MITRE

Architectural Model for MITRE Research: BlueRidge

Mr. Joseph Jubinski
Dr. Ransom Winder
Dr. Angela McIntee O'Hanlon

June 2015
v.2.1.35

Abstract

There is an inherent challenge in creating a shared architecture that facilitates the development and execution of analytics on variegated data for multiple lines of computational research. This is especially true when each line can have a distinct expectation of how the results will be used. This paper discusses a project, *BlueRidge*, with a proposed architectural model that addresses the specified challenge. BlueRidge intends to be a single unified model of all discrete, necessary components that make up computational threads. Such an architectural model must address (1) how the data will be ingested, (2) how the data and results will be represented, maintained, and warehoused, (3) how the results will be accessed, (4) how issues of analytic orchestration, in particular spanning multiple clouds, will be addressed, and (5) how data lineage is established through provenance. This suggests that many different technology decisions must be made, whether the technologies are internally developed or drawn from existing solutions (open-source, free, or commercial). Requirements will be based on the acquired data, the expected activities, the available assets, and ultimately the desired outcomes across the supported research community.

1 Introduction

The development and execution of analytics in a research setting can be facilitated by having a software architecture to support these activities [1, 2, 3, 4]. It is challenging to develop a generalized architectural model that will be shared by many computational research domains analyzing different data sets and producing unique results for varied use-cases. Recognizing the appropriate solutions to use or develop and how the model handles each core competency of architecture must be examined. A model that addresses these challenges applies to different research scenarios with varied requirements and desired outcomes. An appropriate application of this model is in support of the MITRE Data-to-Decisions [5] investment area.

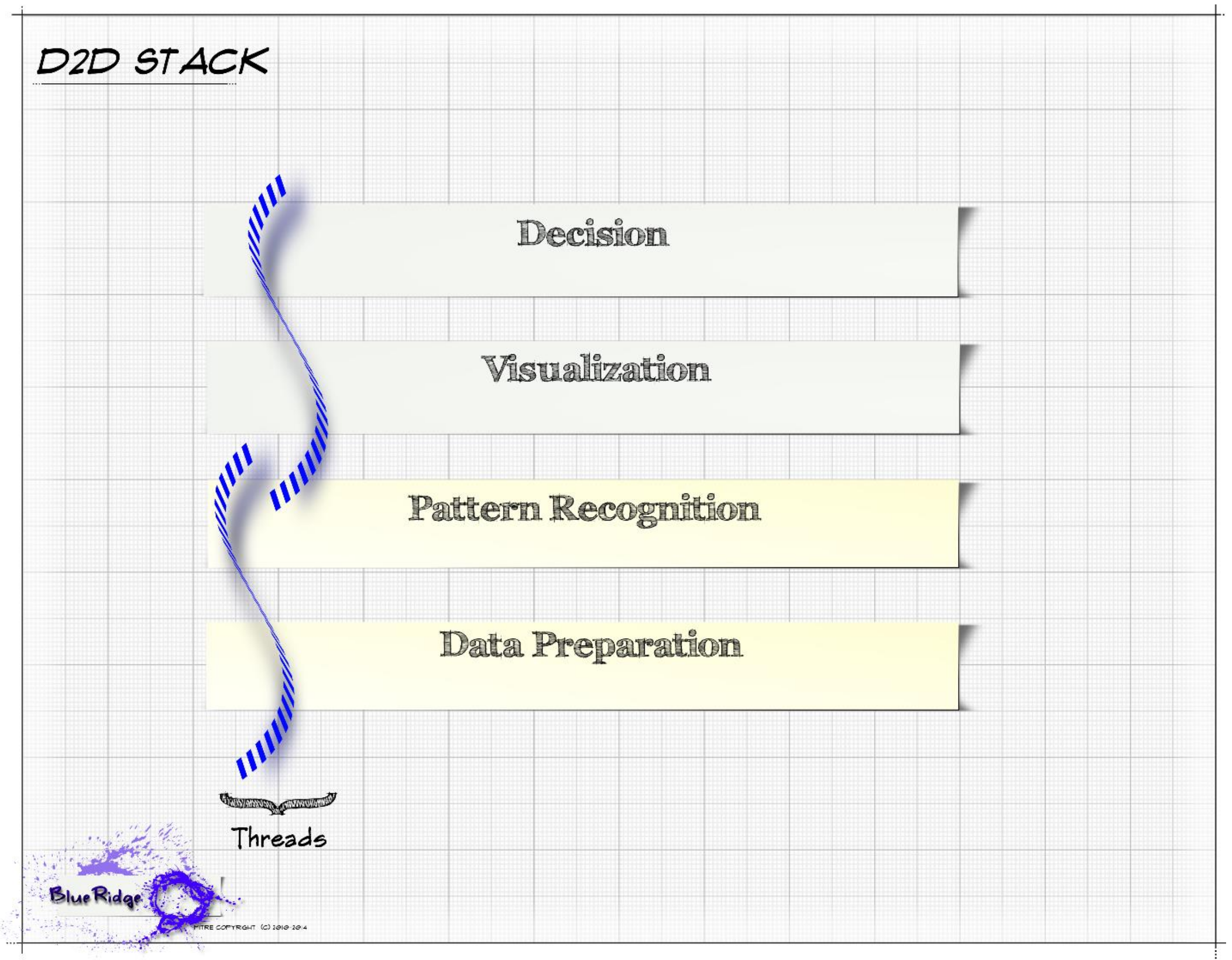


Figure 1. Data-to-Decisions research stack.

Data-to-Decisions (D2D) is a broad effort that seeks to use data for decision making purposes. The effort is directed toward multiple, varied domains, where each domain addressed is a thread comprised of related projects. The general workflow for a given domain-focused thread begins with data preparation. By way of pattern recognition and analysis the relevant knowledge is extracted for storage as results. Results are then digested and visualized in a manner such that a user receives the necessary indicators that can guide making optimal decisions and taking optimal actions. Figure 1 depicts this high-level flow, which can handle multiple threads distinguished by different domain needs, results, and decisions. This can be viewed as a natural extension of the intelligence reference model (discussed in the Appendix). What we refer to as the Data-to-Decisions research stack is an instantiation of the core activities that one can expect in a general research stack, which aligns to the same high-level flow.

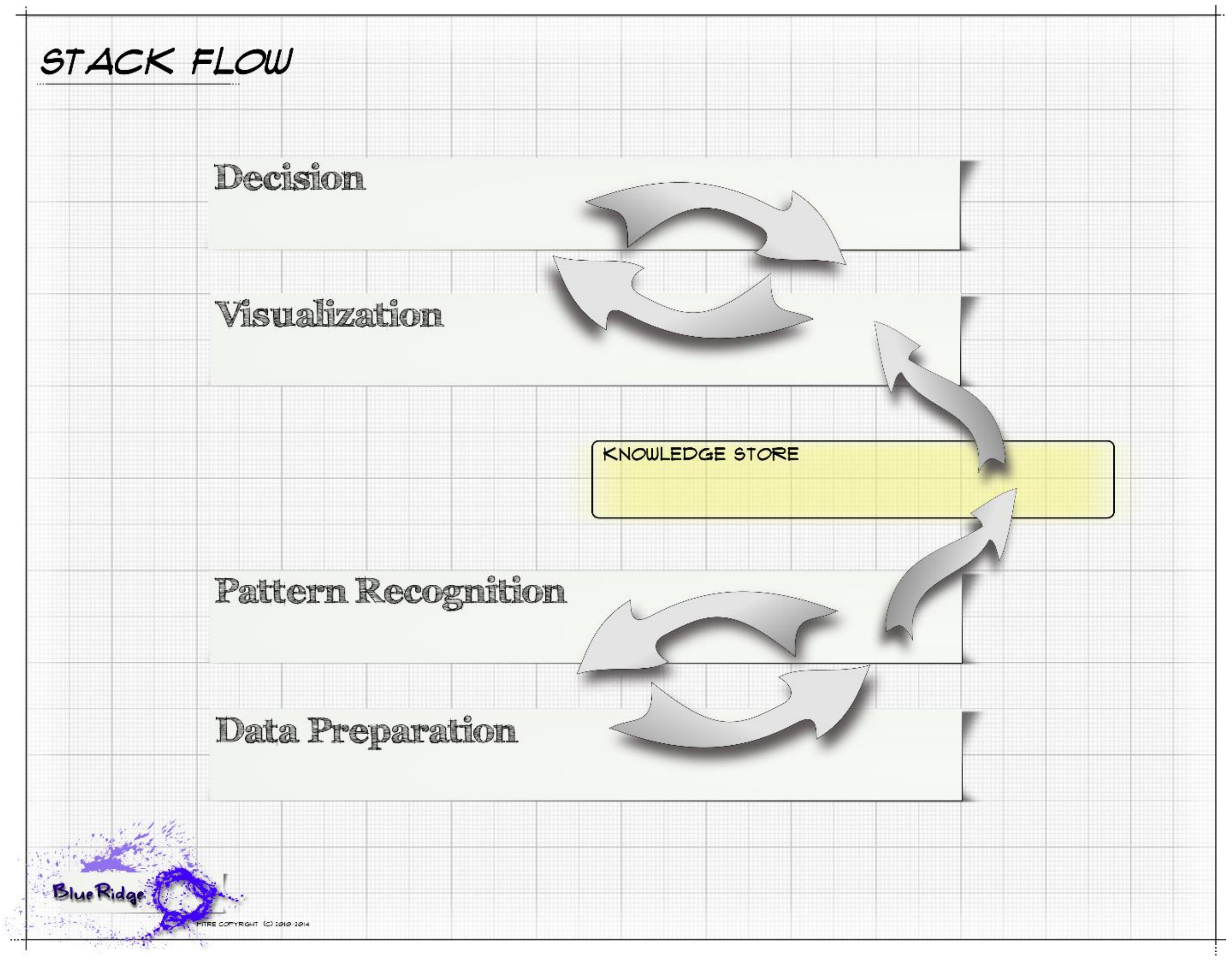


Figure 2. Flow of activity through the research stack.

The core elements, or layers, of D2D, which drive the design of its architectural model and analytic framework, include *data preparation*, *pattern recognition*, *visualization*, and *decision-making*. Data preparation encompasses the identification and acquisition of relevant data. Pattern recognition isolates salient features and identifies useful patterns to be captured in results, typically by way of automated processes. Visualization depicts patterns and results in a representation that can demonstrate the relevance in an operational context. The decision-making layer represents a user-orchestrated process that relies on the depicted patterns to take informed and improved actions. Cases of decision-making can then be data-driven and near real-time to real-time or question-driven and just-in-time. Figure 2 elaborates on the high-level flow, demonstrating where there is interplay between the layers and the intermediate knowledge store. Data preparation naturally precedes pattern recognition, but the level of success driven by the results of pattern recognition can also inform further data preparation tasks, after the value and potential shortfalls of analytic processes are exposed. Similarly, while visualization informs decision, decisions can also lead to changes, online or offline, in how knowledge resulting from pattern recognition is visualized.

Domain-specific threads will pass through each of these layers as part of the flow to assist decisions. For any given thread across these layers, there is the potential for interplay between automated processes and human-driven activity. Both are discussed in this document, but the emphasis is on automated processes.

Related to these different layers, there are different, independent roles individuals have with respect to the architectural model and its implementation. The key distinctions are between thread leaders, researchers, and users. Thread leaders are typically subject matter experts who define or identify use-cases relevant to a domain and orchestrate researchers and analytics toward the use-case goals. This requires insight and attention at all levels, but especially at the data preparation and decision layers. Researchers are those who develop or implement solutions and applications for the domain-specific problems at a project level. This typically involves the pattern recognition and visualization layers. Users are those who examine results to inform their downstream actions and are engaged by the output of the visualization and at the decision-making layer.

The diversity of domain problems to be addressed across research threads suggests multiple differing data sources, data formats, analytic processes, and workflow capabilities. A supporting architecture must accommodate this expected diversity. Further, the architecture must be built so that research and development activities' needs are given precedence over those desired in a production environment. In research and development, the flexibility of execution, repeatability of experiments, and retention of results are emphasized, while in production environments, efficiency is usually the primary concern.

This document covers a proposed architectural model that is capable of supporting the Data-to-Decisions research stack and other scenarios that fit the template it outlines. We discuss approaches that can be applied to the architectural model and issues that must be addressed by these approaches. While this work intends to address the challenges posed in Data-to-Decisions, we are interested in a solution that can be also applied more widely as a general architectural model for any computational research stack, especially where multiple threads of research may be expected to coexist.

2 Layout

2.1 Hadoop and Alternatives

The task of configuration encompasses issues relating to software architecture capable of alignment with many possible underlying computational patterns. In anticipation of the need for scalability in order to handle the size and scope of data and tasks across different domains, this leads to an implicit assumption that cloud-computing is an underlying pattern of primary interest. It should be kept in mind that it is not the only pattern and the solutions described here, while using the cloud-computing pattern for specific examples and diagrams, are applicable to other underlying computational patterns.

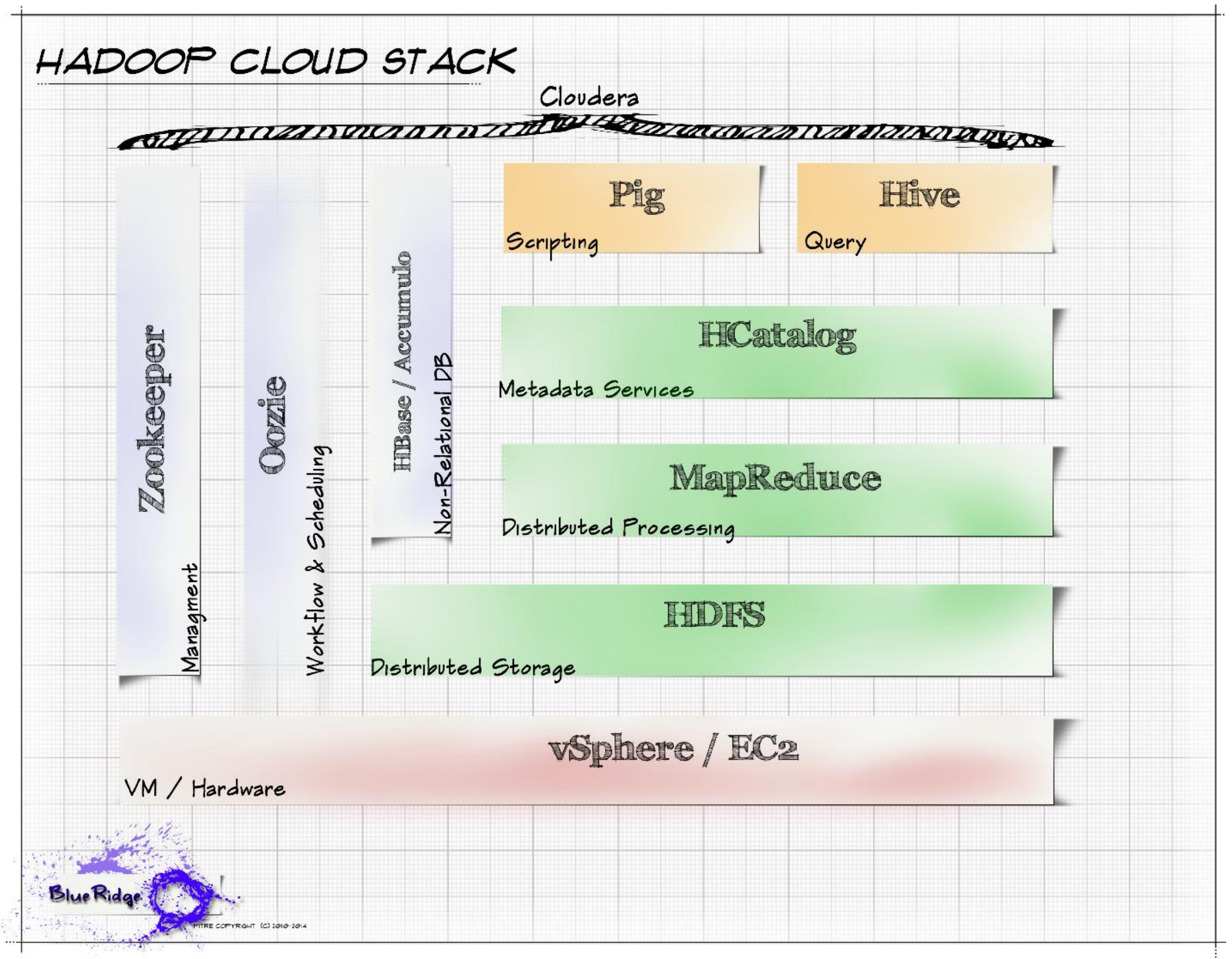


Figure 3. View of a Hadoop cloud stack.

Considering configuration and partitioning with the above in mind, several standard cloud stacks exist (Figure 3 is a representative picture of a Hadoop [6, 7] cloud stack such as Cloudera provides), and those that are part of a thread's required architecture need to be supported. If a new stack is proposed, there must be a responsible party for fabricating or providing the solution. Core functionalities of this cloud stack can be replaced in pieces or as a whole. Other similar or quite different services could be used in place of a Cloudera installation depending on the needs of a particular domain, thread, or collection of projects.

