NIST BDRA Volume 8

Contents

0.1 nistvol8-2.md

NIST Special Publication 1500-9

```
DRAFT NIST Big Data Interoperability Framework:
Volume 8, Reference
Architecture Interfaces
```

NIST Big Data Public Working Group

Reference Architecture Subgroup

Version 3.0.1

December 1, 2018

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

NIST Special Publication 1500-9

Information Technology Laboratory

DRAFT NIST Big Data Interoperability Framework:

Volume 8, Reference Architecture Interfaces

Version 2

NIST Big Data Public Working Group (NBD-PWG)

Definitions and Taxonomies Subgroup

National Institute of Standards and Technology

Gaithersburg, MD 20899

This draft publication is available free of charge from:

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

October 2017

Figure 1: http://physics.nist.gov/Images/doc.bw.gif

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 1500-9

?? pages (December 1, 2018)

or endorsement by NIST, nor is it intended to imply that the entities, materials, or equipment are necessarily the best available for the purpose.

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

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

Comments on this publication may be submitted to Wo Chang

National Institute of Standards and Technology

Attn: Wo Chang, Information Technology Laboratory

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

Email: SP1500comments@nist.gov

Reports on Computer Systems Technology

The Information Technology Laboratory (ITL) at NIST promotes the U.S. economy and public welfare by providing technical leadership for the Nation's measurement and standards infrastructure. ITL develops tests, test methods, reference data, proof of concept implementations, and technical analyses to advance the development and productive use of information technology. 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 Information Technology and its collaborative activities with industry, government, and academic organizations.

1 Abstract

This document summarizes interfaces that are instrumental for the interaction with Clouds, Containers, and High Performance Computing (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*.

2 Keywords

Adoption; barriers; implementation; interfaces; market maturity; organizational maturity; project maturity; system modernization.

3 Acknowledgements

This document reflects the contributions and discussions by the membership of the NBD-PWG, cochaired 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 (Vistronix), and Dan McClary (Oracle), for the Standards Roadmap Subgroup.

The editors for this document were the following:

- ***Version 2.1***: A previous volume used just the definition of the schema based on examples it was easier to read but did only include the definition of the resources and not the interaction with the resources. This volume was in place till June 2018.
- ***Version 2.2***: This version was significantly changed and uses now OpenAPI to specify the Interfaces between the various services and components. Editors of this volume are:
- ***Version 2.3***: The version includes the

Gregor von Laszewski (Indiana University), and Wo Chang (NIST).

Laurie Aldape (Energetics Incorporated) and Elizabeth Lennon (NIST) provided editorial assistance across all NBDIF volumes.

NIST SP 1500-9, Draft NIST Big Data Interoperability Framework: Volume 8, Reference Architecture Interfaces, Version 2 has been collaboratively authored by the NBD-PWG. As of the date of publication, there are over six hundred NBD-PWG participants from industry, academia, and government. Federal agency participants include the National Archives and Records Administration (NARA), National Aeronautics and Space Administration (NASA), National Science Foundation (NSF), and the U.S. Departments of Agriculture, Commerce, Defense, Energy, Census, Health and Human Services, Homeland Security, Transportation, Treasury, and Veterans Affairs.

NIST would like to acknowledge the specific contributions to this volume, during Version 1 and/or Version 2 activities. *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. This includes especially the following NBD-PWG members:

- Gregor von Laszewski, Indiana University
- Wo Chang, National Institute of Standard and Technology,
- Fugang Wang, Indiana University
- Geoffrey C. Fox, Indiana University
- Alicia Zuniga-Alvarado, Consultant
- Robert C. Whetsel, DISA/NBIS
- Pratik Thakkar, Philips

Past contributors of earlier versions of this document include

- Badi Abdhul Wahid, formerly Indiana University

4 Table of Contents

5 Executive Summary

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

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

The resources were categorized in groups that are identified by the NBDRA set forward in the *NBDIF: Volume 6, Reference Architecture* document. While the *NBDIF: Volume 3, Use Cases and General Requirements* document provides *application*-oriented high-level use cases, the use cases defined in this document are subsets of them and focus on *interface* use cases. The interface use cases are not meant to be complete examples, but showcase why the resource has been defined. Hence, the interfaces use cases are only representative, and do not represent the entire spectrum of Big Data usage. All 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 nine 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: Reference Architecture Interfaces
- Volume 9: Adoption and Modernization

The NBDIF will be released in three versions, which correspond to the three development stages of the NBD-PWG work. The three stages aim to achieve the following with respect to the NBDRA.

1. Identify the high-level Big Data reference architecture key components, which are technology-, infrastructure-, and vendor-agnostic.
2. Define general interfaces between the NBDRA components.
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 the NBD-PWG web page (https://bigdatawg.nist.gov).

6 Introduction

6.1 Background

There is broad agreement among commercial, academic, and government leaders about the remarkable potential of Big Data to spark innovation, fuel commerce, and drive progress. Big Data is the common term used to describe the deluge of data in today's networked, digitized, sensor-laden, and information-driven

world. The availability of vast data resources carries the potential to answer questions previously out of reach, including the following:

- How can a potential pandemic reliably be detected early enough to intervene?
- Can new materials with advanced properties be predicted before these materials have ever been synthesized?
- How can the current advantage of the attacker over the defender in guarding against cybersecurity 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 White House Office of Science and Technology Policy, "Big Data is a Big Deal," *OSTP Blog*, accessed February 21, 2014, http://www.whitehouse.gov/blog/2012/03/29/big-data-big-deal). 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) accepted the challenge to stimulate collaboration among industry professionals to further the secure and effective adoption of Big Data. As a 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).

1. Identify the high-level Big Data reference architecture key components, which are technology, infrastructure, and vendor agnostic.
2. Define general interfaces between the NBDRA components.
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 developed.

- 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). The current effort documented in this volume reflects concepts developed within the rapidly evolving field of Big Data.

`Note Section numbers`

6.2 Scope and Objectives of the Reference Architectures Subgroup

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

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

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

The NBDRA is a high-level conceptual model crafted to serve as a tool to facilitate open discussion of the requirements, design structures, and operations inherent in Big Data. The NBDRA is intended to facilitate the understanding of the operational intricacies in Big Data. It does not represent the system architecture of a specific Big Data system, but rather is a tool for describing, discussing, and developing system-specific

architectures using a common framework of reference. The model is not tied to any specific vendor products, services, or reference implementation, nor does it define prescriptive solutions that inhibit innovation.

The NBDRA does not address the following:

- Detailed specifications for any organization's operational systems;
- Detailed specifications of information exchanges or services; and
- Recommendations or standards for integration of infrastructure products.

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

6.3 Report Production

The *NBDIF: Volume 8, References Architecture Interfaces* is one of nine volumes, whose overall aims are to define and prioritize Big Data requirements, including interoperability, portability, reusability, extensibility, data usage, analytic techniques, and technology infrastructure to support secure and effective adoption of Big Data. The overall goals of this volume are to define and specify interfaces to implement the Big Data Reference Architecture. This volume arose from discussions during the weekly NBD-PWG conference calls. Topics included in this volume began to take form in Phase 2 of the NBD-PWG work. This volume represents the groundwork for additional content planned for Phase 3. During the discussions, the NBD-PWG identified the need to specify a variety of interfaces.

To enable interoperability between the NBDRA components, a list of well-defined NBDRA interfaces is needed. These interfaces are documented in this volume. To introduce them, the NBDRA structure will be followed, focusing on interfaces that allow bootstrapping of the NBDRA. The document begins with a summary of requirements that will be integrated into our specifications. Subsequently, each section will introduce a number of objects that build the core of the interface addressing a specific aspect of the NBDRA. A selected number of *interface use cases* will be showcased to outline how the specific interface can be used in a reference implementation of the NBDRA. Validation of this approach can be achieved while applying it to the application use cases that have been gathered in the *NBDIF: Volume 3, Use Cases and Requirements* document. These application use cases have considerably contributed towards the design of the NBDRA. Hence the expectation is that: (1) the interfaces can be used to help implement a Big Data architecture for a specific use case; and (2) the proper implementation. This approach can facilitate subsequent analysis and comparison of the use cases.

This document is expected to grow with the help of contributions from the community to achieve a comprehensive set of interfaces that will be usable for the implementation of Big Data Architectures. To achieve technical and high-quality document content, this document will go through a public comment period along with NIST internal review.

6.4 Report Structure

The organization of this document roughly corresponds to the process used by the NBD-PWG to develop the interfaces. Following the introductory material presented in Section 1, the remainder of this document is organized as follows:

- Section 2 presents the interface requirements;
- Section 3 summarizes the elementary objects that are important to the NBDRA;
- Section 4 presents several objects grouped by functional use; and
- Four appendices provide supplementary information.

6.5 Future Work on this Volume

A number of topics have not been discussed and clarified sufficiently to be included in Version 2.2. Future topics will be identified during discussions within the Reference Architecture Subgroup.

7 NBDRA Interface Requirements

The development of a Big Data reference architecture requires a thorough understanding of current techniques, issues, and concerns. To this end, the NBD-PWG collected use cases to gain an understanding of current applications of Big Data, conducted a survey of reference architectures to understand commonalities within Big Data architectures in use, developed a taxonomy to understand and organize the information collected, and reviewed existing technologies and trends relevant to Big Data. The results of these NBD-PWG activities were used in the development of the NBDRA (Figure 1) and the interfaces presented herein. Detailed descriptions of these activities can be found in the other volumes of the *NBDIF*.

