases have considerably contributed towards the design of the NBDRA. Hence our expectation is that (a) the interfaces can be used to help implementing a big data architecture for a specific use case, and (b) the proper implementation. Through this approach, we can facilitate subsequent analysis and comparison of the use cases. We expect that this document will grow with the help of contributions from the community to achieve a comprehensive set of interfaces that will be usable for the implementation of Big Data Architectures.

2.1. Background

2.2. Scope and Objectives of the Reference Architecture Subgroup

2.3. Report Production

2.4. Report Structure

2.5. Future Work on this Volume

3. NBDRA INTERFACE REQUIREMENTS

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

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 1.

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

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,

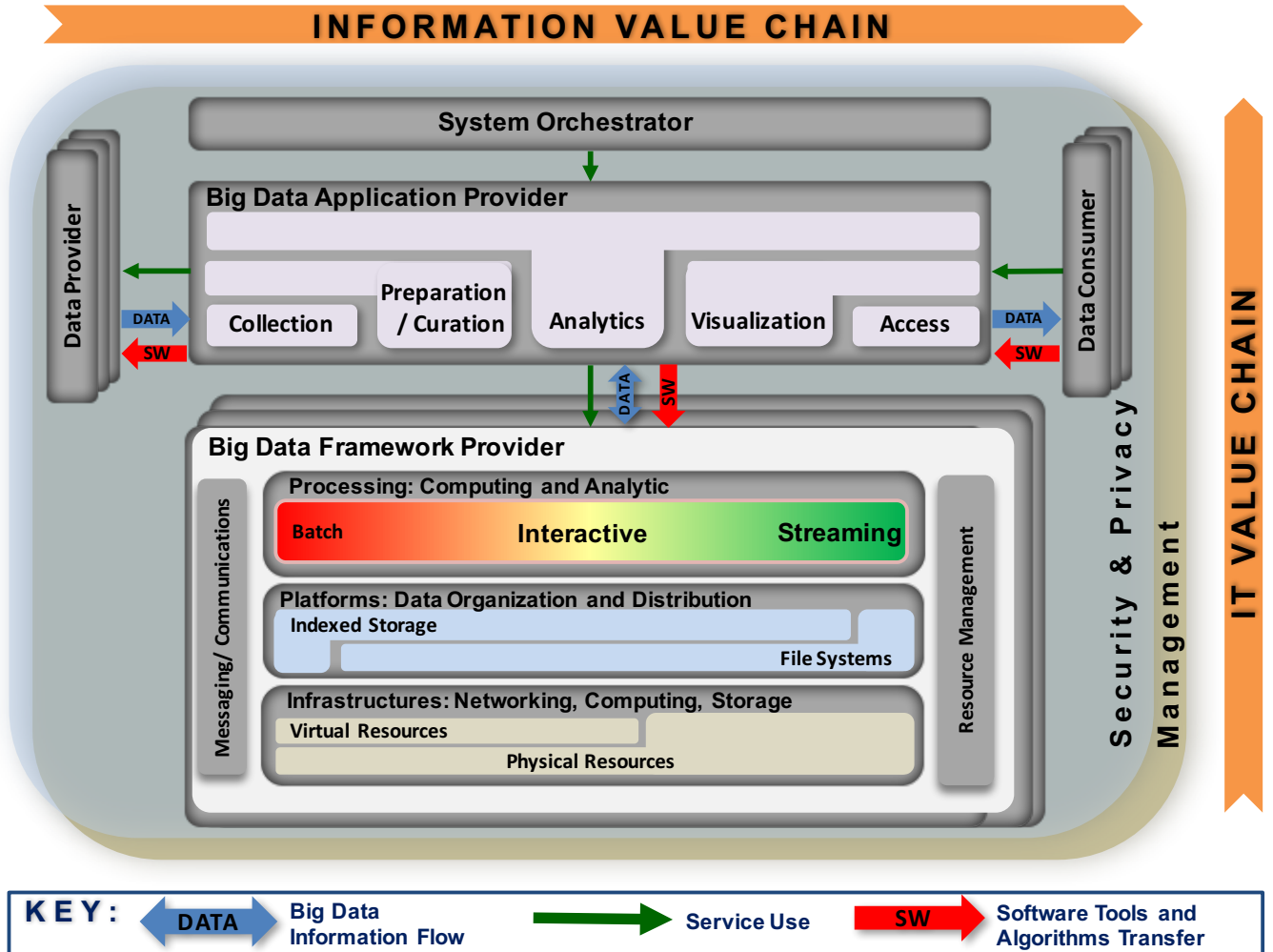


Figure 1: NIST Big Data Reference Architecture (NBDRA)

and clouds, (b) the ability to integrate special purpose hardware such as GPUs and FPGAs that are used in accelerated analysis of data, and (c) the integration of services including micro services that allow the analysis of the data by delegating them to hosted or dynamically deployed services on the infrastructure of choice.

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

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

3.1.7. Security and Privacy Fabric Requirements

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

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 1 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 Requirement

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

3.2.2. Data Provider Interface Requirement

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

141 3.2.3. Data Consumer Interface Requirement

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

148 3.2.4. Big Data Application Interface Provider Requirements

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

157 3.2.4.1 Collection

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

169 3.2.4.2 Preparation

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

179 3.2.4.3 Analytics

180 The analytics activity of the Big Data Application Provider includes the encoding of the low-level business logic
181 of the Big Data system (with higher-level business process logic being encoded by the System Orchestrator).
182 The activity implements the techniques to extract knowledge from the data based on the requirements of
183 the vertical application. The requirements specify the data processing algorithms for processing the data to
184 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. We must be able to provide the following functionality (1) interfaces to files (2) interfaces to virtual data directories (3) interfaces to data streams (4) and interfaces to data filters.

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. As part of the NBDRA we need 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 we need interfaces 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. We need to be able 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

A number of crosscutting interface requirements within the NBDRA provider frameworks include messaging, communication, and resource management. Often these services may actually be hidden from explicit interface use as they are part of larger systems that expose higher level functionality through their interfaces. However, it may be needed to expose such interfaces also on a lower level in case finer grained control is needed. We will identify the need for such crosscutting interface requirements form Volume 3 [4] of this series.

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 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. BD Application Provider to Framework Provider Interface

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

4. SPECIFICATION PARADIGM

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

4.1. Lessons Learned

Originally we used a full REST specification for defining the objets related to the NBDRA [11]. However, we found quickly that at this stage of the document it would introduce too complex of a notation framework.

This would result in (a) a considerable increase in length of this document (b) a more complex framework reducing participation and (c) a more complex framework for developing a reference implementation. Thus we have decided in this version of the document to introduce a design concept by example that is used to automatically create a schema as well as a reference implementation.

4.2. Hybrid and Multiple Frameworks

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

4.3. Design be Research Oriented Architecture

A resource-oriented architecture (ROA) is 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 team we use an approach to define objects and its interfaces by example. These examples can than be taken and a schema can generated from them automatically. The schema is added to the Appendix A.1 of the document.

While focusing first on examples it allows us to speed up our design process and simplify discussions about the objects and interfaces Hence, we eliminate getting lost in complex specifications. The process and specifications used in this document will also allow us to automatically create a implementation of the objects that can be integrated into a reference architecture as provided by for example the cloudmesh client and rest project [9][11].

An example object will demonstrate our approach. The following object defines a JSON object representing a user (see Object 4.1).

Object 4.1: Example object specification

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

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

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

```

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

```

293 Defined objects can also be embedded into other objects by using the *objectid* tag. This is later demonstrated
 294 between the profile and the user objects (see Objects 5.1 and 5.2).

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

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

299
 300

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

302 In our reference implementation these methods are provided automatically.

303 4.5. Interface Compliancy

304 Due to the easy extensibility of our objects and their implicit interfaces it is important to introduce a
 305 terminology that allows us to define interface compliancy. We define it as follows

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

309 **Partially Compliance:** These are reference implementations that provide partial compliance to the objects
 310 defined in this document. A version number will be added to assure the snapshot in time of the objects
 311 is associated with the version. This reference implementation will implement a partial list of the
 312 objects. A document is accompanied that lists all objects defined, but also lists the objects that are not
 313 defined by the reference architecture. A document will outline which objects and interfaces have been
 314 implemented.

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

318 Such documents can than be forwarded to the subgroup for further discussion and for possible future
 319 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 1. Instead we have grouped the objects by functional use as depicted summarized in Figure 2.

Figure 2: NIST Big Data Reference Architecture Interfaces

5.1. Identity

In a multiuser environment we need a simple mechanism of associating objects and data to a particular person or group. While we do not want to replace with our efforts more elaborate solutions such as proposed by eduPerson [5] or others, we need a very simple way of distinguishing users. Therefore we have introduced a number of simple objects including a profile and a user.

5.1.1. Profile

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

Object 5.1: Profile

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

```
11     }
12 }
332
```

333 5.1.2. User

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

Object 5.2: Organization



```
1 {
2     "user": {
3         "profile": "objectid:profile",
4         "roles": ["admin"]
5     }
6 }
337
```

338 5.1.3. Organization

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

Object 5.3: User



```
1 {
2     "organization": {
3         "users": [
4             "objectid:user"
5         ]
6     }
7 }
345
```

346 These concepts allow now the clear definition of various roles such as data provider, data consumer, data curator,
347 and others. It also would allow the creation of services that restrict data access by role, or organizational
348 affiliation.

349 5.1.4. Group/Role

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

Object 5.4: Group



```
1 {
2     "group": {
3         "name": "users",
4         "description": "This group contains all users",
5         "users": [
351
```

```

6         "objectid:user"
7     ]
8 }
9 }

```

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

Object 5.5: Role

```

1 {
2     "role": {
3         "name": "editor",
4         "description": "This role contains all editors",
5         "users": [
6             "objectid:user"
7         ]
8     }
9 }

```

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 be 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 object. Search and Filter functions are integrated to allow identifying objects from it.

381 **Databases** are traditional but also NoSQL databases.

382 **Collections** are agglomeration of any type of data.

