Intro to Apache Pulsar
a cloud-native messaging and event-streaming platform
designed to run in Elastic Cloud environments
horizontally scalable, with clients and data segmented across multiple nodes with no single point of failure
supports the publish-subscribe pattern paradigm
supports low-latency and long-term event capture, storage, and processing

Important vocabulary
Kubernetes
Kubernetes, often abbreviated as K8s, is an open-source container orchestration platform designed for automating the deployment, scaling, and management of containerized applications. Originally developed by Google and now maintained by the Cloud Native Computing Foundation (CNCF), Kubernetes has become the de facto standard for container orchestration and is widely used for managing container-based workloads in cloud and on-premises environments.

Key features and concepts of Kubernetes include:

1. **Container Orchestration**: Kubernetes automates the deployment and scaling of containerized applications. It abstracts the underlying infrastructure and provides a unified API for managing containers.

2. **Containers**: Kubernetes is container-agnostic but is most commonly associated with Docker containers. It can manage containers created with other container runtimes as well.

3. **Cluster**: A Kubernetes cluster consists of a master node and multiple worker nodes. The master node controls and manages the cluster, while worker nodes run the containerized applications.

4. **Pods**: The smallest deployable unit in Kubernetes is a pod. A pod can contain one or more containers that share the same network namespace and storage volume. Containers within a pod are co-located and can communicate with each other through the localhost interface.

5. **Services**: Kubernetes services provide a stable network endpoint to access a group of pods. They abstract the underlying network complexity and load balancing.

6. **Replication and Scaling**: Kubernetes supports replica sets and replication

controllers for ensuring a specified number of pod replicas are running. It can also scale applications automatically based on metrics or manually by the user.

7. **Load Balancing**: Kubernetes can distribute traffic to pods using built-in load balancers. It can also integrate with external load balancers.

8. **Deployment and Rollouts**: Kubernetes enables rolling updates and rollbacks of applications, ensuring minimal downtime during application updates.

9. **Configuration Management**: Configuration data, secrets, and application parameters can be managed and injected into pods as environment variables or mounted volumes.

10. **Auto-Healing**: Kubernetes can automatically restart or replace pods that fail, helping to ensure application availability.

11. **Storage Management**: Kubernetes supports various storage solutions, including local storage, network-attached storage (NAS), and cloud storage.

12. **Logging and Monitoring**: It provides integration with logging and monitoring tools, allowing operators to gain insights into the cluster's health and performance.

13. **Security**: Kubernetes includes features for securing containers and cluster communication, such as Role-Based Access Control (RBAC), network policies, and secrets management.

14. **Extensibility**: Kubernetes is highly extensible and supports the creation of custom resources and controllers through custom resource definitions (CRDs).

Kubernetes is particularly valuable in microservices architectures, where applications are composed of small, independent services that run in containers. It abstracts away many of the complexities of managing containerized applications at scale, making it easier to build, deploy, and manage modern applications.

Kubernetes has a vibrant ecosystem of third-party tools and resources, making it a versatile platform for cloud-native application development and operations.

Elasticsearch
Elasticsearch is an open-source, distributed search and analytics engine designed for horizontal scalability and real-time search. It is part of the Elastic Stack (formerly known as the ELK Stack), which also includes Kibana, Logstash, and Beats. Elasticsearch is commonly used for various search and analytics use cases,

including full-text search, log and event data analysis, and data visualization.

Here are some key features and use cases of Elasticsearch:

1. **Full-Text Search**: Elasticsearch is widely known for its ability to perform full-text search on large volumes of unstructured data. It is commonly used in applications where text search functionality is required, such as e-commerce platforms, content management systems, and search engines.

2. **Real-Time Search**: Elasticsearch provides real-time search capabilities, allowing users to search and retrieve results as soon as the data is indexed. This makes it suitable for applications that require up-to-the-minute data.

3. **Distributed and Scalable**: Elasticsearch is distributed by nature, which means it can scale horizontally by adding more nodes to a cluster. This architecture allows it to handle large datasets and high query loads.

4. **Structured and Unstructured Data**: Elasticsearch can handle both structured and unstructured data. It is not limited to text data and can be used for structured data, geospatial data, and more.

5. **Analytics and Aggregation**: Elasticsearch provides powerful aggregation capabilities, allowing users to perform analytics and statistical operations on data, such as aggregating metrics, summarizing data, and visualizing results.

6. **Log and Event Data Analysis**: Elasticsearch is commonly used for log and event data analysis. It can collect and index log data from various sources, making it easier to search, analyze, and visualize log data for monitoring and troubleshooting purposes.

7. **Near Real-Time Indexing**: Data can be indexed in near real-time, which means that as data is ingested into Elasticsearch, it becomes searchable and available for analysis almost immediately.

8. **Elasticsearch Query DSL**: Elasticsearch uses a JSON-based query domain-specific language (DSL) that allows users to construct complex queries and retrieve specific information from the indexed data.

9. **Data Visualization**: Elasticsearch is often used in conjunction with Kibana, a data visualization tool. Kibana provides a user-friendly interface for creating visualizations, dashboards, and reports based on data stored in Elasticsearch.