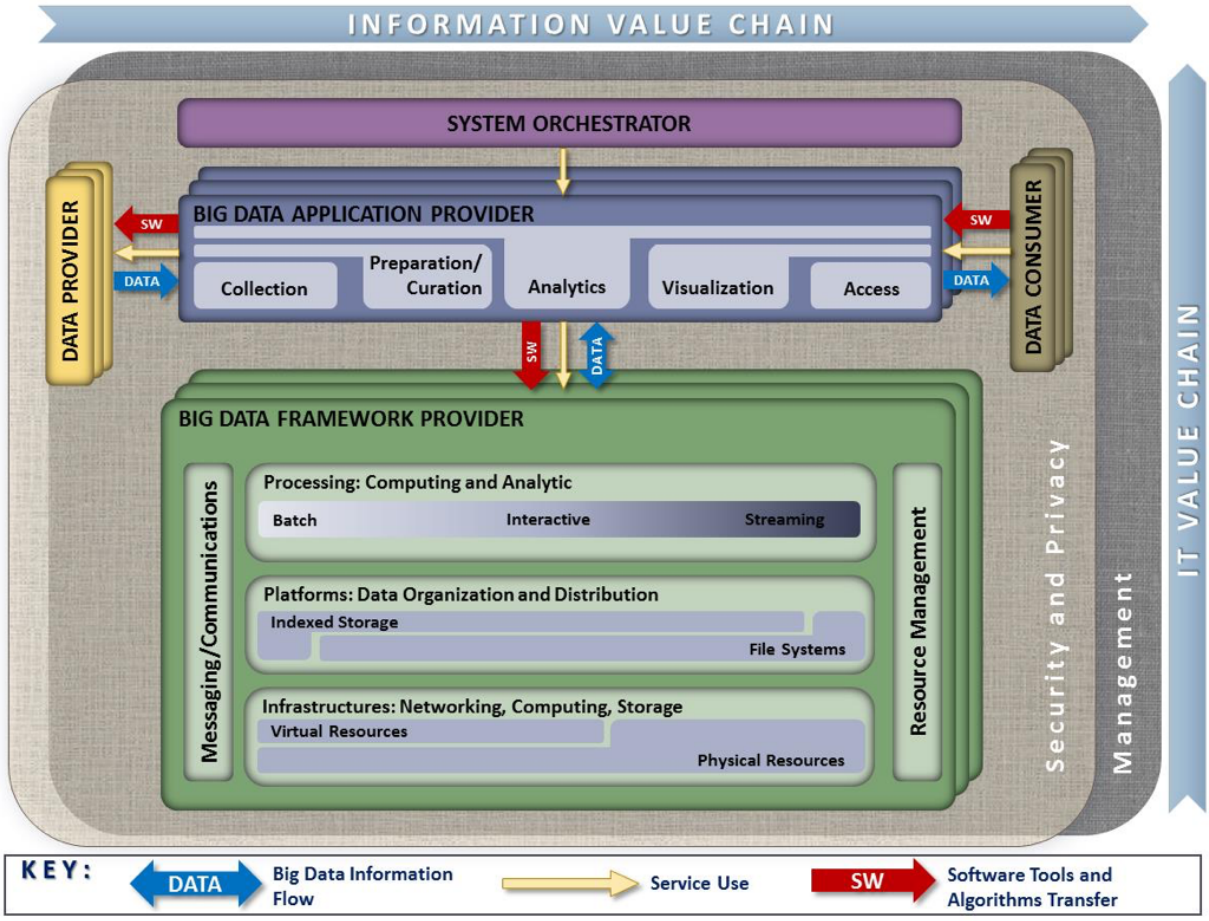

Figure 2: **Figure 1:** NIST Big Data Reference Architecture (NBDRA)

This vendor-neutral, technology- and infrastructure-agnostic conceptual model, the NBDRA, is shown in Figure 1 and represents a Big Data system composed of five logical functional components connected by interoperability interfaces (i.e., services). Two fabrics envelop the components, representing the interwoven nature of management and security and privacy with all five of the components. These two fabrics provide services and functionality to the five main roles in the areas specific to Big Data and are crucial to any Big Data solution. Note: None of the terminology or diagrams in these documents is intended to be normative or

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

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

`:warning: FIGURE 2 was at one point removed by somone, but the text was not updated`

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

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

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

7.1 High-Level Requirements of the Interface Approach

This section focuses on the high-level requirements of the interface approach that are needed to implement the reference architecture depicted in Figure 1.

7.1.1 Technology- and Vendor-Agnostic

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

7.1.2 Support of Plug-In Compute Infrastructure

As Big Data is not just about hosting data, but about analyzing data, the interfaces provided herein 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. These resources and services include the following:

-Access to hierarchy of compute resources from the laptop/desktop, servers, data clusters, and clouds; -The ability to integrate special-purpose hardware such as GPUs and FPGAs that are used in accelerated analysis of data; and -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.

7.1.3 Orchestration of Infrastructure and Services

From review of the use case collection, presented in the *NBDIF: Volume 3, Use Cases and General Requirements* document [4], the need arose to address the mechanism of preparing suitable infrastructures for various use cases. As not every infrastructure is suited for every use case, a custom infrastructure may be needed. As such, this document is not attempting to deliver a single deployed NBDRA, but allow the setup of an infrastructure that satisfies the particular use case. To achieve this task, it is necessary to provision software stacks and services while orchestrating their deployment and leveraging infrastructures. It is not the 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.

7.1.4 Orchestration of Big Data Applications and Experiments

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

7.1.5 Reusability

The interfaces provided must encourage reusability of the infrastructure, services, and experiments described by them. This includes (1) reusability of available analytics packages and services for adoption; (2) deployment of customizable analytics tools and services; and (3) operational adjustments that allow the services and infrastructure to be adapted while at the same time allowing for reproducible experiment execution.

7.1.6 Execution Workloads

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

7.1.7 Security and Privacy Fabric Requirements

Although the focus of this document is not security and privacy, which are documented in the *NBDIF: Volume 4, Security and Privacy* [8], the interfaces defined herein must be capable of integration into a secure reference architecture that supports secure execution, secure data transfer, and privacy. Consequently, the interfaces

defined herein can be augmented with frameworks and solutions that provide such mechanisms. Thus, diverse requirement needs stemming from different use cases addressing security need to be distinguished. To contrast that the security requirements between applications can vary drastically, the following example is provided. Although many of the interfaces and their objects to support Big Data applications in physics are similar to those in healthcare, they differ in the integration of security interfaces and policies. While in physics the protection of data is less of an issue, it is a stringent requirement in healthcare. Thus, deriving architectural frameworks for both may use largely similar components, but addressing security issues will be very different. In future versions of this document, the security of interfaces may be addressed. In the meanwhile, they are considered an advanced use case showcasing that the validity of the specifications introduced here is preserved, even if security and privacy requirements differ vastly among application use cases.

7.2 Component-Specific Interface Requirements

This section summarizes the requirements for the interfaces of the NBDRA components. The five components are listed in Figure 1 and addressed in each of the subsections as part of Section 2.2.1 (System Orchestrator Interface Requirements) and Section 2.2.6 (Big Data Application Provider to Big Data Framework Provider Interface) 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.

NOTE: SECTION NUMBERS

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

7.2.1 System Orchestrator Interface Requirements

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

7.2.2 Data Provider Interface Requirements

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

7.2.3 Data Consumer Interface Requirements

Like the Data Provider, the role of Data Consumer within the NBDRA can be an actual end user or another system. In many ways, this role is the mirror image of the Data Provider, with the entire Big Data framework appearing like a Data Provider to the Data Consumer. The activities associated with the Data Consumer role include the following:

- Search and Retrieve,
- Download,
- Analyze Locally,
- Reporting,
- Visualization, and
- Data to Use for Their Own Processes.

The interface for the data consumer must be able to describe the consuming services and how they retrieve information or leverage data consumers.

7.2.4 Big Data Application Interface Provider Requirements

The Big Data Application Provider role executes a specific set of operations along the data life cycle to meet the requirements established by the System Orchestrator, as well as meeting security and privacy requirements. The Big Data Application Provider is the architecture component that encapsulates the business logic and functionality to be executed by the architecture. The interfaces to describe Big Data applications include interfaces for the various subcomponents including collections, preparation/curation, analytics, visualization, and access. Some of the interfaces used in these subcomponents can be reused from other interfaces, which are introduced in other sections of this document. Where appropriate, application-specific interfaces will be identified and examples provided with a focus on use cases as identified in the *NBDIF: Volume 3, Use Cases and General Requirements*.

7.2.4.1 Collection

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

7.2.4.2 Preparation

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

7.2.4.3 Analytics

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

7.2.4.4 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.

7.2.4.5 Access

The access activity within the Big Data Application Provider is focused on the communication/interaction with the Data Consumer. Like 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.

7.2.5 Big Data Provider Framework Interface Requirements

Data for Big Data applications are delivered through data providers. They can be either local providers, data contributed by a user, or distributed data providers, data on the Internet. This interface must be able to provide the following functionality:

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

7.2.5.1 Infrastructures Interface Requirements

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

7.2.5.2 Platforms Interface Requirements

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

7.2.5.3 Processing Interface Requirements

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

7.2.5.4 Crosscutting Interface Requirements

Several crosscutting interface requirements within the Big Data Framework Provider include messaging, communication, and resource management. Often these services may be hidden from explicit interface use as they are part of larger systems that expose higher-level functionality through their interfaces. However, such interfaces may also be exposed on a lower level in case finer-grained control is needed. The need for such crosscutting interface requirements will be extracted from the *NBDIF: Volume 3, Use Cases and General Requirements* document.