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

385 5.2.1. *TimeStamp*

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

Object 5.6: Timestamp



```
1 {  
2   "timestamp": {  
3     "accessed": "1.1.2017:05:00:00:EST",  
4     "created": "1.1.2017:05:00:00:EST",  
5     "modified": "1.1.2017:05:00:00:EST"  
6   }  
7 }
```

389 5.2.2. *Var*

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

Object 5.7: Var



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

393 5.2.3. *Default*

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

Object 5.8: Default



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

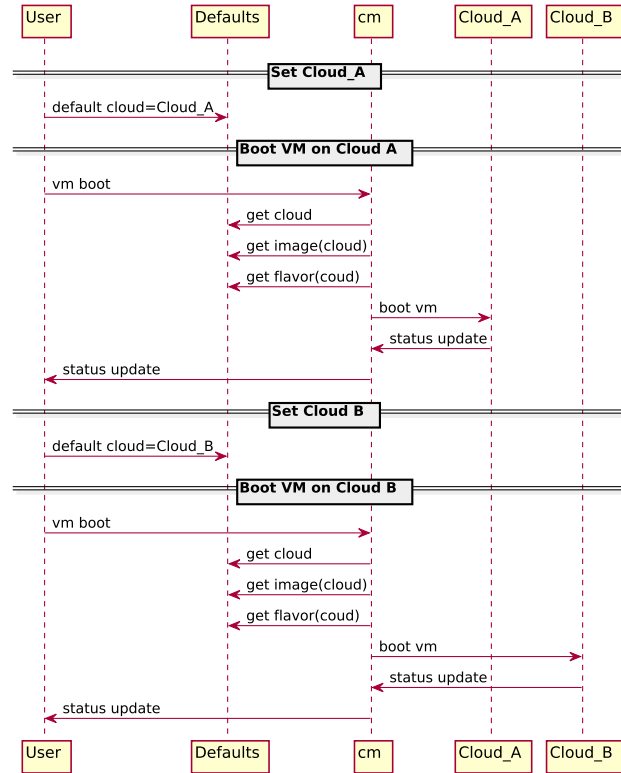


Figure 3: Booting a virtual machine from defaults

398 5.2.4. File

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

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

Object 5.9: File

```

1  {
2      "file": {
3          "name": "report.dat",
4          "endpoint": "file://gregor@machine.edu:/data/report.dat",
5          "checksum": {"sha256": "c01b39c7a35ccc ..... ebfeb45c69f08e17dfe3ef375a7b"},
6          "accessed": "1.1.2017:05:00:00:EST",
7          "created": "1.1.2017:05:00:00:EST",
8          "modified": "1.1.2017:05:00:00:EST",
9          "size": ["GB", "Byte"]
10     }
11 }
  
```

406 5.2.5. *Alias*

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

Object 5.10: File alias



```
1 {  
2   "alias": {  
3     "name": "a better name for the object",  
4     "origin": "the original object name"  
5   }  
6 }
```

411 5.2.6. *Replica*

412 In many distributed systems, it is of importance that a file can be replicated among different systems in order
413 to provide faster access. It is important to provide a mechanism that allows to trace the pedigree of the file
414 while pointing to its original source. A replica can be applied to all data types introduced in this document.

Object 5.11: Replica



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

416 5.2.7. *Virtual Directory*

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

Object 5.12: Virtual directory



```
1 {  
2   "virtual_directory": {  
3     "name": "data",  
4     "endpoint": "http://.../data/",  
5     "protocol": "http",  
6     "collection": [  
420
```

```

7         "report.dat",
8         "file2"
9     ]
10 }
11 }

```

421

422 5.2.8. Database

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

Object 5.13: Database



```

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

```

424

425 5.2.9. Stream

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

Object 5.14: Stream



```

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

```

429

430 Examples for streams could be a stream of random numbers but could also include more complex formats
431 such as the retrieval of data records. Services can subscribe, unsubscribe from a stream, while also applying
432 filters to the subscribed stream.

433 5.2.10. Filter

434 Filters can operate on a variety of objects and reduce and filter information based on a search criterion.

Object 5.15: Filter



```

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

```

435

```

5     }
6     }

```

436

437 5.3. Virtual Cluster

438 One of the essential features for Bid Data is the creation of a Big Data Analysis Cluster. A virtual cluster
 439 combines resources that generally ar used to serve the Big Data Application and can constitute a variety of
 440 data analysis nodes that together build the virtual cluster. Instead of focussing only on the deployment of a
 441 physical cluster the creation of a virtual cluster can be instantiated on a number of different platforms. Such
 442 a platforms can include clouds, containers, physical hardware or a mix thereof to support different aspects of
 443 the big data application.

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

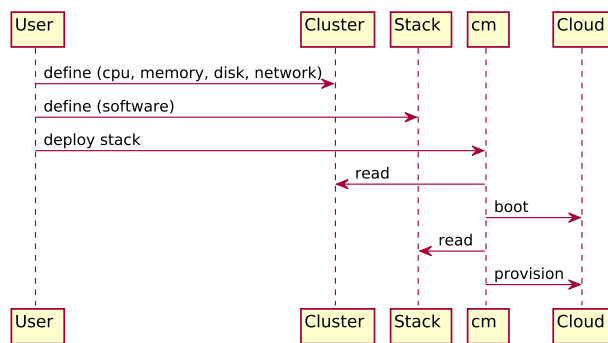


Figure 4: Allocating and provisioning a virtual cluster

449 5.3.1. Virtual Cluster

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

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

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

Object 5.16: Virtual cluster



```

1  {
2    "virtual_cluster": {
3      "name": "myvirtualcluster",
4      "label": "C0",
5      "uuid": "sgdlsjlaj....",

```

459


```

6     "endpoint": {
7         "passwd": "secret",
8         "url": "https:..."
9     },
10    "provider": "virtual_cluster_provider:openstack",
11    "frontend": "objectid:virtual_machine",
12    "authorized_keys": ["objectid:sshkey"],
13    "nodes": [
14        "objectid:virtual_machine"
15    ]
16 }
17 }

```

Object 5.17: Virtual cluster provider



```

1 "virtual_cluster_provider": [ "aws" | "azure" | "google" | "comet" | "openstack"
461

```

462 5.3.2. Compute Node

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

465 Compute nodes are a representation of a computer system (physical or virtual). We are maintaining a very
 466 basic set of information. It is expected that through the endpoint the virtual machine can be introspected
 467 and more detailed information can be retrieved. A compute node has name, label, a flavor, NICs and other
 468 relevant information.

Object 5.18: Compute node of a virtual cluster



```

1 {
2     "compute_node": {
3         "name": "vm1",
4         "label": "gregor-vm001",
5         "uuid": "sgklfgslakj...",
6         "kind": "vm",
7         "flavor": ["objectid:flavor"],
8         "image": "Ubuntu-16.04",
9         "secgroups": ["objectid:secgroup"],
10        "nics": ["objectid:nic"],
11        "status": "",
12        "loginuser": "ubuntu",
13        "status": "active",
14        "authorized_keys": ["objectid:sshkey"],
15        "metadata": {
16            "owner": "gregor",
17            "experiment": "exp-001"
18        }
19    }
20 }

```

470 5.3.3. Flavor

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

Object 5.19: Flavor



```
1 {  
2     "flavor": {  
3         "name": "flavor1",  
4         "label": "2-4G-40G",  
5         "uuid": "sgklfgslakj....",  
6         "ncpu": 2,  
7         "ram": "4G",  
8         "disk": "40G"  
9     }  
10 }
```

473

474 5.3.4. Nic

475 To interact between the node a network is needed. We specify such a network interface on a virtual machine
476 with a Nic (network interface card) object as showcased in Object 5.20.

Object 5.20: Network interface card



```
1 {  
2     "nic": {  
3         "name": "eth0",  
4         "type": "ethernet",  
5         "mac": "00:00:00:11:22:33",  
6         "ip": "123.123.1.2",  
7         "mask": "255.255.255.0",  
8         "broadcast": "123.123.1.255",  
9         "gateway": "123.123.1.1",  
10        "mtu": 1500,  
11        "bandwidth": "10Gbps"  
12    }  
13 }
```

477

478 5.3.5. Key

479 Many services and Frameworks use ssh keys to authenticate. To allow the convenient storage of the public
480 key the sshkey object can be used (see Object 5.21).

Object 5.21: Key



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

481

```

8     }
9 }

```

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 5.22.

Object 5.22: Security Groups

```

1 {
2   "secgroup":
3     {
4       "ingress": "0.0.0.0/32",
5       "egress": "0.0.0.0/32",
6       "ports": 22,
7       "protocols": "tcp"
8     }
9 }

```

5.4. IaaS

Although we have defined in Section 5.3 a general virtual cluster useful for Big Data, we are sometimes in the need to specifically utilize Infrastructure as a Service Frameworks such as Openstack, AWS, Azure, Google and others. To do so it is beneficial to be able to define virtual clusters using these frameworks. Hence, we define in this subsection interfaces related to Infrastructure as a Service frameworks. This includes specific objects useful for OpenStack, Azure, and AWS, as well as others. The definition of the objects between the clouds to manage them are different and not standardized. In this case the objects support functions such as starting, stopping, suspending resuming, migration, network configuration, assigning of resources, assigning of operating systems for and others for the virtual machines.

Learning from others such as *LibCloud* shows the definition of generalized objects, that however are augmented with extra fields to specifically integrate with the various frameworks. When working with Cloudmesh we found that it is sufficient to be able to specify a cloud based on a cloud specific action. Actions include boot, terminate, suspend, resume, assign network ips, add users.

To support such actions we can use objects that are used based on the type of the IaaS when invoked. We list such 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 we can use to interface with different IaaS frameworks. However, as these frameworks are all 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 Disadvantages

We have used LibCloud for some time practically in various versions of Cloudmesh. However we found 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. Thus for advanced applications we not only support the use of LibCloud, but also support the direct utilization of the native objects and

513 interfaces provided by a particular IaaS framework. For this reason we have introduced additional interfaces
514 as showcased in Sections 5.4.2 and 5.4.3. We intend to integrate additional sections addressing other IaaS
515 frameworks in future.