10. **Security and Access Control**: Elasticsearch offers various security features,

including authentication, authorization, and encryption, to protect data and ensure that access is restricted to authorized users.

Elasticsearch is widely used in a variety of industries, including e-commerce, IT operations, security, and business intelligence, for tasks ranging from website search and log analysis to monitoring and data exploration. Its flexibility, scalability, and real-time search capabilities make it a valuable tool for organizations dealing with large and diverse datasets.

Multi-tenancy
Multi-tenancy is a software architecture and deployment model in which a single software application or system serves multiple clients or tenants, often in a shared environment. Each client or tenant typically has its own isolated and secure space within the system, and they can use the software as if it were dedicated to them. This concept is commonly used in cloud computing, Software as a Service (SaaS), and enterprise applications. Here are some key aspects of multi-tenancy:

1. **Shared Environment**: In a multi-tenant system, multiple clients or tenants share the same software and infrastructure. This can lead to resource efficiency and cost savings because resources are pooled.

2. **Isolation**: While the system is shared, there is a strong emphasis on isolating the data and configuration of each tenant. Tenants should not have direct access to or visibility of each other's data or configurations.

3. **Customization**: Multi-tenant systems often allow some level of customization for each tenant. This could include configuring settings, branding, or even some level of code customizations.

4. **Security**: Security is a critical consideration in multi-tenancy. It's essential to ensure that data and configurations are properly isolated and protected. Access controls and encryption are commonly used to maintain security.

5. **Scalability**: Multi-tenant systems need to be designed for scalability, as they may need to accommodate a growing number of tenants. This can involve horizontal scaling of resources.

6. **Efficiency**: By sharing resources, multi-tenancy can be more resource-efficient than maintaining separate instances of the software for each tenant. This can lead to cost savings and easier maintenance.

7. **Examples**: Multi-tenancy is common in various applications, including SaaS platforms (where multiple customers access the same application), cloud

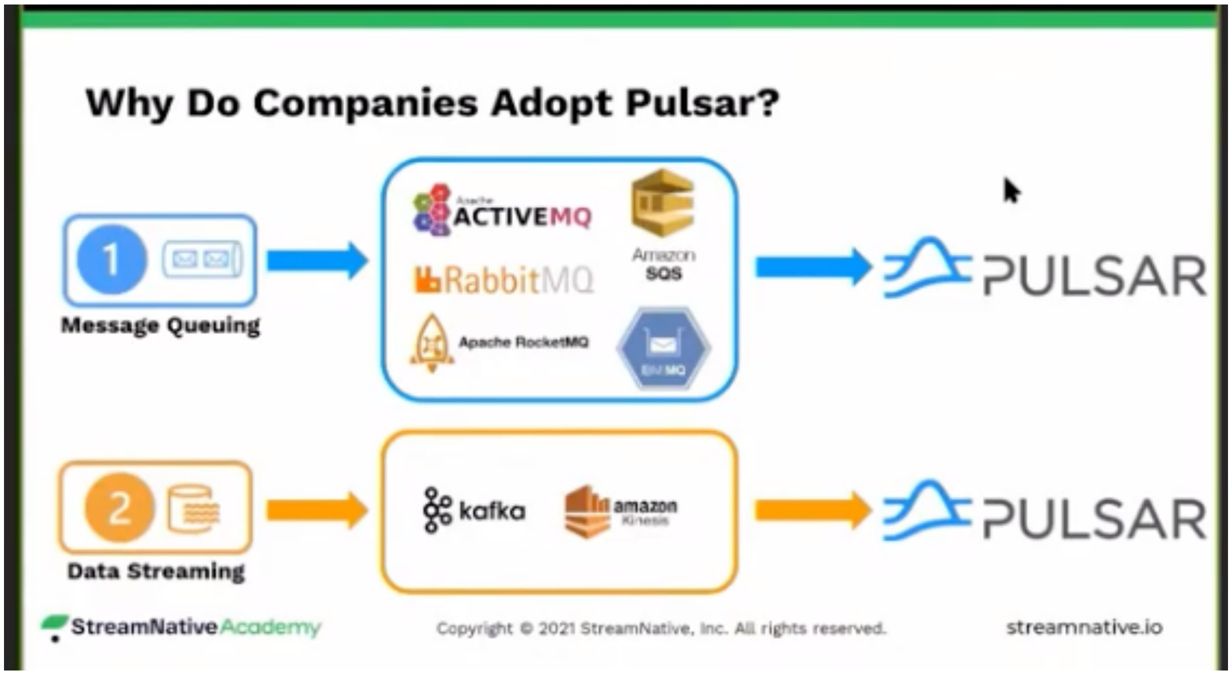
infrastructure (where multiple users share virtualized resources), and enterprise software (where different departments or subsidiaries of a large organization share the same system).

8. **Variants**: There are different levels of multi-tenancy. In a pure multi-tenant model, tenants have no visibility into the shared infrastructure. In a semi-multi-tenant model, there may be some shared components but also dedicated components for each tenant.

Multi-tenancy is an important concept for organizations that provide software services to multiple customers. It allows them to efficiently serve a broad customer base while maintaining data security and customization options. However, it also requires careful design and management to ensure that each tenant's experience is smooth and secure.