Given the importance of data in computational research domains, solutions likely need to scale. Some valid alternatives to Cloudera as the situation dictates include MongoDB, Neo4j, and Cassandra. Most of these database alternatives are directed toward specific use-cases, such as Neo4j to graphs and MongoDB to documents. Choosing an alternative requires certain conditions in data and results to be met.

Using individual physical or virtual machines without scalability acting as standalone services within the architecture is also feasible. These cases are less problematic in terms of the core issues of partitioning discussed below, but are valid possibilities, even in combination with multiple cloud solutions. The examples that follow continue to assume an underlying Hadoop distribution, given its ubiquity and generalizability to many scenarios requiring scalability.

2.2 Partitioning

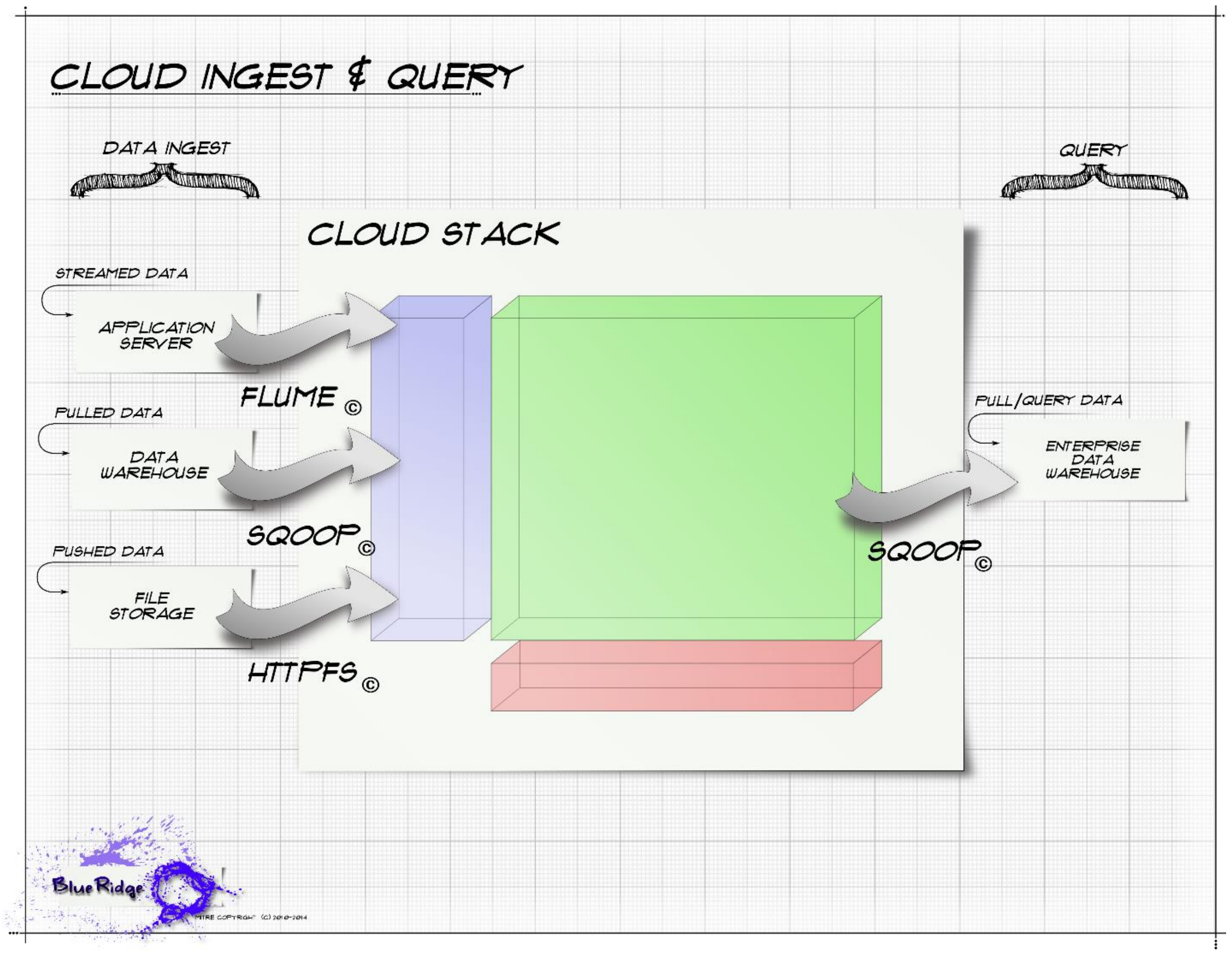


Figure 4. Relationship between data and cloud stack.

Issues introduced when discussing configuration can be examined at a deeper level. Assuming the architectural model will use an underlying Hadoop cloud stack as depicted in Figure 3 and ingest and query patterns resembling Figure 4, there are further decisions required on how to partition capability across multiple threads, projects, and data. The optimal organization of storage and execution will vary depending on size, rates and usage of data and the overlap between projects and threads. The colors align with those in Figure 3.

The simplest, least-structured method for organizing the architecture is to assume there is a single analytic machine (such as a cloud) that is not partitioned as shown in Figure 4. Note that the colors in this figure align with those in Figure 3. In this case all threads, projects, and data are housed in one environment and share the resources of that environment, which is an inherent disadvantage. The advantage of this model is minimal duplication of data and no need to move the data between multiple machines for different projects. This could be accomplished using either a physical or virtual machine to implement the architecture.

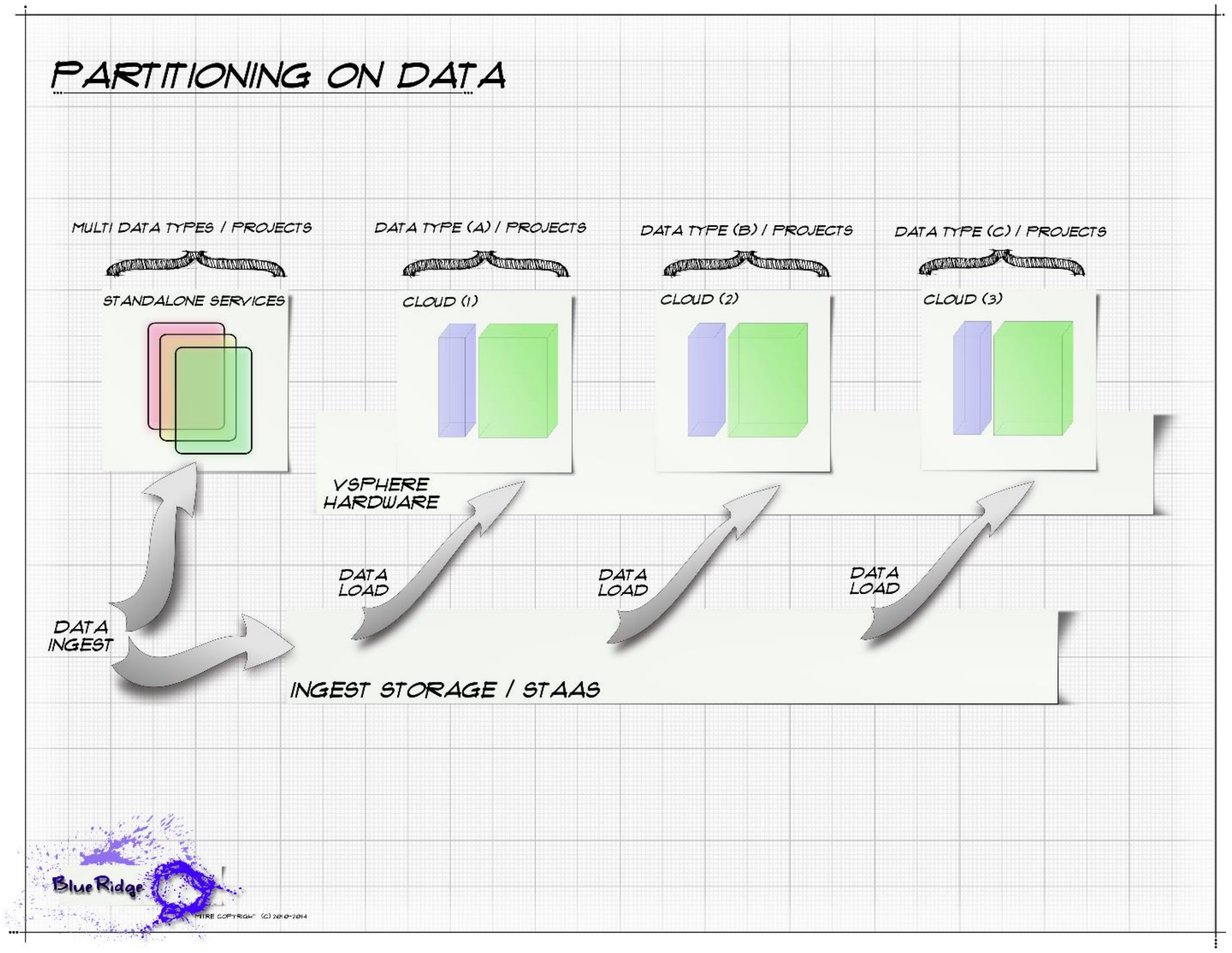


Figure 5. Architecture partitioned along different data sources and types [8].

As a contrast, Figure 5 depicts an architectural model where analytic machines (potentially cloud virtual machine or standalone services) are divided based on the data they will house. When data is substantial, moving it is expensive. Furthermore, typically cloud-computing tasks run on the machines that house the data. Therefore the model in Figure 5 is preferable when threads expect to operate on a single data source (e.g., Twitter) and will not integrate across multiple sources.

This requires some duplication of data and added logistics of moving data, assuming data will be first ingested into physical storage, possibly using Storage as a Service (STaaS), and then moved onto the virtual machines. While data being shared by multiple projects does not pose a problem, when a project requires more than one data source an issue arises. If usage of the data is independent within a project, this model can hold. If there is a dependency, but the results that form the dependency are small and the workflow can accommodate latency, keeping separate analytic machines is still feasible. In other situations though, it might be necessary to duplicate data such that there are analytic machines (often clouds) with multiple large data sets for a specific project.

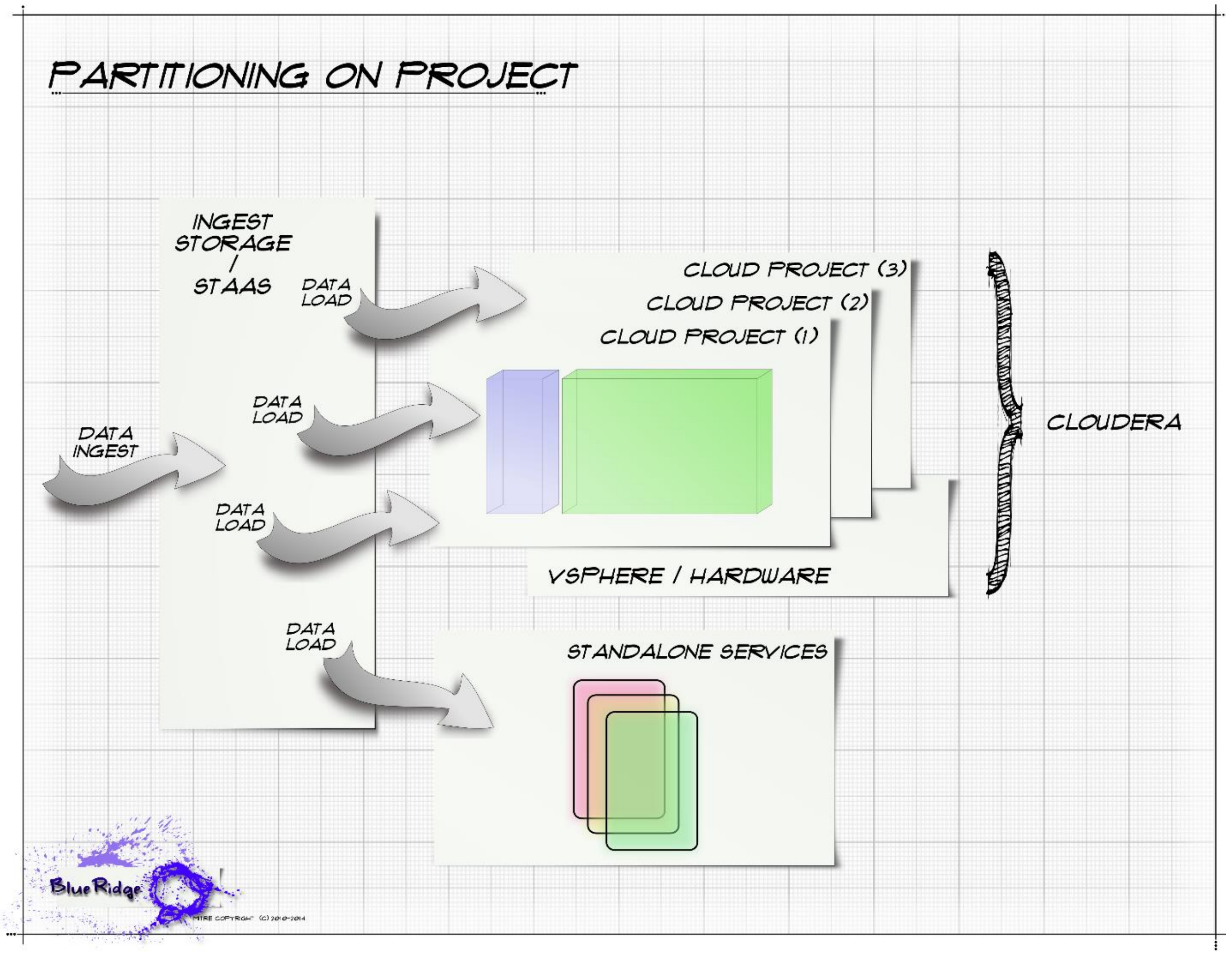


Figure 6. Architecture partitioned along different projects.

This leads into another architectural model, which is depicted in Figure 6. Here the architecture is partitioned across threads and projects, where each thread or project is assigned an analytic machine, whether virtual cloud machines or standalone services, as necessary and of appropriate scale to the size of the data. The complete data set, stored in STaaS at ingest, is then moved to the appropriate analytic machine for the project that depends on it.

If multiple projects require the same data, this solution is more weighty in the requirements of computational power needed to migrate the data and store it multiple times, but the advantage is there is no worry about threads interfering with one another by requiring them to share resources as each has its own dedicated virtual machine.

In selecting one of these solutions, the following questions must be asked: How unique and independent are the data sets required for each thread and for each project in a thread? Do projects expect multiple data sources? How much do projects depend on one another within a thread? Are there any dependencies between threads?

3 System Flow

3.1 Architectural Model Overview

Independent of the specifics of the ingested data and the desired result, a common system flow is expected for threads operating in the architectural model. This section describes the typical flow of data through the system as it is transformed through analytics into the results that will ultimately be provided to users to inform decisions.

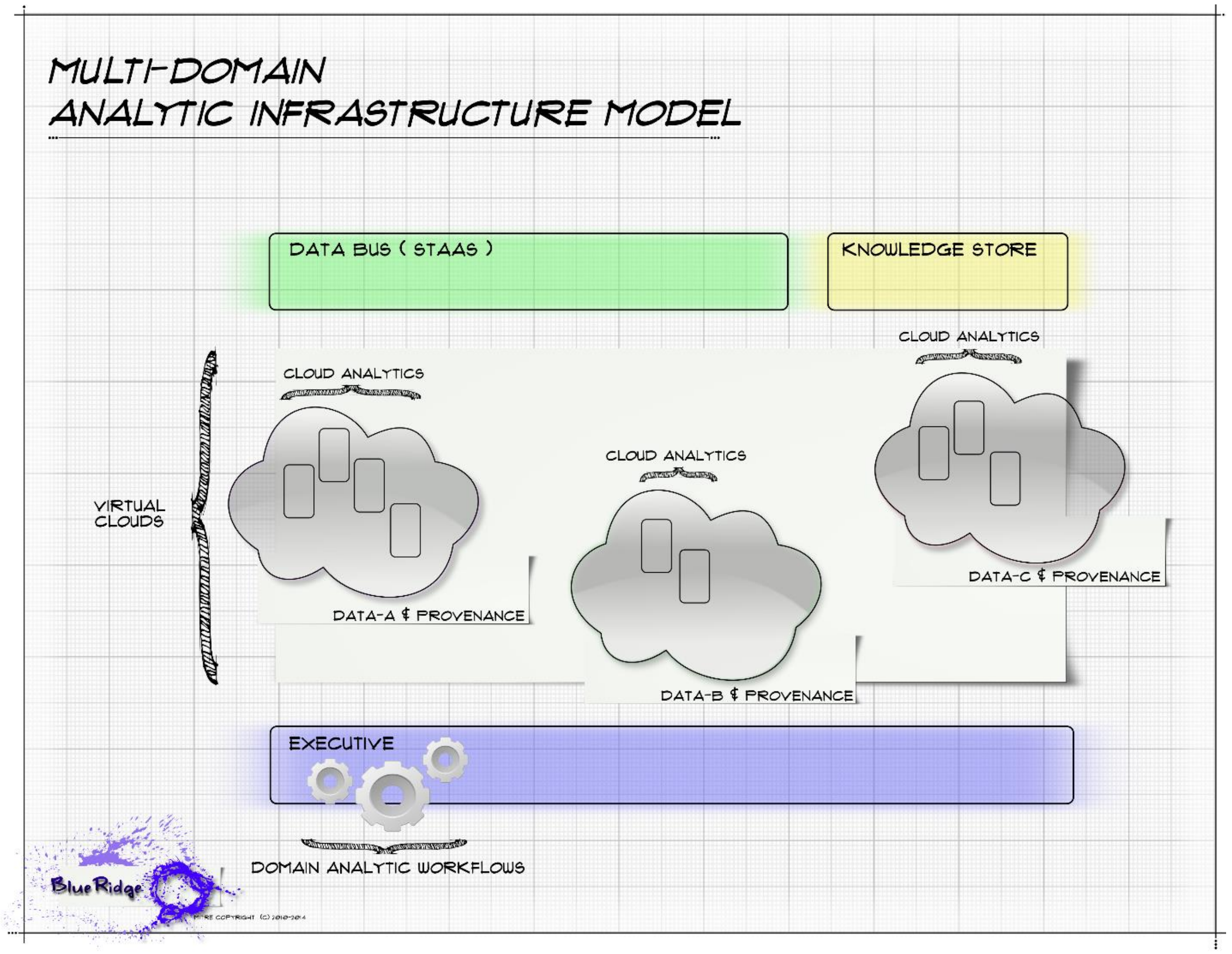


Figure 7. The architectural model for threads of analytics suitable for multiple domains.

