Cloudmesh in support of the NIST Big Data Architecture Framework

Gregor von Laszewski, Fugang Wang, Badi Abdul-Wahid,
Hyungro Lee, Geoffrey C. Fox

CONTENTS

Cloudmesh in support of the NIST Big Data Architecture Framework

Gregor von Laszewski, Fugang Wang,
Badi Abdul-Wahid, Hyungro Lee,
Geoffrey C. Fox
Indiana University
Bloomington, IN
laszewski@gmail.com

Wo Chang
NIST Big Data Public Working Group
National Institute of Standards and Technology
wo.chang@nist.gov

ABSTRACT

The National Institute of Standards and Technology (NIST) has provided a big data reference architecture and is currently attempting to validate that architecture. As part of our current efforts we are developin cloudmesh a tool that sets its goal towards easily managing multiple clouds, container, batch queuesexposed as services to its users. It also allows the integration of defined software stacks that can be used to deploy complex and state-of-the-art frameworks with devOps tools. Cloudmesh is on purpose designed to be vendor agnostic. We evaluate in this paper two aspects. First, based on our rich experience with clouds and other infrastructures, *can we verify the NIST reference architecture from our point of view?* Second, *which limitations may exist in cloudmesh that need to be addressed to potentially improve integration with the NIST efforts?* We will see in this paper that cloudmesh validates the NIST big data architecture which itself motivated further improvements to cloudmesh that we have implemented.

1. INTRODUCTION

Commercial, academic, and government leaders agree about the 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 [1]. Desirable is as a vendor neutral, technology- and infrastructure-agnostic conceptual model used to examine related issues. The focus of this document is on the NIST Big Data Reference Architecture and identify lessons learned from cloudmesh that can augment and verify this architecture through practical use cases. In the next sections we provide some background information to motivate our work and to introduce the two frameworks influencing this paper. This includes cloudmesh [2] [3] and the NIST Big Data Reference Architecture [1].

Our paper is structured as follows. We start by giving a brief introduction to the National Institute of Standards (NIST) Big Data Reference Architecture (Section 1.1) followed by a brief introduction to Cloudmesh (Section 2). We then provide a more in depth description of Cloudmesh while focussing on Cloudmesh's abstraction which are useful for big data analysis (Section 3). Next we investigate some usecases brought forward by the NIST Big Data Working Group and see how cloudmesh can help deploying and executing them (Section 4) .

We conclude our paper with observations made while implementing these usecases impacting the NIST Big Data Architecture and also cloudmesh (Section 6)

1.1 NIST Big Data Reference Architecture

The NIST big data working group is exploring pathways forward in this direction which can be leveraged by the community. The current result reference architecture is summarized in [1]. From this document we gather that the *"conceptual model, referred to as the NIST Big Data Reference Architecture (NBDRA), was crafted by examining publicly available Big Data architectures representing various approaches and products. Inputs from the other NBD-PWG subgroups were also incorporated into the creation of the NBDRA. It is applicable to a variety of business environments, including tightly integrated enterprise systems, as well as loosely coupled vertical industries that rely on cooperation among independent stakeholders. The NBDRA captures the two known Big Data economic value chains: information, where value is created by data collection, integration, analysis, and applying the results to data-driven services, and the information technology (IT), where value is created by providing networking, infrastructure, platforms, and tools in support of vertical data-based applications."* It will produce a number of documents related to definitions [4], taxonomies [5], use cases and general requirements [6], security and privacy [7], architectures white paper survey [8], reference architecture [1], standards roadmap [9]. In addition we expect NIST to work on an interface definition document that is partially influenced by the work presented in this paper.

One of the desired tasks is to identify existing frameworks and to analyze how they correlate to the current reference architecture [1]. This is conducted in order to validate and if needed to improve the architecture. The current NIST big data reference architecture(NBDRA) is shown in Figure 5. However we have augmented the architecture with areas where cloudmesh interfaces with it. We will describe the offerings in relationship to this architecture of cloudmesh in detail in Section **??**. The NBDRA contains according to [1] the following components and services:

System Orchestrator - provides high-level design dataflow between analytics tools and given datasets, computing system requirements, monitoring system resource and performance. At times the system orchestration is performed by the data scientist in an interactive manner.

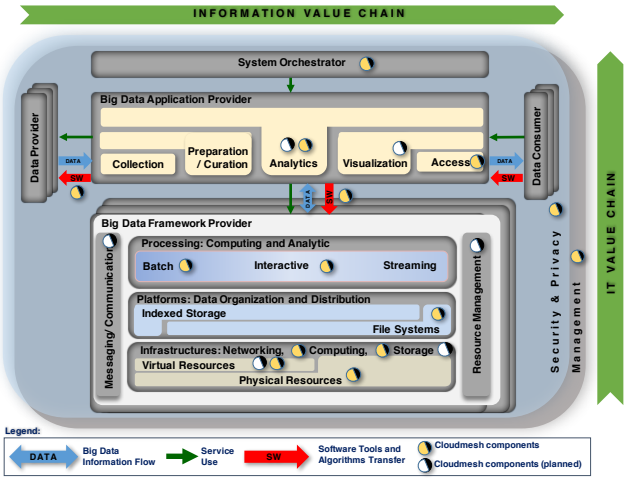

Figure 1: NIST Big Data Reference Architecture (NBDRA) diagram

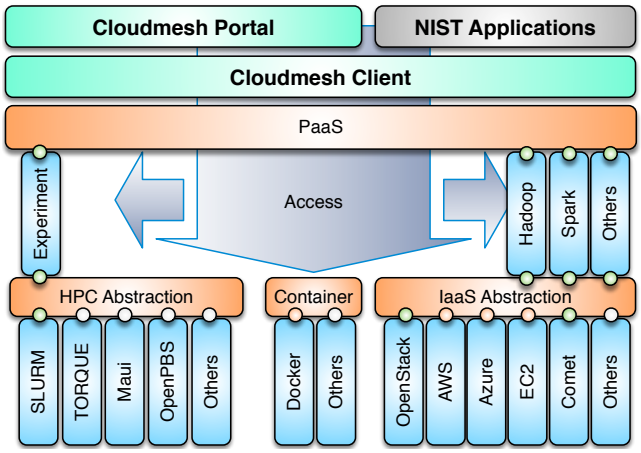

Figure 2: Cloudmesh layered architecture.