Metadata is data about data > descriptive, contextual (purpose, origin), technical (size, type), data management (indexing) governance (ownership), search/retrieval, integration (transforming and mapping between different systems, visualizing, filing (permissions), database management (table structures and relationships), lineage, and warehousing (source/lineage info)

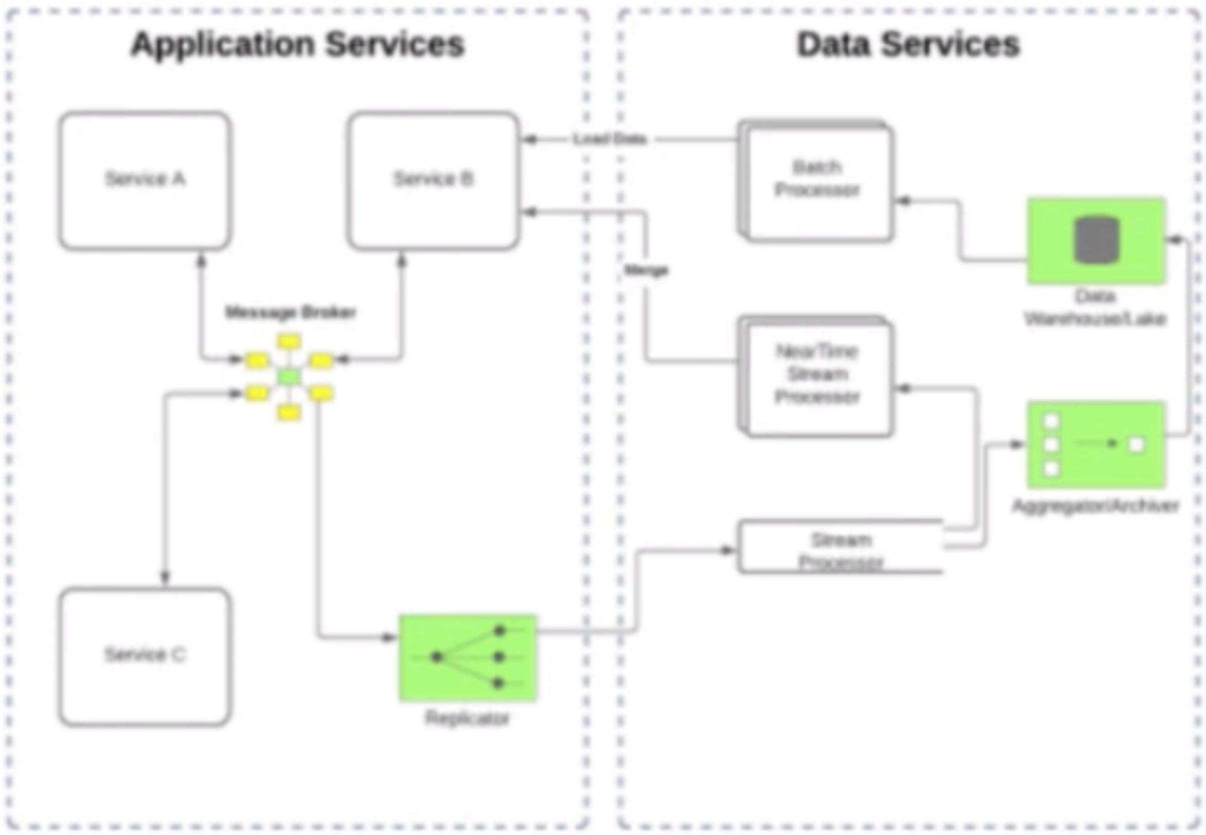
Advantages of Pulsar
provide the ability to achieve both message queuing and data streaming in one architecture



Why Apache Pulsar?

unified messaging platform
guaranteed message delivery
resiliency
infinite scalability

Pulsar helps with communicating between services and moving data within an organization.



‘Message queuing’ helps you deal with these issues because it facilitates interactions between systems by exchangin messages asynchronously.
‘Data streaming’ is a solution to this problem because it can efficiently move large amounts of data in and throughout a system.
Pulsar unifies both of these solutions in one platform.

‘Messaging’ provides a way to decouple services by communicating messages asynchronously.
*Messages are the most basic unit of Pulsar.
Applications send packets of data (think of as a letter)
Pulsar ‘broker’ can be viewed as the central post office
Consumer is the entity receiving the data

Two typical ways to deliver data:

- via a queue
- fanning out messages to multiple interested parties (referred to as a message bus)

Pulsar supports both these ways in addition to scheduled delivery and failure handling

Streaming handles many messages (or joins different streams of messages) at a time.
*these events (messages) are interlinked together

*key difference between messaging and streaming > messaging system applications do not have control over when a message arrives, while streaming applications do control when the data is delivered

Streamin use cases:
- moving large amounts of data to another service (logs to real-time ETL)
- running periodic jobs to move large amounts of data and aggregating the data to more traditional stores (logs to S3)
- computing near real-time aggregate of a message stream (real-time analytics over page views)

Vocab break:
Pub-sub
Publish-Subscribe (Pub-Sub) messaging is a messaging pattern used in distributed systems and messaging systems to facilitate communication between different components, services, or systems. In a pub-sub model, there are two main entities: publishers and subscribers, and an intermediary called the message broker or message bus.