516 5.4.1.2 LibCloud Flavor

517 The object referring to flavors is listed in Object 5.23.

Object 5.23: Libcloud flavor



```
1 {  
2   "libcloud_flavor": {  
3     "bandwidth": "string",  
4     "disk": "string",  
5     "uuid": "string",  
6     "price": "string",  
7     "ram": "string",  
8     "cpu": "string",  
9     "flavor_id": "string"  
10  }  
11 }
```

518

519 5.4.1.3 LibCloud Image

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

Object 5.24: Libcloud image



```
1 {  
2   "libcloud_image": {  
3     "username": "string",  
4     "status": "string",  
5     "updated": "string",  
6     "description": "string",  
7     "owner_alias": "string",  
8     "kernel_id": "string",  
9     "ramdisk_id": "string",  
10    "image_id": "string",  
11    "is_public": "string",  
12    "image_location": "string",  
13    "uuid": "string",  
14    "created": "string",  
15    "image_type": "string",  
16    "hypervisor": "string",  
17    "platform": "string",  
18    "state": "string",  
19    "architecture": "string",  
20    "virtualization_type": "string",  
21    "owner_id": "string"  
22  }  
23 }
```

521

522 5.4.1.4 LibCloud VM

523 The object referring to virtual machines is listed in

Object 5.25: LibCloud VM



```
1 {
2   "libcloud_vm": {
3     "username": "string",
4     "status": "string",
5     "root_device_type": "string",
6     "image": "string",
7     "image_name": "string",
8     "image_id": "string",
9     "key": "string",
10    "flavor": "string",
11    "availability": "string",
12    "private_ips": "string",
13    "group": "string",
14    "uuid": "string",
15    "public_ips": "string",
16    "instance_id": "string",
17    "instance_type": "string",
18    "state": "string",
19    "root_device_name": "string",
20    "private_dns": "string"
21  }
22 }
```

524

525 5.4.1.5 LibCloud Node

526 Virtual machines for the various clouds have additional attributes that we summarize in Object 5.25. These
527 attributes are going to be integrated into the VM object.

Object 5.26: LibCloud Node



```
1 {
2   "LibCloudNode": {
3     "id": "instance_id",
4     "name": "name",
5     "state": "state",
6     "public_ips": ["111.222.111.1"],
7     "private_ips": ["192.168.1.101"],
8     "driver": "connection.driver",
9     "created_at": "created_timestamp",
10    "extra": {
11    }
12  },
13  "ec2NodeExtra": {
14    "block_device_mapping": "deviceMapping",
15    "groups": ["security_group1", "security_group2"],
16    "network_interfaces": ["nic1", "nic2"],

```

528

```

17     "product_codes": "product_codes",
18     "tags": ["tag1", "tag2"]
19 },
20 "OpenStackNodeExtra": {
21     "addresses": ["addresses"],
22     "hostId": "hostId",
23     "access_ip": "accessIPv4",
24     "access_ipv6": "accessIPv6",
25     "tenantId": "tenant_id",
26     "userId": "user_id",
27     "imageId": "image_id",
28     "flavorId": "flavor_id",
29     "uri": "",
30     "service_name": "",
31     "metadata": ["metadata"],
32     "password": "adminPass",
33     "created": "created",
34     "updated": "updated",
35     "key_name": "key_name",
36     "disk_config": "diskConfig",
37     "config_drive": "config_drive",
38     "availability_zone": "availability_zone",
39     "volumes_attached": "volumes_attached",
40     "task_state": "task_state",
41     "vm_state": "vm_state",
42     "power_state": "power_state",
43     "progress": "progress",
44     "fault": "fault"
45 },
46 "AzureNodeExtra": {
47     "instance_endpoints": "instance_endpoints",
48     "remote_desktop_port": "remote_desktop_port",
49     "ssh_port": "ssh_port",
50     "power_state": "power_state",
51     "instance_size": "instance_size",
52     "ex_cloud_service_name": "ex_cloud_service_name"
53 },
54 "GCENodeExtra": {
55     "status": "status",
56     "statusMessage": "statusMessage",
57     "description": "description",
58     "zone": "zone",
59     "image": "image",
60     "machineType": "machineType",
61     "disks": "disks",
62     "networkInterfaces": "networkInterfaces",
63     "id": "node_id",
64     "selfLink": "selfLink",
65     "kind": "kind",
66     "creationTimestamp": "creationTimestamp",

```

```

67     "name": "name",
68     "metadata": "metadata",
69     "tags_fingerprint": "fingerprint",
70     "scheduling": "scheduling",
71     "deprecated": "True or False",
72     "canIpForward": "canIpForward",
73     "serviceAccounts": "serviceAccounts",
74     "boot_disk": "disk"
75 }
76 }
530

```

531 5.4.2. Openstack

532 Objects related to OpenStack virtual machines are summarized in this section.

533 5.4.2.1 Openstack Flavor

534 The object referring to flavors is listed in Object 5.23.

Object 5.27: Openstack flavor



```

1  {
2    "openstack_flavor": {
3      "os_flv_disabled": "string",
4      "uuid": "string",
5      "os_flv_ext_data": "string",
6      "ram": "string",
7      "os_flavor_access": "string",
8      "vcpus": "string",
9      "swap": "string",
10     "rxtx_factor": "string",
11     "disk": "string"
12   }
13 }
535

```

536 5.4.2.2 Openstack Image

537 The object referring to images is listed in Object 5.28.

Object 5.28: Openstack image



```

1  {
2    "openstack_image": {
3      "status": "string",
4      "username": "string",
5      "updated": "string",
6      "uuid": "string",
7      "created": "string",
8      "minDisk": "string",
9      "progress": "string",
10     "minRam": "string",
11     "os_image_size": "string",
538

```

```

12     "metadata": {
13         "image_location": "string",
14         "image_state": "string",
15         "description": "string",
16         "kernel_id": "string",
17         "instance_type_id": "string",
18         "ramdisk_id": "string",
19         "instance_type_name": "string",
20         "instance_type_rxtx_factor": "string",
21         "instance_type_vcpus": "string",
22         "user_id": "string",
23         "base_image_ref": "string",
24         "instance_uuid": "string",
25         "instance_type_memory_mb": "string",
26         "instance_type_swap": "string",
27         "image_type": "string",
28         "instance_type_ephemeral_gb": "string",
29         "instance_type_root_gb": "string",
30         "network_allocated": "string",
31         "instance_type_flavorid": "string",
32         "owner_id": "string"
33     }
34 }
35 }

```

539

540 5.4.2.3 Openstack Vm

541 The object referring to virtual machines is listed in Object 5.29.

Object 5.29: Openstack vm



```

1 {
2     "openstack_vm": {
3         "username": "string",
4         "vm_state": "string",
5         "updated": "string",
6         "hostId": "string",
7         "availability_zone": "string",
8         "terminated_at": "string",
9         "image": "string",
10        "floating_ip": "string",
11        "diskConfig": "string",
12        "key": "string",
13        "flavor__id": "string",
14        "user_id": "string",
15        "flavor": "string",
16        "static_ip": "string",
17        "security_groups": "string",
18        "volumes_attached": "string",
19        "task_state": "string",
20        "group": "string",

```

542


```

21     "uuid": "string",
22     "created": "string",
23     "tenant_id": "string",
24     "accessIPv4": "string",
25     "accessIPv6": "string",
26     "status": "string",
27     "power_state": "string",
28     "progress": "string",
29     "image__id": "string",
30     "launched_at": "string",
31     "config_drive": "string"
32 }
33 }

```

543

544 5.4.3. Azure

545 Objects related to OpenStack virtual machines are summarized in this section.

546 5.4.3.1 Azure Size

547 The object referring to the image size machines is liste in Object 5.30.

Object 5.30: Azure-size



```

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

```

548

549 5.4.3.2 Azure Image

550 The object referring to the images machines is liste in Object 5.31.

Object 5.31: Azure-image



```

1 {
2   "azure_image": {
3     "_uuid": "None",
4     "driver": "libcloud",
5     "extra": {

```

551

```

6     "affinity_group": "",
7     "category": "Public",
8     "description": "Linux VM image with coreclr-x64-beta5-11624 installed to
↵ /opt/dnx. This image is based on Ubuntu 14.04 LTS, with prerequisites of CoreCLR
↵ installed. It also contains PartsUnlimited demo app which runs on the installed
↵ coreclr. The demo app is installed to /opt/demo. To run the demo, please type the
↵ command /opt/demo/Kestrel in a terminal window. The website is listening on port
↵ 5004. Please enable or map a endpoint of HTTP port 5004 for your azure VM.",
9     "location": "East Asia;Southeast Asia;Australia East;Australia Southeast;Brazil
↵ South;North Europe;West Europe;Japan East;Japan West;Central US;East US;East US 2;
↵ North Central US;South Central US;West US",
10    "media_link": "",
11    "os": "Linux",
12    "vm_image": "False"
13  },
14  "id": "03f55de797f546a1b29d1....",
15  "name": "CoreCLR x64 Beta5 (11624) with PartsUnlimited Demo App on Ubuntu Server
↵ 14.04 LTS"
16  }
17  }

```

552

553 5.4.3.3 Azure Vm

554 The object referring to the virtual machines is liste in Object 5.32.

Object 5.32: Azure-vm



```

1  {
2    "azure-vm": {
3      "username": "string",
4      "status": "string",
5      "deployment_slot": "string",
6      "cloud_service": "string",
7      "image": "string",
8      "floating_ip": "string",
9      "image_name": "string",
10     "key": "string",
11     "flavor": "string",
12     "resource_location": "string",
13     "disk_name": "string",
14     "private_ips": "string",
15     "group": "string",
16     "uuid": "string",
17     "dns_name": "string",
18     "instance_size": "string",
19     "instance_name": "string",
20     "public_ips": "string",
21     "media_link": "string"
22   }
23 }

```

555

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 today's 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 latest applications.

Object 5.33: Batchjob

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

5.5.2. Reservation

Some services may consume a considerable amount of resources. In order to allow utilization we need to reserve their use. For this purpose we have introduced a reservation object (see Object 5.34).

Object 5.34: Reservation

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

5.6. Containers

This defines *container* object.

Object 5.35: Container



```

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

```

573

5.7. Deployment

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

Object 5.36: Deployment



```

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

```

577

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.

Object 5.37: Mapreduce



```

1 {
2   "mapreduce": {
3     "function": {

```

584

```

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

```

585

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

591 A function prototype is defined in Listing 5.38. Key properties are that functions describe their input
 592 parameters and generated results. For the former, the “buildInputs” and “systemBuildInputs” respectively
 593 describe the objects which should be evaluated and system packages which should be present before this
 594 function can be installed. The “eval” attribute describes how to apply this function to its input data.
 595 Parameters affecting the evaluation of the function may be passed in as the “args” attribute. The results of
 596 the function application can be accessed via the “outputs” object, which is a mapping from arbitrary keys
 597 (e.g. “data”, “processed”, “model”) to an object representing the result.