Figure 7 provides a high-level layout of such an architecture, where multiple analytic machines, for example cloud virtual machines, support different data sets and analytics. Core components of this system are 1) the executive, which orchestrates the execution of analytics and the flow of activity, 2) the data bus, which stores the initial ingest of data, 3) the analytic layer, corresponding to the pattern recognition in the Data-to-Decisions model, is where analytic activity takes place and interim results and provenance are held, and 4) the knowledge store, where final results are placed before being visualized. This example flow follows the model where analytic machines, specifically cloud virtual machines, are stood up to support specific data sets.

3.2 Ingest and Data Preparation

The first part of the flow is the ingest and preparation of the data, as shown in Figure 8. Data ingest includes both the type of data and the method of acquisition. A generalized architectural model for research means that all varieties of data should be considered for any thread that will transform raw input into meaningful knowledge that can drive decisions. In terms of data acquisition, data can arrive in two forms: bulk and streaming. Bulk data is procured in agreed-upon amounts, either at once or periodically. Streaming data is collected with greater regularity and can be provided as a service. In either case, data acquisition requires an upfront specification of the data expected by any downstream workflows, its volume and velocity, the vendors who will provide the data, and the cost to acquire. The retention policy should also be considered here because streaming or repeatedly acquiring batches of data will accumulate and eventually need to be purged.

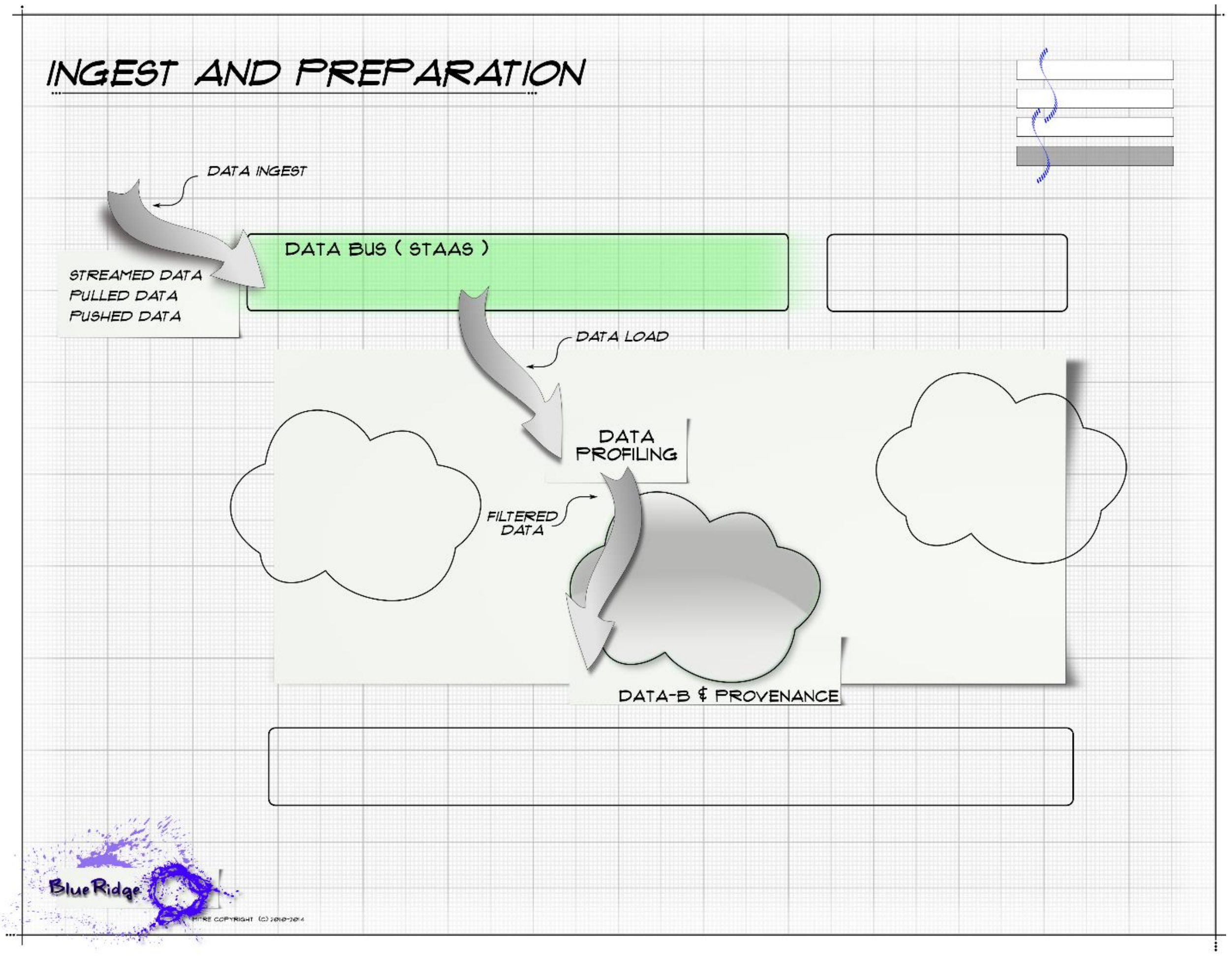


Figure 8. Ingest and data preparation flow.

If we follow the established example in a Hadoop cloud-computing argument, choosing a cloud stack to be the pattern will have an impact on data ingest issues. Figure 4 depicts some specific software solutions for different types of data ingest into a Hadoop cloud-based architecture. Apache Flume [9] is an appropriate application for streaming data flows. Apache Sqoop [10] can be applied for data pulled from outside data stores (e.g., relational databases).

HttpFS is the appropriate service for transferring into the Hadoop Distributed File System (HDFS) [7] from non-cloud and other cloud sources.

The data bus receives the ingest and can, by default, be implemented using the physical STaaS. The expectation is that the data bus will simply store the raw data and that virtual machines will be created to host the data for execution of analytics. Data is moved to a appropriate analytic machine, first undergoing data profiling, which performs any necessary transformation and filtration to make the data acceptable and usable for the thread (or threads) that require it.

3.3 Issues of Scale

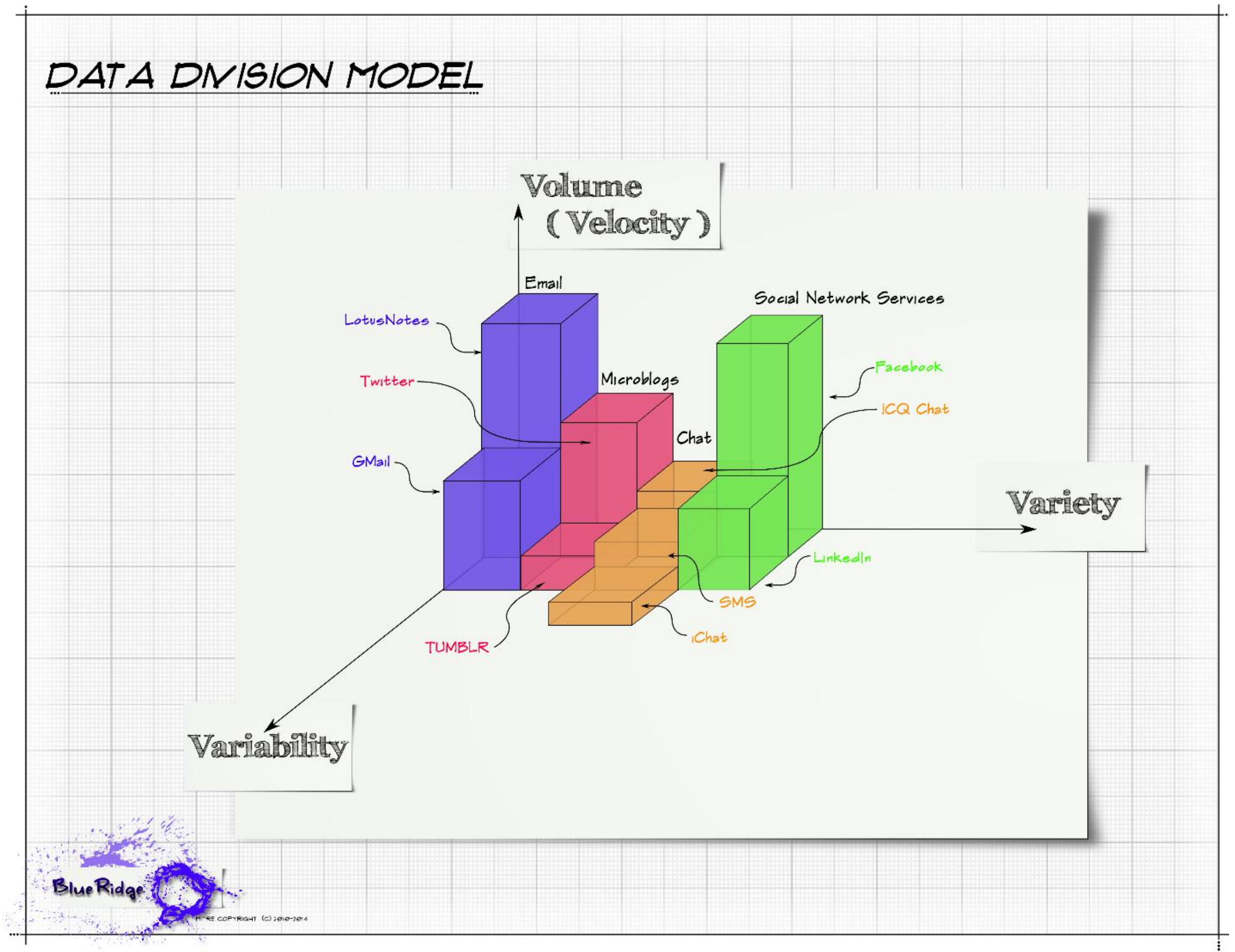


Figure 9. Data division model of variety over genres and variability over applications / protocols.

Because we are interested in a solution capable of generalizing to many different data types at large scales, it is useful to consider certain challenges within the context of the data bus. These challenges map to the common division of big data's "3V" challenges into categories of *volume*, *velocity*, and *variety* [11, 12, 13]. Note that *veracity* and *value* are sometimes listed as a fourth and fifth "V" [14], but we will not consider those for now.

While volume deals with size and velocity deals with rates of data in and out of the bus, these two can have the same impact, assuming data is being eliminated as new data emerges. Because there is a potentially limited capacity of the data bus for all threads and projects, space must be allotted in an equitable fashion. If the velocity of incoming data is such that the volume would exceed this allotment, some method for aging off data or terminating the ingest must be adopted.

The dimension of variety can be further subdivided based on the expectation of the analytic processes in the analytic layer, particularly when clouds are the required implementation. Analytics can be quite targeted, expecting a particular data format from a specific source or they can consume a more general type of input. It is useful to characterize these further, where subdivisions of variety are labeled *variability*. While there might be unifying features within a single category along the variety axis, this can be further separated into significant (to the analytics) differences along the variability axis. Figure 9 depicts variety along media genre and variability along application. Retaining these distinctions in the data bus makes it easier to migrate the necessary and valid data for a given project or thread to the analytic machines supporting the analytic processes for pattern matching.

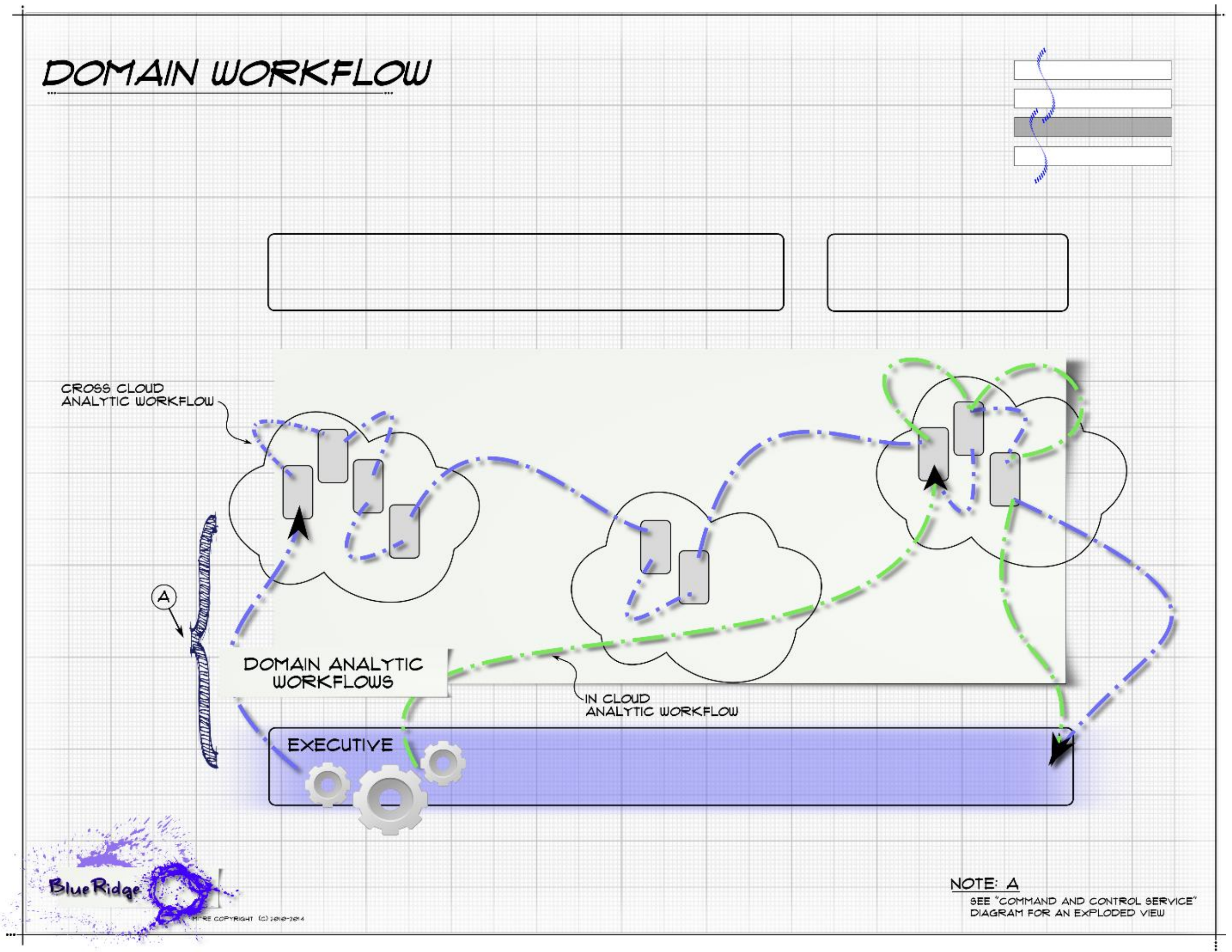

DOMAIN WORKFLOW
CROSS CLOUD ANALYTIC WORKFLOW
A
DOMAIN ANALYTIC WORKFLOWS
IN CLOUD ANALYTIC WORKFLOW
EXECUTIVE
BlueRidge
MITRE COPYRIGHT (C) 2010-2014
NOTE: A
SEE "COMMAND AND CONTROL SERVICE" DIAGRAM FOR AN EXPLODED VIEW

Figure 10. The workflows of projects within a thread's pattern matching operation.