Here's how the Publish-Subscribe pattern works:

1. **Publishers**: Publishers are components or applications that generate messages or events. These messages can represent various types of information, such as data updates, notifications, events, or commands. Publishers are responsible for sending these messages to a message broker.

2. **Message Broker**: The message broker, also known as the message bus, is an intermediary that receives messages from publishers and distributes them to the appropriate subscribers. It acts as a centralized communication hub. The broker is responsible for routing messages to the correct subscribers based on certain criteria or topics.

3. **Subscribers**: Subscribers are components or applications that express their

interest in specific types of messages or events. They subscribe to particular topics or channels provided by the message broker. Subscribers receive messages that match their subscriptions.

Key characteristics of Pub-Sub messaging include:

- **Decoupling**: The Pub-Sub pattern decouples publishers and subscribers. Publishers are unaware of the specific subscribers, and subscribers are unaware of the publishers. This decoupling allows for flexibility and scalability in a distributed system.

- **Scalability**: Pub-Sub systems can easily scale as the number of publishers and subscribers grows. New components can be added without significant changes to the existing infrastructure.

- **Asynchronous Communication**: Communication between publishers and subscribers is typically asynchronous. Publishers send messages to the broker, and subscribers receive messages as they are published.

- **Topic-Based Filtering**: Subscribers can specify their interest in specific topics or channels. The message broker ensures that messages are delivered to subscribers based on their subscriptions.

- **Fan-Out**: Pub-Sub systems can perform fan-out, meaning a single message can be delivered to multiple subscribers interested in the same topic.

- **Event-Driven Architecture**: Pub-Sub is often used in event-driven architectures, where components react to events or messages rather than being tightly coupled through direct function calls.

Pub-Sub messaging is widely used in various applications and scenarios, including:

- Real-time data streaming and event-driven applications.
- IoT (Internet of Things) systems where sensors and devices generate events.
- Distributed systems for managing distributed state and coordination.
- Chat applications and social media platforms.
- Monitoring and alerting systems for notifying stakeholders of important events.

There are many messaging systems and technologies that implement the Pub-Sub pattern, including Apache Kafka, Apache Pulsar, RabbitMQ, and various cloud-based message brokers. These systems provide the infrastructure for building scalable and reliable event-driven applications.

More vocab:
ETL
ETL stands for Extract, Transform, Load, and it refers to a process used in data integration and data warehousing. ETL is a crucial component of the data pipeline in which data is collected from multiple sources, transformed to meet the desired format and structure, and then loaded into a target data store or data warehouse for analysis and reporting. Here's some background on ETL:

1. **Extract (E)**: In the first stage of ETL, data is extracted from various source systems or data repositories. Sources can include databases, log files, spreadsheets, web services, and other structured or unstructured data sources. The goal is to collect data from these diverse sources and bring it together for further processing.

2. **Transform (T)**: After extraction, the data undergoes a transformation process. This can involve data cleaning, filtering, validation, and enrichment. Transformation tasks can include:

   - Data cleansing: Removing or correcting inconsistent or inaccurate data.
   - Data enrichment: Adding additional information to the data, such as geocoding or data from external sources.
   - Data aggregation: Summarizing and aggregating data for reporting.
   - Data normalization: Standardizing data formats, units, or representations.
   - Data deduplication: Identifying and eliminating duplicate records.

   The transformation process ensures that data is in the desired format, structure, and quality for analysis and reporting.

3. **Load (L)**: In the final stage, the transformed data is loaded into a target data store or data warehouse. The data warehouse is a centralized repository designed for analytical purposes. It typically supports complex querying and reporting. Data can be loaded into various tables or data models, making it accessible for business intelligence tools and analytics.

Key points about ETL:

- **Data Integration**: ETL is a critical step in data integration, as it enables organizations to combine data from different sources into a unified view.

- **Batch and Real-Time ETL**: ETL processes can be performed in batch (scheduled, periodic updates) or in real-time (continuous streaming updates) depending on the data and reporting requirements.

- **ETL Tools**: There are ETL tools and platforms that automate and streamline the ETL process. These tools provide graphical interfaces for designing ETL workflows and may include features for data profiling, data quality checks, and job scheduling.

- **Data Quality and Governance**: ETL includes data quality checks and validation, which are essential for ensuring data accuracy and consistency.

- **Historical and Incremental Loading**: ETL can handle historical data loads as well as incremental updates, ensuring that the data warehouse stays up to date.

ETL is a fundamental process in data warehousing and analytics, enabling organizations to make informed decisions based on a consolidated view of their data. It is essential in business intelligence, reporting, and data-driven decision-making.

Amazon S3 (Simple Storage Service)
In the context of big data and cloud computing, "S3" typically refers to Amazon S3 (Simple Storage Service), which is an object storage service provided by Amazon Web Services (AWS). Amazon S3 is widely used as a foundational storage solution for big data and cloud-based data analytics. Here's an overview of what Amazon S3 means in the context of big data:

1. **Object Storage**: Amazon S3 is an object storage service that allows users to store and retrieve data objects, such as files, images, documents, and datasets. It is designed to handle large volumes of data and is highly scalable.