Object 5.38: Mapreduce function



```

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

```

598

```

25     "outputs": {
26         "key": "value"
27     }
28 }
29 }

```

Some example functions include the “NoOp” function shown in Listing 5.39. In the case of undefined arguments, the parameters default to an identity element. In the case of mappings this is the empty mapping while for lists this is the empty list.

Object 5.39: Mapreduce noop

```

1  {
2      "mapreduce_noop": {
3          "name": "noop",
4          "description": "A function with no effect"
5      }
6  }

```

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 5.40)

Object 5.40: Hadoop

```

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

```

609 5.9. Microservice

610 As part of microservices, we are defining a function with parameters that can be invoked. To describe such
611 services we have defined the Object 5.41. As we can define multiple such services we can easily find them
612 and use them as part of a microservice based implementation.

Object 5.41: Microservice



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

613

614 5.9.1. Accounting

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

Object 5.42: Accounting



```
1 {  
2   "accounting_resource": {  
3     "description": "The Description of a resource that we apply accounting to",  
4     "uuid": "unique uuid for this resource",  
5     "name": "the name of the resource",  
6     "charge": "1.1 * parameter1 + 3.1 * parameter2",  
7     "parameters": {"parameter1": 1.0,  
8                     "parameter2": 1.0},  
9     "unites": {"parameter1": "GB",  
10                "parameter2": "cores"},  
11     "user": "username",  
12     "group": "groupname",  
13     "account": "accountname"  
14   }  
15 }
```

623

Object 5.43: Account



```
1 {  
2   "account": {  
3     "description": "The Description of the account",  
4     "uuid": "unique uuid for this resource",  
5     "name": "the name of the account",  
6     "startDate": "10/10/2017:00:00:00",  
7   }  
8 }
```

624

```

7      "endDate": "10/10/2017:00:00:00",
8      "status": "one of active, suspended, closed",
9      "balance": 1.0,
10     "user": ["username"],
11     "group": ["groupname"]
12   }
13 }
625

```

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 then invoked by the user can then consume the resource and are charged accordingly.

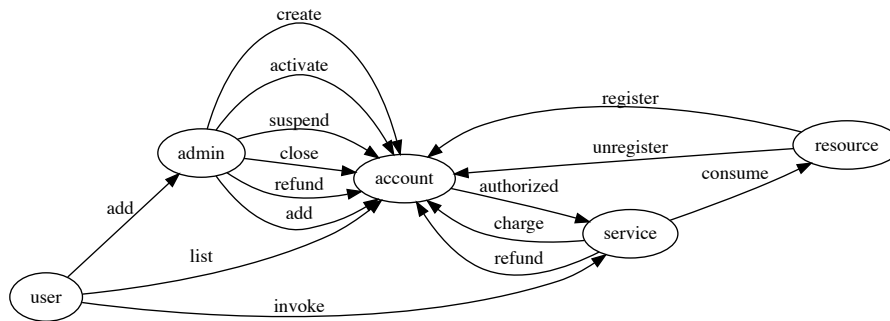


Figure 5: Create Resource

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

6.1. Acronyms and Terms

The following acronyms and terms are used in the paper

ACID	Atomicity, Consistency, Isolation, Durability
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange

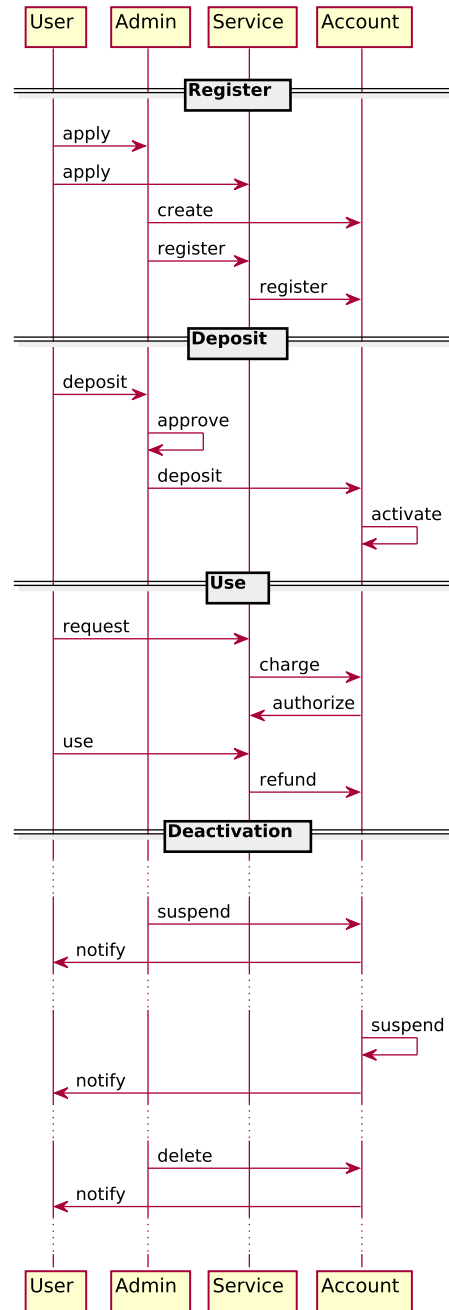


Figure 6: Accounting

- 641 **BASE** Basically Available, Soft state, Eventual consistency
- 642 **Container** see http://csrc.nist.gov/publications/drafts/800-180/sp800-180_draft.pdf
- 643 **Cloud Computing**
- 644 the practice of using a network of remote servers hosted on the Internet to store, manage,
- 645 and process data, rather than a local server or a personal computer. See <http://nvlpubs>.

Table 1: HTTP response codes

HTTP response code	Description
200	<i>OK</i> success code, for GET or HEAD request.
201	<i>Created</i> success code, for POST request.
204	<i>No Content</i> 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.

646		<code>nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf</code>
647	DevOps	A clipped compound of <i>software DEVELOPMENT</i> and <i>information technology OPERATIONS</i>
648	Deployment	The action of installing software on resources.
649	HTTP	HyperText Transfer Protocol HTTPS HTTP Secure
650	Hybrid Cloud	See <code>http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf</code>
651		
652	IaaS	Infrastructure as a Service SaaS Software as a Service
653	ITL	Information Technology Laboratory
654	Microservice Architecture	
655		Is an approach to build applications based on many smaller modular services. Each module
656		supports a specific goal and uses a simple, well-defined interface to communicate with
657		other sets of services.
658	NBD-PWG	NIST Big Data Public Working Group
659	NBDRA	NIST Big Data Reference Architecture
660	NBDRAI	NIST Big Data Reference Architecture Interface
661	NIST	National Institute of Standards
662	OS	Operating System
663	REST	REpresentational State Transfer
664	Replica	A duplicate of a file on another resource in order to avoid costly transfer costs in case of
665		frequent access.
666	Serverless Computing	
667		Serverless computing specifies the paradigm of function as a service (FaaS). It is a
668		cloud computing code execution model in which a cloud provider manages the function
669		deployment and utilization while clients can utilize them. The charge model is based on
670		execution of the function rather than the cost to manage and host the virtual machine or
671		container.

672	Software Stack	A set of programs and services that are installed on a resource in order to support
673		applications.
674	Virtual Filesystem	
675		An abstraction layer on top of a distributed physical file system to allow easy access to
676		the files by the user or application.
677	Virtual Machine	
678		A virtual machine is a software computer that, like a physical computer, runs an operating
679		system and applications. The virtual machine is comprised of a set of specification and
680		configuration files and is backed by the physical resources of a host.
681	Virtual Cluster	
682		A virtual cluster is a software cluster that integrate either virtual machines, containers or
683		physical resources into an agglomeration of compute resources. A virtual cluster allows
684		user to authenticate and authorize to the virtual compute nodes to utilize them for
685		calculations. Optional high level services that can be deployed on a virtual cluster may
686		simplify interaction with the virtual cluster or provide higher level services.
687	Workflow	the sequence of processes or tasks
688	WWW	World Wide Web

A. APPENDIX

A.1. Schema

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

Object A.1: Schema

```
1  container = {
2      'schema': {
3          'ip': {
4              'type': 'string'
5          },
6          'endpoint': {
7              'type': 'string'
8          },
9          'name': {
10             'type': 'string'
11         },
12         'memoryGB': {
13             'type': 'integer'
14         },
15         'label': {
16             'type': 'string'
17         }
18     }
19 }
20
21 stream = {
22     'schema': {
23         'attributes': {
24             'type': 'dict',
25             'schema': {
26                 'rate': {
27                     'type': 'integer'
28                 },
29                 'limit': {
30                     'type': 'integer'
31                 }
32             }
33         },
34         'name': {
35             'type': 'string'
36         },
37         'format': {
38             'type': 'string'
39         }
40     }
41 }
42
43 azure_image = {
44     'schema': {
```

```

45     '_uuid': {
46         'type': 'string'
47     },
48     'driver': {
49         'type': 'string'
50     },
51     'id': {
52         'type': 'string'
53     },
54     'name': {
55         'type': 'string'
56     },
57     'extra': {
58         'type': 'dict',
59         'schema': {
60             'category': {
61                 'type': 'string'
62             },
63             'description': {
64                 'type': 'string'
65             },
66             'vm_image': {
67                 'type': 'string'
68             },
69             'location': {
70                 'type': 'string'
71             },
72             'affinity_group': {
73                 'type': 'string'
74             },
75             'os': {
76                 'type': 'string'
77             },
78             'media_link': {
79                 'type': 'string'
80             }
81         }
82     }
83 }
84
85
86 deployment = {
87     'schema': {
88         'cluster': {
89             'type': 'list',
90             'schema': {
91                 'type': 'dict',
92                 'schema': {
93                     'id': {
94                         'type': 'string'

```

```

95         }
96     }
97 }
98 },
99 'stack': {
100     'type': 'dict',
101     'schema': {
102         'layers': {
103             'type': 'list',
104             'schema': {
105                 'type': 'string'
106             }
107         },
108         'parameters': {
109             'type': 'dict',
110             'schema': {
111                 'hadoop': {
112                     'type': 'dict',
113                     'schema': {
114                         'zookeeper.quorum': {
115                             'type': 'list',
116                             'schema': {
117                                 'type': 'string'
118                             }
119                         }
120                     }
121                 }
122             }
123         }
124     }
125 }
126 }
127 }
128
129 azure_size = {
130     'schema': {
131         'ram': {
132             'type': 'integer'
133         },
134         'name': {
135             'type': 'string'
136         },
137         'extra': {
138             'type': 'dict',
139             'schema': {
140                 'cores': {
141                     'type': 'integer'
142                 },
143                 'max_data_disks': {
144                     'type': 'integer'