7.2.5.5 Messaging/Communications Frameworks

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

7.2.5.6 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.

7.2.6 Big Data Application Provider to Big Data Framework Provider Interface

The Big Data Framework Provider typically consists of one or more hierarchically organized instances of the components in the NBDRA IT value chain (Figure 1). 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 to provide flexibility or meet the complete range of requirements, which are driven from the Big Data Application Provider.

8 Specification Paradigm

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

8.1 Hybrid and Multiple Frameworks

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

8.2 Design by Research Oriented Architecture

A resource-oriented architecture represents a software architecture and programming paradigm for designing and developing software in the form of resources. It is often associated with *REST* interfaces. The resources are software components which can be reused in concrete reference implementations. The service specification is conducted with OpenAPI, allowing use to provide it in a very general form that is independent of the framework or computer language in which the services can be specified. Note that OpenAPI defines services in REpresentational State Transfer (REST) The previous version only specified the resource objects.

8.3 Design by Example

To accelerate discussion among the NBD-PWG members, we encourage contributors to this document to also provide us with examples that we can include in an appendix.

8.4 Version Management

During the design phase and in between versions of this document enhancements are managed through github and community contributions are managed via github issues. This allows us to preserve the history of this document. When a new version is ready, we will tag the version in github. Older version will through this process also be available as historical documents. Discussions about objects in written form are communicated as github issues.

8.5 Interface Compliancy

Due to the easy extensibility of the objects in this document and their implicit interfaces, it is important to introduce a terminology that allows the definition of interface compliancy. We define three levels of interface compliance as follows:

- **Full Compliance:** These are reference implementations that provide full compliance to the objects defined in this document. A version number will be added to assure that the snapshot in time of the objects is associated with the version. This reference implementation will implement all objects.
- **Partial Compliance:** These are reference implementations that provide partial compliance to the objects defined in this document. A version number will be added to assure that the snapshot in time of the objects is associated with the version. This reference implementation will implement a partial list of the objects. A document will be generated during the reference implementation that lists all objects defined, but also lists the objects that are not defined by the reference architecture. The document will outline which objects and interfaces have been implemented.
- **Full and Extended Compliance:** These are interfaces that in addition to the full compliance also introduce additional interfaces and extend them. A document will be generated during the reference implementation that lists the differences to the document defined here.

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

9 Specification

We will provide the specifications to this document through an automated document creation process so that the actual OpenAPI specifications are the source for the document. Thus we will have all OpenAPI specifications located in the following directory in github:

```
Add the link to the github dir
```

Limitations of the current implementation are as follows. It is a demonstration that showcases the generation of a fully functioning REST service based on the specifications provided in this document. However, it is expected that scalability, distribution of services, and other advanced options need to be addressed based on application requirements.

9.1 List of specifications

The following table lists the current set of resource objects that we are defining in this draft. Additional objects are also available at

- https://github.com/cloudmesh-community/nist/tree/master/services

9.2 Identity

As part of services we often need to specify an identity. In addition such persons are often part of groups and have roles within these groups. Thus we distinguish three important terms related to the identity:

- Profile - The information identifying the profile of a person
- Group - A group that a person may belong to that is important to define access to services
- Role - A role given to a person as part of the group that can refine access rights.
- Organization - The information representing an Organization that manages a Big Data Service

9.2.1 Authentication

At this time we have not yet included the mechanisms on how to manage authentication to external services such as clouds that can stage virtual machines. However in cloudmesh we have shown multiple solutions to this

- Local configuration file: A configuration file is managed locally to allow access to the clouds. It is in the designers responsibility not to expose such credentials
- Session based authentication. No passwords are stored in the configuration file and access is granted on a per session basis where the password needs to be entered
- Service based authentication. The authentication is delegated to an external process. One example here is also Auth.
- The service that acts in behalf of the user needs to have access to the appropriate cloud provider credentials

An example for a configuration file is provided at

- https://github.com/cloudmesh-community/cm/blob/master/cm4/etc/cloudmesh4.yaml

9.2.2 Profile

Profiles are used to store information about users. User information can be reused in other services. THis is useful to create virtual organization the depend on user data. Profiles can be added, removed and listed. A group in the profile can be used to augment users to be part of one or more groups. A number of roles can specify a specific role of a user.

Terminology

Group: A Person with profile can be part of a Group

Role: A person with profile can have a role within that Group

9.2.2.1 Properties Profile

Property	Type	Description
uuid	string	A unique id for the profile
username	string	The username associated with the profile
group	array[string]	A list of groups that are associated to the profile
role	array[string]	A list of groups that are associated to the profile
resource	string	A resource this profile may belong to
context	string	The context the profile may belong to
description	string	A description for this profile
firstname	string	The firstanme of the profile user
lastname	string	The lastname of the profile user
publickey	string	The lastname of the profile user
email	string	The email of the profile user

9.2.2.2 Paths

9.2.2.2.1 /cloudmesh/profile/profile

GET /cloudmesh/profile/profile

Returns all profiles

Responses

Code	Description	Schema
200	profile information	Profile

PUT /cloudmesh/profile/profile

Create a new profile

Responses

Code	Description	Schema
201	Created	

Parameters

Name	Located in	Description	Required	Schema
profile	body	The new profile to create	False	Profile

9.2.2.2.2 /cloudmesh/profile/profile/{uuid}

GET /cloudmesh/profile/profile/{uuid}

Returns the profile of a user while looking it up with the UUID

Responses

Code	Description	Schema
200	profile information	Profile

Parameters

Name	Located in	Description	Required	Schema
uuid	path	ERROR: description missing	True	

9.2.2.3 profile.yaml

```
---
swagger: "2.0"
info:
  version: 3.0.3
  x-date: 11-06-2018
  title: "Profile"
  description: |-

    Profiles are used to store information about users. User
    information can be reused in other services. THis is useful to
    create virtual organization the depend on user data.  Profiles can
    be added, removed and listed. A group in the profile can be used
    to augment users to be part of one or more groups.  A number of
    roles can specify a specific role of a user.

    Terminology

    Group: A Person with profile can be part of a Group
```

```
    Role: A person with profile can have a role within that Group

  termsOfService: 'https://github.com/cloudmesh-community/nist/blob/master/LICENSE.txt'
  contact:
    name: "Cloudmesh Profile"
    url: https://cloudmesh-community.github.io/nist/spec/
  license:
    name: "Apache"
host: "localhost:8080"
schemes:
  - "http"
consumes:
  - "application/json"
produces:
  - "application/json"
paths:
  /cloudmesh/profile/profile:
    get:
      tags:
        - "profile"
      description: "Returns all profiles"
      operationId: get_profile
      produces:
        - "application/json"
      responses:
        "200":
          description: "profile information"
          schema:
            $ref: "#/definitions/Profile"
    put:
      tags:
        - "profile"
      summary: "Create a new profile"
      description: "Create a new profile"
      operationId: add_profile
      parameters:
        - in: body
          name: profile
          description: "The new profile to create"
          schema:
            $ref: "#/definitions/Profile"
      responses:
        "201":
          description: Created
  /cloudmesh/profile/profile/{uuid}:
    get:
      tags:
        - "profile"
      description: "Returns the profile of a user while looking it up with the UUID"
      operationId: getProfileByUuid
      parameters:
        - name: uuid
          in: path
          required: true
```

```
          type: string
      produces:
        - "application/json"
      responses:
        "200":
          description: "profile information"
          schema:
            $ref: "#/definitions/Profile"
definitions:
  Profile:
    type: "object"
    properties:
      uuid:
        type: "string"
        description: A unique id for the profile
      username:
        type: "string"
        description: The username associated with the profile
      group:
        type: array
        description: A list of groups that are associated to the profile
        items:
          type: "string"
      role:
        type: array
        description: A list of groups that are associated to the profile
        items:
          type: "string"
      resource:
        type: "string"
        description: A resource this profile may belong to
      context:
        type: "string"
        description: The context the profile may belong to
      description:
        type: "string"
        description: A description for this profile
      firstname:
        type: "string"
        description: The firstanme of the profile user
      lastname:
        type: "string"
        description: The lastname of the profile user
      publickey:
        type: "string"
        description: The lastname of the profile user
      email:
        type: "string"
        description: The email of the profile user
```

9.2.3 Organization

An important concept in many applications is the management of a group of users in an organization that manages a Big Data application or infrastructure. User group management can be achieved through three concepts. First, it can be achieved by using the profile and user resources itself as they contain the ability to manage multiple users as part of the REST interface. The second concept is to create a (virtual) organization that lists all users within the virtual organization. The third concept is to introduce groups and roles either as part of the user definition or as part of a simple list similar to the organization.