2. **Data Storage**: Big data applications often involve the storage of vast amounts of structured and unstructured data, including log files, sensor data, social media content, and more. Amazon S3 provides a reliable and cost-effective storage solution for such data.

3. **Data Ingestion**: In big data pipelines, data is frequently ingested from various sources, such as IoT devices, web applications, and external data feeds. Amazon S3 can serve as a landing zone for incoming data before further processing.

4. **Data Lakes**: Amazon S3 is commonly used as a foundation for building data lakes. A data lake is a central repository for storing raw and structured data that can be accessed and analyzed by various big data tools and services.

5. **Data Distribution**: Amazon S3 allows users to share data by providing

access control mechanisms and generating unique URLs for data objects. This is useful for data distribution and collaboration within and outside an organization.

6. **Data Access**: Big data frameworks and tools, such as Apache Hadoop, Apache Spark, and cloud-based analytics services, can directly access and process data stored in Amazon S3. This enables distributed data processing and analytics at scale.

7. **Data Backup and Archiving**: Amazon S3 is often used for data backup and long-term data archiving. Data objects can be stored with high durability and availability, making it suitable for data retention and compliance needs.

8. **Scalability and Redundancy**: Amazon S3 offers high scalability and redundancy. Data is automatically distributed across multiple data centers, providing fault tolerance and high availability.

9. **Data Security**: Amazon S3 provides features for data security, including encryption options (in transit and at rest), access control, and audit trails. This is essential for protecting sensitive data in big data applications.

10. **Integration**: Amazon S3 integrates with various AWS services and big data platforms, making it a versatile storage solution within the AWS ecosystem and for third-party big data tools.

Amazon S3 is considered a fundamental component of many big data architectures, especially in the cloud. It allows organizations to store, manage, and analyze vast amounts of data efficiently and cost-effectively. Additionally, Amazon S3 provides options for tiered storage, enabling organizations to optimize costs based on data access patterns and retention policies.

Hadoop vs Spark
Certainly! Apache Spark and Apache Hadoop are both open-source frameworks for big data processing, but they have different architectures and use cases. Here's a brief overview of each:

Apache Spark:
- Apache Spark is an open-source, distributed data processing framework designed for high-speed data processing and analytics. It is known for its in-memory processing capabilities, which make it faster than traditional batch processing systems like Apache Hadoop.
- Spark supports various data processing workloads, including batch processing, real-time streaming, machine learning, and graph processing.
- It provides high-level APIs in multiple programming languages, including Scala,

Java, Python, and R, making it accessible to a wide range of developers.
- Spark has a built-in cluster manager (standalone, Mesos, or YARN) and can run on top of Hadoop HDFS, Apache HBase, and other data sources.
- The core abstraction in Spark is the Resilient Distributed Dataset (RDD), a distributed collection of data that can be processed in parallel. RDDs are immutable and fault-tolerant.
- Spark SQL allows for SQL-like queries and data manipulation on structured data, and it can integrate with various data sources.
- Machine learning libraries like MLlib and graph processing libraries like GraphX are available as part of the Spark ecosystem.

Apache Hadoop:
- Apache Hadoop is an open-source framework for distributed storage and processing of large datasets. It's designed for batch processing and storing massive amounts of data on a distributed file system called HDFS (Hadoop Distributed File System).
- Hadoop's core components include HDFS for storage, MapReduce for batch processing, and YARN (Yet Another Resource Negotiator) for cluster resource management.
- MapReduce is a programming model and processing framework used for processing large data sets in parallel. It divides data into smaller chunks and processes them across a cluster of machines.
- Hadoop provides data redundancy and fault tolerance by replicating data across multiple nodes, ensuring that data remains available even if some nodes fail.
- While originally designed for batch processing, Hadoop has added additional modules over time, such as HBase for NoSQL storage, Hive for SQL-like queries, Pig for data scripting, and others.
- Hadoop is suitable for use cases where data processing can be scheduled and executed in batches, such as ETL (Extract, Transform, Load) processes and data warehousing.

In summary, Spark and Hadoop are both used for big data processing, but they excel in different areas. Spark is known for its speed and versatility, making it suitable for real-time analytics and machine learning. Hadoop, on the other hand, is well-suited for batch processing and large-scale data storage. Depending on your specific use case and requirements, you may choose one or the other, or even both in a complementary architecture. It's not uncommon to see Spark and Hadoop used together in a big data ecosystem.

Apache (Apache Software Foundation (ASF)):
Certainly! "Apache" often refers to the Apache Software Foundation (ASF), a nonprofit organization that provides support for open-source software projects. The ASF is not a specific software application but rather a collective of open-

source projects developed and maintained by a global community of contributors. Here's some information about the Apache Software Foundation and its significance in the software industry:

Apache Software Foundation (ASF):

- The Apache Software Foundation, commonly known as Apache, is a community-driven organization established in 1999 to support and oversee the development of a wide range of open-source software projects.
- The ASF operates under a collaborative and consensus-based model, where contributors from around the world work together to develop and maintain software projects.
- The ASF provides a legal framework and infrastructure for hosting open-source projects, including source code repositories, issue tracking, mailing lists, and websites.
- Apache projects cover a diverse array of software categories, including web servers, databases, big data processing, distributed computing, application development frameworks, and more.
- Many well-known and widely used software projects are part of the Apache ecosystem. Some notable examples include:
  - **Apache HTTP Server**: One of the most popular web servers in the world.
  - **Apache Hadoop**: A framework for distributed storage and processing of large datasets.
  - **Apache Spark**: A fast and versatile data processing and analytics framework.
  - **Apache Kafka**: A distributed event streaming platform.
  - **Apache Tomcat**: A Java Servlet and JSP container.
  - **Apache Maven**: A build automation tool.
- Apache projects are governed by merit-based processes, emphasizing collaboration, transparency, and community-driven decision-making.
- Apache projects adhere to open-source licenses, such as the Apache License, which allows for the free use, modification, and distribution of the software.

