Kmesh.io

Multi-Cloud Filesystem and Data Orchestration as a Service

TECHNICAL WHITEPAPER

Table of Contents

Table of Figures

1 The Need for Multi-Cloud Data Orchestration

Many apps and services cannot be deployed on public clouds, because the data they rely on is not portable or accessible in real time. Typically, the data that supports cloud apps and services resides in different locations, whether they are on-premise or cloud-based data sources.

Organizations need to manage and make available data from these multiple sources to applications, a practice we refer to as 'Data Orchestration.' Only through robust data orchestration can they support today's modern workloads that drive competitive advantage – cloud-based DevOps, IoT, machine learning, and high-performance computing.

There are two additional strategic benefits to robust data orchestration. First, it yields app deployment flexibility. Second, it enables a holistic cloud strategy that combines private data centers and public clouds to optimize capabilities and economies of scale.

2 Requirements for Multi-Cloud Data Orchestration

Private and public clouds differ when it comes to architecture, scale, infrastructure availability, storage options, data transfer capabilities, and numerous other operational factors. To make numerous types of data available to apps across such a varied environment, IT organizations require a new method of data management and orchestration that is both simple, flexible and secured.

As such, the following are 'Must Have' features of a multi-cloud data orchestration solution:

1. A simple way of creating data synchronization and access across clouds and cloud regions
2. Easy access to data, without requirements for additional transformation and copying
3. Common data formats across all clouds
4. A design that affords cloud-efficient data transfer; optimizes for cloud-based computing, networking and storage; and provides capabilities to support cloud-friendly use cases, such as a running DevOps, ML/Analytics/HPC on cloud.
5. Ability to handle link failures between clouds and cloud regions
6. Data transfer and access that are scalable and secure

3 Introducing Kmesh

Kmesh Data orchestration service enables organizations to manage their data across cloud, hybrid cloud, and multi-cloud deployments in real-time without the need for costly tools, manual scripting and engineering effort.

Kmesh transforms an enterprise's centralized data into distributed ponds, which cloud exist and operate over multiple clouds, countries, and edges as a single global namespace.

At its core, Kmesh is a data mobility and access service for connecting apps to the data that drives them. Kmesh brings the data to the apps' locations which enables organizations to globalize, organize, share, and synchronize application data. Through this service, customers can deploy robust and efficient data orchestration for a broad set of use cases associated with Hybrid Cloud/Multi-Cloud, Edge Computing, and Data Localization/Data Sovereignty.

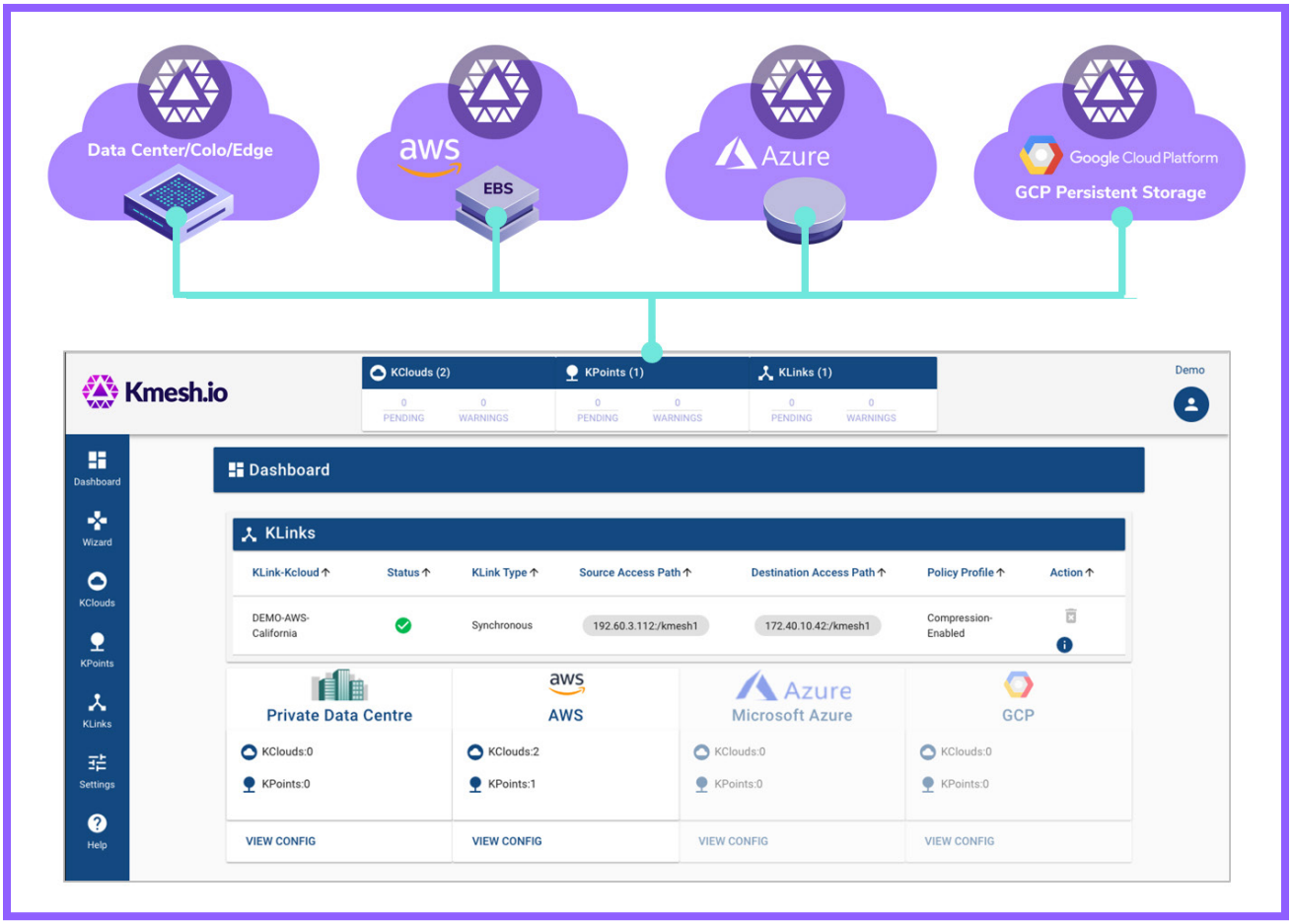

Figure 1: Kmesh Data Orchestration Service Capabilities & Portal

Kmesh delivers the four core benefits for managing applications and data in a Multi-Cloud model:

Multi-Cloud Data Mobility

A simple data orchestration that enables data mobility across different clouds for the purpose of cloud bursting, application migration and mobility.

- Data can be synchronized between Kmesh instances at different locations.
- Data can move between either enterprise data center or public clouds or between cloud regions, e.g., AWS N.California and AWS Ohio or between public clouds, e.g., AWS Oregon and Azure Germany North.

- Data is directly synchronized from Kmesh to any application that is requesting data — with no intermediate "data staging"[1] along the way.
- Efficient and secured data synchronization[2] capabilities include efficient change tracking, in-flight compression, in-flight deduplication and in-flight encryption.

File as a Service

POSIX-Compliant data access across different clouds to standardize on data access.

- Access to data occurs via a POSIX filesystem that is built on top Lustre[3], that is available in 'File-as-a-service'. The remote data is presented as though it is local to the application.
- No need to change your applications when moved from cloud to cloud to fit to the data semantics of each cloud
- Filesystem performance is tuned — pick the right cloud instances, storage and filesystem-level parameters — based on the policies and requirement as specified by the user.
- Kmesh supports a variety of application deployment models such as bare-metal, virtualized, cloud instantiated, and containerized.

Multi-Cloud Data Management

Enterprise data management capabilities to manage data from all the clouds globally.

- Provides key data protection capabilities like Multi-Cloud Snapshots, unified file access management, secure data access and control & audit logs across multiple clouds and regions, including automated security & compliance
- A Multi-Cloud data namespace to bring visibility to entire data sources/targets across om-premises and public clouds.
- The entire solution is highly available both from control and data plane.

Simple Provisioning, Management and Integration

A multi-tenant SaaS provides secured enterprise portal for holistic management of all the data workflows.

- Build end-to-end data workflows for hybrid cloud app deployment, cloud-bursting and other use cases.
- Provision easily without the need to understand the filesystem technology.
- Perform service provisioning using the Kmesh CSI[4] plugin for Kubernetes managed-container deployments.
- Using a REST API, integrate Kmesh into existing enterprise-wide infrastructure, application and data management as well as orchestration systems.

1 - An inefficient practice to copy data, including filesystem/file data, temporarily into object storage and ingest it into cloud instances/ filesystems.

2 - Multiple patent-pending innovations in this area

3 - http://lustre.org : Most popular HPC filesystem used in majority of top supercomputers in the world.

4 - https://kubernetes-csi.github.io/docs/

4 How Kmesh Works

Kmesh is a SaaS platform which users access via the Kmesh portal (portal.kmesh.io). Users are guided by simple and intuitive steps to manage the entire multi-cloud workflow for application data, regardless of the volume or complexity of workflows.

Complex workflows are greatly simplified in Kmesh: for example, user wants an application running on AWS to pull data from an on-premise database, consume the data into the app on AWS, and write results to Azure cloud. Modeling this workflow in Kmesh is just a matter of creating a Kmesh Dataflow between the on-premises and cloud region through a few clicks.

Figure 2 illustrates the provisioning and orchestration of the Kmesh service.