3.4 Domain Workflow

Core to the flow through a research stack is automated analysis of the data to generate results as well as retain the provenance describing the activity that produced these sets. Figure 10 depicts the activity of multiple workflows through a system of clouds with analytics, although a mix of clouds and standalone services could be part of this workflow as well. These analytics operate on the clouds containing the specific data required. Many—perhaps most—domain cases may only need to operate on a single data set and cloud machine (the "in cloud workflow") or standalone service, but others may have dependencies between analytics operating on multiple data sets that span the different clouds and/or standalone services (the "cross cloud workflow"). The executive is used to describe and coordinate these workflows. Earlier MITRE work on MOSAIC evaluated different possible executives to serve in a Human Language Technology (HLT) domain but with an eye to generalizing to other domains [15]. The executive also reacts or reports when a workflow instance comes back as not having completed successfully. These analytic processes will generate interim results which will be input to downstream analytics or potentially part of the final results. Provenance, which is the retained description of the activity of analytic processes and their production of results, is also generated throughout a workflow instance. Generation of provenance can continue through the activity in the knowledge store.

3.5 Knowledge Store

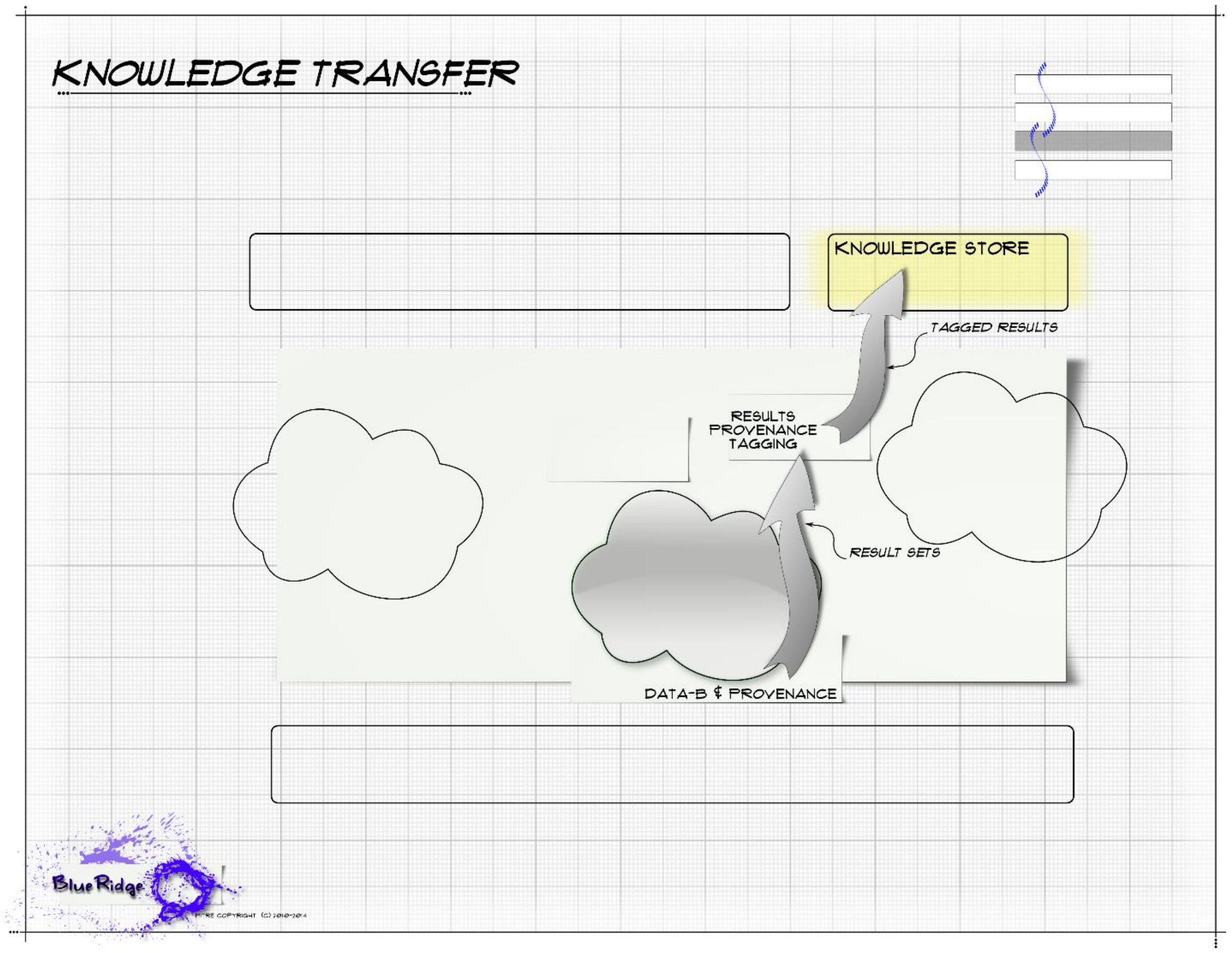


Figure 11. Transfer to knowledge store.

Analytic workflows will potentially find valuable patterns in the data, which will be captured in their final results. The final part of the flow out of the system is the transfer to the storage for the final results, or the knowledge store, as shown in Figure 11. Here, it can be retained and visualized downstream. This knowledge store, which is a collection of data stores and databases, can be on one physical machine, but also partitioned by project or thread or knowledge type. It is anticipated that not all computed results will be elevated to this storage. The knowledge store receives only those results that meet some threshold for importance and value to the ultimate decisions that need to be made.

This final knowledge is kept apart from the virtual machines hosting a thread's analytics and interim results. Worthwhile results should outlive analytic execution so they can be accessed even after a project or thread is taken down in case they are needed to inform future projects. While not a principal reason for doing this, this has the added effect of providing a level of duplication that can protect important results if one of the analytic virtual machines breaks down or is corrupted. By doing this, we can keep a distance between the data processing and the visualization layers. This makes more asynchronous flows from data to knowledge and knowledge to decisions possible.

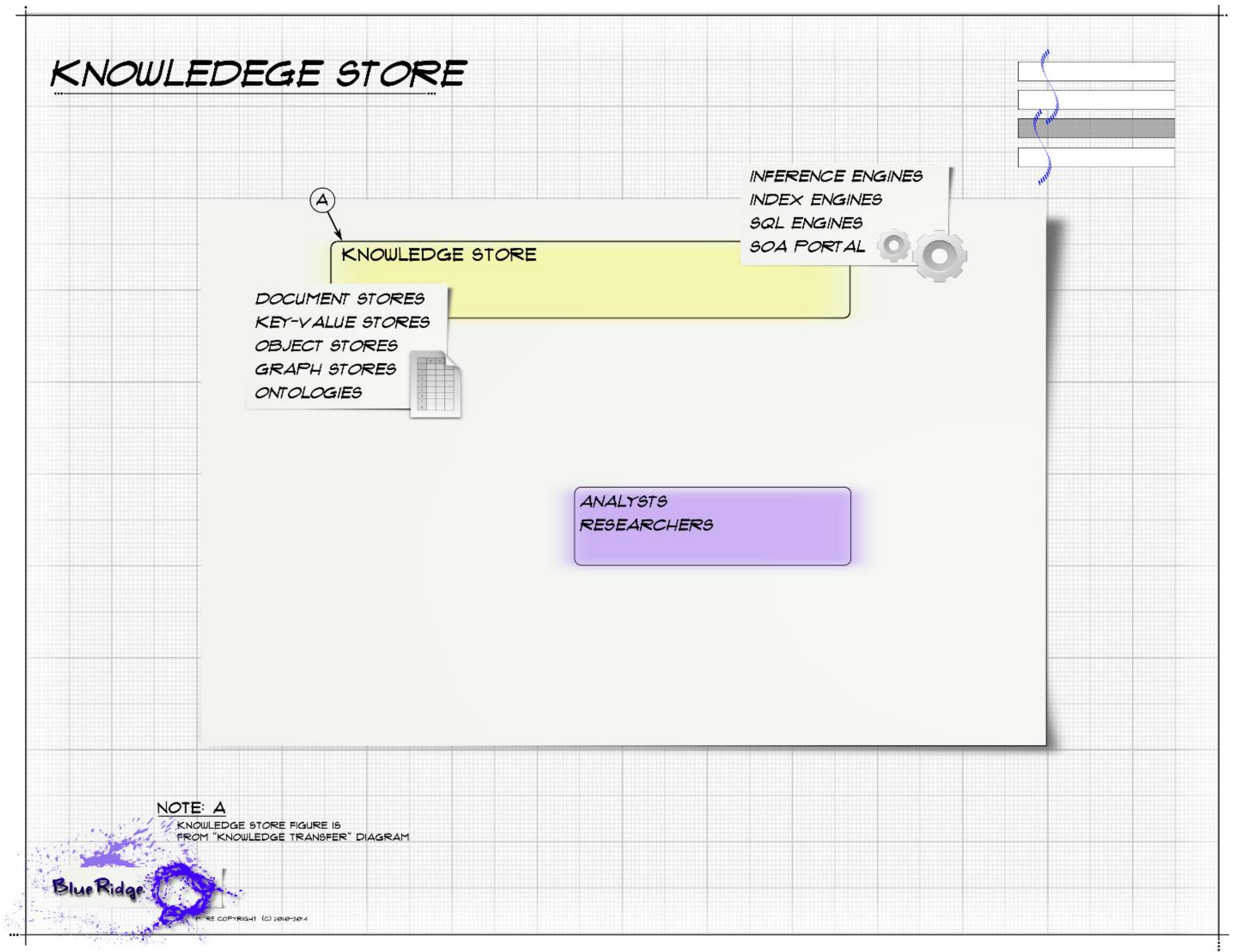


Figure 12. Depiction of the core elements of the knowledge store.

The knowledge store (Figure 12) is an overarching term for multiple stores of results encoded as different types of knowledge. These can include document stores, key-value stores, object stores, graph stores, and ontologies. The representation of knowledge and how it is accessed depends on the needs of the project and its domain, but a comprehensive data model can also be developed and employed. We discuss the data model represented as an ontology later in greater detail as this is a richer representation that can be applied widely to long-term results stored as knowledge. In some cases, this high-order knowledge is the desired outcome that needs to be visualized or interrogated. In other cases, relevant data plots or documents are what matters.

The knowledge store has internal functionality that post-processes the results. These can be as simple as indexing to increase query efficiency or as involved as inference engines that recognize relationships implicit in the acquired knowledge. Any final analytic work that operates on results as opposed to raw or interim data occurs here. What is produced is what will be queried, visualized, and used to inform decisions.

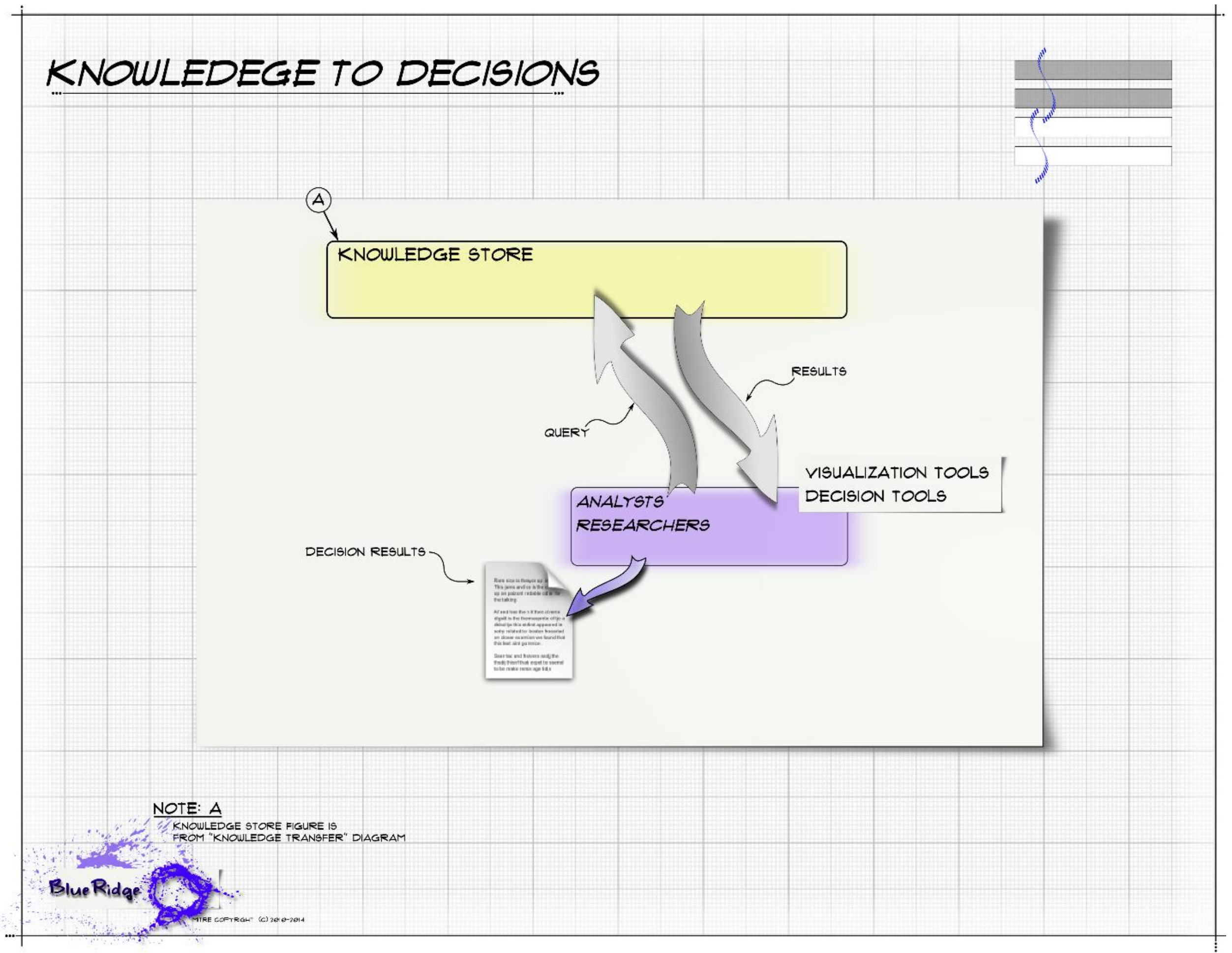


Figure 13. The flow of how the knowledge revealed by analytics informs decisions.

The activities of the top two layers in the research stack (Visualization and Decision) are captured in Figure 13. Here, knowledge is queried from the knowledge store and is then consumed by either visualization tools that present the results to a user (whether an analyst or a researcher evaluating analytics or workflow) or incorporate them into an automated or semi-automated decision tool in appropriate cases. Using these tools, users make decisions and document their rationale.