```

```

145         }
146     }
147 },
148     'price': {
149         'type': 'float'
150     },
151     '_uuid': {
152         'type': 'string'
153     },
154     'driver': {
155         'type': 'string'
156     },
157     'bandwidth': {
158         'type': 'string'
159     },
160     'disk': {
161         'type': 'integer'
162     },
163     'id': {
164         'type': 'string'
165     }
166 }
167 }
168
169 cluster = {
170     'schema': {
171         'provider': {
172             'type': 'list',
173             'schema': {
174                 'type': 'string'
175             }
176         },
177         'endpoint': {
178             'type': 'dict',
179             'schema': {
180                 'passwd': {
181                     'type': 'string'
182                 },
183                 'url': {
184                     'type': 'string'
185                 }
186             }
187         },
188         'name': {
189             'type': 'string'
190         },
191         'label': {
192             'type': 'string'
193         }
194     }

```

695

```

195 }
196
197 computer = {
198     'schema': {
199         'ip': {
200             'type': 'string'
201         },
202         'name': {
203             'type': 'string'
204         },
205         'memoryGB': {
206             'type': 'integer'
207         },
208         'label': {
209             'type': 'string'
210         }
211     }
212 }
213
214 mesos_docker = {
215     'schema': {
216         'container': {
217             'type': 'dict',
218             'schema': {
219                 'docker': {
220                     'type': 'dict',
221                     'schema': {
222                         'credential': {
223                             'type': 'dict',
224                             'schema': {
225                                 'secret': {
226                                     'type': 'string'
227                                 },
228                                 'principal': {
229                                     'type': 'string'
230                                 }
231                             }
232                         },
233                         'image': {
234                             'type': 'string'
235                         }
236                     }
237                 },
238                 'type': {
239                     'type': 'string'
240                 }
241             }
242         },
243         'mem': {
244             'type': 'float'

```

696


```

245     },
246     'args': {
247         'type': 'list',
248         'schema': {
249             'type': 'string'
250         }
251     },
252     'cpus': {
253         'type': 'float'
254     },
255     'instances': {
256         'type': 'integer'
257     },
258     'id': {
259         'type': 'string'
260     }
261 }
262 }
263
264 file = {
265     'schema': {
266         'endpoint': {
267             'type': 'string'
268         },
269         'name': {
270             'type': 'string'
271         },
272         'created': {
273             'type': 'string'
274         },
275         'checksum': {
276             'type': 'dict',
277             'schema': {
278                 'sha256': {
279                     'type': 'string'
280                 }
281             }
282         },
283         'modified': {
284             'type': 'string'
285         },
286         'accessed': {
287             'type': 'string'
288         },
289         'size': {
290             'type': 'list',
291             'schema': {
292                 'type': 'string'
293             }
294         }

```

697

```

295     }
296 }
297
298 reservation = {
299     'schema': {
300         'start_time': {
301             'type': 'list',
302             'schema': {
303                 'type': 'string'
304             }
305         },
306         'description': {
307             'type': 'string'
308         },
309         'service': {
310             'type': 'string'
311         },
312         'end_time': {
313             'type': 'list',
314             'schema': {
315                 'type': 'string'
316             }
317         }
318     }
319 }
320
321 microservice = {
322     'schema': {
323         'function': {
324             'type': 'string'
325         },
326         'endpoint': {
327             'type': 'string'
328         },
329         'name': {
330             'type': 'string'
331         }
332     }
333 }
334
335 flavor = {
336     'schema': {
337         'uuid': {
338             'type': 'string'
339         },
340         'ram': {
341             'type': 'string'
342         },
343         'label': {
344             'type': 'string'

```

698

```

345     },
346     'ncpu': {
347         'type': 'integer'
348     },
349     'disk': {
350         'type': 'string'
351     },
352     'name': {
353         'type': 'string'
354     }
355 }
356 }
357
358 virtual_directory = {
359     'schema': {
360         'endpoint': {
361             'type': 'string'
362         },
363         'protocol': {
364             'type': 'string'
365         },
366         'name': {
367             'type': 'string'
368         },
369         'collection': {
370             'type': 'list',
371             'schema': {
372                 'type': 'string'
373             }
374         }
375     }
376 }
377
378 mapreduce_function = {
379     'schema': {
380         'name': {
381             'type': 'string'
382         },
383         'outputs': {
384             'type': 'dict',
385             'schema': {
386                 'key': {
387                     'type': 'string'
388                 }
389             }
390         },
391         'args': {
392             'type': 'list',
393             'schema': {
394                 'type': 'dict',

```

699

```

395         'schema': {
396             'argument': {
397                 'type': 'string'
398             }
399         }
400     },
401 },
402 'systemBuildInputs': {
403     'type': 'list',
404     'schema': {
405         'type': 'string'
406     }
407 },
408 'source': {
409     'type': 'string'
410 },
411 'install': {
412     'type': 'dict',
413     'schema': {
414         'description': {
415             'type': 'string'
416         },
417         'script': {
418             'type': 'string'
419         }
420     }
421 },
422 'eval': {
423     'type': 'dict',
424     'schema': {
425         'description': {
426             'type': 'string'
427         },
428         'script': {
429             'type': 'string'
430         }
431     }
432 },
433 'buildInputs': {
434     'type': 'list',
435     'schema': {
436         'type': 'string'
437     }
438 },
439 'description': {
440     'type': 'string'
441 }
442 }
443 }

```

```

445 virtual_cluster = {
446     'schema': {
447         'authorized_keys': {
448             'type': 'list',
449             'schema': {
450                 'type': 'objectid',
451                 'data_relation': {
452                     'resource': 'sshkey',
453                     'field': '_id',
454                     'embeddable': True
455                 }
456             }
457         },
458         'endpoint': {
459             'type': 'dict',
460             'schema': {
461                 'passwd': {
462                     'type': 'string'
463                 },
464                 'url': {
465                     'type': 'string'
466                 }
467             }
468         },
469         'frontend': {
470             'type': 'objectid',
471             'data_relation': {
472                 'resource': 'virtual_machine',
473                 'field': '_id',
474                 'embeddable': True
475             }
476         },
477         'uuid': {
478             'type': 'string'
479         },
480         'label': {
481             'type': 'string'
482         },
483         'provider': {
484             'type': 'string'
485         },
486         'nodes': {
487             'type': 'list',
488             'schema': {
489                 'type': 'objectid',
490                 'data_relation': {
491                     'resource': 'virtual_machine',
492                     'field': '_id',
493                     'embeddable': True
494                 }

```

701

```

495         }
496     },
497     'name': {
498         'type': 'string'
499     }
500 }
501 }
502
503 libcloud_flavor = {
504     'schema': {
505         'uuid': {
506             'type': 'string'
507         },
508         'price': {
509             'type': 'string'
510         },
511         'ram': {
512             'type': 'string'
513         },
514         'bandwidth': {
515             'type': 'string'
516         },
517         'flavor_id': {
518             'type': 'string'
519         },
520         'disk': {
521             'type': 'string'
522         },
523         'cpu': {
524             'type': 'string'
525         }
526     }
527 }
528
529 LibCloudNode = {
530     'schema': {
531         'private_ips': {
532             'type': 'list',
533             'schema': {
534                 'type': 'string'
535             }
536         },
537         'extra': {
538             'type': 'dict',
539             'schema': {}
540         },
541         'created_at': {
542             'type': 'string'
543         },
544         'driver': {

```

702

```

545         'type': 'string'
546     },
547     'state': {
548         'type': 'string'
549     },
550     'public_ips': {
551         'type': 'list',
552         'schema': {
553             'type': 'string'
554         }
555     },
556     'id': {
557         'type': 'string'
558     },
559     'name': {
560         'type': 'string'
561     }
562 }
563 }
564
565 sshkey = {
566     'schema': {
567         'comment': {
568             'type': 'string'
569         },
570         'source': {
571             'type': 'string'
572         },
573         'uri': {
574             'type': 'string'
575         },
576         'value': {
577             'type': 'string'
578         },
579         'fingerprint': {
580             'type': 'string'
581         }
582     }
583 }
584
585 timestamp = {
586     'schema': {
587         'accessed': {
588             'type': 'string'
589         },
590         'modified': {
591             'type': 'string'
592         },
593         'created': {
594             'type': 'string'

```

703

```

595     }
596   }
597 }
598
599 mapreduce_noop = {
600   'schema': {
601     'name': {
602       'type': 'string'
603     },
604     'description': {
605       'type': 'string'
606     }
607   }
608 }
609
610 role = {
611   'schema': {
612     'users': {
613       'type': 'list',
614       'schema': {
615         'type': 'objectid',
616         'data_relation': {
617           'resource': 'user',
618           'field': '_id',
619           'embeddable': True
620         }
621       }
622     },
623     'name': {
624       'type': 'string'
625     },
626     'description': {
627       'type': 'string'
628     }
629   }
630 }
631
632 AzureNodeExtra = {
633   'schema': {
634     'ssh_port': {
635       'type': 'string'
636     },
637     'instance_size': {
638       'type': 'string'
639     },
640     'remote_desktop_port': {
641       'type': 'string'
642     },
643     'ex_cloud_service_name': {
644       'type': 'string'

```

704


```

645     },
646     'power_state': {
647         'type': 'string'
648     },
649     'instance_endpoints': {
650         'type': 'string'
651     }
652 }
653 }
654
655 var = {
656     'schema': {
657         'type': {
658             'type': 'string'
659         },
660         'name': {
661             'type': 'string'
662         },
663         'value': {
664             'type': 'string'
665         }
666     }
667 }
668
669 profile = {
670     'schema': {
671         'username': {
672             'type': 'string'
673         },
674         'context': {
675             'type': 'string'
676         },
677         'description': {
678             'type': 'string'
679         },
680         'firstname': {
681             'type': 'string'
682         },
683         'lastname': {
684             'type': 'string'
685         },
686         'publickey': {
687             'type': 'string'
688         },
689         'email': {
690             'type': 'string'
691         },
692         'uuid': {
693             'type': 'string'
694         }