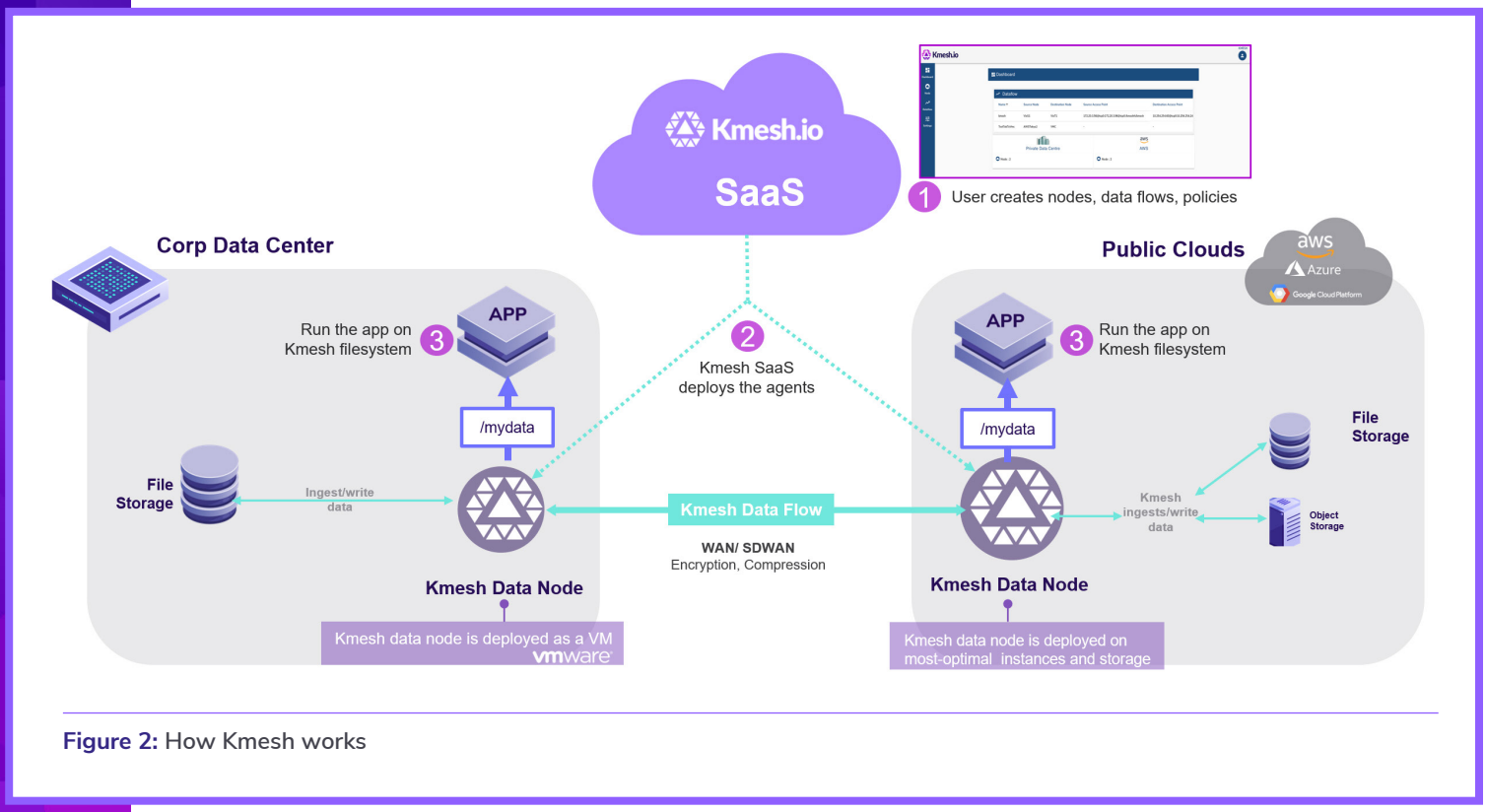

Figure 2: How Kmesh works

4.1 The Two-Step Kmesh Provisioning Process

Provisioning data in Kmesh involves just two steps in the Kmesh portal (portal.kmesh.io), regardless of the single cloud, hybrid cloud, or multi-cloud application scenario a user is deploying.

Note: Prior to Kmesh provisioning, users will need to follow onboarding steps. Please refer to section 7 of this paper, entitled “Kmesh Onboarding.”

Step 1: User creates a Kmesh Instance in each of the clouds involved in a use case:

» User creates Kmesh service instances using *Kmesh Data Nodes* within each cloud or cloud region that requires data access and synchronization (e.g. VMware private cloud, AWS West region, AWS East region, etc.).

» User configures cloud credentials required by Kmesh to instantiate Kmesh nodes on each of the clouds.

Step 2: User configures data workflows for one or more use cases:

» Using one or more *Kmesh Data Flows*, user configures the data access and synchronization policies which are dictated by an application deployment and its associated availability requirements

4.2 Kmesh Service Instantiation

Based on the provisioning inputs, Kmesh orchestrator automatically performs the following functions:

1. Instantiates required Kmesh Data Nodes within on-premises and cloud regions (at all involved clouds/cloud regions). Instances contain both the data orchestration engine for data synchronization and the filesystem for data access.
2. Creates the connectivity to existing data sources across all infrastructure.
3. Ingests the required data (as metadata references, not actual data) from the data sources.
4. Sets up all the connections for real-time data synchronization between all data sources and destinations (apps and services).
5. Creates access points (mount points) in each of the required clouds.

4.3 Application Deployment

At this point, all necessary plumbing is done and the service is ready to use.

1. User deploys application(s) on Kmesh within relevant clouds using the access point.
2. At this point, application is able to access and synchronize the data per the defined use case.

5 Kmesh Technology

Fortune 2000 enterprises are starting to look at Multi-Cloud architectures for many reasons, including proximity to customers, data sensitivity, data localization, and cost efficiency. To meet the application demands for data in multi-cloud environments, Kmesh has developed a globally distributed data service that is purpose-built to make data accessible to applications across multiple clouds. No longer limited by data concerns, enterprises can leverage the Kmesh technology to deploy apps anywhere and to do more in the cloud.

While multi-cloud data synchronization and access capabilities form the core of the Kmesh technology, the Kmesh solution addresses numerous other important aspects of multi-cloud data management and synchronization. These include, but are not limited to, infrastructure variations, cloud infrastructure costs, cloud availability models (I.e. availability zones and regions), and inter-cloud latency. This ensures consistent service behavior based on configurations established by the user.

5.1 Globally Distributed Data Service

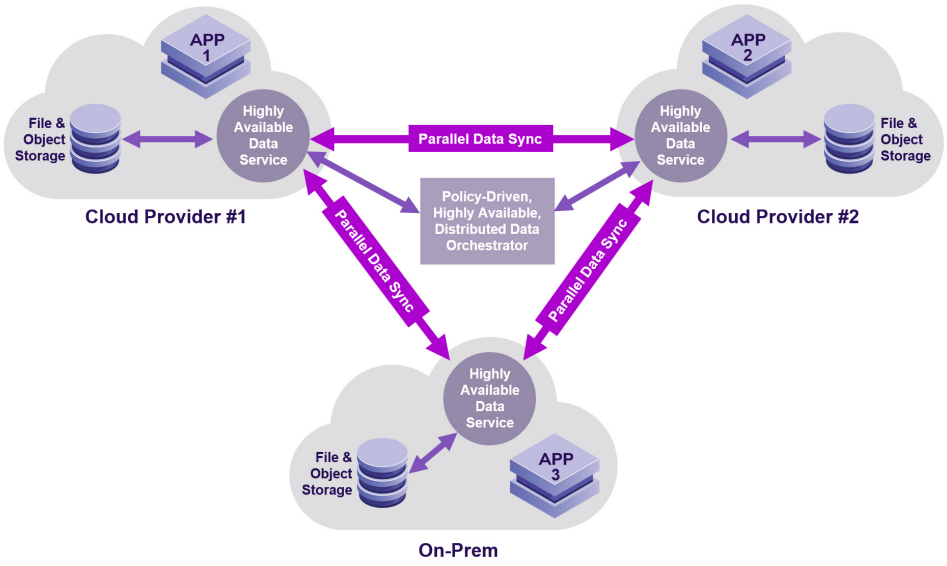

Figure 3: Globally Distributed Data Orchestration

The diagram (left) depicts the Kmesh globally-distributed data service as it operates in a multi-cloud environment. This is the foundation of the entire Kmesh service. The Kmesh technology establishes a loosely-coupled, global namespace and performs policy-driven data orchestration based on the policies applied to that namespace. A notable differentiator from traditional data services is that one portion of the Kmesh data service at each cloud/region can decouple from the global data service, yet at the same time, continue to perform application IO operations on selected parts of the namespace.

This decoupling is the basis for describing the architecture as a "loosely-coupled, global namespace." Referencing Figure 3 (above), the data service in **Cloud Provider #1** (a regional instance) can separate from the **On-Prem** and **Cloud Provider #2** regions (both regional instances) and continue to provide select data services. One advantage to such a design is

that specific applications can continue operating during network connection failures between regions. Policy-driven data orchestration can synchronize and strategically locate data or data copies in a way that minimizes the impact of weaknesses in multi-region/multi-cloud topologies and to take advantage of lower-latency local IO.

Finally, the cloud-agnostic nature of the Kmesh technology means that all Kmesh distributed data services operate the same way, even if the architecture involves multiple different cloud vendors, such as AWS, Azure, GCP and Alibaba. This means a single Kmesh data service can span multiple clouds as well as multiple physical regions of the world.

5.2 Core Technology Components

Kmesh's globally distributed data service is made up of two key elements:

1. **Kmesh Orchestrator with Policy Engine SaaS.** The centralized, multi-tenant SaaS engine and portal help customers to provision, deploy, manage and monitor the Kmesh service.
2. **Cloud-specific Data Nodes.** Deployed on customer's infrastructure within various clouds /cloud regions, nodes can be deployed on VMware, bare metal or openstack-based, on-premises clouds and public clouds like AWS, Azure and GCP.

5.2.1 Kmesh SaaS Orchestrator & Policy Engine

Kmesh Orchestrator and Policy Engine is the core of the Kmesh service. It runs on Kmesh infrastructure and is made up of two major components: the Kmesh Portal and the Kmesh Central Orchestrator.

5.2.1.1 The Kmesh Portal

Users leverage the Kmesh portal to establish data orchestration policies as well as create, configure and manage Kmesh service. The Kmesh Portal allows for simple data orchestration policies that can cater to a wide variety of use cases.

The portal displays the status of the service, from end to end, so that users can continuously monitor their multi-cloud data infrastructure. As requirements change, users can easily modify policies in Kmesh to optimize data usage, minimize costs and comply with evolving enterprise requirements.

5.2.1.2 Kmesh Central Orchestrator

Taking cues from user policies established via the Kmesh Portal, the Central Orchestrator references configurations and automatically installs the necessary data services components on the right infrastructure elements. Installation of data components is done in a way that ensures their high availability.

5.2.1.3 SaaS capabilities and features

5.2.1.3.1 High Availability

Kmesh SaaS stores all configurations and status in a highly available fashion. The configuration and status are replicated across cloud availability zones (AZs) and regions to ensure AZ and region failures do not interrupt service availability.

5.2.1.3.2 Monitoring

Kmesh SaaS pro-actively monitors the state of Kmesh components across different clouds and warns users when they are running low on provisioned storage capacity. Kmesh proactively monitors several different parameters to keep users aware of Kmesh service and the infrastructure. For example, a 'No Activity' alert is sent to users if the service is being used by any application but just consuming infrastructure.

5.2.1.3.3 Multi-tenancy

Kmesh is purpose-built to enable multi-tenancy. Isolations are built for provisioning/ management, dashboards, reporting and billing using the model shown in Figure 4 (below).