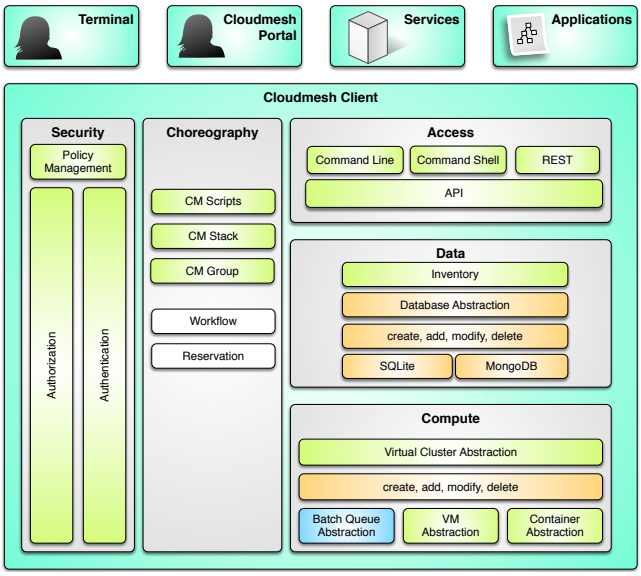

Figure 3: Cloudmesh components.

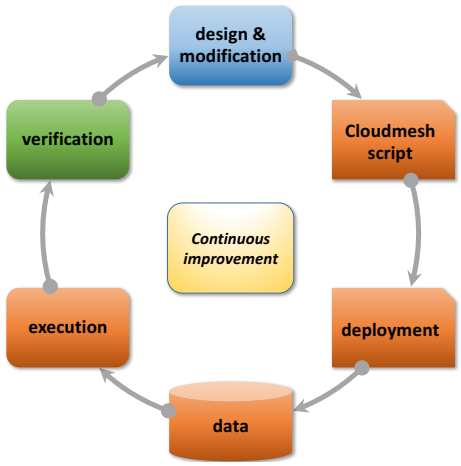

Figure 4: Continuous improvement while using cloudmesh interactively.

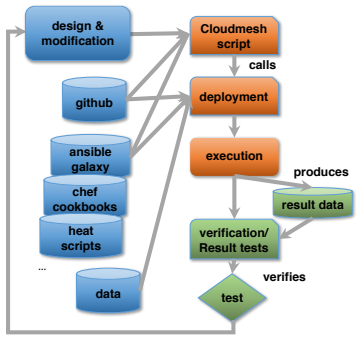

Figure 5: Interaction of the continuous improvement steps with various databases while using ansible deployment scripts.

The system orchestrator is a service or component that acts in behalf of the data scientist, or another data science service that would require a particular configuration.

Data Provider - provides abstraction of various types of data sources (such as raw data or data previously transformed by another system) and makes them available through different functional interfaces. This includes transfer analytics codes to data sources for effective analytic processing.

Big Data Application Provider - provides analytics processing throughout the data lifecycle - acquisition, curation, analysis, visualization, and access - to meet requirements established by the System Orchestrator.

Big Data Framework Provider - provides one or more instances of computing environment to support general Big Data tools, distributed file systems, and computing infrastructure - to meet requirements established by the Big Data Application Provider.

Data Consumer - provides interface to receive the value output from this NBD-RA ecosystem.

Security and Privacy Fabric - provides System Orchestrator the security and privacy interaction to the rest of the NBD-RA components to ensure protection of data and their content.

Management Fabric - provides System Orchestrator the management interaction to the rest of the NBD-RA components with versatile system and software provisioning, resource and performance monitoring, while maintaining a high level of data quality and secure accessibility.

2. CLOUDMESH

The cloudmesh is a lightweight client interface of accessing heterogeneous clouds, clusters, and workstations right from the users computer. The user can manage her own set of resources she would like to utilize. Thus the user has the freedom to customize their cyber infrastructure they use. Cloudmesh includes an API, a command-line client, and a command-line shell. It strives to abstract backends to databases that are used to manage the workflow utilizing the different infrastructure and also the services. Switching for example to stage virtual machines from OpenStack clouds to amazon is as simple as specifying the name of the cloud. Moreover, cloudmesh can be installed on Linux, MacOSX, and in future Windows. Currently cloudmesh supports backends to SLURM, SSH, OpenStack, AWS, and Azure. In the past we supported AWS and Azure. Cloudmesh allows to easily manage virtual machines, containers, HPC tasks, through a convenient client and API. Hence cloudmesh is not only a multi-cloud, but a multi-hpc environment that allows also to use container technologies (under development). Additionally, we have an example code on how to create a Web based portal with the help of cloudmesh.

2.1 General Requirements and Goals

Cloudmesh has from its inception followed the following general design goals and requirements.

Technology agnostic. An important aspect of cloudmesh is to offer access to useful services APIs and interfaces in a technology agnostic fashion. Thus it should be possible for example to switch easily between different IaaS providers. CM has excellently protected us during the changes of the OpenStack interfaces and libraries from the very first version of OpenStack. It also has allowed us to switch easily to different IaaS providers when it became clear that Eucalyptus was replaced by many with OpenStack.

Easy to use. Cloudmesh is supposed to make access of the complex workflow to integrate with IaaS and deploy on them new platforms and software stacks. This advanced feature is not only to be performed by expert IT personal or programmers, but in fact by data scientists which we found in practice have less experience in such areas. As we deal often with many compute nodes it is often insufficient to just provide a graphical user interface or portal, but we need to provide APIs, REST interfaces and especially command-line and shell access in an easy comprehensible fashion.

Expandable. While working over the last years in the area it is obvious that the technology is rapidly evolving and new features need to be integrated, Thus it is important that cloudmesh is easy to expand and new features can be added while leveraging a core set of functionality and services.