```

705

```

695     }
696 }
697
698 virtual_machine = {
699     'schema': {
700         'status': {
701             'type': 'string'
702         },
703         'authorized_keys': {
704             'type': 'list',
705             'schema': {
706                 'type': 'objectid',
707                 'data_relation': {
708                     'resource': 'sshkey',
709                     'field': '_id',
710                     'embeddable': True
711                 }
712             }
713         },
714         'name': {
715             'type': 'string'
716         },
717         'nics': {
718             'type': 'list',
719             'schema': {
720                 'type': 'objectid',
721                 'data_relation': {
722                     'resource': 'nic',
723                     'field': '_id',
724                     'embeddable': True
725                 }
726             }
727         },
728         'RAM': {
729             'type': 'string'
730         },
731         'ncpu': {
732             'type': 'integer'
733         },
734         'loginuser': {
735             'type': 'string'
736         },
737         'disk': {
738             'type': 'string'
739         },
740         'OS': {
741             'type': 'string'
742         },
743         'metadata': {
744             'type': 'dict',

```

706

```

745         'schema': {}
746     }
747 }
748 }
749
750 kubernetes = {
751     'schema': {
752         'items': {
753             'type': 'list',
754             'schema': {
755                 'type': 'dict',
756                 'schema': {
757                     'status': {
758                         'type': 'dict',
759                         'schema': {
760                             'capacity': {
761                                 'type': 'dict',
762                                 'schema': {
763                                     'cpu': {
764                                         'type': 'string'
765                                     }
766                                 }
767                             },
768                             'addresses': {
769                                 'type': 'list',
770                                 'schema': {
771                                     'type': 'dict',
772                                     'schema': {
773                                         'type': {
774                                             'type': 'string'
775                                         },
776                                         'address': {
777                                             'type': 'string'
778                                         }
779                                     }
780                                 }
781                             }
782                         },
783                     'kind': {
784                         'type': 'string'
785                     },
786                     'metadata': {
787                         'type': 'dict',
788                         'schema': {
789                             'name': {
790                                 'type': 'string'
791                             }
792                         }
793                     }
794                 }

```

707

```

795         }
796     }
797 },
798 'kind': {
799     'type': 'string'
800 },
801 'users': {
802     'type': 'list',
803     'schema': {
804         'type': 'dict',
805         'schema': {
806             'name': {
807                 'type': 'string'
808             },
809             'user': {
810                 'type': 'dict',
811                 'schema': {
812                     'username': {
813                         'type': 'string'
814                     },
815                     'password': {
816                         'type': 'string'
817                     }
818                 }
819             }
820         }
821     }
822 }
823 }
824 }
825
826 nic = {
827     'schema': {
828         'name': {
829             'type': 'string'
830         },
831         'ip': {
832             'type': 'string'
833         },
834         'mask': {
835             'type': 'string'
836         },
837         'bandwidth': {
838             'type': 'string'
839         },
840         'mtu': {
841             'type': 'integer'
842         },
843         'broadcast': {
844             'type': 'string'

```

708

```

845     },
846     'mac': {
847         'type': 'string'
848     },
849     'type': {
850         'type': 'string'
851     },
852     'gateway': {
853         'type': 'string'
854     }
855 }
856 }
857
858 openstack_flavor = {
859     'schema': {
860         'os_flv_disabled': {
861             'type': 'string'
862         },
863         'uuid': {
864             'type': 'string'
865         },
866         'os_flv_ext_data': {
867             'type': 'string'
868         },
869         'ram': {
870             'type': 'string'
871         },
872         'os_flavor_acces': {
873             'type': 'string'
874         },
875         'vcpus': {
876             'type': 'string'
877         },
878         'swap': {
879             'type': 'string'
880         },
881         'rxtx_factor': {
882             'type': 'string'
883         },
884         'disk': {
885             'type': 'string'
886         }
887     }
888 }
889
890 azure_vm = {
891     'schema': {
892         'username': {
893             'type': 'string'
894         },

```

709

```

895     'status': {
896         'type': 'string'
897     },
898     'deployment_slot': {
899         'type': 'string'
900     },
901     'group': {
902         'type': 'string'
903     },
904     'private_ips': {
905         'type': 'string'
906     },
907     'cloud_service': {
908         'type': 'string'
909     },
910     'dns_name': {
911         'type': 'string'
912     },
913     'image': {
914         'type': 'string'
915     },
916     'floating_ip': {
917         'type': 'string'
918     },
919     'image_name': {
920         'type': 'string'
921     },
922     'instance_name': {
923         'type': 'string'
924     },
925     'public_ips': {
926         'type': 'string'
927     },
928     'media_link': {
929         'type': 'string'
930     },
931     'key': {
932         'type': 'string'
933     },
934     'flavor': {
935         'type': 'string'
936     },
937     'resource_location': {
938         'type': 'string'
939     },
940     'instance_size': {
941         'type': 'string'
942     },
943     'disk_name': {
944         'type': 'string'

```

710

```

945     },
946     'uuid': {
947         'type': 'string'
948     }
949 }
950 }
951
952 ec2NodeExtra = {
953     'schema': {
954         'product_codes': {
955             'type': 'string'
956         },
957         'tags': {
958             'type': 'list',
959             'schema': {
960                 'type': 'string'
961             }
962         },
963         'network_interfaces': {
964             'type': 'list',
965             'schema': {
966                 'type': 'string'
967             }
968         },
969         'groups': {
970             'type': 'list',
971             'schema': {
972                 'type': 'string'
973             }
974         },
975         'block_device_mapping': {
976             'type': 'string'
977         }
978     }
979 }
980
981 libcloud_image = {
982     'schema': {
983         'username': {
984             'type': 'string'
985         },
986         'status': {
987             'type': 'string'
988         },
989         'updated': {
990             'type': 'string'
991         },
992         'description': {
993             'type': 'string'
994         },

```

711

```

995         'owner_alias': {
996             'type': 'string'
997         },
998         'kernel_id': {
999             'type': 'string'
1000         },
1001         'hypervisor': {
1002             'type': 'string'
1003         },
1004         'ramdisk_id': {
1005             'type': 'string'
1006         },
1007         'state': {
1008             'type': 'string'
1009         },
1010         'created': {
1011             'type': 'string'
1012         },
1013         'image_id': {
1014             'type': 'string'
1015         },
1016         'image_location': {
1017             'type': 'string'
1018         },
1019         'platform': {
1020             'type': 'string'
1021         },
1022         'image_type': {
1023             'type': 'string'
1024         },
1025         'is_public': {
1026             'type': 'string'
1027         },
1028         'owner_id': {
1029             'type': 'string'
1030         },
1031         'architecture': {
1032             'type': 'string'
1033         },
1034         'virtualization_type': {
1035             'type': 'string'
1036         },
1037         'uuid': {
1038             'type': 'string'
1039         }
1040     }
1041 }
1042
1043 user = {
1044     'schema': {

```

712


```

1045     'profile': {
1046         'type': 'objectid',
1047         'data_relation': {
1048             'resource': 'profile',
1049             'field': '_id',
1050             'embeddable': True
1051         }
1052     },
1053     'roles': {
1054         'type': 'list',
1055         'schema': {
1056             'type': 'string'
1057         }
1058     }
1059 }
1060 }
1061
1062 GCENodeExtra = {
1063     'schema': {
1064         'status': {
1065             'type': 'string'
1066         },
1067         'kind': {
1068             'type': 'string'
1069         },
1070         'machineType': {
1071             'type': 'string'
1072         },
1073         'description': {
1074             'type': 'string'
1075         },
1076         'zone': {
1077             'type': 'string'
1078         },
1079         'deprecated': {
1080             'type': 'string'
1081         },
1082         'image': {
1083             'type': 'string'
1084         },
1085         'disks': {
1086             'type': 'string'
1087         },
1088         'tags_fingerprint': {
1089             'type': 'string'
1090         },
1091         'name': {
1092             'type': 'string'
1093         },
1094         'boot_disk': {

```

713

```

1095         'type': 'string'
1096     },
1097     'selfLink': {
1098         'type': 'string'
1099     },
1100     'scheduling': {
1101         'type': 'string'
1102     },
1103     'canIpForward': {
1104         'type': 'string'
1105     },
1106     'serviceAccounts': {
1107         'type': 'string'
1108     },
1109     'metadata': {
1110         'type': 'string'
1111     },
1112     'creationTimestamp': {
1113         'type': 'string'
1114     },
1115     'id': {
1116         'type': 'string'
1117     },
1118     'statusMessage': {
1119         'type': 'string'
1120     },
1121     'networkInterfaces': {
1122         'type': 'string'
1123     }
1124 }
1125 }
1126
1127 group = {
1128     'schema': {
1129         'users': {
1130             'type': 'list',
1131             'schema': {
1132                 'type': 'objectid',
1133                 'data_relation': {
1134                     'resource': 'user',
1135                     'field': '_id',
1136                     'embeddable': True
1137                 }
1138             }
1139         },
1140         'name': {
1141             'type': 'string'
1142         },
1143         'description': {
1144             'type': 'string'

```

714

```

1145     }
1146   }
1147 }
1148
1149 secgroup = {
1150   'schema': {
1151     'ingress': {
1152       'type': 'string'
1153     },
1154     'egress': {
1155       'type': 'string'
1156     },
1157     'ports': {
1158       'type': 'integer'
1159     },
1160     'protocols': {
1161       'type': 'string'
1162     }
1163   }
1164 }
1165
1166 node_new = {
1167   'schema': {
1168     'authorized_keys': {
1169       'type': 'list',
1170       'schema': {
1171         'type': 'string'
1172       }
1173     },
1174     'name': {
1175       'type': 'string'
1176     },
1177     'external_ip': {
1178       'type': 'string'
1179     },
1180     'memory': {
1181       'type': 'integer'
1182     },
1183     'create_external_ip': {
1184       'type': 'boolean'
1185     },
1186     'internal_ip': {
1187       'type': 'string'
1188     },
1189     'loginuser': {
1190       'type': 'string'
1191     },
1192     'owner': {
1193       'type': 'string'
1194     },