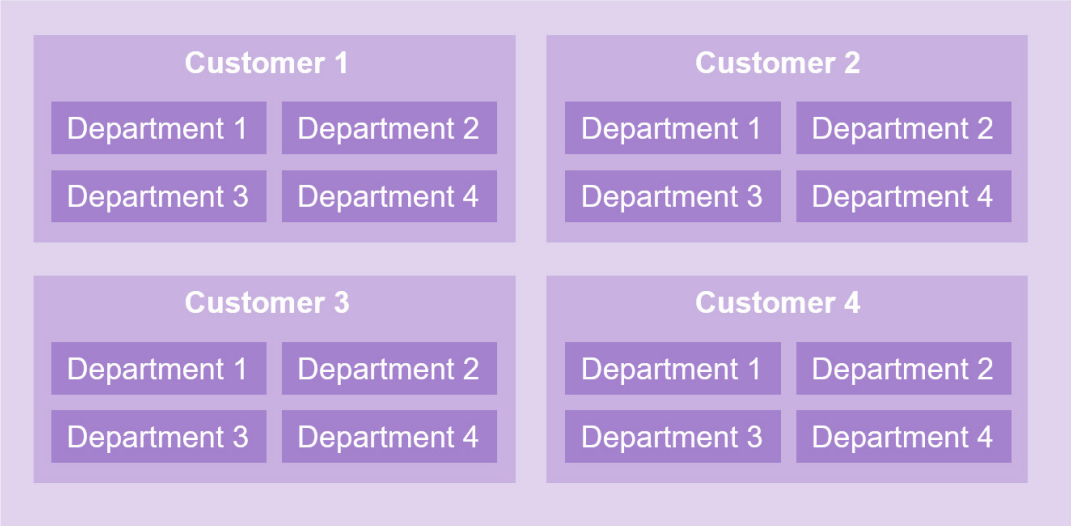

Figure 4: Kmesh SaaS Multi-Tenancy Model

5.2.1.3.4 Multi-Cloud Uniform File Access Policy

Kmesh provides capabilities to enforce uniform file access policy across multiple clouds.

1. Enforcement of identity and access control:
    a. Integration with existing identity and access management (IAM) systems
    b. Integration with cloud-based IAM (AWS & Azure)
    c. Audit Logs from native logging facilities, such as AWS CloudTrail/CloudWatch

5.2.1.4 SaaS Architecture

Customer experience and security drive the architecture of the SaaS platform. The architecture is built around simplicity and scalability. The Kmesh SaaS needs a set of VMs/instances on each cloud per the policy defined by the admin.

Here are interesting aspects of this architecture:

- All of the provisioning and orchestration is automated for public clouds and VMware without requiring action from the customer, except with cases involving unsupported clouds/infrastructure or bare-metal infrastructure.
- No public IP address is required for managing the Kmesh software on the customer's environment.
- No special network requirements
- No need for opening of non-standard ports (firewall holes) on customer infrastructure.
- Most of Kmesh SaaS intelligence is contained within the Kmesh SaaS component

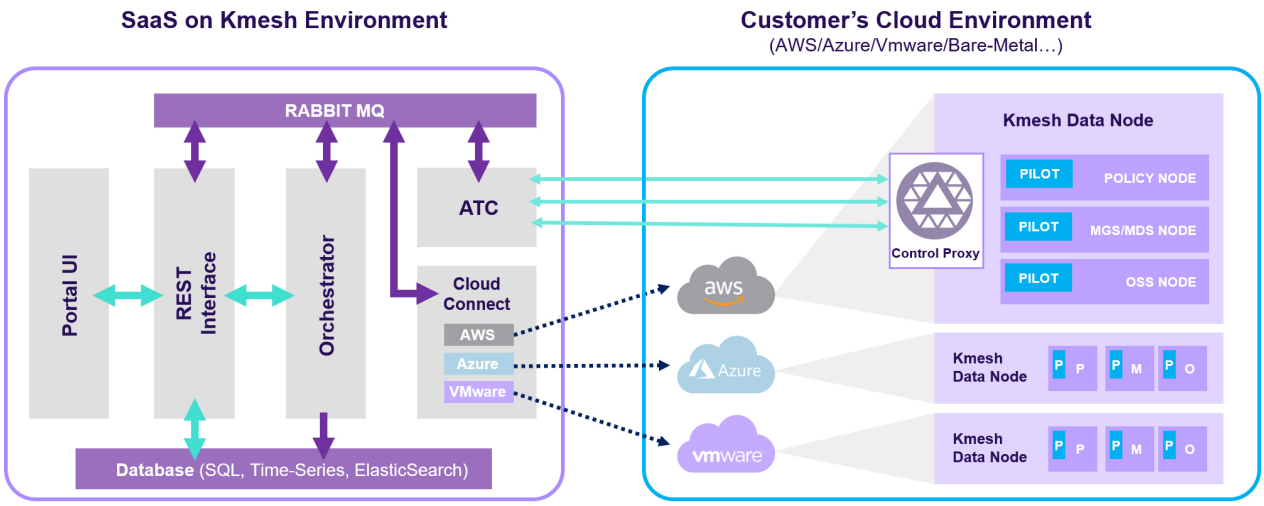

Figure 5: SaaS Architecture

5.2.1.4.1 Components

5.2.1.4.1.1 Pilot

Small Kmesh agent that runs each of the VMs/instances

- Communicates with Air Traffic Control (ATC) over HTTPS
- ATC asks the pilot questions and instructs the pilot to do things:
  - » What kernel is running?
  - » Download an rpm located at X and install it.
  - » Run this lfs command.
  - » Send vsphere api call to create a VM.

5.2.1.4.1.2 Air Traffic Controller (ATC)

Instructs Pilots to perform specialized tasks.

- Runs on our infrastructure at a stable URL (https://atc.kmesh.io).
- Accepts incoming connections from pilots and validates pilot credentials.
- Listens on RabbitMQ message queue for any commands from Orchestrators.
- Pushes responses from pilots to RabbitMQ messaging queue.

5.2.1.4.1.3 Cloud Connectors

Cloud-Specific connectors that use cloud API to perform resource management (VM creation, bring down, etc.):

- For example, AWS connector talks to AWS & Rabbit, Azure-connector talks to Azure & Rabbit.
- Cloud connectors take actions and respond to inquiries:
  - » Create a VM
  - » List VPCs in each region

5.2.1.4.1.4 Orchestrator

Orchestrator is the heart of the SaaS. It is a glue between the user (portal or Rest API) and Kmesh SaaS.

- Translates the user policies and configuration into actionable tasks for ATC/Pilot and other components.
- Carries out all workflows associated with any user-driven changes or pro-active actions by SaaS.

Note: The portal or any orchestrator components DO NOT have access to customer data since Kmesh SaaS does not store any customer data. It only maintains configuration, stats and statistics information for the Kmesh service.

5.2.2 Kmesh Data Node

Kmesh Data Node is the Kmesh software agent automatically deployed on customer infrastructure. It stores the data and carries out the data services in conjunction with the Kmesh SaaS. Users can create more than one node per cloud. A single instance of Kmesh Data Node can consist of one or both of these functionalities.

1. ***Rule-based Data Orchestration Engine:*** An engine for synchronizing the data across clouds, cloud regions and edges based on user-defined rules.
2. ***Data Mesh Filesystem:*** A POSIX filesystem that provides simple access to the data synchronized.

Figure 6 below illustrates how the data node — consisting of filesystem and orchestration engine — is deployed in a two-cloud scenario.

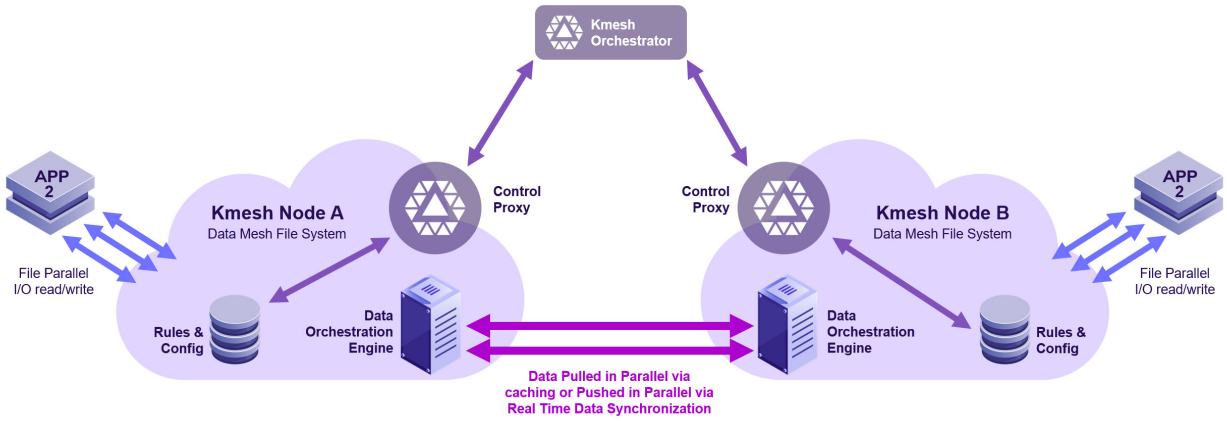

Figure 6: Illustration of Kmesh deployment across 2 clouds

1. User configures policies on Kmesh SaaS/portal
2. Following are examples of policies and data synchronization rules for this specific two-cloud setup:
    a. App1 in Cloud A is consuming data from Node A.
    b. Synchronize data from Cloud A and Cloud B in real-time.
    c. Optionally, ingest existing data from NFS in Cloud A onc or at regular intervals.
    d. App2 consumes the data synchronized on Cloud B and creates new data on Cloud B.
    e. This new data will need to be synchronized back to Cloud A.
3. The synchronization rules are defined using Kmesh Data Flows.
4. Kmesh SaaS deploys the Kmesh node(s) on each cloud/cloud region based on the configuration details, such as number of nodes, Data Node Types, source and destination clouds/regions & data orchestration policies.
5. Data Nodes are deployed as cloud VMs or instances. Kmesh SaaS picks the right cloud VM/instance based on user policy and requirements.
6. Policies are converted to rules and configuration information, and relevant data is pushed to corresponding Kmesh Nodes.
7. The rules and configuration information at the Data Nodes is used to program and configure the Data Orchestration engine & Data Mesh filesystem, with specific instructions for how to treat existing and new data in each Kmesh Node.
8. Data synchronization rules are carried out by Kmesh's orchestration engine and patented data synchronization functionality.