Documented. Furthermore it is important that we provide from the start documentation to existing and new features and make it easy for the developers to contribute documented add ons, but also allow users to have access to documentation including examples. This will include easy to follow documented installation and configuration steps to guarantee successful deployment and use

Repeatable and automated deployment. When we install software stacks an a variety of platforms it is expected that that can easily be replicated and repeated automatically.

Portable. It is important to provide services in cross platforms compatible fashion. It ensures working, executable software deployment on multiple platforms. This includes not only the installation of the software, but the integration of external services and tools such as DevvOps or workflow frameworks that could support the general mission of a data scientist.

Abstractions. To address some of the design issues our requirements implicitly asks for the existence of a number of abstractions and interfaces that can be used to enable portable and crossplatform services and tools.

2.2 Architecture Requirements and Goals

In addition to the general requirements we set some specific architectural requirements and goals that have been growing from previous versions of cloudmesh and our earlier work in this area.

Client based. Cloudmesh is a client based toolkit that is installed and run on the users computers. An add on component to use the client within a portal is available. Thus we distinguish the client that contains most of the functionality and the portal that can access the functionality through a locally maintaine Web portal. Important to note is that the user manages its own credentials and thus security and credential management is done directly on the users machine instead through a hosted Web portal. This increases the security as access to any credential is managed by the user and is not part of a credential management system.

REST. Although Cloudmesh provides a client interface, it will provide a REST interface to many of its services in order to support service based deployments. The basic APIs developed for the client can easily be reused to implement such interfaces.[1]

Layered Architecture. Cloudmesh has a layered architecture that allows easy development of new features. This also allows contribution by the community while developing integrated and smaller sub components. Figure A depicts the various layers. A resource abstraction layer allows the integration of a multitude of resources spanning HPC, Containers, and Cloud resources. (At this time we focus on OpenStack and Slurm resources. We are working on reintegrating resources such as Azure, AWS, Maui, Moab, and oth-

[1]we have demonstrated that cloudmesh APIs can be used to implement REST interfaces in a variety of frameworkks such as Flask, Django, and Cherypy

ers which we previously supported, as well as new resources such as docker).

Management Framework. Cloudmesh contains a management framework, and its components are depicted in Figure B. cloudmesh allows easy management of virtual machines, containers, and the data associated with them. We are currently developing a choreography framework that leverages Ansible, chef, and heat. All of the functionality is easily usable through a comand-shell that also can be used from the command-line, and a Python API. IN future we will be providing a REST API.

Database Agnostic. Cloudmesh contains some state about the resource and environment that a user may want to use. The information is managed in an database abstraction that would allow storing the data in a variety of databases such as SQL and MongoDB. At this time we have chosen SQLite to be the default database as it does not require any additional setup and is universally available on all operating systems without change.

comand-shell and line. Cloudmesh contains a comand-shell allowing scripts to be developed and run. However we designed the comand-shell in such a way that each command can also be called from the command-line. Through the cloudmesh state machine the state between comand-shell, command-client, and the portal is shared.

Cloudmesh Client Portal. Previously, we distributed cloudmesh with client, server, and a portal components in one package. This however turned out to be to complex to be installed for some of our less technically skilled user community. Thus we split up the install into two independent packages. The cloudmesh client and the cloudmesh portal. The portal provides some elementary features to manage virtual machines and HPC jobs. At this time the portal is considered to be alpha technology. Just as the client the portal is to be run on the local user machine in oredr to allow increased security by managing the credentials locally rather than on a server.

Cloudmesh Two Factor Authentication. We have an exploratory project in place that looks at the use of Yubikeys for cloudmesh, client and cloudmesh portal.

Cloudmesh Comet. We have developed the client interface for SDSC's comet supercomputer allowing bare metal provisioning. The interface reuses cloudmesh components and technologies while interfacing with the comet cloud REST interface. The goal here is to manage virtual clusters.

3. CLOUDMESH ABSTRACTIONS

In this paper we will focus our attention to three important abstractions that cloudmesh introduces: Cloudmesh Compute Experiments (Section 3.1) , Cloudmesh Virtual Clusters (Section 3.2), Cloudmesh Stacks (Section 3.6).

3.1 Compute Experiments

For many decades traditional supercomputing has provided large scale resources to the research community. This is done in a shared operating mode and enables researchers to utilize resources that they otherwise would not have access to. Shared resources include memory over a number of processors, shared disks, powerful processors in speed and core numbers, fast interconnect. This has been applied to many modeling applications but can naturally also be applied to big data. In order to integrate such capabilities in a service oriented fashion the Grid community has delivered prior to to e popularization of cloudcomputing useful interfaces, API's, and toolkits. However for research work that we supported with cloudmesh such interfaces and efforts were to complex to use and we designed a simple model for researchers that we worked with. This interface includes the abstract concept of a "job" that is submitted to a queuing system and uses a particular data set. The experiments in our case are repeated multiple times and each run creates its own output directory.

Hence we have created a simple submission interface that submits the script to the cluster. The script will be copied prior to execution into the home directory on the remote machine. If a directory is specified it will be copied into that dir. The name of the output directory is either specified in the script itself, or if not the default naming scheme of cloudmesh is used using and the output directory name is appended with an increasing index. To run such a script we can on our client simply say

```
cms hpc run SCRIPT
```

Some more details about out improved interfaces are given in Figure 1.

3.2 Cloudmesh Virtual Clusters

Traditional HPC provided clusters to the community while the clusters are managed by professional staff and have mostly rigid software stack targeting a broad community. Integration specialized software stack and accessing the newest development based software is often difficult or impossible to manage for a large number of communities. Thus we see the following advantages and disadvantages:

Advantages: provides well defined environment to the community, provides often optimized services for this particular system or the community it serves, allows sharing of resources through queuing system

Disadvantages: Different communities, groups or projects may have different requirements that are not met by the software stack, software stack often not state-of-the art, but tuned for reliability, experimentation with new software and methodologies in such an environment is difficult, queuing system may not provide enough interactivity

Hence we are in the need of a mechanism to provide clusters in a more manageable form. With the advent of virtualization technologies, this can be achieved while users are provided with virtual clusters hosted on cloud infrastructure using virtual machines, or container virtualization software. This provides us with the following advantages and disadvantages:

Advantages: user stack provide a a more flexible and state-of-the-art environment; support different interaction modes with instantaneous access without wait time;

Table 1: Selected Service Description. Items prefixed with a colon (:) indicate parameters e.g. :id, :?.