9.2.3.1 Properties Organization

Property	Type	Description
name	string	Name of the organization
users	array[../user/user.yaml#/definitions/Users]	list of users

9.2.3.2 Paths

9.2.3.2.1 /cloudmesh/organization

GET /cloudmesh/organization

Returns all users of the organization

Responses

Code	Description	Schema
200	organization info	Organization

PUT /cloudmesh/organization

Create a new organization

Responses

Code	Description	Schema
201	Created	

Parameters

Name	Located in	Description	Required	Schema
organization	body	The new organization to create	False	Organization

9.2.3.2.2 /cloudmesh/organization/{name}

GET /cloudmesh/organization/{name}

Returns the organization

Responses

Code	Description	Schema
200	organization info	Organization

Parameters

Name	Located in	Description	Required	Schema
name	path	The name of the organization	True	

9.2.3.3 organization.yaml

```
swagger: '2.0'
info:
  version: 3.0.3
  title: organization
  description: |-

    An important concept in many applications is the management of a
    group of users in an organization that manages a Big Data
    application or infrastructure. User group management can be
    achieved through three concepts. First, it can be achieved by
    using the profile and user resources itself as they contain the
    ability to manage multiple users as part of the REST
    interface. The second concept is to create a (virtual)
    organization that lists all users within the virtual
    organization. The third concept is to introduce groups and roles
    either as part of the user definition or as part of a simple list
    similar to the organization.

  termsOfService: 'https://github.com/cloudmesh-community/nist/blob/master/LICENSE.txt'
  contact:
    name: Cloudmesh RESTful Service Example
    url: https://cloudmesh-community.github.io/nist/spec/
  license:
    name: Apache
host: 'localhost:8080'
schemes:
  - http
consumes:
  - application/json
produces:
  - application/json
paths:
  /cloudmesh/organization:
    get:
      description: Returns all users of the organization
      operationId: get_organization
      produces:
        - application/json
      responses:
        '200':
          description: organization info
```

```
          schema:
            $ref: '#/definitions/Organization'
    put:
      summary: Create a new organization
      description: Create a new organization
      operationId: add_organization
      parameters:
        - in: body
          name: organization
          description: The new organization to create
          schema:
            $ref: '#/definitions/Organization'
      responses:
        '201':
          description: Created
  '/cloudmesh/organization/{name}':
    get:
      summary: Returns the organization
      description: Returns the organization
      operationId: get_organization_by_name
      parameters:
        - name: name
          description: The name of the organization
          in: path
          required: true
          type: string
      produces:
        - application/json
      responses:
        '200':
          description: organization info
          schema:
            $ref: '#/definitions/Organization'
definitions:
  Organization:
    type: object
    properties:
      name:
        description: Name of the organization
        type: string
      users:
        description: list of users
        type: array
        items:
          $ref: '../user/user.yaml#/definitions/Users'
```

9.2.4 Keystore

A service to store key, value, type information. All of them are stored as Strings.

9.2.4.1 Properties Key

Property	Type	Description
uuid	string	The uuid of the key, tha uuid must be unique
name	string	The name to access the key. The name must be unique
description	string	A description of the key
value	string	The string representing the key
kind	string	The type of the key

9.2.4.2 Paths

9.2.4.2.1 /cloudmesh/keystore/key

GET /cloudmesh/keystore/key

Returns all keys

Responses

Code	Description	Schema
200	key info	Key

PUT /cloudmesh/keystore/key

Create a new key

Responses

Code	Description	Schema
201	Created	

Parameters

Name	Located in	Description	Required	Schema
key	body	The new key to create	False	Key

9.2.4.2.2 /cloudmesh/keystore/key/{name}

GET /cloudmesh/keystore/key/{name}

Returns the key by name

Responses

Code	Description	Schema
200	key info	Key

Parameters

Name	Located in	Description	Required	Schema
name	path	ERROR: description missing	True	

9.2.4.3 keystore.yaml

```
---
swagger: '2.0'
info:
  version: 3.0.3
  x-date: 11-06-2018
  title: key
  description: |-

    A service to store key, value, type information. All of them are
    stored as Strings.

  termsOfService: 'https://github.com/cloudmesh-community/nist/blob/master/LICENSE.txt'
  contact:
    name: Cloudmesh RESTful Service Example
    url: https://cloudmesh-community.github.io/nist/spec/
  license:
    name: Apache
host: 'localhost:8080'
schemes:
  - http
consumes:
  - application/json
produces:
  - application/json
paths:
  /cloudmesh/keystore/key:
    get:
      tags:
        - "keystore"
      description: Returns all keys
      operationId: get_key
      produces:
        - application/json
      responses:
        '200':
          description: key info
          schema:
            $ref: '#/definitions/Key'
    put:
      tags:
        - "keystore"
      summary: Create a new key
      description: Create a new key
      operationId: add_key
      parameters:
        - in: body
          name: key
          description: The new key to create
          schema:
            $ref: '#/definitions/Key'
      responses:
        '201':
```

```
          description: Created
  /cloudmesh/keystore/key/{name}:
    get:
      tags:
        - "keystore"
      description: Returns the key by name
      operationId: get_key_by_name
      parameters:
        - name: name
          in: path
          required: true
          type: string
      produces:
        - application/json
      responses:
        '200':
          description: key info
          schema:
            $ref: '#/definitions/Key'
definitions:
  Key:
    type: object
    properties:
      uuid:
        type: string
        description: The uuid of the key, tha uuid must be unique
      name:
        type: string
        description: The name to access the key. The name must be unique
      description:
        type: string
        description: A description of the key
      value:
        type: string
        description: The string representing the key
      kind:
        type: string
        description: The type of the key
```

9.3 General Resources

9.3.1 Timestamp

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

- TODO: assign and review, we do not need a service for this but some mechanism so each resource has timestamps

9.3.1.1 Properties Timestamp

Property	Type	Description
accessed	string	ERROR: description missing

Property	Type	Description
created	string	ERROR: description missing
modified	string	ERROR: description missing

9.3.1.2 Paths

9.3.1.2.1 /cloudmesh/timestamps

GET /cloudmesh/timestamps

Returns all timestamps

Responses

Code	Description	Schema
200	timestamp info	Timestamp

PUT /cloudmesh/timestamps

ERROR: missing

Responses

Code	Description	Schema
201	Created	

Parameters

Name	Located in	Description	Required	Schema
timestamp	body	The new timestamp to create	False	Timestamp

9.3.1.2.2 /cloudmesh/timestamp/{name}

GET /cloudmesh/timestamp/{name}

Returns a timestamp

Responses

Code	Description	Schema
200	timestamp info	Timestamp

Parameters

Name	Located in	Description	Required	Schema
name	path	ERROR: description missing	True	

9.3.1.3 timestamp.yaml

```
swagger: '2.0'
info:
  version: 3.0.2
  x-date: 10-30-2018
  title: timestamp
  description: |-

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

    * TODO: assign and review, we do not need a service for this
      but some mechanism so each resource has timestamps

  termsOfService: 'https://github.com/cloudmesh-community/nist/blob/master/LICENSE.txt'
  contact:
    name: Cloudmesh RESTful Service Example
    url: https://cloudmesh-community.github.io/nist/spec/
  license:
    name: Apache
host: 'localhost:8080'
schemes:
  - http
consumes:
  - application/json
produces:
  - application/json
paths:
  /cloudmesh/timestamps:
    get:
      description: Returns all timestamps
      operationId: get_timestamp
      produces:
        - application/json
      responses:
        '200':
          description: timestamp info
          schema:
            $ref: '#/definitions/Timestamp'
    put:
      summary: Create a new timestamp
      operationId: add_timestamp
      parameters:
        - in: body
          name: timestamp
          description: The new timestamp to create
          schema:
            $ref: '#/definitions/Timestamp'
      responses:
        '201':
          description: Created
  '/cloudmesh/timestamp/{name}':
    get:
      description: Returns a timestamp
```

```
      operationId: get_timestamp_by_name
      parameters:
        - name: name
          in: path
          required: true
          type: string
      produces:
        - application/json
      responses:
        '200':
          description: timestamp info
          schema:
            $ref: '#/definitions/Timestamp'
definitions:
  Timestamp:
    type: object
    description: the timestamp
    properties:
      accessed:
        type: string
        format: date
      created:
        type: string
        format: date
      modified:
        type: string
        format: date
```

9.3.2 Alias

A user may be interested to create an alias for a resource. This is a name useful to the user. Users can deploy an alias server in which they store such aliasses for resources. Such aliasses could naturally be shared with othere. A resource could have one or more aliasses. The reason for an alias is that a resource may have a complex name but a user may want to refer to the resource using a name that is suitable for the user's application.