5.2.2.1 Data Node: Types & Sizes

A Kmesh Node can support multiple applications and simultaneous data synchronizations across clouds. Alternatively, users can create multiple nodes to manage different applications or data synchronizations.

Additionally, the Data node provides the following capabilities:

1. Node Types
    - » Users can create different types and sizes of nodes depending on the application requirements (storage capacity, performance, etc.) and cost of running the nodes on clouds.
2. Scale-out Architecture
    - » A Data Node is built on a scale-out model. A node has the smallest configuration (Figure 7: Kmesh Data Node Types) for each node type. Smallest configuration specifies the capacity, performance and cost parameters for each Data Node type.
    - » A node can be manually scaled up and down based on the node's capacity and performance utilization. Metadata performance or data performance can be scaled up together or independently based on application requirements.

Figure 7: MDS: Metadata Server OSS: Object Storage Service.

3. Auto-scaling:
    - » Optionally, a Data Node can be auto-scaled based on certain criteria. If the capacity or performance is 75% utilized, scale up by one smallest configuration.
4. At-rest compression, encryption & deduplication:
    - » The node supports at-rest compression, encryption and deduplication.
5. HA and Non-HA mode:
    - » HA adds additional underlying VM/instance availability to a node. If an instance/VM were to fail, the node can still serve IOs to an application. By default, the nodes are in non-HA mode. Nodes can be configured in HA node if required.

6. Data Tiering:

» Data tiering reduces total storage cost by using cheaper storage while optimizing performance.

» Tiering the Node data to cheaper tier S3 or Glacier (AWS) and equivalent, cheaper storage.

» The non-hot data is automatically moved to the cheaper tier.

» When the data is needed by the application, the data is read from the cheaper tier into fast tier.

» The data movement is controlled by "Tiering percentage" which is set by the user. This is the portion of the capacity that will come from the cheaper tier.

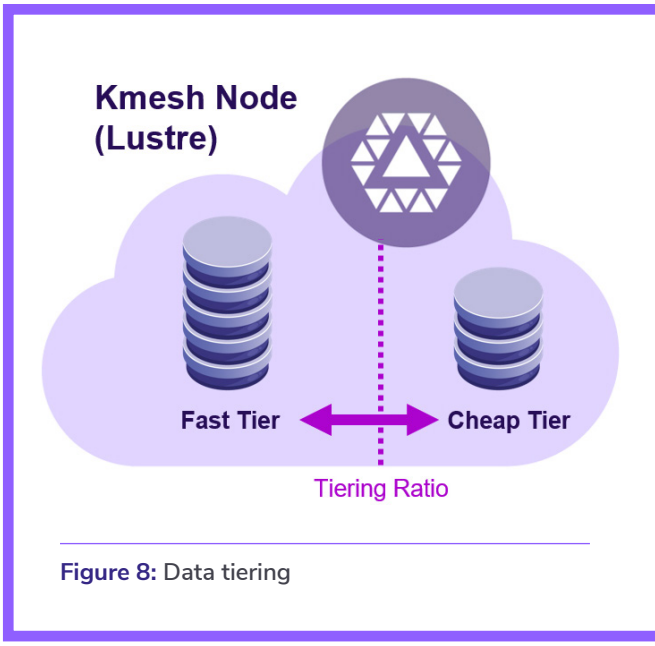

Figure 8: Data tiering

Figure 9 shows different Node types and relevant information presented to users during the provisioning process. This way, users can select the right node type. It also helps to create the right size of the node using the capacity increments and performance details while obtaining the most cost-optimized selection. Capacity increments define the smallest unit of the scale-out architecture.

5.2.2.2 Data Mesh Filesystem

Kmesh Data Mesh Filesystem (DMFS) is a geographically distributed, POSIX-compliant filesystem capable of supporting a wide spectrum of applications with varying requirements

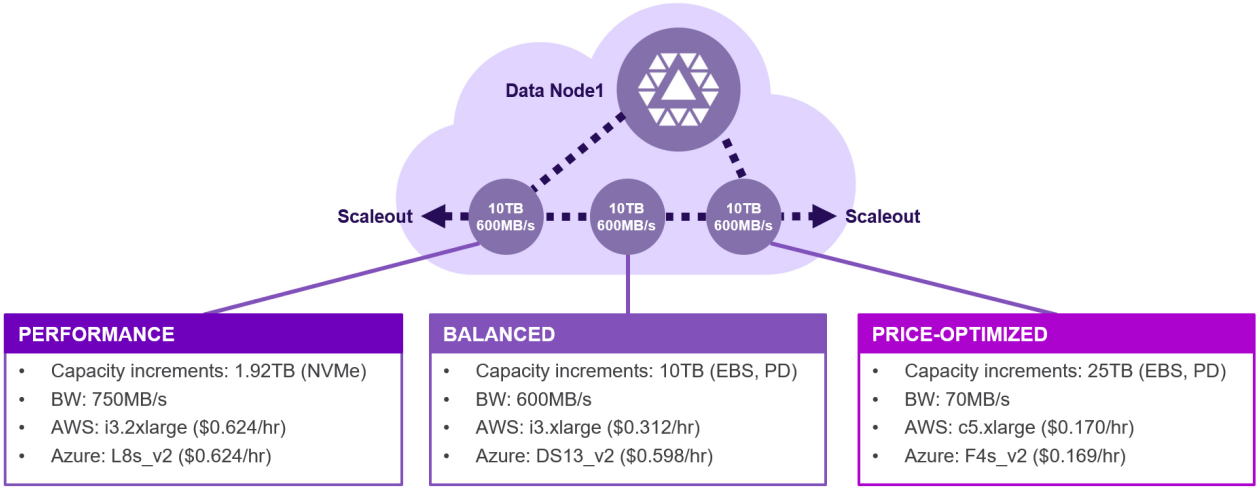

Figure 9: Kmesh Data Node types

for performance, availability and cost-efficiency. DMFS spans multiple clouds as a single namespace.

The core filesystem technology is built on top of Lustre, the most popular high-performance computing (HPC) filesystem. Following are key indicators of Lustre's importance to HPC which led Kmesh to develop DMFS using Lustre.

1. 3 of the top 5 supercomputers in the world use Lustre[5]
2. 60% of the top 100 supercomputers use Lustre[6]
3. Lustre has proven, industry-leading scale and performance
4. Top filesystem in IO 500 Node challenge[7] — 6 of the top 14 positions
5. Many US National Laboratories, universities, and enterprises have deployed Lustre in a highly scalable manner.
6. For example, Oak Ridge National Laboratories' supercomputer, Spider, runs Lustre with over 25,000 clients, over 10PB of storage, and total IO throughput of 240 GB/sec.[8]

Using Lustre, Kmesh has developed many patent-pending innovations, as show in Figure 10 (below).

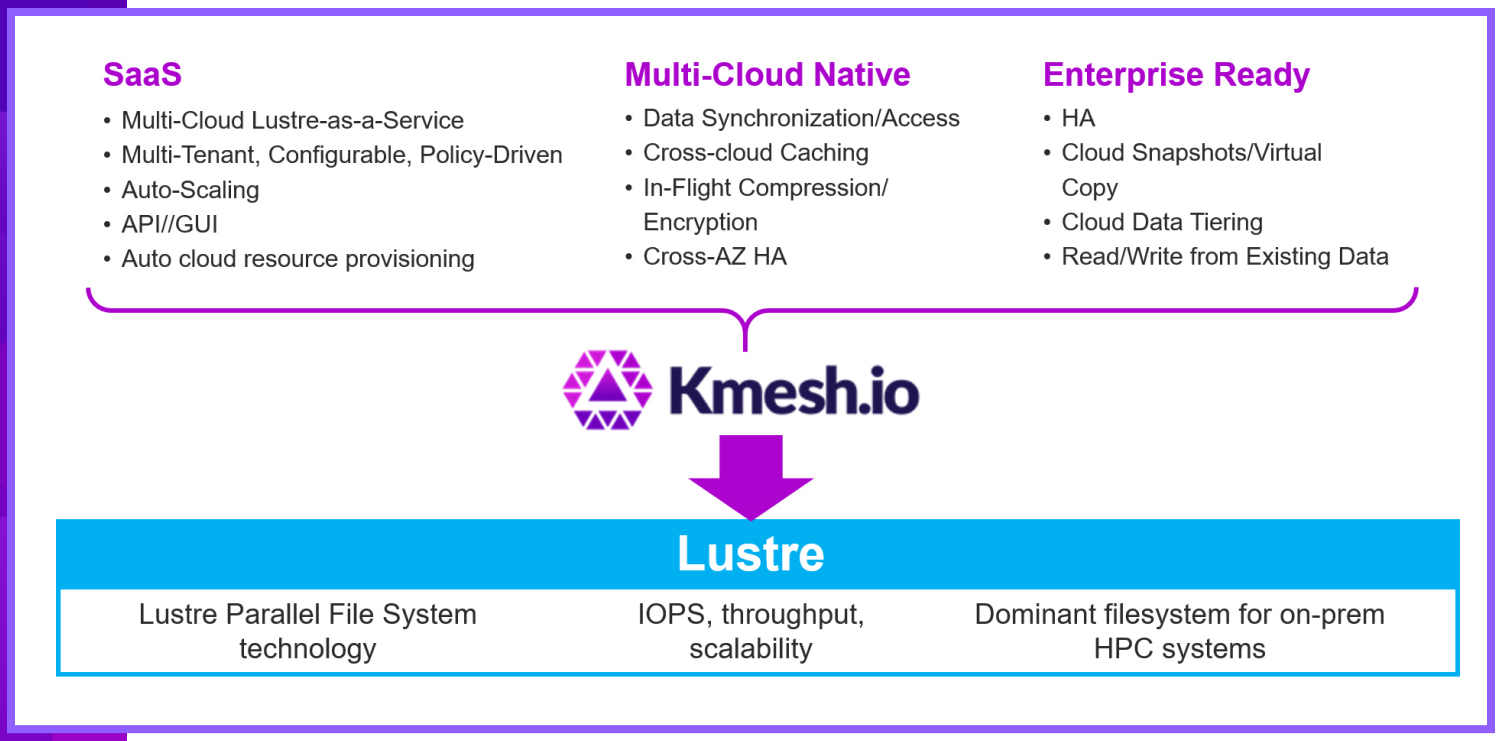

Figure 10: Kmesh innovations built on top of Lustre

5 - https://en.wikipedia.org/wiki/TOP500#Top_500_ranking & various resources
6 - https://wiki.whamcloud.com/display/PUB/Why+Use+Lustre
7 - https://www.vi4io.org/io500/list/18-11/10node
8 - https://www.hpcwire.com/2018/08/31/the-convergence-of-big-data-and-extreme-scale-hpc/

Here is a summary of the innovations we have built on top of Lustre (in words):

1. Numerous enterprise-focused functional capabilities like HA, Snapshots/Virtual Copies (clone) and Data Tiering.
2. Lustre works in a distributed environment over WAN, and thus, operates well in a multi-cloud environment.
3. Data orchestration capabilities that simplify multi-cloud data synchronization and access.
4. SaaS solution to deploy Lustre clusters on-demand in any cloud without need for manual installation.