```

715

```

1195     'cores': {
1196         'type': 'integer'
1197     },
1198     'disk': {
1199         'type': 'integer'
1200     },
1201     'ssh_keys': {
1202         'type': 'list',
1203         'schema': {
1204             'type': 'dict',
1205             'schema': {
1206                 'from': {
1207                     'type': 'string'
1208                 },
1209                 'decrypt': {
1210                     'type': 'string'
1211                 },
1212                 'ssh_keygen': {
1213                     'type': 'boolean'
1214                 },
1215                 'to': {
1216                     'type': 'string'
1217                 }
1218             }
1219         }
1220     },
1221     'security_groups': {
1222         'type': 'list',
1223         'schema': {
1224             'type': 'dict',
1225             'schema': {
1226                 'ingress': {
1227                     'type': 'string'
1228                 },
1229                 'egress': {
1230                     'type': 'string'
1231                 },
1232                 'ports': {
1233                     'type': 'list',
1234                     'schema': {
1235                         'type': 'integer'
1236                     }
1237                 },
1238                 'protocols': {
1239                     'type': 'list',
1240                     'schema': {
1241                         'type': 'string'
1242                     }
1243                 }
1244             }
1245         }
1246     }
1247 }

```

```

1245     }
1246 },
1247 'users': {
1248     'type': 'dict',
1249     'schema': {
1250         'name': {
1251             'type': 'string'
1252         },
1253         'groups': {
1254             'type': 'list',
1255             'schema': {
1256                 'type': 'string'
1257             }
1258         }
1259     }
1260 }
1261 }
1262 }
1263
1264 batchjob = {
1265     'schema': {
1266         'output_file': {
1267             'type': 'string'
1268         },
1269         'group': {
1270             'type': 'string'
1271         },
1272         'job_id': {
1273             'type': 'string'
1274         },
1275         'script': {
1276             'type': 'string'
1277         },
1278         'cmd': {
1279             'type': 'string'
1280         },
1281         'queue': {
1282             'type': 'string'
1283         },
1284         'cluster': {
1285             'type': 'string'
1286         },
1287         'time': {
1288             'type': 'string'
1289         },
1290         'path': {
1291             'type': 'string'
1292         },
1293         'nodes': {
1294             'type': 'string'

```

717

```

1295     },
1296     'dir': {
1297         'type': 'string'
1298     }
1299 }
1300 }
1301
1302 account = {
1303     'schema': {
1304         'status': {
1305             'type': 'string'
1306         },
1307         'startDate': {
1308             'type': 'string'
1309         },
1310         'endDate': {
1311             'type': 'string'
1312         },
1313         'description': {
1314             'type': 'string'
1315         },
1316         'uuid': {
1317             'type': 'string'
1318         },
1319         'user': {
1320             'type': 'list',
1321             'schema': {
1322                 'type': 'string'
1323             }
1324         },
1325         'group': {
1326             'type': 'list',
1327             'schema': {
1328                 'type': 'string'
1329             }
1330         },
1331         'balance': {
1332             'type': 'float'
1333         },
1334         'name': {
1335             'type': 'string'
1336         }
1337     }
1338 }
1339
1340 libcloud_vm = {
1341     'schema': {
1342         'username': {
1343             'type': 'string'
1344         },

```

718

```

1345     'status': {
1346         'type': 'string'
1347     },
1348     'root_device_type': {
1349         'type': 'string'
1350     },
1351     'private_ips': {
1352         'type': 'string'
1353     },
1354     'instance_type': {
1355         'type': 'string'
1356     },
1357     'image': {
1358         'type': 'string'
1359     },
1360     'private_dns': {
1361         'type': 'string'
1362     },
1363     'image_name': {
1364         'type': 'string'
1365     },
1366     'instance_id': {
1367         'type': 'string'
1368     },
1369     'image_id': {
1370         'type': 'string'
1371     },
1372     'public_ips': {
1373         'type': 'string'
1374     },
1375     'state': {
1376         'type': 'string'
1377     },
1378     'root_device_name': {
1379         'type': 'string'
1380     },
1381     'key': {
1382         'type': 'string'
1383     },
1384     'group': {
1385         'type': 'string'
1386     },
1387     'flavor': {
1388         'type': 'string'
1389     },
1390     'availability': {
1391         'type': 'string'
1392     },
1393     'uuid': {
1394         'type': 'string'

```

719

```

1395     }
1396 }
1397 }
1398
1399 compute_node = {
1400     'schema': {
1401         'status': {
1402             'type': 'string'
1403         },
1404         'authorized_keys': {
1405             'type': 'list',
1406             'schema': {
1407                 'type': 'objectid',
1408                 'data_relation': {
1409                     'resource': 'sshkey',
1410                     'field': '_id',
1411                     'embeddable': True
1412                 }
1413             }
1414         },
1415         'kind': {
1416             'type': 'string'
1417         },
1418         'uuid': {
1419             'type': 'string'
1420         },
1421         'secgroups': {
1422             'type': 'list',
1423             'schema': {
1424                 'type': 'objectid',
1425                 'data_relation': {
1426                     'resource': 'secgroup',
1427                     'field': '_id',
1428                     'embeddable': True
1429                 }
1430             }
1431         },
1432         'nics': {
1433             'type': 'list',
1434             'schema': {
1435                 'type': 'objectid',
1436                 'data_relation': {
1437                     'resource': 'nic',
1438                     'field': '_id',
1439                     'embeddable': True
1440                 }
1441             }
1442         },
1443         'image': {
1444             'type': 'string'

```

720


```

1445     },
1446     'label': {
1447         'type': 'string'
1448     },
1449     'loginuser': {
1450         'type': 'string'
1451     },
1452     'flavor': {
1453         'type': 'list',
1454         'schema': {
1455             'type': 'objectid',
1456             'data_relation': {
1457                 'resource': 'flavor',
1458                 'field': '_id',
1459                 'embeddable': True
1460             }
1461         }
1462     },
1463     'metadata': {
1464         'type': 'dict',
1465         'schema': {
1466             'owner': {
1467                 'type': 'string'
1468             },
1469             'experiment': {
1470                 'type': 'string'
1471             }
1472         }
1473     },
1474     'name': {
1475         'type': 'string'
1476     }
1477 }
1478
1479
1480 database = {
1481     'schema': {
1482         'endpoint': {
1483             'type': 'string'
1484         },
1485         'protocol': {
1486             'type': 'string'
1487         },
1488         'name': {
1489             'type': 'string'
1490         }
1491     }
1492 }
1493
1494 default = {

```

721

```

1495     'schema': {
1496         'context': {
1497             'type': 'string'
1498         },
1499         'name': {
1500             'type': 'string'
1501         },
1502         'value': {
1503             'type': 'string'
1504         }
1505     }
1506 }
1507
1508 openstack_image = {
1509     'schema': {
1510         'status': {
1511             'type': 'string'
1512         },
1513         'username': {
1514             'type': 'string'
1515         },
1516         'updated': {
1517             'type': 'string'
1518         },
1519         'uuid': {
1520             'type': 'string'
1521         },
1522         'created': {
1523             'type': 'string'
1524         },
1525         'minDisk': {
1526             'type': 'string'
1527         },
1528         'progress': {
1529             'type': 'string'
1530         },
1531         'minRam': {
1532             'type': 'string'
1533         },
1534         'os_image_size': {
1535             'type': 'string'
1536         },
1537         'metadata': {
1538             'type': 'dict',
1539             'schema': {
1540                 'instance_uuid': {
1541                     'type': 'string'
1542                 },
1543                 'image_location': {
1544                     'type': 'string'

```

722

```

1545 },
1546 'image_state': {
1547     'type': 'string'
1548 },
1549 'instance_type_memory_mb': {
1550     'type': 'string'
1551 },
1552 'user_id': {
1553     'type': 'string'
1554 },
1555 'description': {
1556     'type': 'string'
1557 },
1558 'kernel_id': {
1559     'type': 'string'
1560 },
1561 'instance_type_name': {
1562     'type': 'string'
1563 },
1564 'ramdisk_id': {
1565     'type': 'string'
1566 },
1567 'instance_type_id': {
1568     'type': 'string'
1569 },
1570 'instance_type_ephemeral_gb': {
1571     'type': 'string'
1572 },
1573 'instance_type_rxtx_factor': {
1574     'type': 'string'
1575 },
1576 'image_type': {
1577     'type': 'string'
1578 },
1579 'network_allocated': {
1580     'type': 'string'
1581 },
1582 'instance_type_flavorid': {
1583     'type': 'string'
1584 },
1585 'instance_type_vcpus': {
1586     'type': 'string'
1587 },
1588 'instance_type_root_gb': {
1589     'type': 'string'
1590 },
1591 'base_image_ref': {
1592     'type': 'string'
1593 },
1594 'instance_type_swap': {

```

723

```

1595         'type': 'string'
1596     },
1597     'owner_id': {
1598         'type': 'string'
1599     }
1600 }
1601 }
1602 }
1603 }
1604
1605 OpenStackNodeExtra = {
1606     'schema': {
1607         'vm_state': {
1608             'type': 'string'
1609         },
1610         'addresses': {
1611             'type': 'list',
1612             'schema': {
1613                 'type': 'string'
1614             }
1615         },
1616         'availability_zone': {
1617             'type': 'string'
1618         },
1619         'service_name': {
1620             'type': 'string'
1621         },
1622         'userId': {
1623             'type': 'string'
1624         },
1625         'imageId': {
1626             'type': 'string'
1627         },
1628         'volumes_attached': {
1629             'type': 'string'
1630         },
1631         'task_state': {
1632             'type': 'string'
1633         },
1634         'disk_config': {
1635             'type': 'string'
1636         },
1637         'power_state': {
1638             'type': 'string'
1639         },
1640         'progress': {
1641             'type': 'string'
1642         },
1643         'metadata': {
1644             'type': 'list',

```

724

```

1645         'schema': {
1646             'type': 'string'
1647         }
1648     },
1649     'updated': {
1650         'type': 'string'
1651     },
1652     'hostId': {
1653         'type': 'string'
1654     },
1655     'key_name': {
1656         'type': 'string'
1657     },
1658     'flavorId': {
1659         'type': 'string'
1660     },
1661     'password': {
1662         'type': 'string'
1663     },
1664     'access_ip': {
1665         'type': 'string'
1666     },
1667     'access_ipv6': {
1668         'type': 'string'
1669     },
1670     'created': {
1671         'type': 'string'
1672     },
1673     'fault': {
1674         'type': 'string'
1675     },
1676     'uri': {
1677         'type': 'string'
1678     },
1679     'tenantId': {
1680         'type': 'string'
1681     },
1682     'config_drive': {
1683         'type': 'string'
1684     }
1685 }
1686 }
1687
1688 mapreduce = {
1689     'schema': {
1690         'function': {
1691             'type': 'dict',
1692             'schema': {
1693                 'source': {
1694                     'type': 'string'