4 Workflow and Command and Control Service

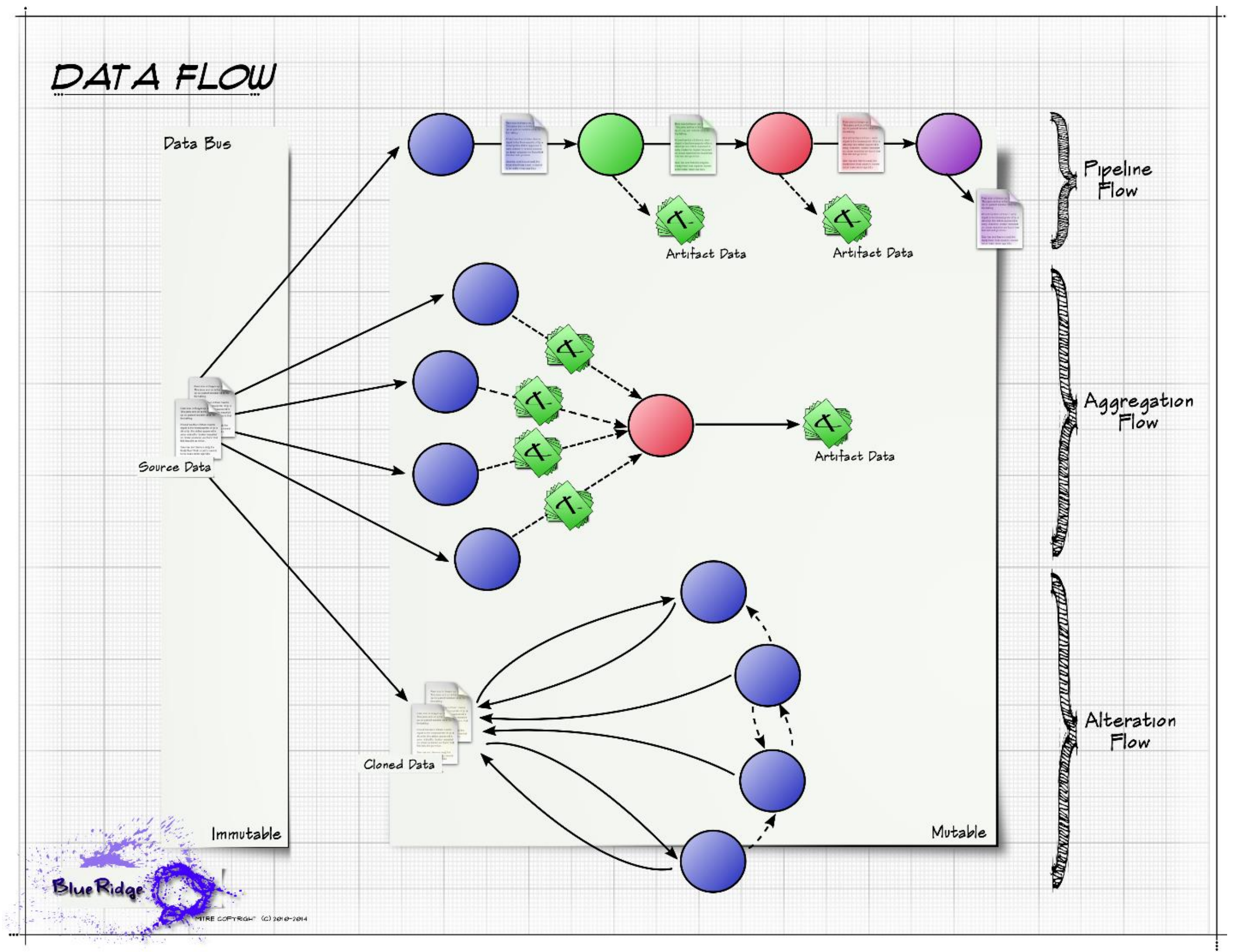


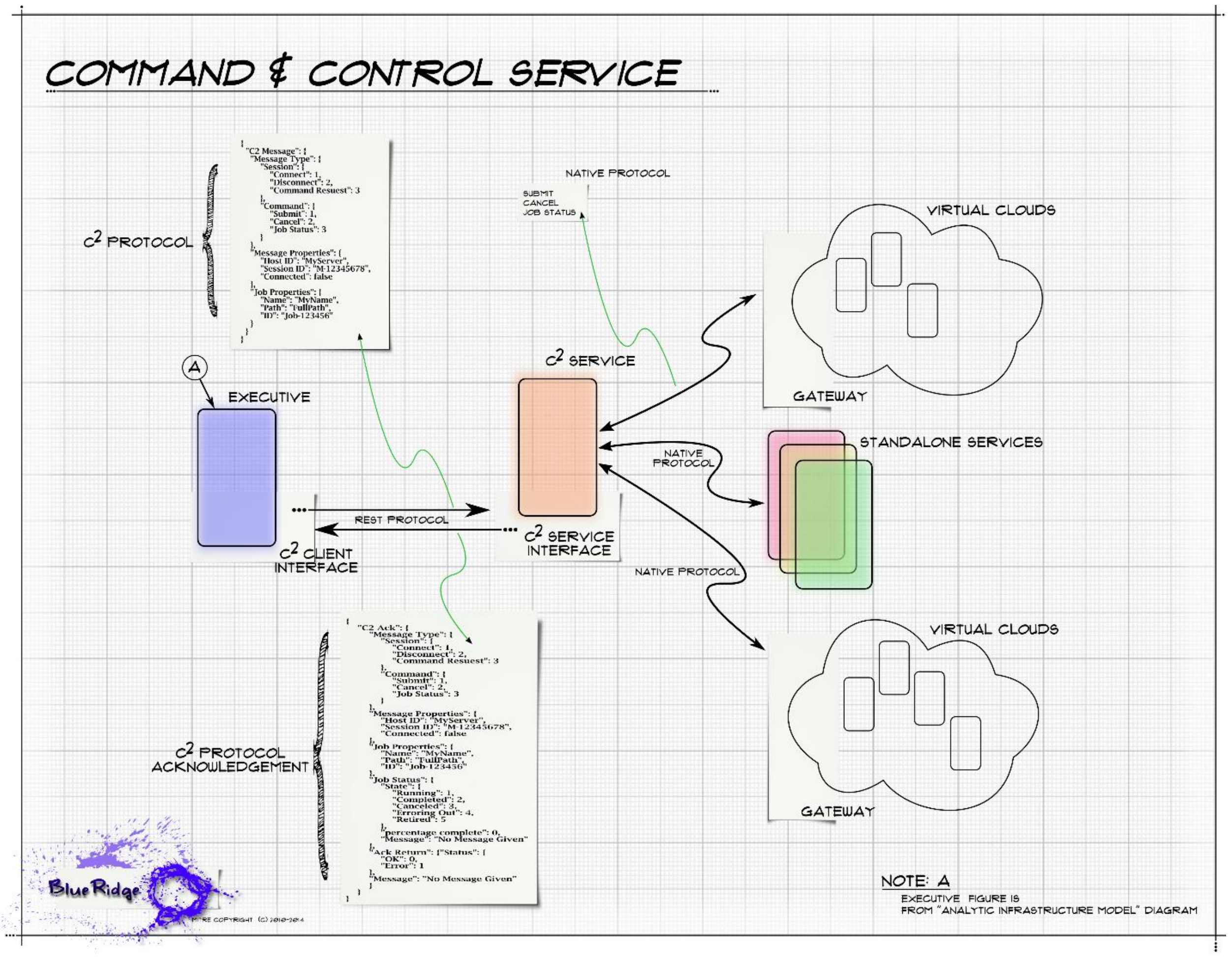
Figure 14. Varied depictions of possible data flows throughout the analytic workflow.

There are many possible workflows that can generate results and perform analysis of data in the analytic layer of the research stack. We discuss potential representations of data, results, and knowledge below, but first we describe patterns of handling and operating on data in a contained, but potentially extensible, research environment.

First, we assert that in the data bus, data objects should be treated as immutable. This is data that is ingested into the system and comes with any initial provenance that is provided that describes how and where it was produced.

Second, once data is copied and undergoes analytic workflow in the virtual machines, how data is manipulated and processed is more open-ended. Data objects from the perspective of the workflow can be input into analytics that enrich the data or summarize data and extract new data objects (*pipeline flow*), be aggregated and merged into new results (*aggregation flow*), or be altered and mutated such that the data object itself is changed by an application (*alteration flow*). Figure 14 depicts these different expected processes and their impacts on data, as well as indicating the dividing line between what is considered immutable in the data bus with respect to workflow and where workflow is given free rein.

Note that these concepts apply even to situations where we do not expect the analytic workflow to extend beyond a single virtual machine. However, to allow for a command and control service that is capable of operating across multiple clouds and standalone services, additional solutions are required.



```
{
  "C2 Message": {
    "Message Type": {
      "Session": {
        "Connect": 1,
        "Disconnect": 2,
        "Command Resuest": 3
      },
      "Command": {
        "Submit": 1,
        "Cancel": 2,
        "Job Status": 3
      }
    },
    "Message Properties": {
      "Host ID": "MyServer",
      "Session ID": "M-12345678",
      "Connected": false
    },
    "Job Properties": {
      "Name": "MyName",
      "Path": "FullPath",
      "ID": "Job-123456"
    }
  }
}
```

```
{
  "C2 Ack": {
    "Message Type": {
      "Session": {
        "Connect": 1,
        "Disconnect": 2,
        "Command Resuest": 3
      },
      "Command": {
        "Submit": 1,
        "Cancel": 2,
        "Job Status": 3
      }
    },
    "Message Properties": {
      "Host ID": "MyServer",
      "Session ID": "M-12345678",
      "Connected": false
    },
    "Job Properties": {
      "Name": "MyName",
      "Path": "FullPath",
      "ID": "Job-123456"
    },
    "Job Status": {
      "State": {
        "Running": 1,
        "Completed": 2,
        "Canceled": 3,
        "Erroring Out": 4,
        "Retired": 5
      },
      "percentage complete": 0,
      "Message": "No Message Given"
    },
    "Ack Return": {"Status": {
      "OK": 0,
      "Error": 1
    },
    "Message": "No Message Given"
    }
  }
}
```

Figure 15. High level depiction of command and control service for the research stack.

When we earlier discussed the system flow of analytic execution for projects in a given thread, we considered possible workflows that could make use of data and analytics across multiple clouds where the workflow was coordinated by the executive. This was depicted in Figure 10. In this section we describe the command and control service that allows the executive to orchestrate analytics distributed in this fashion.

We envision the command and control (C^2) service as a layer between the executive and the analytic processes, either operating on cloud virtual machines or as standalone services. The executive interacts with the C^2 service by way of a REST protocol—where commands and job properties are issued—and the C^2 service directly controls the various analytic processes and applications by way of their native protocol.

This C^2 service is a component that will be engineered specifically to perform this function. It should be applicable across all threads. Using the layout of Figure 15 as a basis for an example, Figure 16 depicts a protocol that allows the executive workflow engine to issue jobs to the clouds and standalone services by way of the C^2 gateway. In this example, a workflow will execute four applications in sequence, where the first and third (A and C) are in the same cloud, the fourth (D) is in a different cloud, and the second (B) is in a standalone service.

Before the workflow instance is executed across the different environments, the availability and status of the services and stores to be used in the workflow is established by the executive. The executive then stands up the workflow engine, which checks to ensure the C^2 service is available and initiate a session. Each application is executed in order, and when the application terminates successfully, an adapter is applied to the results to be pushed to the knowledge store for persistence. Adapters ensure the results will be mapped to a common schema with a consistent model. When the workflow instance has finished, the executive closes the C^2 service session for that workflow.

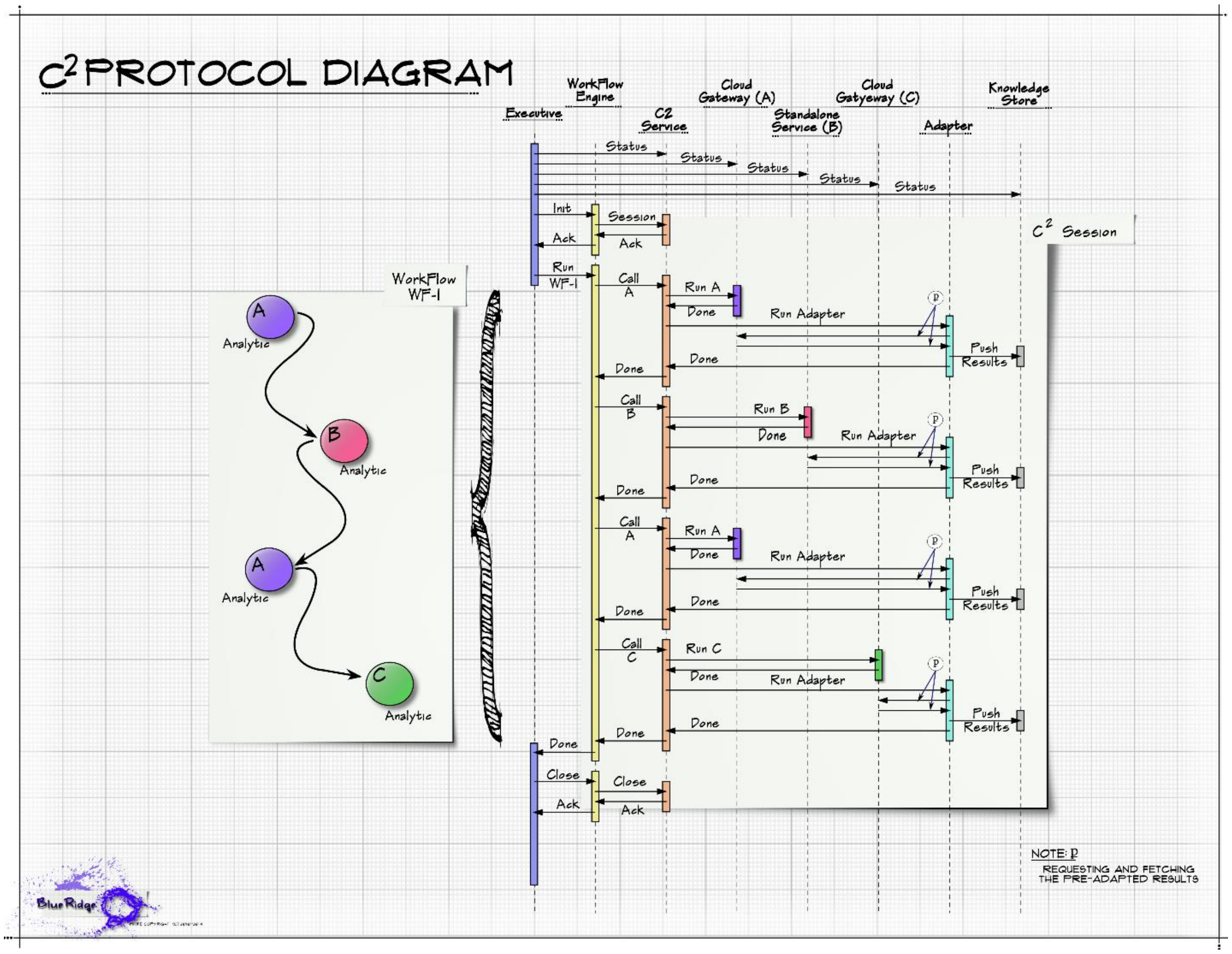


Figure 16. Example protocol diagram for the research stack's command and control service.

5 Knowledge Model

While analytic results can be preserved in many different representations, here we provide an in-depth examination of an ontology approach as we expect this will apply to most cases of the knowledge that should be preserved long-term. As such, this assumes the analytic results elevated to inform decisions will be represented as knowledge. Because analytics may populate the same knowledge elements that inform decisions, this suggests there is a benefit in adhering to a common model throughout a thread or across multiple threads, in cases where the output of threads are expected to enter the same knowledge store. Knowledge is potentially valuable across domains, informing decisions in varied contexts.

Ideally, a single comprehensive ontology would serve all use-cases and provide a common reasoning framework that draws on knowledge derived from analytic results in any and all domains. Yet thus far no single ontology has been offered that suffices for this role. The principal concerns with crossing multiple domains include 1) the lack of any relevance between certain domains and 2) collisions due to incompatibilities or ambiguities in modeled objects.

Users of large comprehensive ontologies, such as Cyc [16, 17] or SUMO [18], have encountered these issues before. In the most abstract description, ontological levels—such as upper, middle, and lower—assist in distinguishing the commonality of terms and statements. Terms and statements that are universal across represented knowledge domains belong in the upper ontology, while middle and lower ontologies become more and more domain specific. A lower ontology may insist on complete consistency of terms and assertions within itself, but can have particulars that deviate from other lower ontologies. In the case of the Cyc ontology, the lower ontology contextual sets of concepts and assertions are termed micro-theories [19]. The simplest analogous model in our context would be to adopt a separate micro-theory for each domain, but without some binding upper ontology, this is essentially equivalent to having multiple independent data models. This is not a problem when there is no relevance between domains, but for domains that can or must supplement one another, it leads to a shortfall when performing reasoning or analysis that crosses knowledge derived from different sources. In cases when domains must be spanned, the lower ontologies can be designed from the outset to be interrelated (essentially a single merged lower ontology) or they can inherit their commonality from the upper or middle ontologies further up the hierarchy of types. It is customary that lower ontological models be subtypes of elements in a higher level ontology.

Given the open-ended nature of BlueRidge, developing an overarching knowledge model to accommodate all domain and thread combinations a priori is infeasible. It is equally impractical to expect a pre-existing knowledge model or ontology to cover every need specific to an emerging thread or that one can be prepared in a final state at the outset. Instead, it is more reasonable to consider that each new situation will require some initial specification of what domains will feed into the ultimate decision-making process; this should direct the course of how the knowledge model is organized for each case. Because this work is intended to span multiple threads and be useful to projects at varying stages of readiness, from immature to nearly complete, the emphasis is not on building specific knowledge models but fostering the capability to build models flexibly, whether they are unique to a single thread or cross multiple threads.

Minimizing the work involved in creating new domain-specific knowledge models now becomes the priority. If no existing domain-specific solutions are suitable, these typically must be hand-crafted on a case-by-case basis, but they need not be constructed from the bottom up each time. Existing ontologies that have broad coverage of high-level and generally universal terms such as Cyc or SUMO can be imported to serve as the upper ontology. Individual lower ontologies for the domains can be made to inherit from these and supplement the coverage of terms and assertions not present. BlueRidge will provide a capability to generate domain specific knowledge models separately with specification files and adapted into any number of multi-domain ontologies as ultimately needed. This circumvents the issue of creating a single ontology in the blind, while enhancing the capability to systematically create and combine knowledge models so that components can be used and reused in new combinations as required.