The end result is a geographically distributed, high-performance filesystem (see Figure 11 below) with key capabilities to deploy applications in multi-cloud environments.

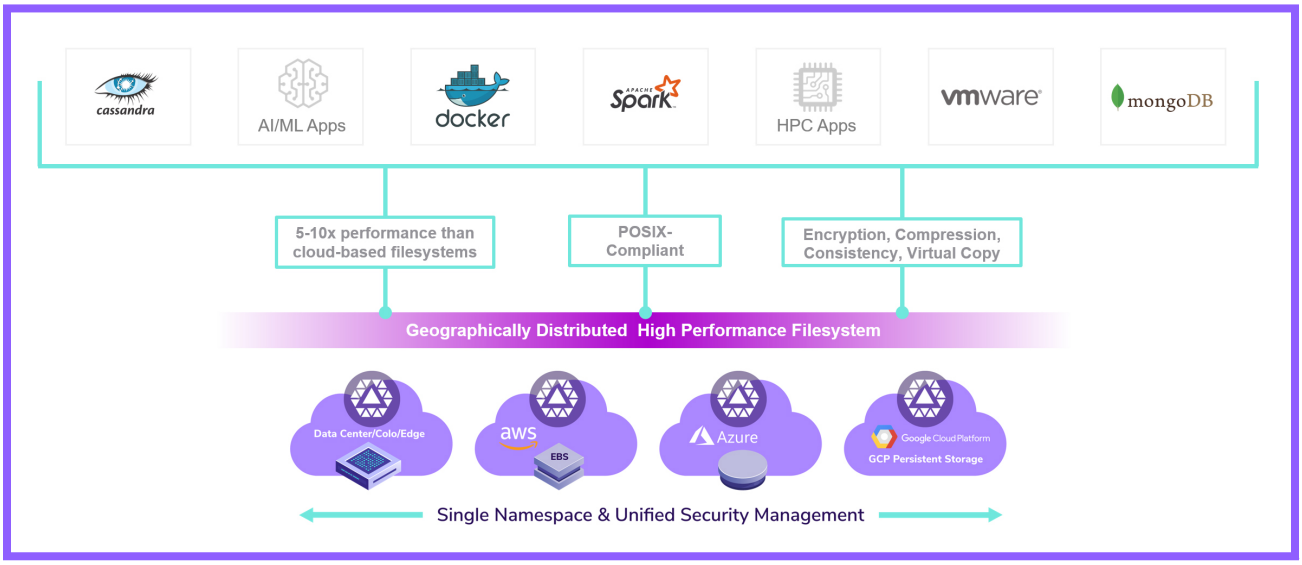

Figure 11: Geographically distributed, high-performance filesystem

5.2.2.2.1 Data Mesh Filesystem Features

5.2.2.2.1.1 POSIX-compliant

Lustre is POSIX-compliant. DMFS is based on Lustre and hence is POSIX-compliant, which means any application that is deployed on a POSIX-compliant Filesystem (NFS, EFS, ext4, etc.) can be deployed on Kmesh.

5.2.2.2.1.2 Single, Loosely-Coupled Global Namespace

The multiple distributed filesystem instances are presented as a single namespace. The exact namespace is presented based on the data synchronization policies set by the user. For example, certain subsets of data may be visible only on certain clouds or cloud regions according to the data orchestration policies. This way, users can control the synchronization and visibility of their enterprise data, globally.

5.2.2.2.1.3 Performance

Metadata performance

Instances	Slots	Directory Creation	Directory Stat	Directory Removal	File Creation	File Stat	File Read	File Removal	Tree Creation	Tree Removal	CPU Utilizaton
i3.xlarge	8	4,827.42	17,709.11	6,363.59	6,660.80	9,780.99	9,780.85	9,737.73	2,276.53	2,601.46	77.10%
i3.2xlarge	8	5,876.86	19,147.13	8,928.47	8,168.89	10,642.64	10,460.31	11,516.76	2,412.13	2,570.56	67.70%
M5d.4xlarge	8	9,861.89	25,202.48	12,248.51	11,300.76	13,988.44	13,963.48	15,866.89	3,431.61	3,821.88	32.80%
C5d.4xlarge	8	4,034.69	30,153.35	14,768.59	13,628.94	16,969.22	17,973.16	18,410.46	4,152.59	4,449.19	19.60%

```
Command Used: mpiexec <host> /usr/bin/mdtest -I 10 -i 2 -z5 -b 5 -i 3 -d /mnt/kmesh/mdtest5
```

Bandwidth Performance

Model	vCPU*	Mem (GiB)	Ephemeral	Ephemeral Storage	Available Storage (1 Ephemeral Used)	RecordSize	Slots	Read MB/s	Write MB/s
i3.xlarge	4	30.5	1 x 1.9 NVMe SSD	1945.6	590	Lustre=128K ZFS=128K	8	601.16	439.99
i3.2xlarge	8	61	1 x 1.9 NVMe SSD	1945.6	1228.8	Lustre=128K ZFS=128K	8	621.31	620.20
i3.4xlarge	16	53	1 x 1.9 NVMe SSD	3891.2	1228.8	Lustre=128K ZFS=128K	8	621.33	621.63
m5d.4xlarge	16	64	2 x 300 NVMe SSD	600	188	Lustre=128K ZFS=128K	8	621.16	301.65
c5d.4xlarge	16	32	1 x 400 NVMe SSD	400	254	Lustre=128K ZFS=128K	8	837.14	386.57
i3.xlarge	4	30.5	1 x 1.9 NVMe SSD	1945.6	590	Lustre=128K ZFS=128K	16	614.27	418.33
i3.2xlarge	8	61	1 x 1.9 NVMe SSD	1945.6	1228.8	Lustre=128K ZFS=128K	16	620.75	619.78
i3.4xlarge	16	53	1 x 1.9 NVMe SSD	3891.2	1228.8	Lustre=128K ZFS=128K	16	621.23	621.42
m5d.4xlarge	16	64	2 x 300 NVMe SSD	600	188	Lustre=128K ZFS=128K	16	568.29	281.43
c5d.4xlarge	16	32	1 x 400 NVMe SSD	400	254	Lustre=128K ZFS=128K	16	823.93	365.05
i3.xlarge	4	30.5	1 x 1.9 NVMe SSD	1945.6	590	Lustre=1MBK ZFS=128K	8	597.39	438.40
i3.2xlarge	8	61	1 x 1.9 NVMe SSD	1945.6	1228.8	Lustre=1MBK ZFS=128K	8	621.30	619.51
i3.4xlarge	16	53	1 x 1.9 NVMe SSD	3891.2	1228.8	Lustre=1MBK ZFS=128K	8	621.32	621.55
m5d.4xlarge	16	64	2 x 300 NVMe SSD	600	188	Lustre=1MBK ZFS=128K	8	621.05	302.83
c5d.4xlarge	16	32	1 x 400 NVMe SSD	400	254	Lustre=1MBK ZFS=128K	8	839.62	386/69

Test	Data Size	vs HDFS	vs AWS EFS
Tera Gen	100GB	1.98x	**26.57x**
Tera Sort	100GB	1.30x	**20.96x**
Tera Gen	400GB	3.32x	**28.99x**
Tera Sort	400GB	3.28x	**16.37x**
Tera Gen	800GB	3.42x	**28.21x**
Tera Sort	800GB	3.15x	**14.38x**

AWS, 32 Executors, Kmesh - 4 x C4.8xlarge w 3x200GB gp2, Spark - 4 x C4.8xlarge w 1x500BG st1

Spark performance

Kmesh filesystem can provide 5-10x performance improvements over cloud-native filesystems like EFS. The table on the left shows a performance comparison of Spark running on Kmesh, HDFS and EFS.

5.2.2.2.1.4 Availability

The filesystem is built using Kmesh Node. A Kmesh Node is a multi-node (VM or instance) entity that optionally contains data mirroring. Apart from that, users can enable cross-AZ & cross-region HA on the Kmesh Node to protect against AZ or region failures.

5.2.2.2.1.5 Data Efficiency

Data efficiency within and across cloud and cloud regions becomes a huge requirement because of economics of storage and moving data in and out of the cloud. Kmesh supports granular at-rest and in-flight compression as well as deduplication.

5.2.2.2.1.6 Encryption

Sensitive data needs to be safeguarded from end to end. Kmesh provides both granular at-rest and in-flight data encryption.

5.2.2.2.1.7 File Access Privileges

Kmesh can integrate with existing AD or LDAP IAM solutions or new cloud-based IAM solutions like AWS IAM to create uniform file access privilege management across multiple clouds.

5.2.2.2.1.8 Scalability

The filesystem uses a scale-out model that scales to 100s of TBs and 100GB/s of throughput per cloud. In addition, 100s of simultaneous data synchronizations and access can be supported.

5.2.2.3 Rule-based Data Orchestration Engine

The key part of the Distributed Data Service is the rule-based Data Orchestration Engine. The data synchronization and access are controlled by the rules set by users based on use case/app requirements.

Unlike other cloud data management solutions, Kmesh Data Flow provides a simple way to setup the data orchestration.

5.2.2.3.1 Kmesh Data Flow

Kmesh Data Flow provides users with a simple method for managing data orchestration across multiple clouds. Orchestration can be performed by defining data synchronization and cross-cloud remote data access. For example, a user can establish real-time synching of data from on-prem to AWS East for consumption by Big Data applications and, simultaneously, establish synching of data on AWS West to AWS central for the purpose of protecting against the impact of a potential failure in the AWS West region.

Figure 12 show the capabilities of Data Flows. Following are common uses of a Kmesh Data Flow:

- Connect a data source on one cloud to a data source on a second cloud.
- Connect a data source on one cloud to an application (filesystem) on the same cloud or on a different cloud.
- Connect an application (filesystem) on one cloud to an application (filesystem) on a different cloud.
- Connect an application (filesystem) to a remote data source.