The Apache Software Foundation has played a significant role in the development of open-source software and has contributed to the success of many projects that have become integral to the software industry. These projects are widely adopted across a range of sectors, including web development, data processing, and infrastructure management. The ASF's commitment to open-source principles and community collaboration has made it a well-respected organization in the world of software development.

RabbitMQ:
RabbitMQ is an open-source message broker software that provides a messaging

and message queuing service for distributed and scalable applications. It is designed to facilitate the exchange of data and messages between different components of a distributed system, making it a fundamental tool for building robust and asynchronous communication in various software applications. Here are some key aspects of RabbitMQ:

1. **Message Queuing**: RabbitMQ follows a message queuing model, where messages are produced by senders (producers) and consumed by receivers (consumers). The broker, RabbitMQ in this case, acts as an intermediary that receives and routes messages to their intended destinations.

2. **Messaging Patterns**: RabbitMQ supports various messaging patterns, including publish-subscribe (pub-sub), request-reply, and point-to-point messaging. These patterns allow developers to implement different communication scenarios in their applications.

3. **Message Broker**: RabbitMQ is a central message broker that handles the distribution and routing of messages. It ensures that messages are delivered to the correct consumers based on the specified routing rules.

4. **Queueing**: Messages are placed in queues by producers and then consumed by consumers. Queues act as temporary storage for messages, allowing consumers to process them at their own pace. This decouples message production and consumption.

5. **Routing and Exchange**: RabbitMQ uses exchanges to route messages to the appropriate queues. The exchange determines how messages are distributed based on routing keys and message headers.

6. **Support for Multiple Protocols**: RabbitMQ supports various messaging protocols, including Advanced Message Queuing Protocol (AMQP), Simple/Streaming Text Oriented Messaging Protocol (STOMP), and others. This makes it interoperable with a wide range of applications and platforms.

7. **Reliability**: RabbitMQ is known for its reliability and message durability. Messages can be persisted to disk, ensuring that they are not lost even in the event of system failures.

8. **Scalability**: RabbitMQ can be deployed in clusters to achieve high availability and scalability. It can handle a large number of concurrent connections and deliver messages with low latency.

9. **Plugins and Extensions**: RabbitMQ provides a plugin architecture, allowing

developers to extend its functionality and integrate it with other systems and services.

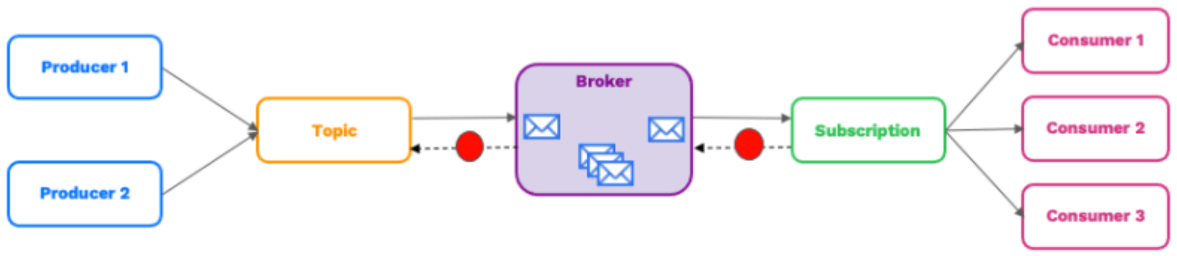
10. **Management and Monitoring**: RabbitMQ offers a web-based management interface that allows administrators to configure and monitor the message broker, view statistics, and manage queues and exchanges.

RabbitMQ is widely used in various scenarios, including microservices architectures, cloud-based applications, data processing pipelines, and event-driven systems. It provides a flexible and reliable means of enabling communication and coordination between different components of a distributed application, enhancing the scalability, robustness, and flexibility of modern software systems.