```

725

```

1695         },
1696         'args': {
1697             'type': 'dict',
1698             'schema': {}
1699         }
1700     },
1701     'fault_tolerant': {
1702         'type': 'boolean'
1703     },
1704     'data': {
1705         'type': 'dict',
1706         'schema': {
1707             'dest': {
1708                 'type': 'string'
1709             },
1710             'source': {
1711                 'type': 'string'
1712             }
1713         }
1714     },
1715     'backend': {
1716         'type': 'dict',
1717         'schema': {
1718             'type': {
1719                 'type': 'string'
1720             }
1721         }
1722     }
1723 }
1724
1725 }
1726
1727 filter = {
1728     'schema': {
1729         'function': {
1730             'type': 'string'
1731         },
1732         'name': {
1733             'type': 'string'
1734         }
1735     }
1736 }
1737
1738 alias = {
1739     'schema': {
1740         'origin': {
1741             'type': 'string'
1742         },
1743         'name': {
1744             'type': 'string'

```

726

```

1745     }
1746   }
1747 }
1748
1749 replica = {
1750   'schema': {
1751     'endpoint': {
1752       'type': 'string'
1753     },
1754     'name': {
1755       'type': 'string'
1756     },
1757     'checksum': {
1758       'type': 'dict',
1759       'schema': {
1760         'md5': {
1761           'type': 'string'
1762         }
1763       }
1764     },
1765     'replica': {
1766       'type': 'string'
1767     },
1768     'accessed': {
1769       'type': 'string'
1770     },
1771     'size': {
1772       'type': 'list',
1773       'schema': {
1774         'type': 'string'
1775       }
1776     }
1777   }
1778 }
1779
1780 openstack_vm = {
1781   'schema': {
1782     'vm_state': {
1783       'type': 'string'
1784     },
1785     'availability_zone': {
1786       'type': 'string'
1787     },
1788     'terminated_at': {
1789       'type': 'string'
1790     },
1791     'image': {
1792       'type': 'string'
1793     },
1794     'diskConfig': {

```

727

```

1795         'type': 'string'
1796     },
1797     'flavor': {
1798         'type': 'string'
1799     },
1800     'security_groups': {
1801         'type': 'string'
1802     },
1803     'volumes_attached': {
1804         'type': 'string'
1805     },
1806     'user_id': {
1807         'type': 'string'
1808     },
1809     'uuid': {
1810         'type': 'string'
1811     },
1812     'accessIPv4': {
1813         'type': 'string'
1814     },
1815     'accessIPv6': {
1816         'type': 'string'
1817     },
1818     'power_state': {
1819         'type': 'string'
1820     },
1821     'progress': {
1822         'type': 'string'
1823     },
1824     'image__id': {
1825         'type': 'string'
1826     },
1827     'launched_at': {
1828         'type': 'string'
1829     },
1830     'config_drive': {
1831         'type': 'string'
1832     },
1833     'username': {
1834         'type': 'string'
1835     },
1836     'updated': {
1837         'type': 'string'
1838     },
1839     'hostId': {
1840         'type': 'string'
1841     },
1842     'floating_ip': {
1843         'type': 'string'
1844     },

```

728


```

1845         'static_ip': {
1846             'type': 'string'
1847         },
1848         'key': {
1849             'type': 'string'
1850         },
1851         'flavor__id': {
1852             'type': 'string'
1853         },
1854         'group': {
1855             'type': 'string'
1856         },
1857         'task_state': {
1858             'type': 'string'
1859         },
1860         'created': {
1861             'type': 'string'
1862         },
1863         'tenant_id': {
1864             'type': 'string'
1865         },
1866         'status': {
1867             'type': 'string'
1868         }
1869     }
1870 }
1871
1872 organization = {
1873     'schema': {
1874         'users': {
1875             'type': 'list',
1876             'schema': {
1877                 'type': 'objectid',
1878                 'data_relation': {
1879                     'resource': 'user',
1880                     'field': '_id',
1881                     'embeddable': True
1882                 }
1883             }
1884         }
1885     }
1886 }
1887
1888 hadoop = {
1889     'schema': {
1890         'deployers': {
1891             'type': 'dict',
1892             'schema': {
1893                 'ansible': {
1894                     'type': 'string'

```

729

```

1895         }
1896     }
1897 },
1898 'requires': {
1899     'type': 'dict',
1900     'schema': {
1901         'java': {
1902             'type': 'dict',
1903             'schema': {
1904                 'implementation': {
1905                     'type': 'string'
1906                 },
1907                 'version': {
1908                     'type': 'string'
1909                 },
1910                 'zookeeper': {
1911                     'type': 'string'
1912                 },
1913                 'supervisord': {
1914                     'type': 'string'
1915                 }
1916             }
1917         }
1918     }
1919 },
1920 'parameters': {
1921     'type': 'dict',
1922     'schema': {
1923         'num_resourcemanager': {
1924             'type': 'integer'
1925         },
1926         'num_namenodes': {
1927             'type': 'integer'
1928         },
1929         'use_yarn': {
1930             'type': 'boolean'
1931         },
1932         'num_datanodes': {
1933             'type': 'integer'
1934         },
1935         'use_hdfs': {
1936             'type': 'boolean'
1937         },
1938         'num_historyservers': {
1939             'type': 'integer'
1940         },
1941         'num_journalnodes': {
1942             'type': 'integer'
1943         }
1944     }

```

```

1945     }
1946   }
1947 }
1948
1949 accounting_resource = {
1950   'schema': {
1951     'account': {
1952       'type': 'string'
1953     },
1954     'group': {
1955       'type': 'string'
1956     },
1957     'description': {
1958       'type': 'string'
1959     },
1960     'parameters': {
1961       'type': 'dict',
1962       'schema': {
1963         'parameter1': {
1964           'type': 'float'
1965         },
1966         'parameter2': {
1967           'type': 'float'
1968         }
1969       }
1970     },
1971     'uuid': {
1972       'type': 'string'
1973     },
1974     'charge': {
1975       'type': 'string'
1976     },
1977     'unites': {
1978       'type': 'dict',
1979       'schema': {
1980         'parameter1': {
1981           'type': 'string'
1982         },
1983         'parameter2': {
1984           'type': 'string'
1985         }
1986       }
1987     },
1988     'user': {
1989       'type': 'string'
1990     },
1991     'name': {
1992       'type': 'string'
1993     }
1994   }

```

731

```

1995 }
1996
1997
1998
1999 eve_settings = {
2000     'MONGO_HOST': 'localhost',
2001     'MONGO_DBNAME': 'testing',
2002     'RESOURCE_METHODS': ['GET', 'POST', 'DELETE'],
2003     'BANDWIDTH_SAVER': False,
2004     'DOMAIN': {
2005         'container': container,
2006         'stream': stream,
2007         'azure_image': azure_image,
2008         'deployment': deployment,
2009         'azure-size': azure_size,
2010         'cluster': cluster,
2011         'computer': computer,
2012         'mesos-docker': mesos_docker,
2013         'file': file,
2014         'reservation': reservation,
2015         'microservice': microservice,
2016         'flavor': flavor,
2017         'virtual_directory': virtual_directory,
2018         'mapreduce_function': mapreduce_function,
2019         'virtual_cluster': virtual_cluster,
2020         'libcloud_flavor': libcloud_flavor,
2021         'LibCloudNode': LibCloudNode,
2022         'sshkey': sshkey,
2023         'timestamp': timestamp,
2024         'mapreduce_noop': mapreduce_noop,
2025         'role': role,
2026         'AzureNodeExtra': AzureNodeExtra,
2027         'var': var,
2028         'profile': profile,
2029         'virtual_machine': virtual_machine,
2030         'kubernetes': kubernetes,
2031         'nic': nic,
2032         'openstack_flavor': openstack_flavor,
2033         'azure-vm': azure_vm,
2034         'ec2NodeExtra': ec2NodeExtra,
2035         'libcloud_image': libcloud_image,
2036         'user': user,
2037         'GCENodeExtra': GCENodeExtra,
2038         'group': group,
2039         'secgroup': secgroup,
2040         'node_new': node_new,
2041         'batchjob': batchjob,
2042         'account': account,
2043         'libcloud_vm': libcloud_vm,
2044         'compute_node': compute_node,

```

732

```

2045     'database': database,
2046     'default': default,
2047     'openstack_image': openstack_image,
2048     'OpenStackNodeExtra': OpenStackNodeExtra,
2049     'mapreduce': mapreduce,
2050     'filter': filter,
2051     'alias': alias,
2052     'replica': replica,
2053     'openstack_vm': openstack_vm,
2054     'organization': organization,
2055     'hadoop': hadoop,
2056     'accounting_resource': accounting_resource,
2057 },
2058 }
733

```

734 B. CLOUDMESH REST

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

738 B.1. Prerequisites

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

```
744     pip install cloudmesh.rest
```

745 B.2. REST Service

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

749 A valid json resource specification looks like this:

```

750 {
751     "profile": {
752         "description": "The Profile of a user",
753         "email": "laszewski@gmail.com",
754         "firstname": "Gregor",
755         "lastname": "von Laszewski",
756         "username": "gregor"
757     }
758 }

```

759 here we define an object called profile, that contains a number of attributes and values. The type of the

760 values are automatically determined. All json specifications are contained in a directory and can easily be
761 converted into a valid schema for the eve rest service by executing the commands

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

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

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

767 B.3. Limitations

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

771 C. CONTRIBUTING

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

777 C.1. Conversion to Word

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

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

781 Managing the document in github has provided us with the advantage that a reference implementation can
782 be automatically derived from the specified objects. Also it is easy to contribute as all text is written in
783 ASCII while using \LaTeX syntax to allow for formating in PDF.

784 Contributions can be makes as follows:

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

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

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

795 C.2. Object Specification

796 All objects are located in

797 `cloudmesh.rest/cloudmesh/specification/examples`

798 And can be modified there

799 C.3. Creation of the PDF document

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

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

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

806 To compile the document please use

807 `cd docs`
808 `make`

809 This will generate the PDF file

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

811 On OSX we have also integrated a quick view whit

812 `make view`

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

814 * <http://pdf2docx.com/>

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

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

820 C.4. Code Generation

821 This section is intended for experts and guidance on using it can be obtained by contacting Gregor von
822 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
```

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