Resource	REST Method	Description
Virtual Cluster: /cluster		
/	GET	List available clusters
/	POST	Launch a cluster on the provider
/	DELETE	Delete all available clusters
/:id	DELETE	Delete and destroy a cluster
/:id	GET	View the status of a cluster (nodes, node type, etc)
/:id/properties/:property	GET, PUT	Get/set a property (provider, name, description) of the cluster
/:id/inventory/:format	GET	Obtain an inventory of the cluster
Stack Composition: /stack		
/	GET	List available compositions
/	POST	Create a new composition
/:id	GET	Show information about the composition
/:id	DELETE	Delete a composition
/:id/name	GET, PUT	Get/set the name of the composition
/:id/add?deployer=:?&source=:?	POST	Add a layer to the composition
/:id/layers	GET	List layers of the composition
/:id/layers/:id	DELETE	Delete the layer of the composition
Stack Deployment: /stack		
/	GET	List the available stacks with discription
/:id/deployments/:cluster	POST	Deploy a stack onto a cluster
/:id/status	GET	Current status
/:id/deployments/:cluster	GET	Current status on given cluster
Batch Experiments: /hpc		
/	GET	List all jobs started with the run command
/:id	DELETE	Deletes the experiment with the given id
/run?script=:?&cluster=:?	POST	Submits an experiment to the named cluster
/:id/status	GET	Returns the status of the job started with the run command
File Connections: /connections/file TODO		
Database Connections: /connections/db TODO		

support the use of heterogeneous platforms to allow increase in availability as well as features; and support the easy management of such an environment

Disadvantages: the user needs to manage the stacks themselves; a steep learning curve is needed to achieve this; the environment are feature rich and are difficult to manage

Naturally, cloudmesh is targeting to ease the disadvantages laid out.

3.3 Cloudmesh Groups

One of the essential abstractions in cloudmesh is the definition of groups on which actions can be performed. This allows us to agglomerate named objects into a group where the objects could have even different types. An example would be the resources that are part of a virtual cluster and could include virtual machines, object stores, or even queuing system services. On such groups we can apply actions that work in three different modes. Either individually, in groups or subgroups, or in an orchestrated fashion through workflows.

This simple abstraction makes it possible to quickly assemble high level representations of virtual clusters than can be deployed, monitored and managed by appropriate software.

3.4 Uniform Access Interfaces

Versioning. In case different versions are uses the version number is a prefix to the general query

Info. At time we may not only return results offered by the service that we contact, but receive information about the service itself, this si achieved with the info url. This has been introduced for the first time in [10]

Query. a query returns a subset of items. What kind of queries may be allowed may be communicated through an info command.

Pageing. In some instances it is important to restrict the returned values which can be achieved with paging

```
?offset=15&limit=5>
```

3.5 Virtual Clusters

Definition: A set of compute, storage and network services that interact with each other to serve an application or community for a specific amount of time to support one or more experiments. A virtual cluster is managed by the community or application user. They are typically built on top of HPC, Grids, Clouds, and Containers. The virtual cluster is managed by the user.

Contrasting Grids: targeted towards the support of a virtual organization introducing high overheads on the deployment and management of such infrastructure. Grids are a natural expansion of traditional supercomputer centers that are managed by professional staff. Contrasting Cloud IaaS: offer typically low level IaaS services allowing users to combine them. They are a valuable building block for virtual clusters. Clouds are managed by professional staff.

Table 2: Additional abstractions

Resource	REST Method	Description
Groups:		
/	GET	List all groups
/:id	DELETE	Deletes the group with the given id
/:id/add/...	POST	Add elements to the group
/:id/status	GET	Returns the status of elements in the group
/:id/delete	GET	deletes elements in the group
/:id/delete/:member	GET	delete a member from the group in the group
/:id/list	GET	list the elements in the group in the group
/:id/list/:member	GET	list the element in the group in the group
/:id/run/:action	GET	apply an action on a group.
/:id/attach/:order	GET	apply an action on a group.
Groups: /group		
/	GET	List all groups
/:id	GET	Returns the status of elements in the group
/:id	DELETE	Deletes the group with the given id
/:id/member	POST	Add a new member to the group
/:id/member	GET	List the members of a group
/:id/member/:member	PATCH	Update a member from the group in the group
/:id/member/:member	DELETE	Deletes elements in the group

Contrasting Cloud PaaS: offer enhanced services to users targeting a particular platform. They are offered by professional staff.

Contrasting Containers: offer an abstraction to share the existing hardware and OS while using containers making the it not necessary to us OS virtualization, thus saving space. Containers provide very useful enhancements for creating virtual clusters through add ons such as Kubernetes and Docker Swarm.

3.5.1 *Cloudmesh Access and Management of virtual clusters*

Cloudmesh is an API, command line tool, and command shell allowing the easy utilization of HPC, Clouds, Containers through machine abstractions Switching between alternative services can be achieved by updating a single variable.