Back to Pulsar
-unified storage with access to underlying data
-native tiered storage
-single system to exchange data
-teams share toolset

*goal is to centralize, simplify, and make more efficient the data services architecture and in turn how these services communicate with application services

How Pulsar brings the application and data domains together?
- consumes the raw data from all the services over a unified transport
- either offloads the data to long-term storage or uses a real-time stream processor to extract, transform, and load (ETL) the data
- reads back the data using a parallel batch process
- sends the stream of data back to the other systems so the services can consume the data



Producer is a process that publishes messages to a topic
Consumer is a process that establishes a subscription to a topic and processes messages ppublished to that topic
Subscription is a named configuration rule that determines how messages are

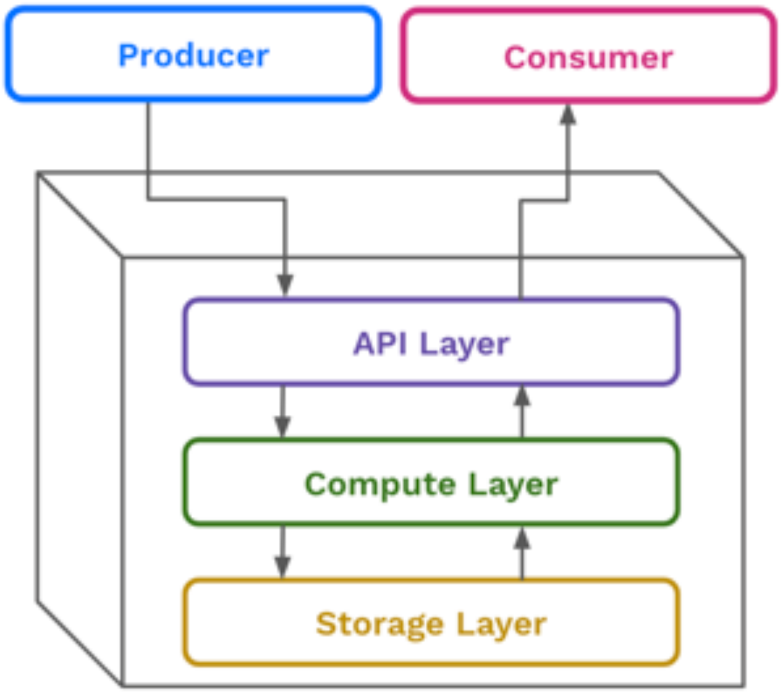
delivered to consumers
- Four types of Pulsar subscriptions > exclusive, shared, failover, and key-shared

Brokers (Pulsar) handle the connections and routes messages
Topics are named channels for transmitting messages from producers to consumers
Partitioned topics are ‘virtual’ topics composed of multiple topics
Messages belong to a topic and contain an arbitrary payload
Payload is the actual data carried by the message
- can be of any type or format

Pulsar uses ‘acknowledgments’ for reliable messaging.
- like a postmark, acknowledgements confirme receipt

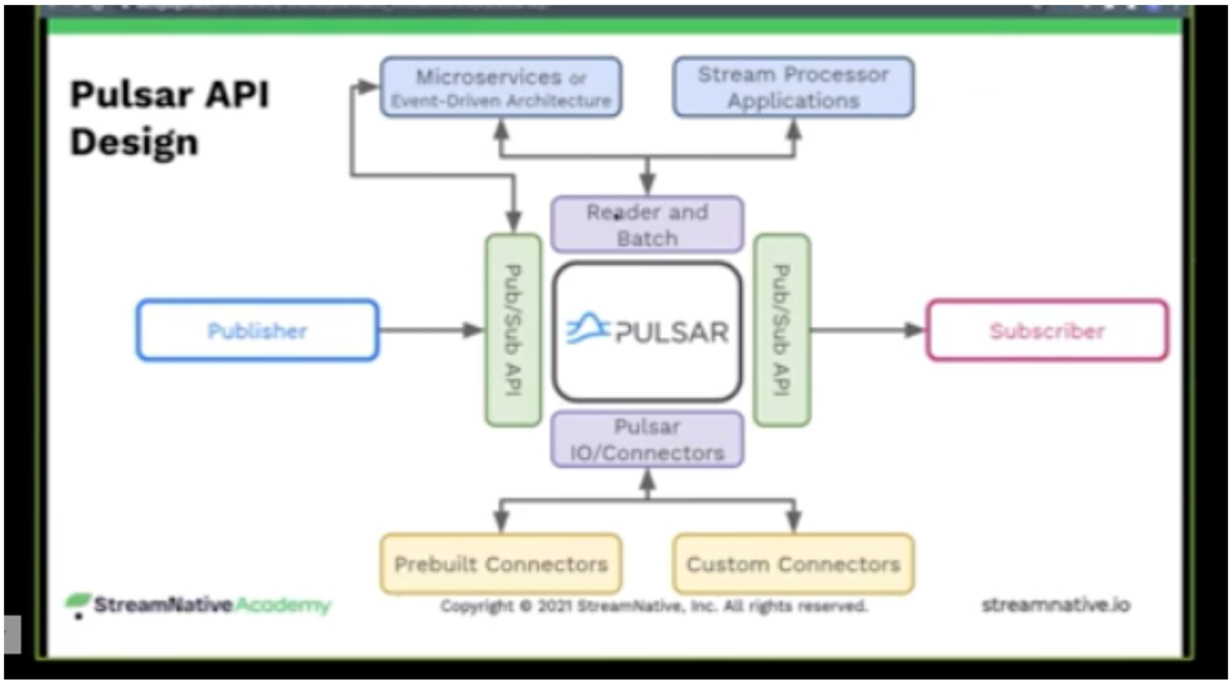
Subscriptions
Exclusive > allow only a single consumer to be connected at a time (that consumer is guaranteeing order)
Failover > similar to exclusive, but if a consumer fails a second one pick up where the first one left off
Shared > subscriptions consume a subset of the messages, there is no ordering guarantee but it allows for distributed work
Key-shared > allow multiple consumers to attach and each of these consumers have guaranteed ordering using a key