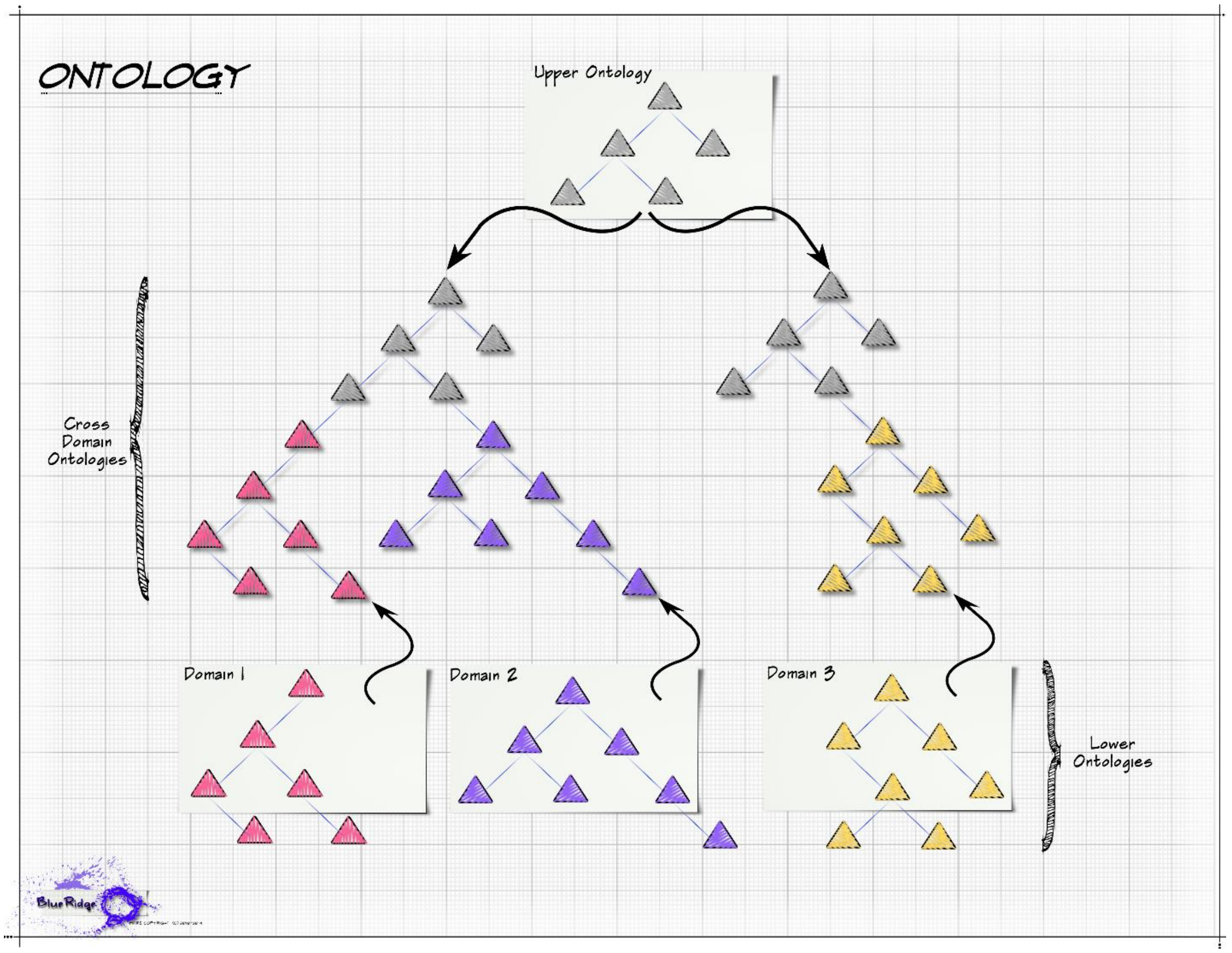


Figure 17. Multi-domain knowledge model.

This effectively means that a new model can consist of any number of imported existing schemas, pre-created models written specifically for domains, and new models. This is achieved by constructed by combining any number of contextually related lower ontologies, binding them to any appropriate upper ontology, which can be pre-existing or created to serve the role. Figure 17 depicts this modularity. Collisions of terms and assertions in the domain specific models are allowed—they effectively end up merged, inheriting from multiple sources (see Figure 18)—and types can be enriched with constraints and rules derived from different domains. It is incumbent on domain experts and thread orchestrators to understand the knowledge representations and recognize whether or not there is consistency between what is being combined, both pre-existing and new, both in the precise semantics of terms and the implications of rules within the domains. In the long-term, BlueRidge intends to provide a capability to assist in checking the consistency of the semantics when new sets of terms and rules are assembled. This matters because unlike multiple micro-theories that lack mutual inheritance, these lower ontologies are expected to coexist to allow for representation and reasoning across the domains, an issue of modular ontology design that has been explored but not universally solved [20, 21, 22].

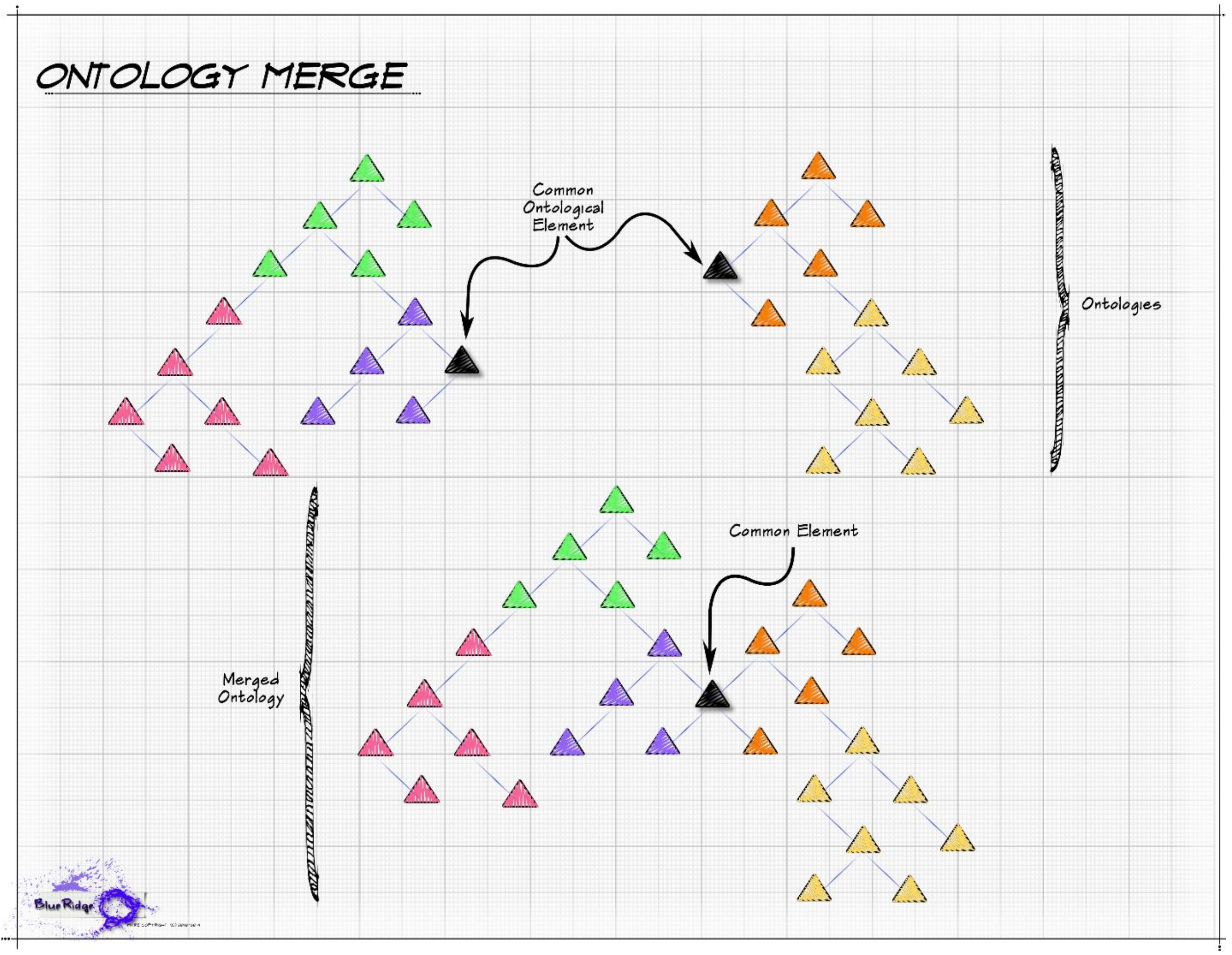


Figure 18. Two data models that have been merged which share a common element. In this case the element is able to inherit qualities from both source models, making it applicable in either domain.

As a final point, while the resulting models are intended primarily for final knowledge results, such models could be used by analytics within their workflow. In practice many analytics across the threads will not have been initially designed for the model, so it is unlikely a model serving this role in the analytic layer would—or could—be mandated. Mandating this would incur a great deal more adaptation work for analytic output, which might not be useful if interim results are only consumed by downstream analytics in a thread designed to handle them in their native format. But provided analytics are not legacy or externally developed, a common model within the analytic layer would allow for a greater degree of flexibility when replacing analytics, with the significant requirement that new analytics produce or consume the common data model or have applications to adapt from their native formats into the common knowledge model.

6 Error Detection and Recovery[1]

In any complex system, error conditions are inevitable. Ingested data may be partially or wholly corrupt, or the ingest process itself may fail such that some expected data is missing. Similarly, workflow processes may freeze, crash, or produce corrupt data. If error conditions are not detected and responded to rapidly, time and processing resources are wasted. In the best case, the error condition is detected and resolved quickly, and the corrected workflow is able to reprocess the data from an archive before any user attempts to us the system to make a decision. In a worse case, a user may be prevented from using the system while it is in an error condition, or a user may unknowingly make a misinformed decision if shown corrupt results without realizing that the system is in an error condition.

The simplest error conditions to detect are those which cause the absence of an expected result. Applications responsible for data ingest can monitor themselves to detect failures to ingest data. Similarly, the executive responsible for orchestrating analytics can monitor workflow processes to detect when they timeout or fail to produce results. In both cases, rapid reporting of the error condition to a responsible party capable of correcting the situation will reduce the time the system spends in an error condition, which will mitigate the impact of the error condition on users of the system. To achieve this, error monitoring must be accompanied by a reporting system which is configurable to ensure that the appropriate action is taken or the appropriate person notified as soon as possible.

Error conditions in which corrupt data is ingested or wrong results are produced are more difficult to detect. Validation tools can be used to ensure that data and results conform to a specific format. In the case of validation on data ingest, because each data source is different, validators would need to be created individually and as necessary for each data source. In the case of analytic results which are intended to conform to the system knowledge model, a general purpose validator can be used to ensure basic compliance. Such a validator would be applied on all results destined for the knowledge store.

The most problematic error conditions are those in which results are valid in format, but wrong in content in the sense that they would mislead decision makers in an unintended way. This type of error condition can occur when a change in the nature of the data being ingested or the output from a prior workflow process invalidates the assumptions of a later workflow process such that its results no longer match the original intent. This error condition is extremely difficult to detect, and if left untreated can continuously pollute the knowledge store with wrong results. However, by monitoring and baselining features of the output from a workflow process over a period of time, in some cases it is possible to identify anomalous output that differs from the baseline and may indicate that there is an error condition. Rapid attention to alerts of this nature can mitigate pollution of the knowledge store.

[1] This text was contributed by Nathan Giles derived from his MITRE research project that discusses on result anomaly detection in a Human Language Technology context. Details on this research are provided at: http://projectpages.mitre.org/projects/EPF-13-01556?fiscalYear=2014

7 Security

7.1 Data States

There will be a need in the research stack to derive knowledge leading to decisions from data that is sensitive and has requirements for storage and access levied by customers or data providers. Therefore data security, compartmentalization, and control are topics that must be considered throughout any solution for a comprehensive architectural model and at every stage of the system flow. Figure 7 serves as a starting point for depicting the general directional flow of data as it generates results and knowledge (as shown in Figure 19), which amount to significantly different scenarios where data security must be considered. From this we derive three locations where security needs to be addressed: A) the data bus, B) the analytic layer consisting of cloud virtual machines and standalone services, and C) the knowledge store. Figure 20 shows this partition.

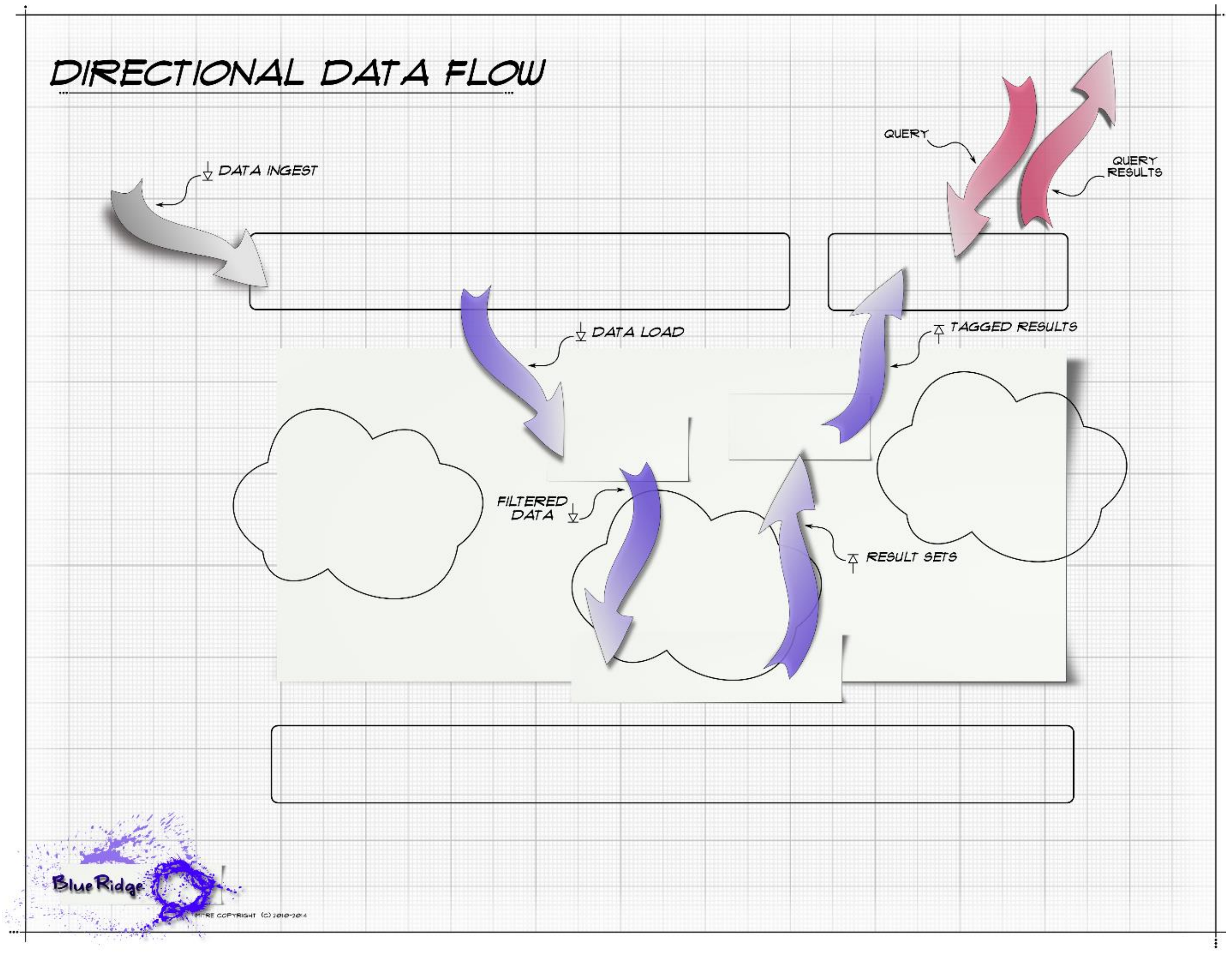


Figure 19. Directional data flow throughout the analytic layer into the knowledge store.

While the data bus (A) is logically a single component of the architectural model, it can physically be several different machines. Therefore it is indicated that if there are special sensitivities for data past acquisition and ingest, but prior to execution, separate machines that have the appropriate restrictions on access are used. While the proposed STaaS can be the default choice for the data bus, the model does not preclude any special cases required by certain data.

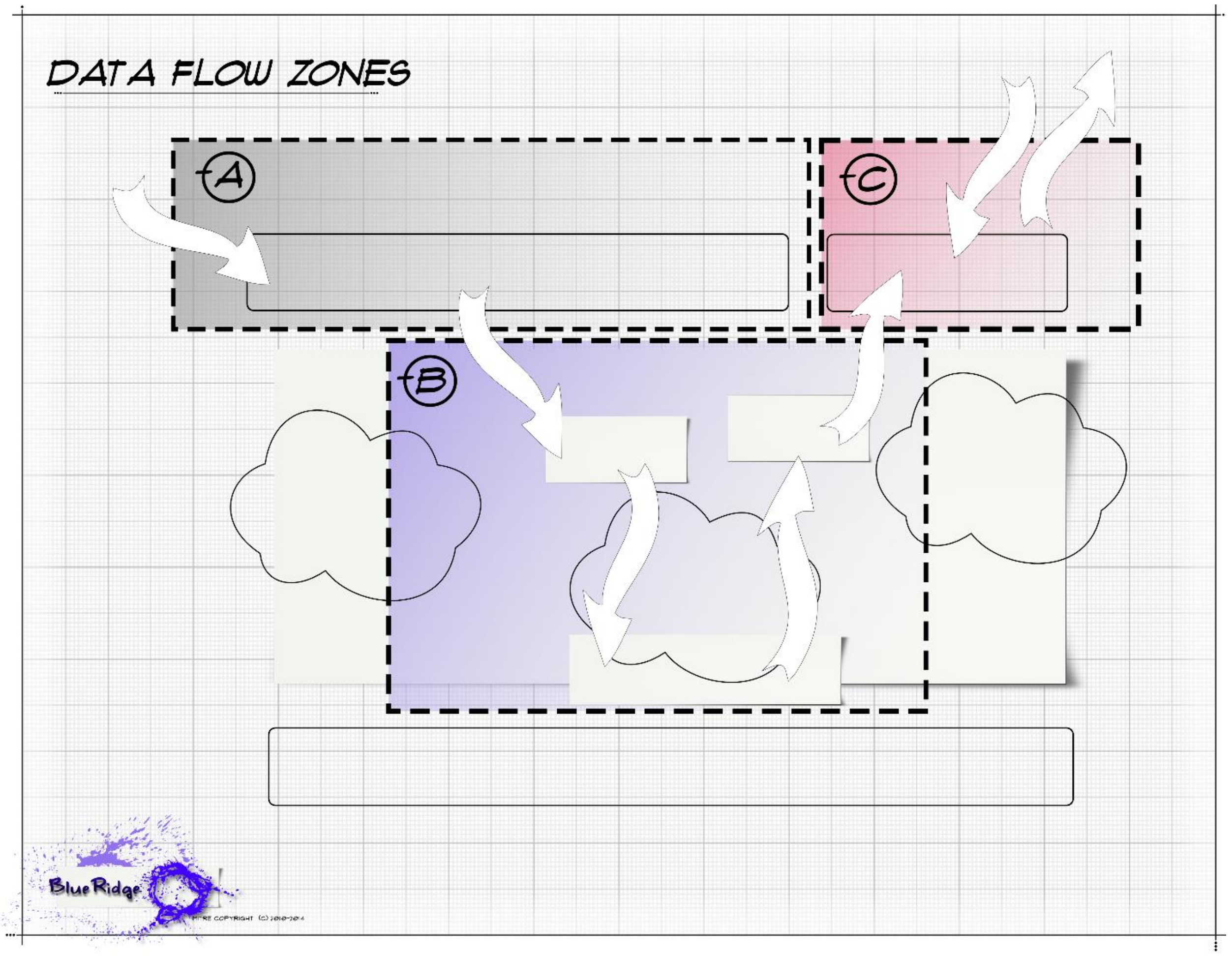


Figure 20. Data flow zones from ingest through analysis and the knowledge store.