9.3.2.1 Properties Alias

Property	Type	Description
name	string	The name of the alias
origin	string	The original object name

9.3.2.2 Paths

9.3.2.2.1 /cloudmesh/alias

GET /cloudmesh/alias

Returns all aliases

Responses

Code	Description	Schema
200	alias information	Alias

PUT /cloudmesh/alias

ERROR: missing

Responses

Code	Description	Schema
201	Created	

Parameters

Name	Located in	Description	Required	Schema
alias	body	The new alias to create	False	Alias

9.3.2.2.2 /cloudmesh/alias/{name}

GET /cloudmesh/alias/{name}

Returns an alias

Responses

Code	Description	Schema
200	alias info	Alias

Parameters

Name	Located in	Description	Required	Schema
name	path	ERROR: description missing	True	

9.3.2.3 alias.yaml

```
swagger: '2.0'
info:
  version: 3.0.2
  x-date: 10-30-2018
  title: alias
  description: |-

    A user may be interested to create an alias for a resource. This
    is a name useful to the user. Users can deploy an alias server in
    which they store such aliasses for resources. Such aliasses could
    naturally be shared with othere. A resource could have one or more
    aliasses.  The reason for an alias is that a resource may have a
    complex name but a user may want to refer to the resource using a
    name that is suitable for the user's application.
```

```
  termsOfService: 'http://bin.io/terms/'
  contact:
    name: Cloudmesh RESTful Service Example
    url: https://cloudmesh-community.github.io/nist/spec/
  license:
    name: Apache
host: 'localhost:8080'
schemes:
  - http
consumes:
  - application/json
produces:
  - application/json
paths:
  /cloudmesh/alias:
    get:
      description: Returns all aliases
      operationId: get_alias
      produces:
        - application/json
      responses:
        '200':
          description: alias information
          schema:
            $ref: '#/definitions/Alias'
    put:
      summary: Create a new alias
      operationId: add_alias
      parameters:
        - in: body
          name: alias
          description: The new alias to create
          schema:
            $ref: '#/definitions/Alias'
      responses:
        '201':
          description: Created
  '/cloudmesh/alias/{name}':
    get:
      description: Returns an alias
      operationId: get_alias_by_name
      parameters:
        - name: name
          in: path
          required: true
          type: string
      produces:
        - application/json
      responses:
        '200':
          description: alias info
          schema:
            $ref: '#/definitions/Alias'
```

```
definitions:
  Alias:
    type: object
    description: the alias
    properties:
      name:
        type: string
        description: The name of the alias
      origin:
        type: string
        description: The original object name
```

9.3.3 Variables

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

- TODO: assign and review

9.3.3.1 Properties Variables

Property	Type	Description
name	string	name of the variable
value	string	type of the variable
kind	string	value of the variable

9.3.3.2 Paths

9.3.3.2.1 /cloudmesh/variables

GET /cloudmesh/variables

Returns all variabless

Responses

Code	Description	Schema
200	variables info	Variables

PUT /cloudmesh/variables

Create a new variables

Responses

Code	Description	Schema
201	Created	

Parameters

Name	Located in	Description	Required	Schema
variables	body	The new variables to create	False	Variables

9.3.3.2.2 /cloudmesh/variables/{name}

GET /cloudmesh/variables/{name}

Returns a variables

Responses

Code	Description	Schema
200	variables info	Variables

Parameters

Name	Located in	Description	Required	Schema
name	path	Name of the variable	True	

9.3.3.3 variables.yaml

```
swagger: '2.0'
info:
  version: 3.0.1
  title: variables
  description: |-

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

    * TODO: assign and review

  termsOfService: 'https://github.com/cloudmesh-community/nist/blob/master/LICENSE.txt'
  contact:
    name: Cloudmesh RESTful Service Example
    url: https://cloudmesh-community.github.io/nist/spec/
  license:
    name: Apache
host: 'localhost:8080'
schemes:
  - http
consumes:
  - application/json
produces:
  - application/json
paths:
  /cloudmesh/variables:
    get:
      description: Returns all variabless
```

```
      operationId: get_variables
      produces:
        - application/json
      responses:
        '200':
          description: variables info
          schema:
            $ref: '#/definitions/Variables'
    put:
      description: Create a new variables
      operationId: add_variables
      parameters:
        - in: body
          name: variables
          description: The new variables to create
          schema:
            $ref: '#/definitions/Variables'
      responses:
        '201':
          description: Created
  '/cloudmesh/variables/{name}':
    get:
      description: Returns a variables
      operationId: get_variables_by_name
      parameters:
        - name: name
          description: Name of the variable
          in: path
          required: true
          type: string
      produces:
        - application/json
      responses:
        '200':
          description: variables info
          schema:
            $ref: '#/definitions/Variables'
definitions:
  Variables:
    type: object
    description: the variables
    properties:
      name:
        type: string
        description: name of the variable
      value:
        type: string
        description: type of the variable
      kind:
        type: string
        description: value of the variable
```

9.3.4 Default

A default is a special variable that has a context associated with it. This allows one to define values that can be easily retrieved based on the associated context. For example, a default could be the image name for a cloud where the context is defined by the cloud name.

- TODO: assign for review and improvement

9.3.4.1 Properties Default

Property	Type	Description
name	string	name of the default
value	string	type of the default
kind	string	value of the default

9.3.4.2 Paths

9.3.4.2.1 /cloudmesh/defaults

GET /cloudmesh/defaults

Returns all defaults

Responses

Code	Description	Schema
200	default info	Default

PUT /cloudmesh/defaults

ERROR: missing

Responses

Code	Description	Schema
201	Created	

Parameters

Name	Located in	Description	Required	Schema
default	body	The new default to create	False	Default

9.3.4.2.2 /cloudmesh/default/{name}

GET /cloudmesh/default/{name}

Returns a default

Responses

Code	Description	Schema
200	default info	Default

Parameters

Name	Located in	Description	Required	Schema
name	path	ERROR: description missing	True	

9.3.4.3 default.yaml

```
swagger: '2.0'
info:
  version: 3.0.2
  x-date: 10-30-2018
  title: default
  description: |-

    A default is a special variable that has a context associated with
    it. This allows one to define values that can be easily retrieved
    based on the associated context. For example, a default could be
    the image name for a cloud where the context is defined by the
    cloud name.

    * TODO: assign for review and improvement

  termsOfService: 'https://github.com/cloudmesh-community/nist/blob/master/LICENSE.txt'
  contact:
    name: Cloudmesh RESTful Service Example
    url: https://cloudmesh-community.github.io/nist/spec/
  license:
    name: Apache
host: 'localhost:8080'
schemes:
  - http
consumes:
  - application/json
produces:
  - application/json
paths:
  /cloudmesh/defaults:
    get:
      description: Returns all defaults
      operationId: get_default
      produces:
        - application/json
      responses:
        '200':
          description: default info
          schema:
            $ref: '#/definitions/Default'
    put:
```

```
      summary: Create a new default
      operationId: add_default
      parameters:
        - in: body
          name: default
          description: The new default to create
          schema:
            $ref: '#/definitions/Default'
      responses:
        '201':
          description: Created
  '/cloudmesh/default/{name}':
    get:
      description: Returns a default
      operationId: get_default_by_name
      parameters:
        - name: name
          in: path
          required: true
          type: string
      produces:
        - application/json
      responses:
        '200':
          description: default info
          schema:
            $ref: '#/definitions/Default'
definitions:
  Default:
    type: object
    description: the defaults
    properties:
      name:
        type: string
        description: name of the default
      value:
        type: string
        description: type of the default
      kind:
        type: string
        description: value of the default
```

9.4 Data Management

9.4.1 Database

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

- TODO: assign for review and improvement

9.4.1.1 Properties Database

Property	Type	Description
name	string	name of the database
endpoint	string	endpoint of the database
kind	string	the kind of the database

9.4.1.2 Paths

9.4.1.2.1 /cloudmesh/databases

GET /cloudmesh/databases

Returns all databases

Responses

Code	Description	Schema
200	database info	Database

PUT /cloudmesh/databases

ERROR: missing

Responses

Code	Description	Schema
201	Created	

Parameters

Name	Located in	Description	Required	Schema
database	body	The new database to create	False	Database

9.4.1.2.2 /cloudmesh/database/{name}

GET /cloudmesh/database/{name}

Returns a database

Responses

Code	Description	Schema
200	database info	Database

Parameters

Name	Located in	Description	Required	Schema
name	path	ERROR: description missing	True	

9.4.1.3 database.yaml

```
swagger: '2.0'
info:
  version: 3.0.2
  x-date: 10-30-2018
  title: database
  description: |-

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

    * TODO: assign for review and improvement

  termsOfService: 'http://bin.io/terms/'
  contact:
    name: Cloudmesh RESTful Service Example
    url: https://cloudmesh-community.github.io/nist/spec/
  license:
    name: Apache
host: 'localhost:8080'
schemes:
  - http
consumes:
  - application/json
produces:
  - application/json
paths:
  /cloudmesh/databases:
    get:
      description: Returns all databases
      operationId: get_database
      produces:
        - application/json
      responses:
        '200':
          description: database info
          schema:
            $ref: '#/definitions/Database'
    put:
      summary: Create a new database
      operationId: add_database
      parameters:
        - in: body
          name: database
          description: The new database to create
          schema:
            $ref: '#/definitions/Database'
      responses:
        '201':
          description: Created
  '/cloudmesh/database/{name}':
    get:
      description: Returns a database
```

```
      operationId: get_database_by_name
      parameters:
        - name: name
          in: path
          required: true
          type: string
      produces:
        - application/json
      responses:
        '200':
          description: database info
          schema:
            $ref: '#/definitions/Database'
definitions:
  Database:
    type: object
    description: the database
    properties:
      name:
        type: string
        description: name of the database
      endpoint:
        type: string
        description: endpoint of the database
      kind:
        type: string
        description: the kind of the database
```

9.4.2 Virtual Directory

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

9.4.2.1 Properties UnauthorizedError

Property	Type	Description
code	string	Code form of the error
message	string	Human readable form of the error

9.4.2.2 Properties Virtualdirectory

Property	Type	Description
name	string	The name of the virtual directory
description	string	description of the virtual directory
host	string	remote host of the virtual directory
location	string	remote location, e.g., a directory with full path on a host
protocol	string	access protocol, e.g., HTTP, FTP, SSH, etc.
credential	object	credential to access, e.g., username, password

9.4.2.3 Paths

9.4.2.3.1 /cloudmesh/virtualdirectory

GET /cloudmesh/virtualdirectory

Returns all virtualdirectorys

Responses

Code	Description	Schema
200	virtualdirectory info	Virtualdirectory

Code	Description	Schema
401	unauthorized error	UnauthorizedError

PUT /cloudmesh/virtualdirectory

ERROR: missing

Responses

Code	Description	Schema
201	Created	

Parameters

Name	Located in	Description	Required	Schema
virtualdirectory	body	The new virtualdirectory to create	False	Virtualdirectory