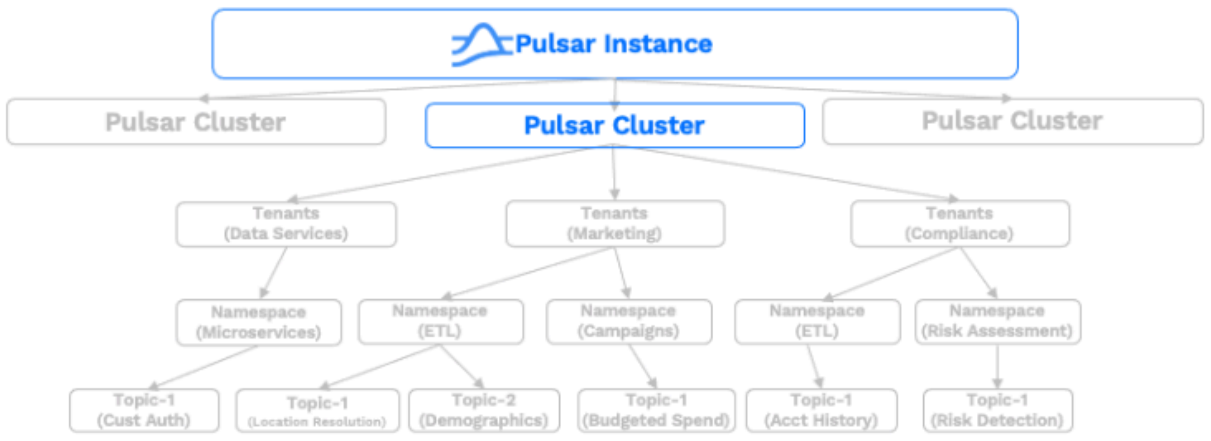
Three layers of the Pulsar architecture



API layer - data serving is handled by brokers
Compute layer - physical disk, RAM, CPU
Storage layer (message retention) - data storage is handled by bookies

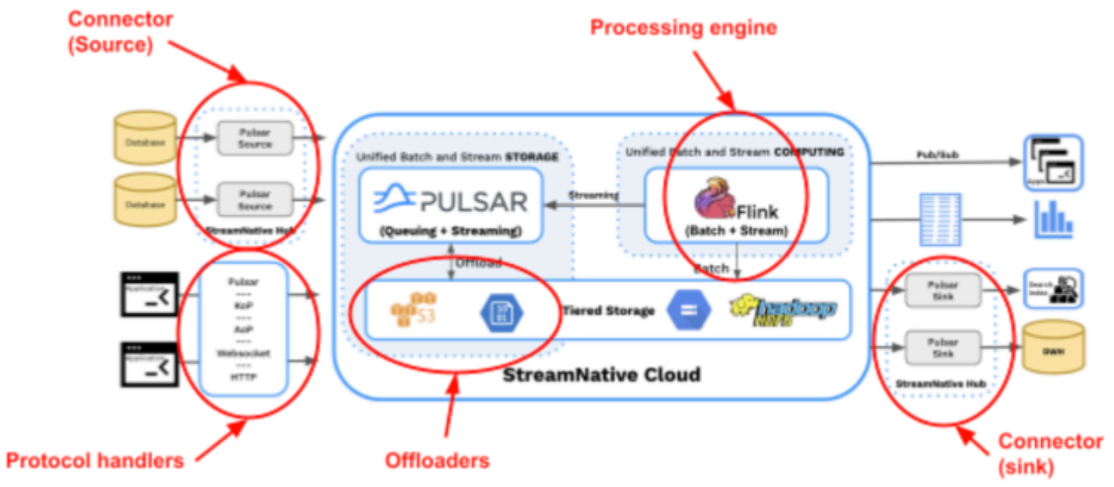
Pulsar API Design





Multi-tenancy
- provides clear structure for mapping topics to different teams, applications, or use cases
- serves as the foundation of security
- allows for unified, global management of multiple clusters

Instance is a group of clusters that act together as a single unit
Cluster is a set of Pulsar brokers, Zookeeper quorum, and an ensemble of BookKeeper bookies
Tenants are the administrative unit within a shared environment that provides the ability to isolate namespaces, specify quotas, and configure authentication and authorization on a per-tenant basis
Namespaces are a grouping mechanism for related topics
Topics are named channels for transmitting messages from producers to consumers


Connector (Source)
Processing engine
Database
Database
Pulsar Source
Pulsar Source
StreamNative Hub
Unified Batch and Stream STORAGE
Unified Batch and Stream COMPUTING
PULSAR (Queuing + Streaming)
Streaming
Flink (Batch + Stream)
Pub/Sub
Offload
Batch
Pulsar
KoP
AoP
Websocket
HTTP
S3
Tiered Storage
hadoop HDFS
Pulsar Sink
Pulsar Sink
StreamNative Hub
DWH
StreamNative Cloud
Protocol handlers
Offloaders
Connector (sink)