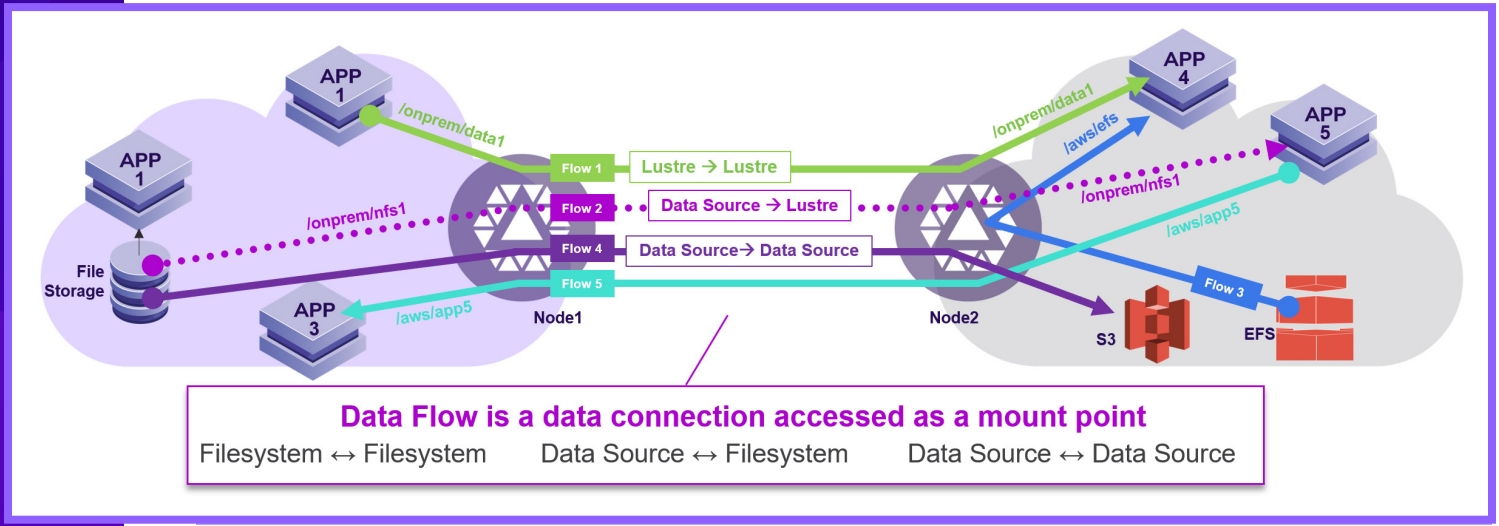

Figure 12: Kmesh Data Flows

5.2.2.3.2 Application Data Access

Data Flows make the data accessible as mountable filesystem access points (e.g. /mnt/Data Flow1/) to any Linux-based client system in a cloud or data center. These access points can be defined by the user in order to manage the global namespace. Applications can perform IO operations to this file system using standard POSIX APIs. In addition, based on the Data Flow policy and type, the data is shared with other cloud applications at multiple locations, in real time.

5.2.2.3.3 Flexible Data Mobility

Leveraging core Data Flow capabilities, users gain maximum flexibility in their cross-cloud data orchestration for purposes of data access, synchronization, availability, and aggregation (see Figure 13).

5.2.2.3.4 Data Flow Policies

5.2.2.3.4.1 In-Flight Compression

- Enabled per Data Flow
- At-rest compression can reduce storage requirements by more than 50 percent.[9]
- Both compression types support LZ4 and GZIP algorithms.
- In-flight compression uses a scale-out model to support multiple data flows.

9 - Based on the workload profile

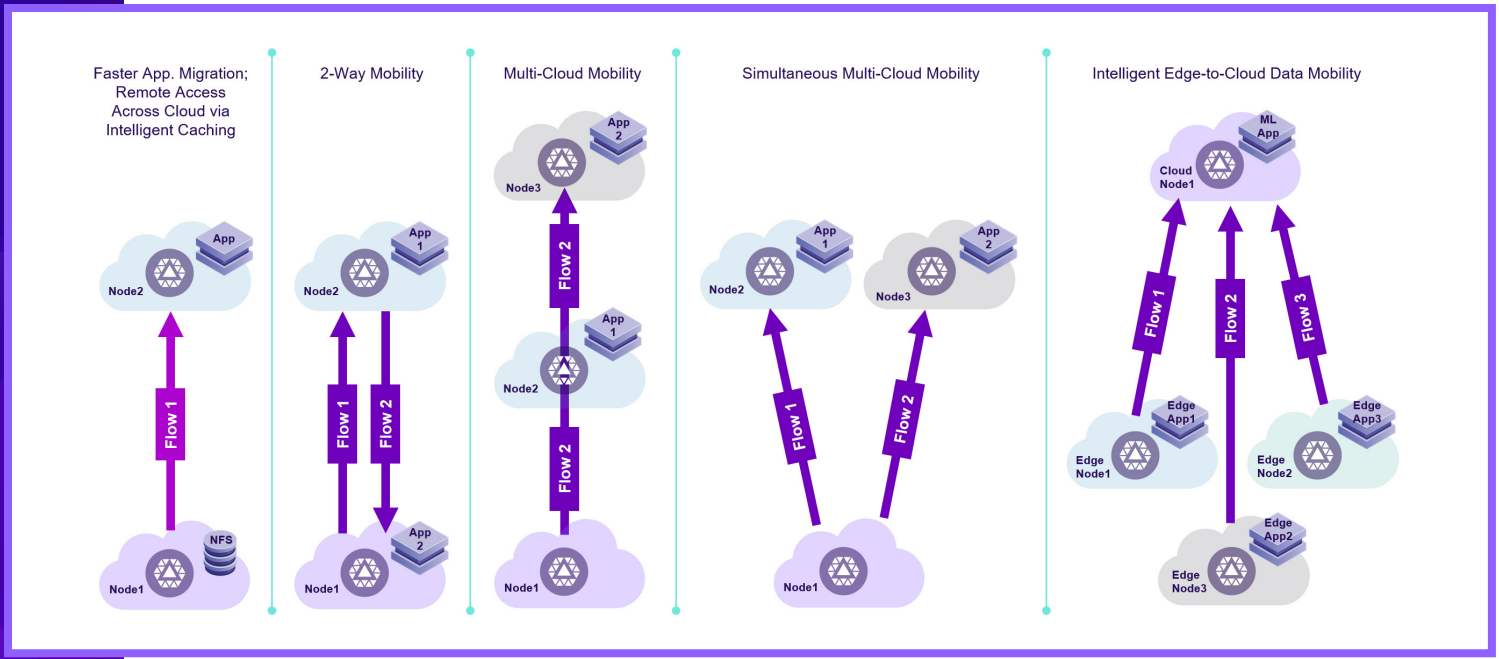

Figure 13: Kmesh flexible data mobility

5.2.2.3.4.2 In-Flight Encryption

- Enabled per Data Flow
- 128/192/256-bit encryption key
- Key Management: user-specified key management, both enterprise and public cloud-based (e.g. AWS Key Management Service)
- In-flight encryption using a scale-out model to support multiple data flows.

5.2.2.3.4.3 In-Flight Deduplication

- Enabled per Data Flow
- Duplicate blocks are detected and only metadata information is sent to the other side.

5.2.2.3.4.4 Delete Data After Use policy

When this setting is activated, data is deleted in the destination cloud when the last app is unmounted from the Data Flow.

5.2.2.3.5 Kmesh filesystem to filesystem Data Synchronization

Data synchronization is a method of copying data from one cloud to another and keeping the destination in sync with the source of any changes. The existing data is copied, and any subsequent changes—caused by application write or data ingest updates—are synchronized. If the application is deployed on the Kmesh filesystem, the data is synchronized in real time. In the event that data changes occur at the external data source (NFS, EFS or S3), the changes are synchronized as Kmesh detects the changes.

As illustrated in the Figure 14, the whole design extends Lustre's highly parallel architecture to metadata and data synchronization.

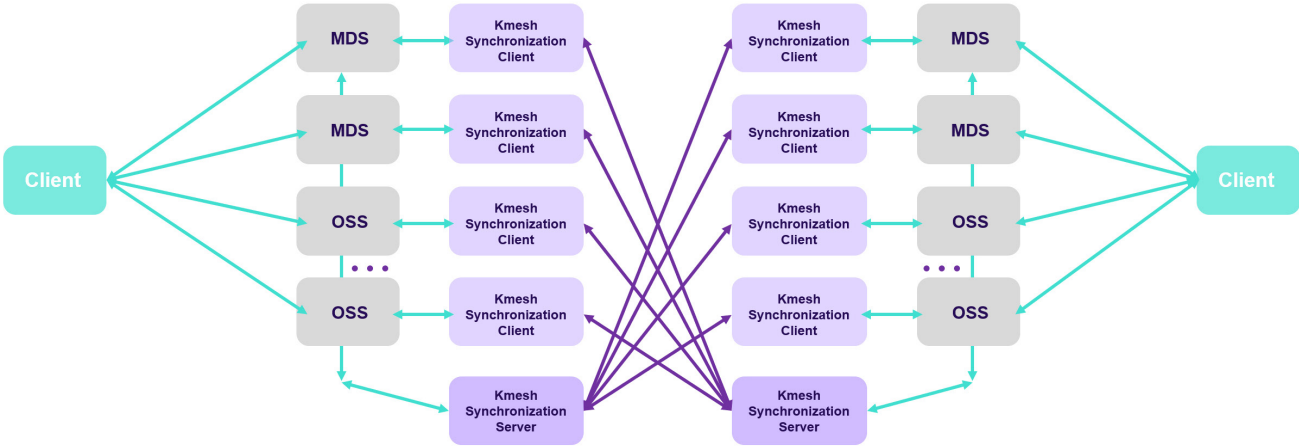

Figure 14: Kmesh Filesystem to Filesystem Data Synchronization Architecture

Parallel file systems split up their workload, so many processors, usually on different servers, can operate on file system operations or IOs coming from different clients in parallel. They can also split up large IOs from a single client and return/accept the large IOs to/from multiple processes on different servers. In this way, parallel file systems can parallelize operations and obtain a scale-out architecture, by adding more metadata and data stack servers. The following is one way to implement a parallel file system architecture. In other technology discussion documents for this patent, this described architecture is referenced as a Parallel File System Cluster Node.

In Figure 14, we have two applications, each sending file system operations to two clients, respectively. The clients are on different servers; and from this point, only the clients will be discussed, because applications must pass all their file system operations through the clients. They are performing independent IOs to a parallel file system. In the following example, there are two metadata stack processes (MDS) existing on two compute & storage servers. There are three data stack processes (OSS) existing on three compute & storage servers. Each of the processes has its own unique storage infrastructure devices.