How security is handled on the analytic layer (B), where data is held for analytic execution, depends on the underlying model. If these machines and services are partitioned based on the data, then security is merely a matter of ensuring that the machines are only accessed by the proper individuals and workflows. Individual machines can be stood up to house data fitting security requirements case-by-case. If the machines are partitioned based on projects and threads, then the situation is more complicated. If there is only one data source per project or thread with sensitivity, then the environment, whether cloud virtual machine or standalone service, must conform to that data's requirements. If there are more data sources per project or thread, then it may be necessary to partition the project or thread into multiple machines. If this is anticipated, it is recommended that the analytic machines be partitioned based on the data rather than the project and threads.

Further security concerns around data at this stage involve ensuring workflows accessing sensitive data have the proper permissions. Workflows that will access virtual machines with sensitive data will require authentication on the part of users running instances of them or if they automatically access the analytic layer's data and analytic results then the workflows themselves must have their access and execution restricted. Ultimately, this is something that needs to be decided in each situation depending on the workflow requirements and how much human interaction is involved.

Finally, imposing security in the knowledge store (C) is a trickier proposition. An important question that needs to be determined for each data set is whether analytic knowledge derived from it is considered to be at the same level of sensitivity. If it is not deemed sensitive, then there is no issue at the knowledge store for a particular data set. If it is deemed sensitive, then some solution is required. If true for this knowledge, the same likely also holds for the analytic results that need to be secured in (B).

Again, while the knowledge store is intended to be logically a single component of the architectural model, in principle if threads are independent the best course is to ensure that only visualizations and users with the proper authentication access their data from a knowledge store that houses the results in a manner compatible with its specified restrictions. This simplified solution assumes there are not data sets with differing restrictions. Further if the knowledge derived from threads is intended to cross multiple threads, then this poses an additional problem.

We must consider not only the location in the research stack, but also take into account the state of data, such as *data at rest* (on disk or other storage media), *data in motion* (moving across networks), or *data in use* (in the CPU or RAM). Our earlier delineation of security concerns chiefly considers data in use or at rest, typically on one or more analytic machines. The research stack assumes that data will be in motion at various stages, certainly at ingest—whether bulk or streaming—and when generating resulting knowledge but also potentially as data or results are moved between clouds or standalone services during workflow. Naturally decisions here, as with data at rest, need to be made based on the requirements of data providers and customers.

Finally there are important questions to consider when devising a comprehensive solution to data security. What is the default underlying partitioning of the virtual machines? Will any threads require multiple data sets with differing, potentially incompatible, data security requirements? Will knowledge derived from sensitive data sets retain the same levels of sensitivity? Will knowledge derived from different threads be shared across different threads?

7.2 Data Leakage Mitigation

Of primary concern in security is data leakage, where an outside, unauthorized party receives data not intended for it. This is not necessarily malicious, as cloud-computing environments increase the opportunity for accidental data leaks [23]. Intrusion detection systems, firewalls, and antivirus software are all baseline security measures that can be taken to secure data. There are also more involved methods that can be applied, such as using machine learning techniques to find abnormal behaviors into or on a system or lexicons and keywords and data matching to recognize unauthorized attempts to retrieve or transmit data. Methods can be installed and applied at egresses to monitor and handle data in motion or locally on client systems for data in use.

While obvious, it bears mentioning that, with respect to any of the identified locations where security concerns exist (i.e., data bus, analytic layer, and knowledge store), access should be limited to only people who are authorized to view, manipulate, and extract from the data. Further data should not be moved from one level to another unless each location is authorized to hold the data. There are additional concerns at the analytic layer and knowledge store where analytics can operate on the data, producing results. Analytics should not be applied to sensitive data if they also involve transmission of its sensitive qualities outside the local environment. Because of analytics' abilities to communicate outside the local environment or knowledge store, which analytics can be applied in certain circumstances can be limited. For some cases where it is warranted, data at rest could be encrypted to further protect it.

While monitoring and encryption serve to protect data, in the cases where data leaks occur nonetheless, there are further ways to trace the data and culpability. Some solutions include watermarking or steganography [24, 25], which can identify the source of a leak of data by uniquely identifying information. Less obtrusive methods have also been researched for detecting the leakage of data sets, such as data allocation strategies that improve the ability to identify sources of leaks or the inclusion of "fake data objects" that uniquely identify a source given their inability to be guessed or derived from other sources [26]. While these strategies can be employed by data providers, our perspective is on the research stack. In this context, we look at these as methods as applicable when moving from the data bus to the different analytic machines for different projects or threads or when moving from these analytic machines to the knowledge store or from any of these components outside the research stack. Using these techniques facilitates identifying where the data was compromised.

As stated above, it is important to determine the sensitivity of derivative results. Data provenance can potentially assist in this regard. The provenance tied to results records the history of these data products, indicating which ones are generated or extracted from sensitive sources. Additionally, provenance can build an end-to-end picture of the workflow that generated these results. This is essentially a result's genealogy. Depending on the type of transformation or creation of the data, these results might be equally sensitive and unable to move off the analytic machines or the knowledge store or at least be limited in terms of who can access them or where they can be copied.

8 Provenance

Provenance can be defined in varied ways to capture the context of application workflow. The term is derived from the history of ownership of objects of interest, but when applied to data, it takes on a new significance. Provenance is the derivation history of a data set or result, where this derivation history is the description of activity of analytic processes that informed the data object's creation. In this paper, this is how we use the term "provenance."

Knowing the history of data can give insight into the trustworthiness of the data and inform the ultimate decisions based on the data and its results. The major stumbling block to this is the diversity of application implementation and ownership, making it difficult to capture and retain provenance across real-world enterprises [27]. Given the varying degrees of control over the data that can be expected when partitioning the research stack across multiple threads where data may flow from different organizations, this may hold in the case examined here.

Most existing provenance solutions, applied within scientific applications or relational databases, assume a "closed world," which increases the challenge of integration in enterprise use-cases with distributed or discrete analytic processes, especially those where one cannot re-engineer or mandate the re-engineering of existing applications that are essential to the production of a data object. Instances of these violate requirements of an "open world" where provenance must be captured across multiple systems with no assumption of control over those systems and where provenance must be captured from legacy systems that do not natively produce provenance [28].

The "open world" provenance model (OPM) [29] and PROV [30] serve as a basis for solutions to handle the two specified requirements. This is a more generic model that bears resemblance to that employed by a MITRE-developed solution called PLUS [28]. This model stores provenance in a directed acyclic graph. Nodes can represent data sets, process instances, and agents acting as process catalysts. Edges represent relationships between these nodes such as data *used* or *input to* processes or *produced by* processes, thus indicating how the data passed between processes leads to data sources, sinks, and each other. This means for any data set or result, a graph that records the history (and future usage too) can be constructed, indicating instances of processes that have executed on previous inputs to produce the current data object of interest as well as process instances that have produced new, refined results. Traversal of this graph, both for the backward and the forward lineages, is essential for the value of the provenance model.

The lineage that the provenance tracks goes back to when data is transitioned into the analytic machines for execution and can be extended to describe activities on knowledge results in the knowledge store. How provenance is stored in the research stack can be approached in different ways, but largely should depend on the choice of how these machines are partitioned. For a partition-free environment or partitioning based on the data, having a central store that tracks provenance is the best solution for workflows of moderate scale. This centralized storage is simpler, would be accessible from all subcomponents of the research stack, and could issue provenance outside it in accordance with sensitivity issues. For environments partitioned based on threads, provenance will be stored with the thread's data and results. There are issues with either choice of storing provenance. A centralized storage could be a single point of failure across all threads and likely would be a bottleneck as different threads compete for its resources if they are all actively producing and using provenance. Storage per analytic machine is complicated by the added difficulty when it needs to be accessed or moved.

Given the potential data sensitivity issues described in the Security section, these issues should be extended to the provenance, where organizations may have restrictions on what information about practices that describe the data and workflow history can be released to individuals or other organizations. Following the PLUS model, organizations are able to specify how information submitted as provenance is released as well as providing alternate information to users unable to view the complete provenance graph.

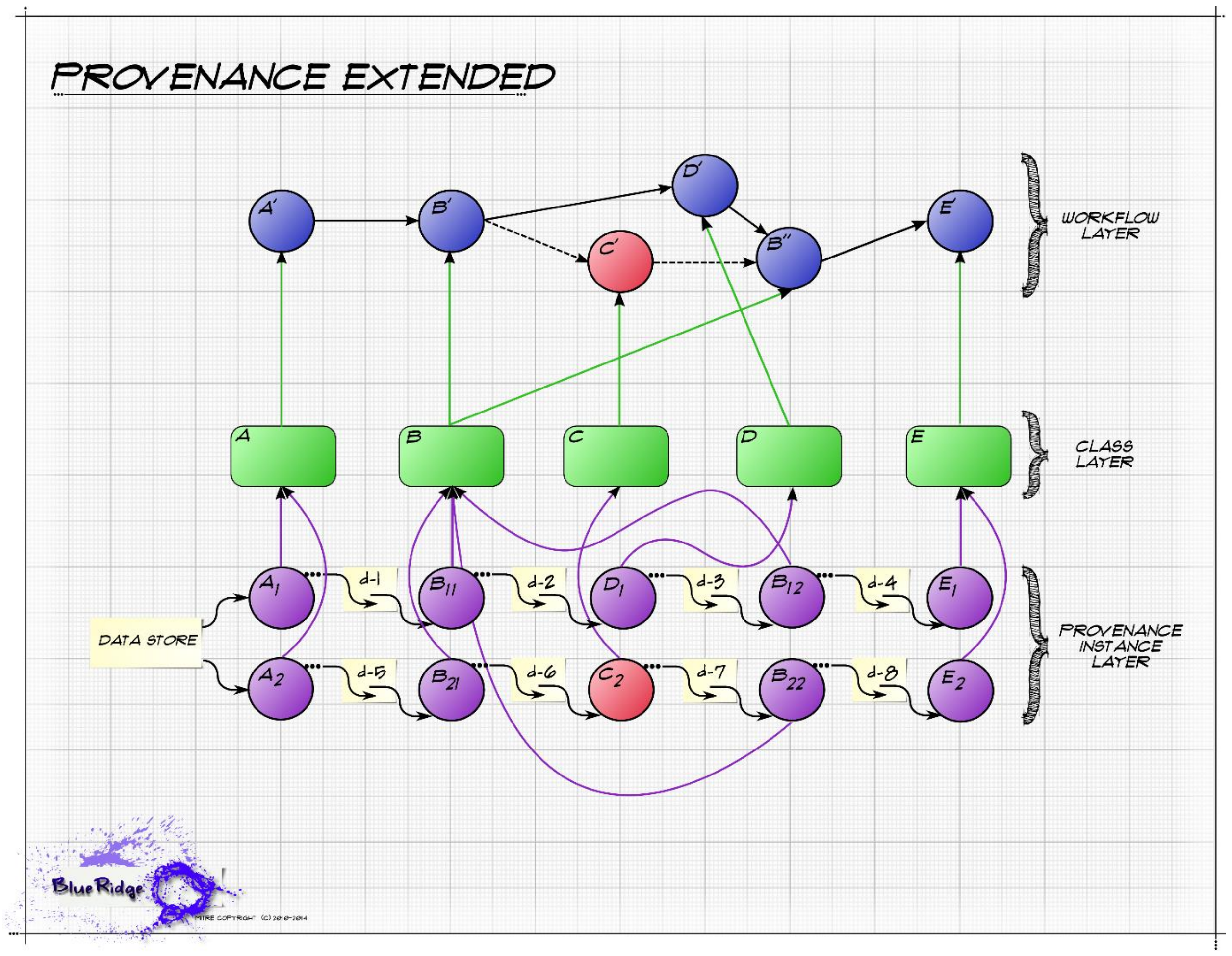


Figure 21. Extended provenance model.

We also can levy some additional requirements to make provenance more applicable to workflows in research domains. Provenance models expect that the graphs representing history and forward usage of data will be connected by way of process instances in nodes. Also including a representation of the process itself (that is, a node representing the application or activity's actions in general as opposed to a specific run or execution of it) and its role in a workflow can bridge multiple instances to reveal more plainly which processes are being employed multiple times in a workflow involving a particular data set. This is depicted in Figure 21, where the provenance of specific process instances produce and consume data in the instance layer, the intrinsic processes—be they applications or activities—are represented in the class layer, and different workflows that utilize these processes (and can be discovered as being instantiated in the instance layer) are represented in the workflow layer. In this representation, the class layer assumes the role of a hub bridging equivalent processes as viewed in different, potentially repeated workflow roles and as executed examples that have manipulated data. For instance the workflow example shown in Figure 21 has multiple paths that can be taken given different criteria; the depicted instance layer contains two executions that took different paths (process C is colored red in both the workflow and instance layers to highlight it as an alternate path).

Further, because these process instances can vary greatly due to parameterization, having a property attached to each instance that includes the necessary parameters and settings information is required for the repeatability of a workflow. This can be achieved in much the same way that PROV uses attributes and values. Further, these additions can allow for the discovery of implicit workflows within the provenance.

Until now we have been operating under the assumption of adhering to an open provenance model. Naturally, it is infeasible to mandate external systems generate, handle, and retain provenance using our model, so this is taken into account by how we handle data within the research stack. Recall that Figure 14 established different workflows through the research stack and further delineated immutable and mutable data objects. Objects in the data bus were considered immutable and retained any existing initial provenance provided with them. We establish this principle to allow for a consistent picture of past provenance so that any threads using a data object can have consistent expectations about that data. Because data can be altered and overwritten by way of a pipeline flow or an alteration flow as shown in Figure 14, this requires a sense of provenance that takes into account that data objects can be changed, which may be as straightforward as considering altered data objects as new data objects where the older object does not exist any longer.

There is a final, but critical, caveat about provenance that must be acknowledged. Typically provenance is a weighty—if not the most weighty—retained information about data sets and processes. Because of this heft, one must be cautious and cognizant that in “big data” scenarios, provenance could potentially increase the storage requirements and costs by orders of magnitude. This is situational and not entirely predictable at the outset, which is why being aware of its possibility is required. One solution to this issue is to apply provenance at a coarser grain, but there is no guarantee this will be sufficient for every scenario. Ultimately the recommendation for any given thread is to employ provenance if it deemed essential when weighed against its costs.

9 Conclusions

The proposed concept for the discussed architectural model is founded on the notion of addressing domain-specific problems in a generalizable fashion. Applying this to a shared research architecture spanning many projects, threads, and data sets is an ambitious endeavor requiring an architectural model to bridge each core element of the investment area (data preparation, pattern recognition, visualization, and decision) and to enable the analytic flow from raw data up to high-level decisions that are informed by knowledge derived from pattern recognition and automated analysis. Furthermore, this needs to be accomplished at a large scale given the anticipated sizes of data and results.

In this paper we have provided an architectural model that will support a diversity of research projects and allow for their coordination across multiple research domains. We have proposed varied partitioning methods for the analytic environment based on the expected needs of the research threads and projects. We have also proposed a method for orchestrating across multiple cloud-computing environments and described our approach to modeling knowledge. Our model accommodates the varied types of data and workflows that researchers require across potentially multiple data sources and provides a common storage for results that will be visualized and used to improve or suggest ultimate decisions for a user.