9.4.2.3.2 /cloudmesh/virtualdirectory/{name}

GET /cloudmesh/virtualdirectory/{name}

Returns a virtualdirectory

Responses

Code	Description	Schema
200	virtualdirectory info	Virtualdirectory

Parameters

Name	Located in	Description	Required	Schema
name	path	ERROR: description missing	True	

9.4.2.4 virtualdirectory.yaml

```
---
swagger: '2.0'
info:
  version: 3.0.3
  x-date: 06-11-2018
  title: virtualdirectory
  description: |-

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

  termsOfService: 'http://bin.io/terms/'
  contact:
    name: Cloudmesh RESTful Virtual Directory Service Example
    url: https://cloudmesh-community.github.io/nist/spec/
  license:
    name: Apache
host: 'localhost:8080'
schemes:
  - http
consumes:
  - application/json
produces:
  - application/json
securityDefinitions:
  httpbasic:
    type: basic
  apikey:
    type: apiKey
    in: query
    name: api_key
  apisecret:
    type: apiKey
    in: query
    name: api_secret
paths:
  /cloudmesh/virtualdirectory:
    get:
      tags:
        - "virtualdirectory"
      description: Returns all virtualdirectorys
      operationId: get_virtualdirectory
      produces:
        - application/json
      security:
        - httpbasic: []
      responses:
        '200':
          description: virtualdirectory info
```

```
          schema:
            $ref: '#/definitions/Virtualdirectory'
        '401':
          description: unauthorized error
          schema:
            $ref: '#/definitions/UnauthorizedError'
    put:
      tags:
        - "virtualdirectory"
      summary: Create a new virtualdirectory
      operationId: add_virtualdirectory
      parameters:
        - in: body
          name: virtualdirectory
          description: The new virtualdirectory to create
          schema:
            $ref: '#/definitions/Virtualdirectory'
      security:
        - apikey: []
          apisecret: []
      responses:
        '201':
          description: Created
  '/cloudmesh/virtualdirectory/{name}':
    get:
      tags:
        - "virtualdirectory"
      description: Returns a virtualdirectory
      operationId: get_virtualdirectory_by_name
      parameters:
        - name: name
          in: path
          required: true
          type: string
      produces:
        - application/json
      security:
        - httpbasic: []
      responses:
        '200':
          description: virtualdirectory info
          schema:
            $ref: '#/definitions/Virtualdirectory'
definitions:
  UnauthorizedError:
    type: object
    description: A specific error
    properties:
      code:
        type: string
        description: Code form of the error
      message:
        type: string
        description: Human readable form of the error
```

```
Virtualdirectory:
  type: "object"
  description: the virtualdirectory
  properties:
    name:
      description: The name of the virtual directory
      type: "string"
    description:
      description: description of the virtual directory
      type: "string"
    host:
      description: remote host of the virtual directory
      type: "string"
    location:
      description: remote location, e.g., a directory with full path on a host
      type: "string"
    protocol:
      description: access protocol, e.g., HTTP, FTP, SSH, etc.
      type: "string"
    credential:
      description: credential to access, e.g., username, password
      type: "object"
```

9.4.3 File

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

- TODO: assign for review and improvement

9.4.3.1 Properties File

Property	Type	Description
name	string	the name of the file
endpoint	string	The location of the file
checksum	string	The checksum of the file
size	integer	The size of the file in byte
timestamp		ERROR: description missing

9.4.3.2 Paths

9.4.3.2.1 /cloudmesh/files

GET /cloudmesh/files

Returns all files

Responses

Code	Description	Schema
200	file info	File

PUT /cloudmesh/files

ERROR: missing

Responses

Code	Description	Schema
201	Created	

Parameters

Name	Located in	Description	Required	Schema
file	body	The new file to create	False	File

9.4.3.2.2 /cloudmesh/file/{name}

GET /cloudmesh/file/{name}

Returns a file

Responses

Code	Description	Schema
200	file info	File

Parameters

Name	Located in	Description	Required	Schema
name	path	ERROR: description missing	True	

9.4.3.3 file.yaml

```
swagger: '2.0'
info:
  version: 3.0.2
  x-date: 10-30-2018
  title: file
  description: |-

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

```
    MB, Byte, checksum for integrity check, and last accessed
    timestamp.

    * TODO: assign for review and improvement

  termsOfService: 'http://bin.io/terms/'
  contact:
    name: Cloudmesh RESTful Service Example
    url: https://cloudmesh-community.github.io/nist/spec/
  license:
    name: Apache
host: 'localhost:8080'
schemes:
  - http
consumes:
  - application/json
produces:
  - application/json
paths:
  /cloudmesh/files:
    get:
      description: Returns all files
      operationId: get_file
      produces:
        - application/json
      responses:
        '200':
          description: file info
          schema:
            $ref: '#/definitions/File'
    put:
      summary: Create a new file
      operationId: add_file
      parameters:
        - in: body
          name: file
          description: The new file to create
          schema:
            $ref: '#/definitions/File'
      responses:
        '201':
          description: Created
  '/cloudmesh/file/{name}':
    get:
      description: Returns a file
      operationId: get_file_by_name
      parameters:
        - name: name
          in: path
          required: true
          type: string
      produces:
        - application/json
      responses:
```

```
        '200':
          description: file info
          schema:
            $ref: '#/definitions/File'
definitions:
  File:
    type: object
    description: the file
    properties:
      name:
        type: string
        description: the name of the file
      endpoint:
        type: string
        description: The location of the file
      checksum:
        type: string
        description: The checksum of the file
      size:
        type: integer
        description: The size of the file in byte
      timestamp:
          $ref: '../timestamp/timestamp.yaml#/definitions/Timestamp'

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

9.4.4 Replica

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

- TODO: assign and improve

9.4.4.1 Properties Replica

Property	Type	Description
name	string	the name of the replica
filename	string	the original filename
endpoint	string	The location of the file
checksum	string	The checksum of the file
size	integer	The size of the file in byte
timestamp		ERROR: description missing

9.4.4.2 Paths

9.4.4.2.1 /cloudmesh/replicas

GET /cloudmesh/replicas

Returns all replicas

Responses

Code	Description	Schema
200	replica info	Replica

PUT /cloudmesh/replicas

ERROR: missing

Responses

Code	Description	Schema
201	Created	

Parameters

Name	Located in	Description	Required	Schema
replica	body	The new replica to create	False	Replica

9.4.4.2.2 /cloudmesh/replica/{name}

GET /cloudmesh/replica/{name}

Returns a replica

Responses

Code	Description	Schema
200	replica info	Replica

Parameters

Name	Located in	Description	Required	Schema
name	path	ERROR: description missing	True	

9.4.4.3 replica.yaml

```
swagger: '2.0'
info:
  version: 3.0.2
  x-date: 10-30-2018
  title: replica
  description: |-
```

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

    * TODO: assign and improve

  termsOfService: 'http://bin.io/terms/'
  contact:
    name: Cloudmesh RESTful Service Example
    url: https://cloudmesh-community.github.io/nist/spec/
  license:
    name: Apache
host: 'localhost:8080'
schemes:
  - http
consumes:
  - application/json
produces:
  - application/json
paths:
  /cloudmesh/replicas:
    get:
      description: Returns all replicas
      operationId: get_replica
      produces:
        - application/json
      responses:
        '200':
          description: replica info
          schema:
            $ref: '#/definitions/Replica'
    put:
      summary: Create a new replica
      operationId: add_replica
      parameters:
        - in: body
          name: replica
          description: The new replica to create
          schema:
            $ref: '#/definitions/Replica'
      responses:
        '201':
          description: Created
  '/cloudmesh/replica/{name}':
    get:
      description: Returns a replica
      operationId: get_replica_by_name
      parameters:
        - name: name
          in: path
          required: true
          type: string
```