The data stripe size in this example is 3 MiB, which means a 3-MiB write, starting at address 0, will span all three data stack processes such that the first 1 MiB of 3 MiB would be written to data stack (451); the second 1 MiB of 3 MiB would be written to data stack (452); and the third 1 MiB of 3 MiB gets written to data stack (453).

All initial metadata open calls are sent to Metadata Stack process, and this process tells the calling client which metadata stack process handles all subsequent metadata operations for a specific file. Metadata stack processes are interconnected in order to solve multi-file operations, like file rename. Network outages or process failures are not discussed.

The exact nature of synchronization and data consistency depend on the type of synchronization used for a data flow.

5.2.2.3.5.1 Synchronous

A data write or change is considered successful only when it is written to the destination successfully. This follows a semi-strict consistency approach whereby a successful update/write on the source means that the data read on the target will reflect the same update.

This data synchronization method delivers a semi-strict consistency model. Kmesh adopted this model to ensure producer/consumer models can be supported seamlessly. The consumer (destination) should be able to work independently when there is a communication issue between the source and the destination. For example, the data generated on-prem is consumed by an analytics application in the cloud. The analytics application will continue to run, even in the event of a communication failure with the on-prem source.

Note: the IO latency between source and destination may impact the performance of a producer application. If such a consistency model is not required and latency is a concern, then an async method may be selected.

5.2.2.3.5.2 Asynchronous

The data is synced to the other side, independent of the changes and updates at the source. This provides an eventual consistency model. This can be used for the producer/consumer deployment model.

5.2.2.3.5.3 Multi-Master

While the data is changed in one point for sync and async, there are scenarios in which the same data can be changed simultaneously at multiple locations. For example, an ecommerce application deployed across multiple geographic locations to serve specific regions needs certain data (I.e. order & inventory data) to be updated at a global level. Multi-master data synchronization is particularly useful in this scenario by providing a strict consistency model.

5.2.2.3.6 Any Data Source to Any Data Source Synchronization

The Kmesh filesystem to filesystem Data Synchronization works best when an application needs to be deployed on both sides of synchronization. In cases where this is not needed and to synchronize from any data source to filesystem or any other data sources, Kmesh has created the optimal synchronization solution, as illustrated in Figure 15 below.

This efficient data synchronization (Data Sync) between two regions reduces the amount of data transferred between the data source region and the data target region. In order to connect to and transfer data between source and target data regions with different data semantics, data can be canonicalized into a common data interchange format. The following is a partial list of data manipulation algorithms that improve network bandwidth to keep two data regions synchronized. Each item will reduce data network traffic, even if implemented in isolation.

1. Split objects or files into multiple smaller blocks and track differences on smaller data regions—can now transfer small blocks that differ rather than entire files/objects.

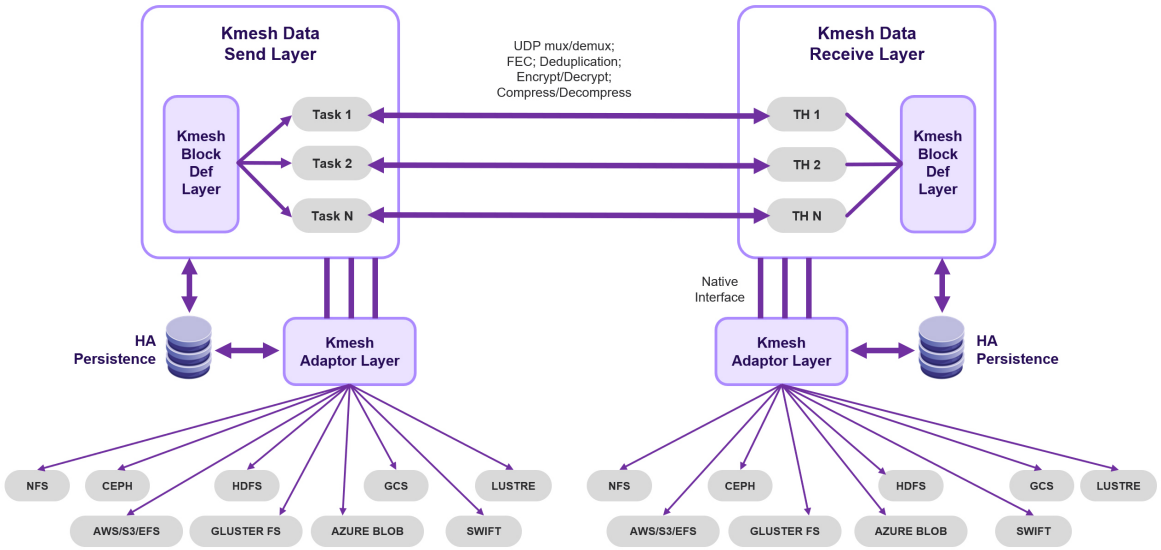

Figure 15: Any Data Source to Kmesh Filesystem Synchronization

2. Perform deduplication on the data transmitted over the wire.
3. Perform compression on the data over the wire.
4. UDP mux/demux of network connections to increase effective throughput over a 'lossy' network.
5. Forward Error Correction (FEC) will improve network bandwidth by reducing the number of retries in a lossy network. Reducing the number of retries will decrease synchronization times.

Transferring data in parallel between regions will also decrease data synchronization time. This optimization is orthogonal to data reduction techniques.

To provide security over WAN networks, the ability to encrypt/decrypt data is required but can be optional in a LAN network environment.

This automated data synchronization system works between multiple data source regions, possibly hybrid cloud endpoints.

- Control Engine will notify the system of files which have changed.
- Data Send Layer and Data Recv Layer are responsible for sending and receiving the data across the endpoints.
- The Data Send Layer and Data Recv layer interface with respective Adapter Layer to handle multiple Data sources like NFS, Ceph, HDFS, GCS, Lustre, AWS S3/EFS, GlusterFs, Azure Blob and Swift. Data Send Layer and Data Recv Layer also interact with local HA Persistence services that provide highly available persistent storage for the system.

- To determine how data in each file has changed, the Data Send Layer has a Block Meta Layer which breaks files into fixed-size chunks; and using meta information about the chunks, decides which chunks have been modified. In this manner, only modified chunks must be transmitted to the target region.
- Schedule Layer is responsible for handling multiple tasks in which data is pushed in parallel to the Destination endpoint (or Data Recv Layer). Schedule Layer will serialize tasks to maintain correct ordering.
- Data move tasks run in parallel over multiplexed connections to provide efficient data transfers over UDP protocol.
- Poll Layer is responsible for handling incoming data and update events to Open Desc Metadata Layer and to the Adapter Layer for appending/modifying data at the destination data sources.

5.2.2.3.7 Cross-Cloud Caching for Remote Data Access (Intelligent Caching)

In many cases, users want an application deployed in the cloud to access data from a source located in their datacenter, without making a permanent copy of the data outside the datacenter. Kmesh Cross-Cloud Caching helps with remote data access and applies intelligent caching to reduce IO latency.

When the Data Flow is active, the filesystem namespace is made active without waiting for the data to be copied. The data is pulled when the application reads certain portions of it. Kmesh caching applies intelligent pre-fetching algorithms to reduce the latency overhead.

The solution will evict the data in the cache when the allocated space is reached.

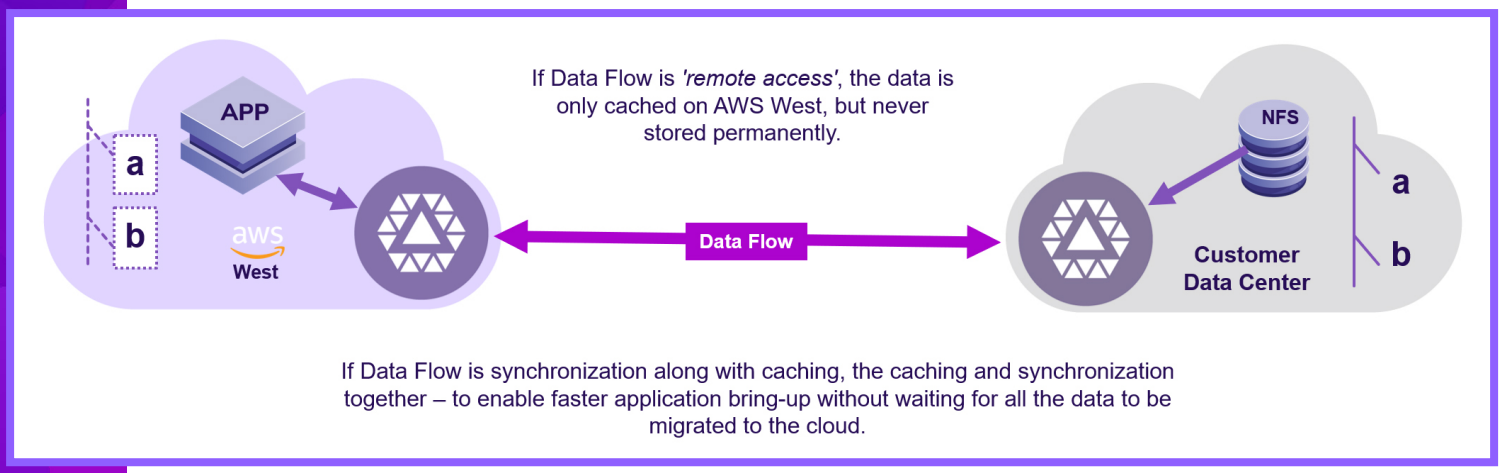

Figure 16: Kmesh synchronization

5.2.2.3.7.1 Cross-Cloud Caching with Synchronization for Faster App Migration

One of the primary issues with app migration to cloud is the time it normally takes to copy the application data to the cloud. When intelligent caching is combined with data synchronization in Kmesh, an app can be started as soon as the Data Flow is active.

5.2.2.3.8 Ingest and Write to Existing File Storage and Object Storage

As part of the Data Flow, users can set up ingest data sources so that existing data can be used in the cloud. Kmesh supports ingest from any file storage, like NFS and EFS, and object storage like S3 and Azure Blob. Users can set up one-time & recurring ingests. Kmesh can also write back to these external sources.