Demonstrated usages of Comet (HPC integrated Cloud) FutureSystems, Chameleon, CloudLab, Bridges, Jetstream, (National Openstack Clouds) Cybera (CA), Karslruhe Openstack Cloud (KIT, Germany), (International) EC2, AWS, Azure (Commercial Non Openstack) Devstack, Trystack, Virtualbox (Desktop Clouds) A user could use all of them

3.5.2 *Cloudmesh and Comet Cloud Virtual Cluster*

Comet is a NSF sponsored super computer offered by SDSC to the community. It operates in two modes: HPC and virtual clusters Comet virtual clusters are low level and offer the HPC administrator the view of a cluster as if it were hardware. This is achieved via virtualization and low level exposure of network services. The administrator has full control over the cluster. This is achieved by the integration of virtualization and SRIOV within Rocks. Thus the same cluster can not only be used in virtualized mode, but also in HPC mode Comet users can therefore use HPC and virtualized clusters on the same hardware. Virtual clusters are treated as a special kind of compute job Based on our long experience in this field we have delivered to SDSC an extension to cloudmesh that allows the creation and management of virtual clusters within comet. A subset of commands that showcase how easy the interaction with comet is shown in Table 3

3.6 Cloudmesh Stack

Important is that cloudmesh introduces an abstraction for creating and managing software stacks. Such stacks are used to tackle the problem of reproducibly deploying and configuring software on virtual clusters. They are also important to define customization of the software stack that typically can not be provided by traditional supercomputing centers and have become an integral part of cloud computing and development efforts of large scale open source software projects.

Fix citations

A *Stack* is a way to modify a collection of accessible resources in order to bring them to a desired state. It is desirable that this execution unit can interface with various deployment tools and approaches. This includes scripts, programs, ssh, Make, OpenStack Heat [**?**], Ansible [**?**], Chef [**?**], Puppet [**?**], Salt [**?**], Vagrant [**?**], Dockerfile [**?**], and NixOS/NixOps [**?**]. The reason for this is that users can use their own preferred deployment tools, but also allows them to leverage efforts conducted by the open source community while integrating large and complex software stacks. Hence, if available a variety of different launch platforms can be integrated into cloudmesh and work from different community contributions can be supported.

These deployment units are then composed as part of a stack *Composition* into layers to form a new deployment unit that can be itself used recursively as a layer within a different composition (illustrated in Figure 6 and Figure 9).

To support the goals of a sophisticated deployment frame-

Table 3: Commands to interact with XSEDE comet virtual machine management

Command	Description
cms comet cluster ID	Show the cluster details
cms comet power on ID vm-ID -[0-3] –walltime=6h	Power 3 nodes on for 6 hours
cms comet image attach image.iso ID vm-ID-0	Attach an image
cms comet boot ID vm-ID-0	Boot node 0
cms comet console vc4	Start interactive console for vc4
cms var cloud=jetstream / bridges / aws / comet	easy switching of infrastructure
cms boot	booting of a virtual machine with default parameters

work, stack need to support the following properties:

Idempotent. Deployment of a stack should only bring the system to the desired state: applying a stack multiple times should be indistinguishable from applying it a single time. This also provides a measure of fault tolerance to the deployment.

Reproducible. Deployments of a given stack onto multiple identical targets should yield equivalent systems.

Self contained. All components need to bring a system to the stack's desired state should be declared by the stack.

Configurable. When a stack a defined default values may be assumed to take effect on deployment. In order to allow a system to be tuned to the desired state, depending on the user's requirements, these values must be configurable.

Composable. The semantics of deploying two stacks should be clear.

Linkable. Stacks should be able to link to each other to define choreography roles and avoid duplication.

Extensible. Stacks need to be easily extensible in order to be adapted to specific deployment needs.

To support this effort we define a cloudmesh stack command *stack* command that deploys and configures a user-defined subset of the available modules while leveraging existing deployment technologies.

In addition, Cloudmesh provides a number of predefined stacks such as Apache Spark and Apache Drill that can be reused and customize virtual clusters while enhancing them with sophisticated software stacks.

This results in a minimization of work by the user as they can leverage a number of existing deployments that are available through open source repositories. Hence cloudmesh integrates valuable contributions form the community while not only pointing to them, but also making selected stacks available while vetting and improving them [11].

The stack command was initially developed to aid students and researchers in completing data analytics projects. They were faced with exploring various technologies and would get stuck during the installation and configuration phase, unable to progress due to the complexity involved in managing it. While making cloudmesh stack available and targeting often used deployment stacks we wer able to assist these groups significantly.

To illustrate the simplicity of use of sophisticated stacks that would be otherwise not achievable by many users we provide the following example.

In this example we will create and deploy the stack shown in Figure 9 that deploys one of our test use cases described in Section **??**. To achieve this we use the commands listed in Figure **??** and create the *fingerprint* stack from preexisting *spark*, *hbase*, and *drill* stacks and compose it with the predefined *hadoop* and *web* stacks. Then, a cluster of four nodes is launched onto which the newly created stack is deployed. We use indentations to illustrate better the association of a stack containing a composition of layers.[2] As part of this example we use the convenient stack command introduced by cloudmesh. A portion of its manual page is shown in Figure 8.

3.6.1 Stack Composition and Linking

A *Composition* creates a new stack by combining multiple other stacks. The unit of deployment is a *Stack*, but each stack may be a composed of one or more layers of other stacks, forming a graph structure whose evaluation results in deployment onto the target cluster.

Figure 9 illustrates how Figure 6 (while expanding on it) may be evaluated. Evaluation of *My Analytics Stack* requires the evaluation of the *Fingerprint* and *Web* stacks, each of which are compositions of other stacks.

3.6.2 Stack Linking

Stack linking allows other stacks to reuse stacks not just by copying the entire content of them into the stack, but by simply linking them with a url into the stack. This will than allow the reuse of compassable stacks more easily. Care must be taken when changes take place in linkable stack, thus they can be augmented with version information to avoid undesirable side effects.

3.6.3 Stack Programming

Building virtual clusters running hadoop becomes easy for users while leveraging cloudmesh stack. First we utilize the virtual cluster command to create a cluster (for example with 10 nosed). It also will launch the deployment of the stack called hadoop which is ocated in our cloudmesh stack repository

[2]the term layer may need to be replaced with the term add. We also need to document that the last stack is used by default.