```
      produces:
        - application/json
      responses:
        '200':
          description: replica info
          schema:
            $ref: '#/definitions/Replica'
definitions:
  Replica:
    type: object
    description: the replica
    properties:
      name:
        type: string
        description: the name of the replica
      filename:
        type: string
        description: the original filename
      endpoint:
        type: string
        description: The location of the file
      checksum:
        type: string
        description: The checksum of the file
      size:
        type: integer
        description: The size of the file in byte
      timestamp:
          $ref: '../timestamp/timestamp.yaml#/definitions/Timestamp'
```

9.5 Compute Management - Virtual Clutsers

9.5.1 Virtual Cluster

Virtual Cluster example

9.5.1.1 Properties VirtualCluster

Property	Type	Description
name	string	The name of the virtual cluster
description	string	A description of the virtual cluster
nnodes	integer	number of nodes of the virtual cluster
owner	string	owner of the virtual cluster
fe		Front-end node of the virtual cluster
nodes	array[#/definitions/Node]	List of nodes of the virtual cluster

9.5.1.2 Properties Node

Property	Type	Description
name	string	name of the node

Property	Type	Description
state	string	power state of the node
ncpu	integer	number of virtual CPUs of the node
ram	string	RAM size of the node
disk	string	Disk size of the node
nics	array[#/definitions/NIC]	List of network interfaces of the node

9.5.1.3 Properties NIC

Property	Type	Description
mac	string	MAC address of the node
ip	string	IP address of the node

9.5.1.4 Paths

9.5.1.4.1 /cloudmesh/virtualcluster/virtualcluster

GET /cloudmesh/virtualcluster/virtualcluster

Returns all virtualcluster

Responses

Code	Description	Schema
200	profile info	VirtualCluster

PUT /cloudmesh/virtualcluster/virtualcluster

ERROR: missing

Responses

Code	Description	Schema
201	Created	

Parameters

Name	Located in	Description	Required	Schema
virtualcluster	body	The new virtualcluster to create	False	VirtualCluster

9.5.1.4.2 /cloudmesh/virtualcluster/virtualcluster/{virtualclustername}

GET /cloudmesh/virtualcluster/virtualcluster/{virtualclustername}

Returns a virtualcluster by its name

Responses

Code	Description	Schema
200	virtualcluster info	VirtualCluster

Parameters

Name	Located in	Description	Required	Schema
virtualclustername	path	ERROR: description missing	True	

9.5.1.4.3 /cloudmesh/virtualcluster/virtualcluster/{virtualclustername}/fe

GET /cloudmesh/virtualcluster/virtualcluster/{virtualclustername}/fe

Returns the front-end node info of the specified virtualcluster

Responses

Code	Description	Schema
200	virtualcluster front-end info	Node

Parameters

Name	Located in	Description	Required	Schema
virtualclustername	path	ERROR: description missing	True	

9.5.1.4.4 /cloudmesh/virtualcluster/virtualcluster/{virtualclustername}/{nodename}

GET /cloudmesh/virtualcluster/virtualcluster/{virtualclustername}/{nodename}

Returns the specified node info of the specified virtualcluster

Responses

Code	Description	Schema
200	virtualcluster node info	Node

Parameters

Name	Located in	Description	Required	Schema
virtualclustername	path	ERROR: description missing	True	
nodename	path	ERROR: description missing	True	

9.5.1.5 vc.yaml

```
---
swagger: "2.0"
info:
  version: 3.0.3
  x-date: 06-11-2018
```

```
  title: "Virtual Cluster"
  description: |-

    Virtual Cluster example

  termsOfService: "https://github.com/cloudmesh-community/nist/blob/master/LICENSE.txt"
  contact:
    name: "Cloudmesh REST Service Example"
    url: https://cloudmesh-community.github.io/nist/spec/
  license:
    name: "Apache"
host: "localhost:8080"
schemes:
  - "http"
consumes:
  - "application/json"
produces:
  - "application/json"
paths:
  /cloudmesh/virtualcluster/virtualcluster:
    get:
      tags:
        - "virtualcluster"
      description: "Returns all virtualcluster"
      operationId: get_virtualcluster
      produces:
        - "application/json"
      responses:
        "200":
          description: "profile info"
          schema:
            $ref: "#/definitions/VirtualCluster"
    put:
      tags:
        - "virtualcluster"
      summary: "Create a new virtualcluster"
      operationId: add_virtualcluster
      parameters:
        - in: body
          name: virtualcluster
          description: "The new virtualcluster to create"
          schema:
            $ref: "#/definitions/VirtualCluster"
      responses:
        "201":
          description: Created
  /cloudmesh/virtualcluster/virtualcluster/{virtualclustername}:
    get:
      tags:
        - "virtualcluster"
      description: "Returns a virtualcluster by its name"
      operationId: getVirtualclusterByName
      parameters:
        - name: virtualclustername
```

```
          in: path
          required: true
          type: string
      produces:
        - "application/json"
      responses:
        "200":
          description: "virtualcluster info"
          schema:
            $ref: "#/definitions/VirtualCluster"
  /cloudmesh/virtualcluster/virtualcluster/{virtualclustername}/fe:
    get:
      tags:
        - "virtualcluster"
      description: "Returns the front-end node info of the specified virtualcluster"
      operationId: getVirtualclusterFeByName
      parameters:
        - name: virtualclustername
          in: path
          required: true
          type: string
      produces:
        - "application/json"
      responses:
        "200":
          description: "virtualcluster front-end info"
          schema:
            $ref: "#/definitions/Node"
  /cloudmesh/virtualcluster/virtualcluster/{virtualclustername}/{nodename}:
    get:
      tags:
        - "virtualcluster"
      description: "Returns the specified node info of the specified virtualcluster"
      operationId: getVirtualclusterNodeByName
      parameters:
        - name: virtualclustername
          in: path
          required: true
          type: string
        - name: nodename
          in: path
          required: true
          type: string
      produces:
        - "application/json"
      responses:
        "200":
          description: "virtualcluster node info"
          schema:
            $ref: "#/definitions/Node"
definitions:
  VirtualCluster:
    type: "object"
    properties:
```

```
    name:
      description: The name of the virtual cluster
      type: "string"
    description:
      type: "string"
      description: A description of the virtual cluster
    nnodes:
      type: "integer"
      description: number of nodes of the virtual cluster
    owner:
      type: "string"
      description: owner of the virtual cluster
    fe:
      description: Front-end node of the virtual cluster
      $ref: "#/definitions/Node"
    nodes:
      description: List of nodes of the virtual cluster
      type: array
      items:
        $ref: "#/definitions/Node"
Node:
  type: "object"
  properties:
    name:
      type: "string"
      description: name of the node
    state:
      type: "string"
      description: power state of the node
    ncpu:
      type: "integer"
      description: number of virtual CPUs of the node
    ram:
      type: "string"
      description: RAM size of the node
    disk:
      type: "string"
      description: Disk size of the node
    nics:
      type: "array"
      description: List of network interfaces of the node
      items:
        $ref: "#/definitions/NIC"
NIC:
  type: "object"
  properties:
    mac:
      type: "string"
      description: MAC address of the node
    ip:
      type: "string"
      description: IP address of the node
```

9.5.2 Scheduler

A service to store scheduler, value, type information. All of them are stored as Strings.

- TODO: assign and improve

9.5.2.1 Properties Scheduler

Property	Type	Description
name	string	name of the scheduler
value	string	value of the scheduler
kind	string	the scheduler kind or type

9.5.2.2 Paths

9.5.2.2.1 /cloudmesh/schedulers

GET /cloudmesh/schedulers

Returns all schedulers

Responses

Code	Description	Schema
200	scheduler info	Scheduler

PUT /cloudmesh/schedulers

ERROR: missing

Responses

Code	Description	Schema
201	Created	

Parameters

Name	Located in	Description	Required	Schema
scheduler	body	The new scheduler to create	False	Scheduler

9.5.2.2.2 /cloudmesh/scheduler/{name}

GET /cloudmesh/scheduler/{name}

Returns a scheduler

Responses

Code	Description	Schema
200	scheduler info	Scheduler

Parameters

Name	Located in	Description	Required	Schema
name	path	ERROR: description missing	True	

9.5.3 scheduler.yaml

```
swagger: '2.0'
info:
  version: 3.0.2
  x-date: 10-30-2018
  title: scheduler
  description: |-

    A service to store scheduler, value, type information. All of them
    are stored as Strings.

    * TODO: assign and improve

  termsOfService: 'https://github.com/cloudmesh-community/nist/blob/master/LICENSE.txt'
  contact:
    name: Cloudmesh RESTful Service Example
    url: https://cloudmesh-community.github.io/nist
  license:
    name: Apache
host: 'localhost:8080'
schemes:
  - http
consumes:
  - application/json
produces:
  - application/json
paths:
  /cloudmesh/schedulers:
    get:
      description: Returns all schedulers
      operationId: get_scheduler
      produces:
        - application/json
      responses:
        '200':
          description: scheduler info
          schema:
            $ref: '#/definitions/Scheduler'
    put:
      summary: Create a new scheduler
      operationId: add_scheduler
      parameters:
        - in: body
          name: scheduler
          description: The new scheduler to create
          schema:
```

```
            $ref: '#/definitions/Scheduler'
      responses:
        '201':
          description: Created
  '/cloudmesh/scheduler/{name}':
    get:
      description: Returns a scheduler
      operationId: get_scheduler_by_name
      parameters:
        - name: name
          in: path
          required: true
          type: string
      produces:
        - application/json
      responses:
        '200':
          description: scheduler info
          schema:
            $ref: '#/definitions/Scheduler'
definitions:
  Scheduler:
    type: object
    description: the scheduler
    properties:
      name:
        type: string
        description: name of the scheduler
      value:
        type: string
        description: value of the scheduler
      kind:
        type: string
        description: the scheduler kind or type
```

9.6 Compute Management - Virtual Machines

This section is planned for a future version.

9.6.1 Image

An operating system image.

Image objects are typically returned by the driver for the cloud provider in response to the list_images function.

- TODO: this is a demo of the to do line

9.6.1.1 Properties Image

Property	Type	Description
id	string	A unique id for the image

Property	Type	Description
name	string	The name of the image
tag	string	A tag that can be defined by the user for the image
description	string	A description for the image
cloud	string	A cloud provider for which the image is designed. If multiple are using the image, they are passed along as space seperated strings
os	string	The OS of the image
osVersion	string	The OS version of the image
status	string	The status of the image
visibility	string	The visibility of the image
extra	string	Extra object of the image

9.6.1.2 Paths

9.6.1.2.1 /cloudmesh/image

GET /cloudmesh/image

Returns all general description images

Responses

Code	Description	Schema
200	default image info	Image

PUT /cloudmesh/image

ERROR: missing

Responses

Code	Description	Schema
201	Created	

Parameters

Name	Located in	Description	Required	Schema
image	body	The new image to create	False	Image

9.6.1.2.2 /cloudmesh/image/{name}

GET /cloudmesh/image/{name}

Returns a general description of an image

Responses

Code	Description	Schema
200	image info	Image

Parameters

Name	Located in	Description	Required	Schema
name	path	ERROR: description missing	True	

9.6.1.3 image.yaml

```
swagger: '2.0'
info:
  version: 3.0.2
  x-date: 10-30-2018
  title: image
  description: |-

    An operating system image.