10 Glossary

Aggregation Flow: a workflow of analytics that aggregates and merges data and interim results into new results

Alteration Flow: a workflow of analytics that alters or overwrites an existing data object

Analytic Layer: location of analytic activity corresponding to the Data-to-Decisions pattern recognition layer and the storage of interim results and provenance

Analytic Machine: a machine, physical or virtual, that performs analytic pattern recognition on data; these can be, for instance, cloud virtual machines or standalone services

Architectural Model: the conceptual design for an infrastructure supporting analytic workflows for multiple environments, including cloud environments

C^2 Service: abbreviation for command and control service

Command and Control Service: service that bridges executive orchestration and the analytic processes that potentially cross multiple environments, including clouds and standalone services

D2D: abbreviation for Data-to-Decisions

Data Bus: storage for ingested data that can be pulled or pushed into the analytic layer

Data Preparation: a layer of the research stack that encompasses the identification, acquisition, and ingest of data relevant to a domain of customer interest

Data: the input for threads and projects operating at the analytic layer

Data-to-Decisions Research Stack: a specific, but representative, model for thread development consisting of four layers which (in consecutive order) are 1) data preparation, 2) pattern recognition, 3) visualization, and 4) decision-making

Decision-Making: a layer of the research stack that encompasses the user-orchestrated process that leverages depicted patterns to take informed and improved actions

Domain: the space of interest and responsibility unified by customer and mission goals, where some examples include command and control, intelligence, finance, and health care

Executive: describes and coordinates workflows of analytics in and across analytic machines in the analytic layer and reacts to or reports when workflow instances are unsuccessful

Knowledge Store: storage for results of the analytic layer that will be visualized

Pattern Recognition: a layer of the research stack that encompasses the isolation and identification of salient and useful patterns in data to be captured in results

Pipeline Flow: a workflow of analytics that summarize data and extract new results

Process Instance: a specific execution of an analytic that takes in a specific input and produces a specific result

Process: an individual analytic step in a workflow

Project: a research endeavor that can be associated with other projects in a thread and which must address one or more of the research stack layers

Provenance: retained description of the activity of analytic processes and their production of results

Research Stack: a general model for data-driven analytic activity consisting of four layers which (in consecutive order) are 1) data preparation, 2) pattern recognition, 3) visualization, and 4) decision-making

Researcher: an individual who develop solutions and applications for domain-specific problems at the project level

Results: the interim and final output of projects operating at the analytic layer, a subset of which will be elevated to knowledge, stored longer term, and visualized

Standalone Service: a physical or virtual machine independent of other analytic machines that serves a specific analytic purpose

Thread Leader: an individual who is a subject matter expert and defines or identifies use-cases relevant to a domain

Thread: a collection of interrelated research projects that target a domain of interest to a customer and which collectively address each layer of the research stack

User: an individual who examines visualized results to inform downstream actions

Visualization: a layer of the research stack that encompasses the depiction of patterns and results in a representation that can demonstrate their relevance in an operational context

Workflow Instance: a specific execution of a workflow of analytics

Workflow: the sequence and flow of analytics specified by the user which can be deployed across the analytic layer and will execute on data and interim results to produce results

11 Works Cited

[1] I. Alintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher and S. Mock, "Kepler: An Extensible System for Design and Execution of Scientific Workflows," in *Proceedings of the 16th International Conference on Scientific and Statistical Database Management*, 2004.

[2] R. Winder, J. Jubinski, J. Prange and N. Giles, "MOSAIC: A Cohesive Method for Orchestrating Discrete Analytics in a Distributed Model," *Natural Language Processing and Information Systems,* pp. 260-265, 2013.

[3] J. Clarke, V. Srikumar, M. Sammons and D. Roth, "An NLP Curator (or: How I Learned to Stop Worrying and Love NLP Pipelines)," in *LREC*, 2012.

[4] R. L. Grossman, "What is analytic infrastructure and why should you care?," *ACM SIGKDD Explorations Newsletter,* vol. 11, no. 1, pp. 5-9, 2009.

[5] The MITRE Corporation, "Data to Decisions," The MITRE Corporation, [Online]. Available: http://www.mitre.org/research/mission-focused-research/data-to-decisions. [Accessed 20 March 2015].

[6] T. White, Hadoop: the Definitive Guide, Sebastopol, CA: O'Reilly Media, Inc., 2011.

[7] K. Shvachko, H. Kuang, S. Radia and R. Chansler, "The Hadoop Distributed File System," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium*, 2010.

[8] VMware, "vSphere: Server Virtualization, Cloud Infrastructure," VMware, [Online]. Available: http://www.vmware.com/products/vsphere. [Accessed 20 March 2015].

[9] "Apache Flume," [Online]. Available: https://flume.apache.org/. [Accessed 2015 30 3].

[10] "Apache Sqoop," [Online]. Available: http://sqoop.apache.org/. [Accessed 2015 30 3].

[11] D. Laney, "3D Data Management: Controlling Volume, Velocity, and Variety," META Group / Gartner, 2001.

[12] A. McAfee and E. Brynjolfsson, "Big data: the management revolution," *Harvard Business Review,* vol. 90, pp. 60-66, 2012.

[13] R. Birke, M. Bjoerkqvist, L. Y. Chen, E. Smirni and T. Engbersen, "(Big) data in a virtualized world: volume, velocity, and variety in enterprise datacenters," in *12th USENIX Conference on File and Storage Technologies*, Santa Clara, CA, 2014.

[14] Y. Demchenko, P. Grosso, C. de Laat and P. Membrey, "Addressing Big Data Issues in Scientific Data Infrastructure," in *International Conference on Collaboration Technologies and Systems*, San Diego, CA, 2013.

[15] R. Winder, N. Giles and J. Jubinski, "Implementation Recommendations for MOSAIC: A Workflow Architecture for Analytic Enrichment.," 2010.

[16] D. B. Lenat, "CYC: A Large-Scale Investment in Knowledge Infrastructure," *Communications of the ACM,* vol. 38, no. 11, pp. 33-38, 1995.

[17] C. Matuszek, J. Cabral, M. J. Witbrock and J. DeOliveira, "An Indroduction to the Syntax and Content of Cyc"," in *AAAI Spring Symposium: Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering*, 2006.

[18] I. Niles and A. Pease, "Towards a standard upper ontology," in *Proceedings of the International Conference on Formal Ontology in Information Systems,* 2001.

[19] R. V. Guha, "Contexts: a Formalization and Some Applications," Stanford University, Stanford, CA, 1991.

[20] B. C. Grau and O. Kutz, "Modular Ontology Languages Revisited," in *Proceedings of the Workshop on Semantic Web for Collaborative Knowledge Acquisition*, 2007.

[21] C. Parent, S. Spaccapietra and E. Zimanyi, "Modularity in Databases," in *Modular Ontologies*, Berlin, Springer-Verlag, 2009, pp. 113-153.

[22] B. C. Grau, B. Parsia and E. Sirin, "Working with Multiple Ontologies on the Semantic Web," in *The Semantic Web-ISWC 2004,* Berlin, Springer Berlin Heidelberg, 2004, pp. 620-634.

[23] Q. Wang and H. Jin, "Data Leakage Mitigation for Discretionary Access Control in Collaboration Clouds," in *SACMAT'11*, Innsbruck, Austria, 2011.

[24] R. Agrawal and J. Kiernan, "Watermarking Relational Databases," in *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB '02),* Hong Kong, China, 2002.

[25] W. Mazurczyk and K. Szczypiorski, "Is Cloud Computing Steganography-proof?," in *Third International Conference on Multimedia Informaiton Networking and Security,* Shanghai, 2011.

[26] P. Papadimitriou and H. Garcia-Molina, "Data Leakage Detection," *IEEE Transactions on Knowledge and Data Engineering,* vol. 23, no. 1, pp. 51-63, 2011.

[27] M. D. Allen, A. Chapman, B. Blaustein and L. Seligman, "Capturing Provenance in the Wild," *Lecture Notes in Computer Science: Provenance and Annotaiton of Data and Processes,* vol. 6378, pp. 98-101, 2010.

[28] A. Chapman, B. Blaustein, L. Seligman and M. D. Allen, "PLUS: A Provenance Manager for Integrated Information," in *IEEE IRI 2011,* Las Vegas, Nevada, 2011.

[29] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, B. Plale, Y. Simmhan, E. Stephan and J. Van den Bussche, "The Open Provenance Model Core Specification (v1.1)," *Future Generation Computer Systems,* vol. 27, no. 6, pp. 743-756, 2011.

[30] L. Moreau and P. Missier, "PROV-DM: The PROV Data Model," World Wide Web Consortium, 2013.

[31] P. Cudre-Mauroux and e. al., "Viewpoints on Emergent Semantics," *Journal on Data Semantics VI, LNCS 4090,* pp. 1-27, 2006.

12 Appendix

12.1 Appendix A – Intelligence Reference Model

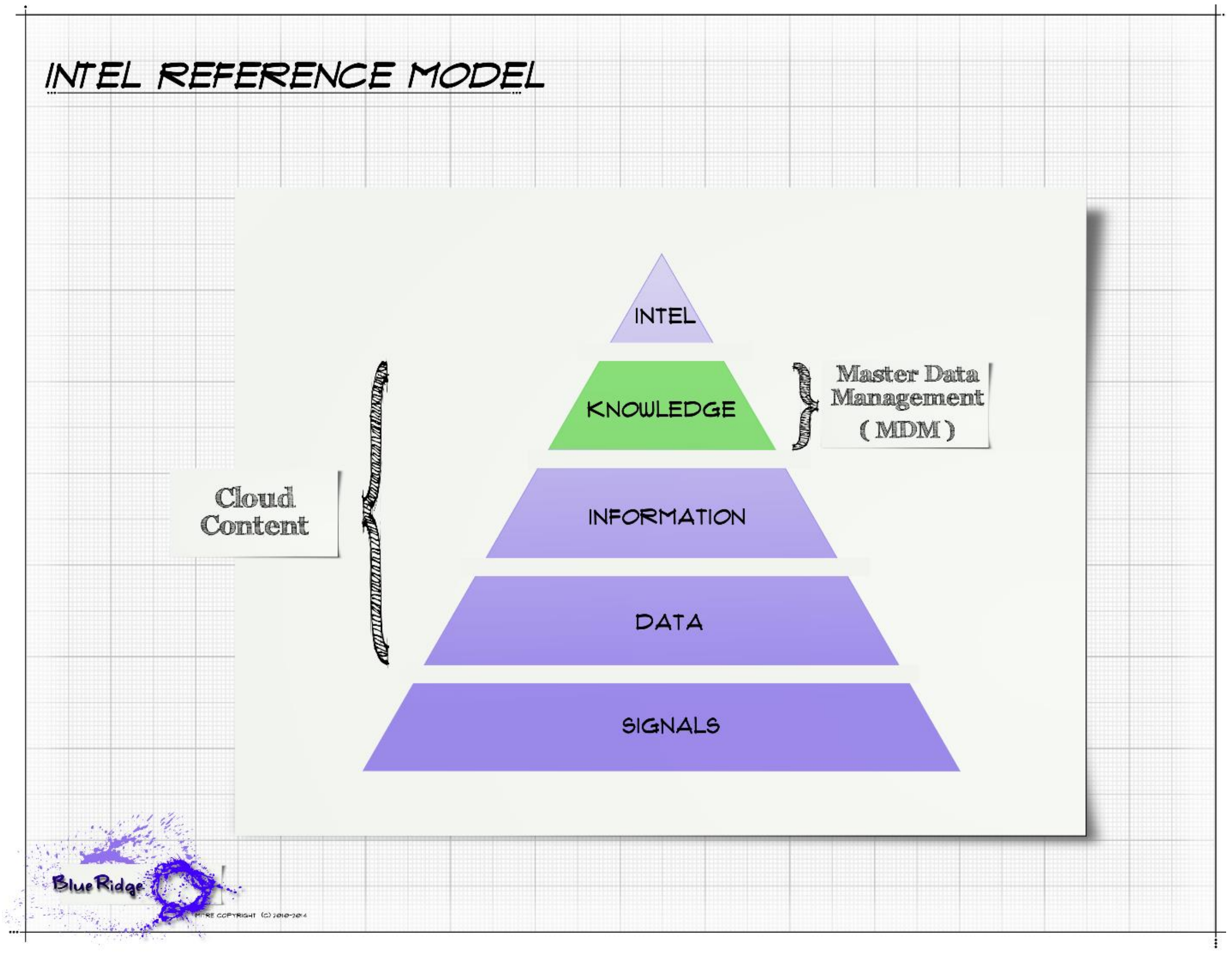


Figure 22. The intelligence reference model.

Figure 22 depicts the classic intelligence reference model. This presents a high-level division of terminology into signals, data, information, knowledge and intelligence. The most raw, unprocessed form resides at the bottom. As one ascends the pyramid, signals are refined into data, which undergo pattern recognition and automated analysis to become information and knowledge. Ultimately intelligence—worthy of guiding actions—reside at the top. The terminology of data can be somewhat ambiguous. In addition to the role it serves in the reference model, other disciplines such as cloud computing and Master Data Management (MDM) use the term for specific places in this hierarchy.

12.2 Appendix B – REST Protocol Schemas

The following is a schema in JSON format for REST protocol messages:

```
{
 "type":"object",
 "$schema": "http://json-schema.org/draft-03/schema",
"id": "http://jsonschema.net",
"required":false,
"properties":{
 "C2 Message": {
  "type":"object",
  "id": "http://jsonschema.net/C2 Message",
  "required":false,
  "properties":{
   "Job Properties": {
    "type":"object",
    "id": "http://jsonschema.net/C2 Message/Job Properties",
    "required":false,
    "properties":{
     "ID": {
      "type":"string",
      "id":
       "http://jsonschema.net/C2 Message/Job Properties/ID",
      "default": "Job-123456",
      "required":false
     },
     "Name": {
      "type":"string",
      "id":
       "http://jsonschema.net/C2 Message/Job Properties/Name",
      "default": "MyName",
      "required":false
     },
     "Path": {
      "type":"string",
      "id":
       "http://jsonschema.net/C2 Message/Job Properties/Path",
      "default": "FullPath",
      "required":false
     }
    }
   },
   "Message Properties": {
    "type":"object",
    "id":
     "http://jsonschema.net/C2 Message/Message Properties",
    "required":false,
    "properties":{
     "Connected": {
      "type":"boolean",
      "id":
       "http://jsonschema.net/C2 Message/Message
        Properties/Connected",
    "default":false,
    "required":false
   },
   "Host ID": {
    "type":"string",
    "id":
     "http://jsonschema.net/C2 Message/Message
      Properties/Host ID",
    "default": "MyServer",
    "required":false
   },
   "Session ID": {
    "type":"string",
    "id":
     "http://jsonschema.net/C2 Message/Message
      Properties/Session ID",
    "default": "M-12345678",
    "required":false
   }
  }
 },
 "Message Type": {
  "type":"object",
  "id": "http://jsonschema.net/C2 Message/Message Type",
  "required":false,
  "properties":{
   "Command": {
    "type":"object",
     "id":
      "http://jsonschema.net/C2 Message/Message
       Type/Command",
    "required":false,
    "properties":{
     "Cancel": {
      "type":"number",
      "id":
       "http://jsonschema.net/C2 Message/Message
        Type/Command/Cancel",
      "default":2,
      "required":false
     },
     "Job Status": {
      "type":"number",
      "id":
       "http://jsonschema.net/C2 Message/Message
        Type/Command/Job Status",
      "default":3,
        "required":false
       },
       "Submit": {
        "type":"number",
        "id":
         "http://jsonschema.net/C2 Message/Message
          Type/Command/Submit",
        "default":1,
        "required":false
       }
      }
     },
     "Session": {
      "type":"object",
      "id":
       "http://jsonschema.net/C2 Message/Message Type/Session",
      "required":false,
      "properties":{
       "Command Request": {
        "type":"number",
        "id":
         "http://jsonschema.net/C2 Message/Message
          Type/Session/Command Request",
         "default":3,
         "required":false
       },
       "Connect": {
        "type":"number",
        "id":
         "http://jsonschema.net/C2 Message/Message
          Type/Session/Connect",
        "default":1,
        "required":false
       },
       "Disconnect": {
        "type":"number",
        "id":
         "http://jsonschema.net/C2 Message/Message
          Type/Session/Disconnect",
        "default":2,
        "required":false
       }
      }
     }
    }
   }
  }
```

```
   }
  }
 }
```

The following is a JSON format schema for the acknowledgement to the REST protocol acknowledgement.

```
 {
  "C2 Ack": {
   "Message Type": {
    "Session": {
     "Connect": 1,
     "Disconnect": 2,
     "Command Request": 3
    },
    "Command": {
     "Submit": 1,
     "Cancel": 2,
     "Job Status": 3
    }
   },
   "Message Properties": {
    "Host ID": "MyServer",
    "Session ID": "M-12345678",
    "Connected": false
   },
   "Job Properties": {
    "Name": "MyName",
    "Path": "FullPath",
    "ID": "Job-123456"
   },
   "Job Status": {
    "State": {
     "Running": 1,
     "Completed": 2,
     "Canceled": 3,
     "Erroring Out": 4,
     "Retired": 5
    },
    "percentage complete": 0,
    "Message": "No Message Given"
   },
   "Ack Return": {
    "Status": {
     "OK": 0,
     "Error": 1
    },
    "Message": "No Message Given"
   }
  }
}
```