5.2.2.3.9 Reverse Synchronization from Virtual Copy

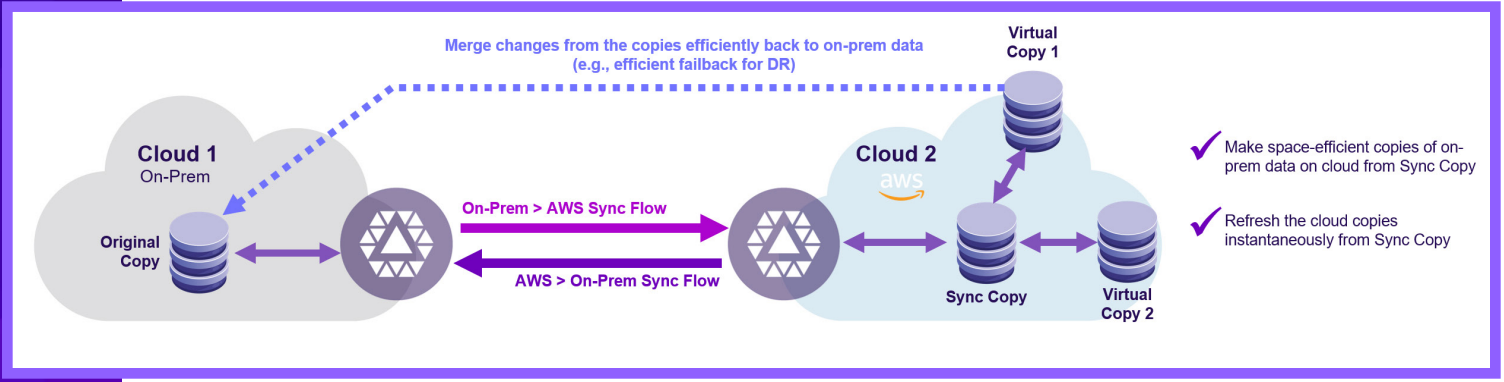

Figure 17: Kmesh reverse synchronization from virtual copy

Any time data is moved to another cloud, minimizing data movement saves money and reduces operational times. The data from one cloud is first synchronized to another cloud. From there, the data is snapshot or cloned. During this time, no changes have occurred to the original data, only to the snapshot/clone. Traditionally, merging the data from the snapshot/clone back to the original data requires a data comparison of most or all of the data on both original data and the clone. This operation will consume a lot of time and will consume even more if it is performed over two cloud connections.

5.2.2.3.10 Dataflow Snapshots

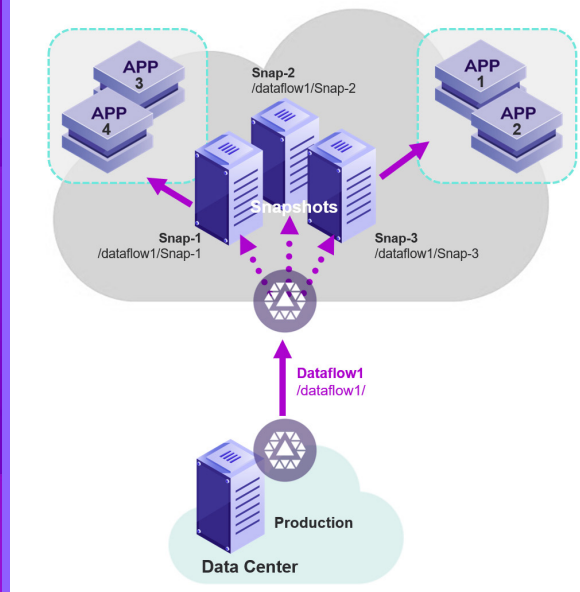

Figure 18: Space-efficient Dataflow Snapshots in Kmesh

Data Flows represent the data for the applications. User can make snapshots, which are the space-efficient, read-only copies of Data Flows.

Following are several important convenience features of the snapshots:

- Copies can be mounted directly.
- Snapshots can be scheduled:
  - » Option to schedule daily, weekly, and monthly snapshots.
  - » Option to change number of saved snapshots. Older ones are auto-deleted once the threshold number is reached. For example: Hourly (default:24) Daily (Default: 7) Weekly (Default: 5) Monthly (Default 12).

- The copies can be further copied to external sources (NFS, EFS, or S3).
- App integrated versions provide hooks to insert customer-specific scripts for quiescing of apps running on a Data Flow or Node before making a version.
- The snapshots can be created within a single cloud or across multiple clouds.

5.2.2.3.11 Dataflow Virtual Copies

A virtual copy is a point-in-time, space-efficient copy of the Data Flow representing original data on the source cloud. It is an independent copy of the parent Data Flow. Virtual Copies have the following properties:

1. Instantly refresh a virtual copy from the original data on the source cloud using the data synchronization
2. Changes on vIrtual Copies can be efficiently merged back to original copy on the source cloud using the reverse synchronization capability.

Kmesh virtual copy technology along with our patent-pending solution that tracks the relationships of the copies and the changes related to original data.

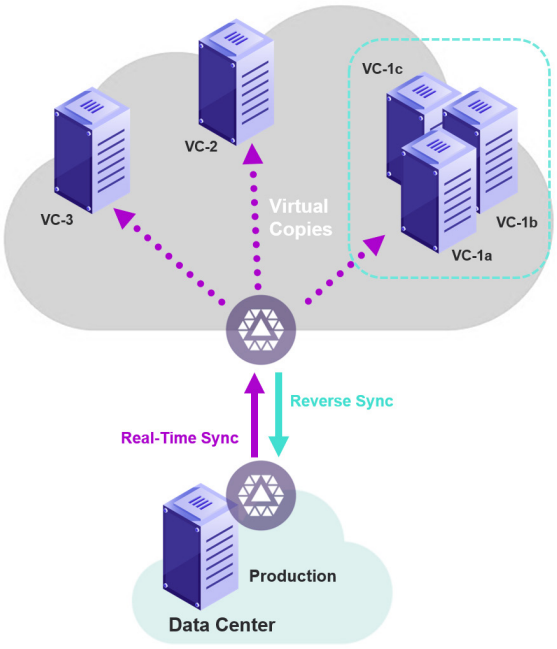

Figure 19: Virtual Dataflow Copies in Kmesh

Use cases for Data Flow virtual copy

1. App Migration: Make on-prem data available in the cloud as read-write data so that the application can be migrated.
2. DevOps in the Cloud: Deploy and streamline the DevOps process in the cloud.

6 Manageability of Kmesh

In addition to managing the Kmesh service using the Kmesh Portal, the service can also be managed using REST and via the Kubernetes Dynamic Volume plugin.

6.1 REST API

Kmesh can be provisioned and managed using REST APIs. These are the APIs the portal uses to manage the service as well.

6.2 Kubernetes Integration

Customers can deploy Kubernetes PODs on Kmesh with Kmesh dynamic provisioner. Following capabilities are provided by Kmesh dynamic provisioner.

- Deploy different K8s PODs or instances of PODs on different storage types, storage location. cloud regions or even clouds using policies
- A single POD can be deployed across hybrid cloud.
- Deploy PODs uniformly load balanced across all the available storage infrastructure

The following diagram illustrates how different PODs are assigned to different Data Flows from different policies.

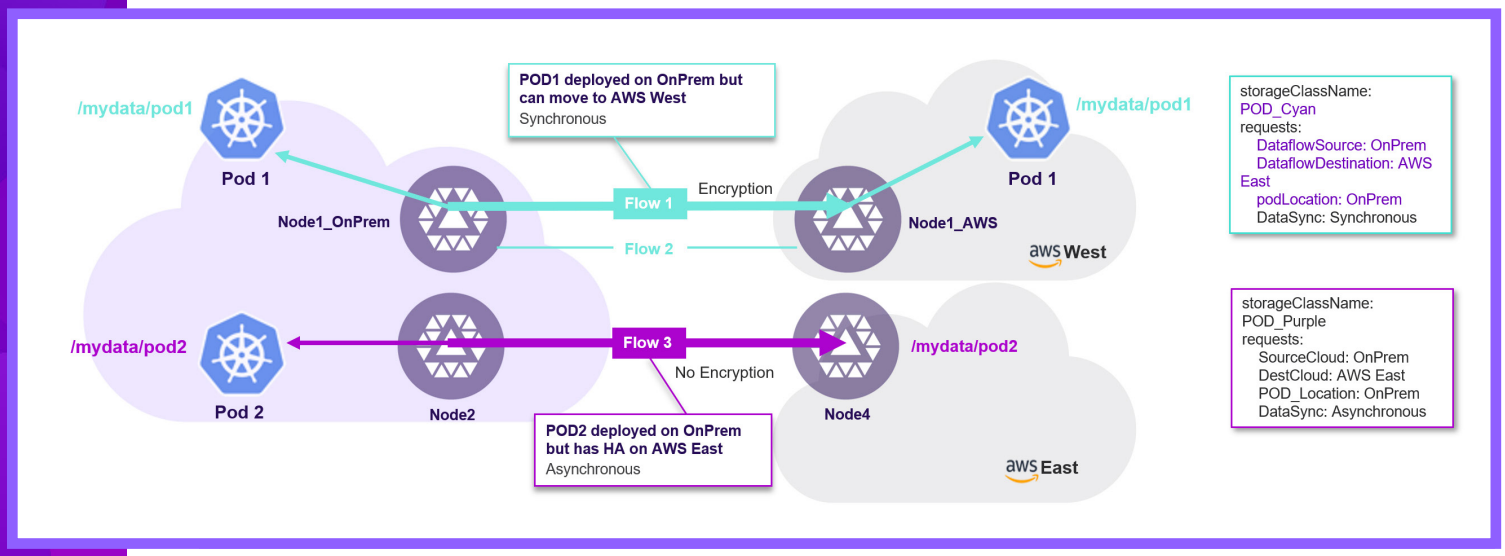

Figure 20: Dynamic provisioning of K8s PODs with Kmesh

7 Kmesh Onboarding

Kmesh orchestrator provides fully automated installation of Kmesh instances on private and public clouds. There are several prerequisite configurations and steps users take prior to Kmesh installation. In addition, users configure credentials so that Kmesh can access the infrastructure.

7.1 Private cloud

- Kmesh requires access to vCenter to create the VMs and install Kmesh instances. Alternatively, users can create the required instances and configure the IP addresses for the VMs.
- Network connectivity for Kmesh service to access the Kmesh instance.

7.2 Public cloud

- Kmesh is installed in a VPC on the public cloud and requires shared credentials to create the VPC.
- Kmesh may also require network connectivity to on-prem via VPN or DirectConnect, if the use case involves on-prem.

Visit us on the web at
https://kmesh.io/partners/,
or email us today at info@kmesh.io.

Kmesh, Inc.
4655 Old Ironsides Drive #460
Santa Clara, CA 95054

info@kmesh.io

Kmesh India Pvt Ltd.
Suite 412, HQ10, 4th Floor,
Plot No.10, Bristol IT Park,
Thiru Vi Ka Industrial Estate, Guindy,
Chennai, Tamil Nadu 600032