    Image objects are typically returned by the driver for the
    cloud provider in response to the list_images function.

    * TODO: this is a demo of the to do line

  termsOfService: 'http://bin.io/terms/'
  contact:
    name: Cloudmesh RESTful Service Example
    url: https://cloudmesh-community.github.io/nist/spec/
  license:
    name: Apache
host: 'localhost:8080'
schemes:
  - http
consumes:
  - application/json
produces:
  - application/json
paths:
  /cloudmesh/image:
    get:
      description: Returns all general description images
      operationId: get_image
      produces:
        - application/json
      responses:
        '200':
          description: default image info
          schema:
            $ref: '#/definitions/Image'
    put:
      summary: Create a new image
      operationId: add_image
      parameters:
        - in: body
          name: image
          description: The new image to create
```

```
          schema:
            $ref: '#/definitions/Image'
      responses:
        '201':
          description: Created
  '/cloudmesh/image/{name}':
    get:
      description: Returns a general description of an image
      operationId: get_image_by_name
      parameters:
        - name: name
          in: path
          required: true
          type: string
      produces:
        - application/json
      responses:
        '200':
          description: image info
          schema:
            $ref: '#/definitions/Image'
definitions:
  Image:
    type: object
    properties:
      id:
        description: A unique id for the image
        type: string
      name:
        description: The name of the image
        type: string
      tag:
        description: A tag that can be defined by the user for the image
        type: string
      description:
        description: A description for the image
        type: string
      cloud:
        description: A cloud provider for which the image is designed. If multiple are using the image,
        type: string
      os:
        description: The OS of the image
        type: string
      osVersion:
        description: The OS version of the image
        type: string
      status:
        description: The status of the image
        type: string
      visibility:
        description: The visibility of the image
        type: string
      extra:
        description:  Extra object of the image
```

```
type: string
```

9.6.2 Flavor

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

x-date: 10-30-2018

9.6.2.1 Properties Flavor

Property	Type	Description
name	string	name of the flavor
id	string	the id of the flavor
label	string	a label that a user can set for this flavor
metadata	string	A dictionary with additional metadata
uuid	string	the uuid of the flavor
ram	integer	number of bytes used for the image in RAM
disk	integer	number of bytes used for the disk
price	string	price for the flavor
cloud	string	name of the cloud this flavor is used
cloud_id	string	an id used by the cloud

9.6.2.2 Paths

9.6.2.2.1 /cloudmesh/flavors

GET /cloudmesh/flavors

Returns all flavors

Responses

Code	Description	Schema
200	flavor info	Flavor

PUT /cloudmesh/flavors

ERROR: missing

Responses

Code	Description	Schema
201	Created	

Parameters

Name	Located in	Description	Required	Schema
flavor	body	The new flavor to create	False	Flavor

9.6.2.2.2 /cloudmesh/flavor/{name}

GET /cloudmesh/flavor/{name}

Returns a flavor

Responses

Code	Description	Schema
200	flavor info	Flavor

Parameters

Name	Located in	Description	Required	Schema
name	path	ERROR: description missing	True	

9.6.2.3 image.yaml

```
swagger: '2.0'
info:
  version: 3.0.3
  title: flavor
  description: |-

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

    x-date: 10-30-2018

  termsOfService: 'https://github.com/cloudmesh-community/nist/blob/master/LICENSE.txt'
  contact:
    name: Cloudmesh RESTful Service Example
    url: https://cloudmesh-community.github.io/nist
  license:
    name: Apache
    url: https://github.com/cloudmesh-community/nist/blob/master/LICENSE.txt
host: 'localhost:8080'
schemes:
  - http
consumes:
  - application/json
produces:
  - application/json
paths:
  /cloudmesh/flavors:
    get:
      description: Returns all flavors
      operationId: get_flavor
      produces:
        - application/json
      responses:
```

```
        '200':
          description: flavor info
          schema:
            $ref: '#/definitions/Flavor'
    put:
      summary: Create a new flavor
      operationId: add_flavor
      parameters:
        - in: body
          name: flavor
          description: The new flavor to create
          schema:
            $ref: '#/definitions/Flavor'
      responses:
        '201':
          description: Created
  '/cloudmesh/flavor/{name}':
    get:
      description: Returns a flavor
      operationId: get_flavor_by_name
      parameters:
        - name: name
          in: path
          required: true
          type: string
      produces:
        - application/json
      responses:
        '200':
          description: flavor info
          schema:
            $ref: '#/definitions/Flavor'
definitions:
  Flavor:
    type: object
    description: the flavor
    properties:
      name:
        type: string
        description: name of the flavor
      id:
        type: string
        description: the id of the flavor
      label:
        type: string
        description: a label that a user can set for this flavor
      metadata:
        type: string
        description: A dictionary with additional metadata
      uuid:
        type: string
        description: the uuid of the flavor
      ram:
        type: integer
```

```
      description: number of bytes used for the image in RAM
    disk:
      type: integer
      description: number of bytes used for the disk
    price:
      type: string
      description: price for the flavor
    cloud:
      type: string
      description: name of the cloud this flavor is used
    cloud_id:
      type: string
      description: an id used by the cloud
```

9.6.3 Vm

A service to manage virtual machines

9.6.3.1 Properties VM

Property	Type	Description
provider	string	Name of the provider
id	string	a unique id for the vm
name	string	the name of the vm
image	string	the image for the vm
region	string	an optional region
size	string	The size of the vm
state	string	The state of the vm
private_ips	string	The private IPs
public_ips	string	The public IPS
metadata	string	The meta data passed along to the VM

9.6.3.2 Paths

9.6.3.2.1 /vm

GET /vm

Returns the list of the vms

Responses

Code	Description	Schema
200	Listing the VMs	VM

Parameters

Name	Located in	Description	Required	Schema
cloud	query	specify the cloud from which we list, if ommitted all clouds are returned.	False	

9.6.3.3 vm.yaml

```
swagger: '2.0'
info:
  description: |-

    A service to manage virtual machines

  version: 3.0.5
  x-date: 11-16-2018
  title: Coludmesh VM
  termsOfService: 'https://github.com/cloudmesh-community/nist/blob/master/LICENSE.txt'
  contact:
    name: Gregor von Laszeewski
  license:
    name: Apache
host: 'localhost:8080'
basePath: /cloudmesh/v3
schemes:
  - http
consumes:
  - application/json
produces:
  - application/json
paths:
  /vm:
    get:
      tags:
        - vm
      description: Returns the list of the vms
      operationId: vm_list
      produces:
        - application/json
      parameters:
        - name: cloud
          in: query
          type: string
          description: 'specify the cloud from which we list, if ommitted all clouds are returned.'
          required: false
      responses:
        '200':
          description: Listing the VMs
          schema:
            $ref: '#/definitions/VM'
definitions:
  VM:
    type: object
    properties:
      provider:
```

```
      type: string
      description: Name of the provider
    id:
      type: string
      description: a unique id for the vm
    name:
      type: string
      description: the name of the vm
    image:
      type: string
      description: the image for the vm
    region:
      type: string
      description: an optional region
    size:
      type: string
      description: The size of the vm
    state:
      type: string
      description: The state of the vm
    private_ips:
      type: string
      description: The private IPs
    public_ips:
      type: string
      description: The public IPS
    metadata:
      type: string
      description: The meta data passed along to the VM
  example:
    image: image
    metadata: metadata
    size: size
    provider: provider
    name: name
    private_ips: private_ips
    id: id
    state: state
    region: region
    public_ips: public_ips
```

9.7 Compute Management - Containers

This section is planned for a future version.

9.8 Compute Management - Functions

This section is planned for a future version.

9.9 Others

Please notify us if you would like to add other specifications.

10 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 following:

- The HTTP response code;
- An accompanying message for the HTTP response code; and
- A field or object where the error occurred.

Table 1: HTTP Response Codes

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

11 Acronyms and Terms

The following acronyms and terms are used in this volume.

ACID - Atomicity, Consistency, Isolation, Durability

API - Application Programming Interface

ASCII American Standard Code for Information Interchange

BASE Basically Available, Soft state, Eventual consistency

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

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

DevOps A clipped compound of software DEVelopment and information technology OPerationS

Deployment The action of installing software on resources

HTTP HyperText Transfer Protocol HTTPS HTTP Secure

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

IaaS Infrastructure as a Service SaaS Software as a Service

ITL Information Technology Laboratory

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

NBD-PWG NIST Big Data Public Working Group

NBDRA NIST Big Data Reference Architecture

NBDRAI NIST Big Data Reference Architecture Interface

NIST National Institute of Standards and Technology

OS Operating System

REST REpresentational State Transfer

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

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

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

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

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

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

Workflow The sequence of processes or tasks

WWW World Wide Web

12 Bibliography

- [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, Reference Architecture Interfaces. Special Publication (NIST SP) - 1500-9 1500-9, National Institute of Standards, 100 Bureau Drive, Gaithersburg, MD 20899, October 2015. URL: https://laszewski.github.io/papers/NIST.SP.1500-9-draft.pdf, doi:NIST.SP.1500-9.
- [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.