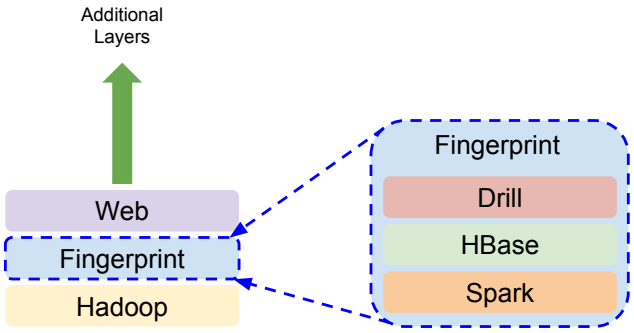

Figure 6: Multiple stacks may be composed. Here, the *Fingerprint* stack is a composition of the Apache *Drill*, *HBase*, and *Spark* stacks. The *Fingerprint* stack is then included as a layer withing another composition built upon *Hadoop* and extended with the *Web* layers. The *Web* layer itself may be composed of other layers.

```
cm stack create my-analytics-stack
   cm stack layer hadoop
cm stack create fingerprint
   cm stack layer spark hbase drill
   cm stack layer web
   cm cluster create --count 4
cm stack deploy
```

Figure 7: Commands to deploy the fingerprint N_1 stack

Badi: stack commands to deploy the fingerprint N1 stack

```
stack check [--stack=bds]
stack init [--no-activate]
           [--branch=master]
           [--user=$\$$USER]
           [--name=<project>] <ip>...
stack list [--sort=<field=date>]
           [--list=<field,...=all>]
stack project [<name>]
stack deploy [<play>...]
             [--define=<define>...]
```

Figure 8: Excerpt of the Cloudmesh stack manual page

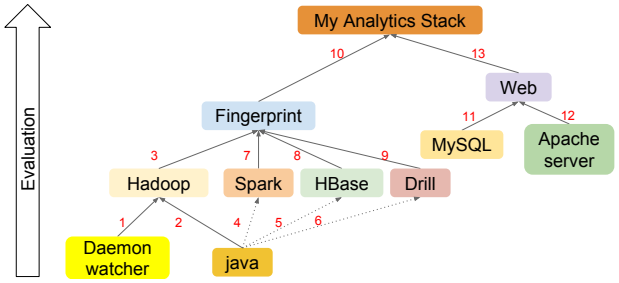

Figure 9: Evaluation of a stack. The *My Analytics Stack* is composed of the *Fingerprint* and *Web* layers. These in turn are themselves compositions. Numbers indicate order of evaluation. While *spark*, *hbase*, and *drill* depend on *java*, re-evaluation (*4 - 6*) is not problematic due to the idempotency property.

```
cm stack hadoop -n 10 --name=cluster_1 --cloud=futuresystems
```

To specify our face detection stack we can say

```
cm stack facedetection -n 10 --cloud=futuresystems
```

FutureSystems is one of the Clouds we maintain at Indiana University and is based on OpenStack [**?**]. To list the available stacks we can issue the command

cm stack list [–url=...]

It is possible to specify an optional list of URLs to add more stack repositories.

Once we have defined the virtaul cluster and deployed the needed software stack, we need to make sure that we can interact easily with it while for example starting services or deleting the cluster

```
cm hadoop start --name=cluster_1
cm delete --name=cluster_1 --all
```

The Cloudmesh big data stack commands are leveraging abstractions and can therefore implemented while utilizing and interacting with various software packages from different layers. This enables us to accomplish diverse tasks that have to be achieved in order to assist the scientists trying to use big data as part of their application missions. While providing them easily to the scientists complex big data tasks can be achieved more easily.

3.7 Stack Repository

We are providing at this time a number of software stacks while leveraging ansible. However previously we also showcased the use of OpenStack Heat demonstrating that it is possible to use different deployment tools within cloudmesh. The current stacks that we developed include Drill [**?**], Ganglia [**?**], Hadoop [**?**], Hbase [**?**], Hive [**?**], Java [**?**], Limits [**?**], Maven [**?**], Mysql [**?**], Nagios [**?**], Pig [**?**], Spark [**?**], Supervisord [**?**], Zookeeper [**?**]. Layerd stacks which reuse them include Hadoop [**?**], Hbase [**?**], Pig [**?**], Spark [**?**], Drill [**?**], and Hive [**?**].

The use and further development of cloudmesh stacks is not only done by our research and application teams, but also integrated in ongoing classes with a large number of students taught at Indiana University while additional stacks are being tested and added. This will guarantee use and further development of the components by hundreds of users.

4. BIG DATA USE CASES

The NIST Big Data Working group has identified 51 benchmarking examples for Big Data[**?**] spanning application areas such as:

Government Operation: National Archives and Records Administration, Census Bureau

Commercial: Finance in Cloud, Cloud Backup, Mendeley (Citations), Netflix, Web Search, Digital Materials, Cargo shipping (as in UPS)

Defense: Sensors, Image surveillance, Situation Assessment Healthcare

Life Sciences: Medical records, Graph and Probabilistic analysis, Pathology, Bioimaging, Genomics, Epidemiology, People Activity models, BiodiversityDeep Learning and Social Media: Driving Car, Geolocate images/-cameras, Twitter, Crowd Sourcing, Network Science, NIST benchmark datasets

The Ecosystem for Research: Metadata, Collaboration, Language Translation, Light source experiments

Astronomy and Physics: Sky Surveys compared to simulation, Large Hadron Collider at CERN, Belle Accelerator II in JapanEarth,

Environmental and Polar Science: Radar Scattering in Atmosphere, Earthquake, Ocean, Earth Observation, Ice sheet Radar scattering, Earth radar mapping, Climate simulation datasets, Atmospheric turbulence identification, Subsurface Biogeochemistry (microbes to watersheds), AmeriFlux and FLUXNET gas sensors

Energy: Smart grid

In addition we have 81 student projects from classes taught at Indiana University on the topic of Big Data. From these examples we have examined on six of the NIST uses case projects to identify the technologies used as shown in Table 4.

Deployment of a single technology may result in development of several ansible roles comprising dependencies. The implementation of the Fingerprint usecases (N_1) Ansible playbook uses 19 such roles that may be reused in other projects and Face Detection (N_2) uses five. Furthermore, The remaining four NIST use cases N_4 - N_6 (Twitter analysis, Healthcare, Spatial data, Data Wharehousing) may contribute an addition 27 roles.

In summary, we have looked at 6 projects from the 51 NIST use cases to identify 51 Ansible roles. Looking through 81 class projects over two semesters at Indiana University showed 62 roles, of which a subset were found to be repeatedly used across various projects.

4.1 Fingerprint Matching (N_1)

Fingerprint recognition refers to the automated method for verifying a match between two fingerprints and that is used to identify individuals and verify their identity. Fingerprints

Table 4: Technology used in a subset of usecases. A ✓ indicates that the technology is used in the given project. See Table 5 for details on a specific project. The final row aggregates ✓ across projects.

ID	Hadoop	Mesos	Spark	Storm	Pig	Hive	Drill	HBase	Mysql	MongoDB	Mahout	D3 and Tableau	nltk	MLlib	Lucene/Solr	OpenCV	Python	Java	Ganglia	Nagios	zookeeper	AlchemyAPI	R
N_1	✓		✓			✓	✓	✓	✓									✓	✓	✓	✓		
N_2		✓	✓													✓	✓				✓		
N_3				✓				✓		✓		✓	✓				✓	✓			✓	✓	✓
N_4	✓		✓					✓			✓	✓		✓	✓			✓			✓		
N_5	✓		✓								✓	✓		✓				✓					
N_6	✓		✓		✓	✓		✓		✓	✓	✓		✓	✓			✓			✓		
count	4	1	5	1	1	2	1	4	1	2	3	4	1	3	2	1	2	5	1	1	5	1	1

are the most widely used form of biometric used to identify individuals. The automated fingerprint matching generally required the detection of different fingerprint features (aggregate characteristics of ridges, and minutia points) and then the use of fingerprint matching algorithm, which can do both one-to- one and one-to- many matching operations. Based on the number of matches a proximity score (distance or similarity) can be calculated. Furthermore, NIST is providing via the the NIST Fingerprint dataset a special database. The goal for this usecase is the following: given a set of *probe* and *gallery* images, compare the probe images to the gallery images, and report the matching scores. The dataset comprises 54,000 images and their metadata. It uses MINDTCT [12] preprocesses the images to identify minutae of the prints automatically locating and recording ridge ending and bifurcations in a fingerprint image; and BOZORTH3 [13] to identify matches. Both are are part of the NIST Biometric Image Software (NBIS) [14].

To execute this usecase[15] we need to deploy an application to analyze the dataset. It internally uses cloudmesh to deploy an the software stack The implemented [15] solution uses a software stack comprising of Hadoop HDFS[16], YARN[16], Apache Spark[17], Apache HBase[18], and Apache drill[19], Scala[20], and the NBIS software[21]. A Hadoop cluster is deployed and YARN used to schedule Spark jobs that load the images into HBase, process the images, and compute the matches. Apache Drill, with the HBase plugin, can then be used to generate reports with the NBIS tools[14]. The results are stored in HBase and Apache Drill is used to query the results. The code leverages tools and services is based on [22], while significantly enhancing it with cloudmesh deployment strategies and services.

4.2 Face Detection (N_2)

Human detection and face detection have been studied during the last several years and models for them have improved along with Histograms of Oriented Gradients (HOG) [23] for Human Detection.

We use[24] OpenCV [25], a Computer Vision library includ-

Table 5: Dataset used in the various use cases.

ID	Use Case	Dataset	Size (GB)
N_1	Fingerprint Matching	Special Database 14 - NIST Mated Fingerprint Card Pairs 2	2.1
N_2	NIST Human and Face Detection	INRIA Person Dataset	0.96
N_3	NIST Twitter Analysis	Twitter	-
N_4	NIST Analytics for Healtcare Data / Health Informatics	Medicare Part-B in 2014 from Center for Medicare and Medicaid Services (CMS)	0.1
N_5	NIST Spatial Big Data/Spatial Statistics/Geographic Information Systems	Uber	0.2
N_6	NIST Data Warehousing and Data Mining	United States 2010 Census data	-

ing the Support Vector Machine (SVM) classifier, and the Histogram of Oriented Gradient (HOG) [23]object detector for pedestrian detection and INRIA Person Dataset is one of popular samples for both training and testing purposes. HOG with SVM model is used used as object detectors and classifiers while the python libraries from OpenCV provide these models for human detection. The OpenCV Python code runs with Spark Map function to perform distributed job processing on the Mesos scheduler.

To enable this analysis we use cloudmesh to deployed Apache Spark on a Mesos clusters and install the OpenCV software and its Python API. We also update the python software stack. Then we to train and apply detection models from OpenCV using Python API. We use the INRIA Person Dataset [26]. This dataset contains positive and negative images for training and test purposes with annotation files for upright persons in each image. 288 positive test images, 453 negative test images, 614 positive training images and 1218 negative training images are included along with normalized 64x128 pixel formats. The size of the dataset is 970MB.

Cloudmesh deploys and builds the clusters for batch-processing large datasets, Internally cloudmesh uses for this ansible scripts to support installation and configuration while leveraging available cloud compute resources. We have for this example developed or are reusing five ansible roles that we developed for other usecases:: Apache Spark Role [27] Apache Mesos Mesos [28], Apache Zookeeper Role [29], OpenCV Role (with Python) [30].

5. STATUS

We can replicate on different infrastructures including AWS, Azure, Openstack, and XSEDE/SDSC Comet.

The *stack* command has been implemented, while we used to focus initially on heat we found thia was a too limiting approach. Since than we have expanded our activities on using for many of the stacks ansible and provide a significant set as part of our open source solutions. This includes:

- Hadoop HDFS
- Hadoop YARN
- Apache Spark
- Apache Pig
- Apache Hive
- Apache HBase
- Apache Drill
- OpenCV

Cloudmesh is also used in the production system of XSEDE comet and is used to interact with virtual clusters on that system showcasing the general approach that we take in cloudmesh to integrate with vastly different clouds and IaaS frameworks.

6. CONCLUSION

Ansible + Cloudmesh Will enable re-usable specification of Big Data Stack applications in 87 use cases and 62 unique roles from which NIST has 27 BDS status From this a total part of BDS are vetted (the most fundamental ones): 14 Roles (about 50% done of NIST roles) 6 Playbooks

Acknowledgement

We woudl like to thank for his contributions to an earlier version of this paper

discuss additional work

server less computing [31]

server less computing [32]

7. REFERENCES

[1] NIST, "NIST Big Data Interoperability Framework: Volume 6, Reference Architecture," National Institute of Standards and Technology, U.S. Department of Commerce, NIST Special Publication 1500-6, Sep.15 2015.

[2] G. von Laszewski, J. Diaz, F. Wang, and G. C. Fox, "Comparison of Multiple Cloud Frameworks," in Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on, June 2012, pp. 734–741.

[3] G. von Laszewski, Web Page. [Online]. Available: http://cloudmesh.github.io/client/

[4] NIST, "NIST Big Data Interoperability Framework: Volume 1, Definitions," National Institute of Standards and Technology, U.S. Department of Commerce, NIST Special Publication 1500-1, Sep.15 2015.

[5] ——, "NIST Big Data Interoperability Framework: Volume 2, Taxonomies," National Institute of Standards and Technology, U.S. Department of Commerce, NIST Special Publication 1500-2, Sep.15 2015.

[6] ——, "NIST Big Data Interoperability Framework: Volume 3, Use Cases and General Requirements," National Institute of Standards and Technology, U.S. Department of Commerce, NIST Special Publication 1500-3, Sep.15 2015.

[7] ——, "NIST Big Data Interoperability Framework: Volume 4, Security and Privacy," National Institute of Standards and Technology, U.S. Department of Commerce, NIST Special Publication 1500-4, Sep.15 2015.

[8] ——, "NIST Big Data Interoperability Framework: Volume 5, Architectures White Paper Survey," National Institute of Standards and Technology, U.S. Department of Commerce, NIST Special Publication 1500-5, Sep.15 2015.

[9] ——, "NIST Big Data Interoperability Framework: Volume 7, Standards Roadmap," National Institute of Standards and Technology, U.S. Department of Commerce, NIST Special Publication 1500-7, Sep.15 2015.

[10] G. von Laszewski, J. Gawor, C. J. Peña, and I. Foster, "Infogram: A grid service that supports both information queries and job execution," in Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing, ser. HPDC '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 333–. [Online]. Available: https://www.globus.org/sites/default/files/vonLaszewski--infogram.pdf

[11] "Cloudemsh stack repository," github. [Online]. Available: https://github.com/futuresystems/big-data-stack

[12] NIST. minutiae detector from nbis (nist biometric image software). [Online]. Available: http://www.nist.gov/itl/iad/ig/nbis.cfm

[13] M. D. Garris, C. I. Watson, R. M. McCabe, and C. L. Wilson, "User's guide to nist fingerprint image software(nfis)," 2001., 2001.

[14] C. I. Watson, M. D. Garris, E. Tabassi, C. L. Wilson, R. M. Mccabe, S. Janet, and K. Ko, "User's guide to nist biometric image software (nbis)," 2007.

[15] B. Abdul-Wahid. Nist fingerprint matching ansible playbooks. [Online]. Available: https://github.com/cloudmesh/example-project-nist-fingerprint-matching

[16] Apache hadoop. Web Page. [Online]. Available: http://hadoop.apache.org/

[17] Apache spark. Web Page. [Online]. Available: http://spark.apache.org/

[18] Apache hbase. Web Page. [Online]. Available: https://hbase.apache.org/

[19] Apache drill. Web Page. [Online]. Available: https://drill.apache.org/

[20] The scala programming language. Web Page. [Online]. Available: https://www.scala-lang.org/

[21] P. Flanagan, "Nist biometric image software (nbis)," 2010.

[22] Afzal Godil and Wo Chang, "NIST Big Data Public Working Group draft Possible Big Data Use Cases Implementation using NBDRA," National Institute of Standards and Technology, U.S. Department of Commerce, NIST.

[23] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), vol. 1. IEEE, 2005, pp. 886–893.

[24] H. Lee. Nist human and face detection ansible playbooks. [Online]. Available: https://github.com/futuresystems/pedestrian-and-face-detection

[25] G. Bradski et al., "The opencv library," Doctor Dobbs Journal, vol. 25, no. 11, pp. 120–126, 2000.

[26] N. Dalal and B. Triggs, "Inria person dataset," 2005.

[27] H. Lee. Ansible role for apache spark. [Online]. Available: https://github.com/VirtualClusters/ansible-role-spark-for-mesos

[28] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center." in NSDI, vol. 11, 2011, pp. 22–22.

[29] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "Zookeeper: Wait-free coordination for internet-scale systems." in USENIX Annual Technical Conference, vol. 8, 2010, p. 9.

[30] H. Lee. Ansible role for opencv. [Online]. Available: https://github.com/futuresystems/ansible-role-opencv

[31] J. Hennessey, S. Tikale, A. Turk, E. U. Kaynar, C. Hill, P. Desnoyers, and O. Krieger, "Hil: Designing an exokernel for the data center," in Proceedings of the Seventh ACM Symposium on Cloud Computing, ser. SoCC '16. New York, NY, USA: ACM, 2016, pp. 155–168. [Online]. Available: http://doi.acm.org/10.1145/2987550.2987588

[32] A. Vahdat. (2016, May) Serverless computing and cloud 3.0. Video. Google talk. [Online]. Available: https://drive.google.com/file/d/0BwMOJChROzavTE4yZXBsMWdmY0U/view

Notes

- [ ] Fix citations . . . . . .
- [ ] Badi: stack commands to deploy the fingerprint N1 stack . . . . . .
- [ ] discuss additional work . . . . . .
- [ ] server less computing [31] . . . . . .
- [ ] server less computing [32] . . . . . .