Leveraging Kafka Event Stream for Financial Services

Matt Hejnas - Principal Architect, Cloud Integration Lead

Today’s Schedule

1. Cedrus Overview (5 min)
2. Kafka/Event Stream Overview (10 min)
3. Kafka Use Cases (5 min)
4. Financial Institute use case (10 min)
5. Value of Kubernetes to Kafka (5 min)
6. Scaling/operation best practices (5 min)
7. QA (10 min)

Welcome to Cedrus | Digital Transformation Solutions

Cloud Native & IoT

Discovers user stories and designs MVPs, UIs and solutions for the Cloud. Their expertise ranges from web and mobile development to IoT and cognitive / voice activated devices. Developing / Implementing Cloud and Hybrid Cloud strategies. Operational excellence. Technology stack includes Kafka, Cassandra, Kubernetes, Spring...

Cognitive Automation & Artificial Intelligence

Designs and builds digital process solutions. In addition, this group focuses on Artificial Intelligence and Machine Learning services to help automate and optimize processes. Included in this group is also our IBM, Chatbot and Blockchain skills. As well as Robotic Automation (RPA - Blue Prism, Pega, Automation Anywhere)

Cloud Security

Within Cloud security we focus on Identity and Access (ISAM / SailPoint / Okta / Auth0) within the specific domain of CASB (Cloud Access Security Broker - Netskope) and how it works with existing corporate policies, systems, and the target SaaS / IaaS / PaaS

Cedrus Lab

Responsible for research and development our products and business accelerators.

- API Czar
- Code Fusion
- Smart Data Intake
- Home Energy control solution

Design Thinking is at the heart of everything we do at Cedrus

Why Should You Care About a Vendor's Ecosystem?

You care about what your business needs to achieve – not whether all the technology components come from one provider. You care about having partners who give you every advantage.

Ecosystems provide the ability to assemble the best solutions and be CREDIBLE.

cedrus

Kafka/Event Stream Overview

cedrus

Kafka’s Origins

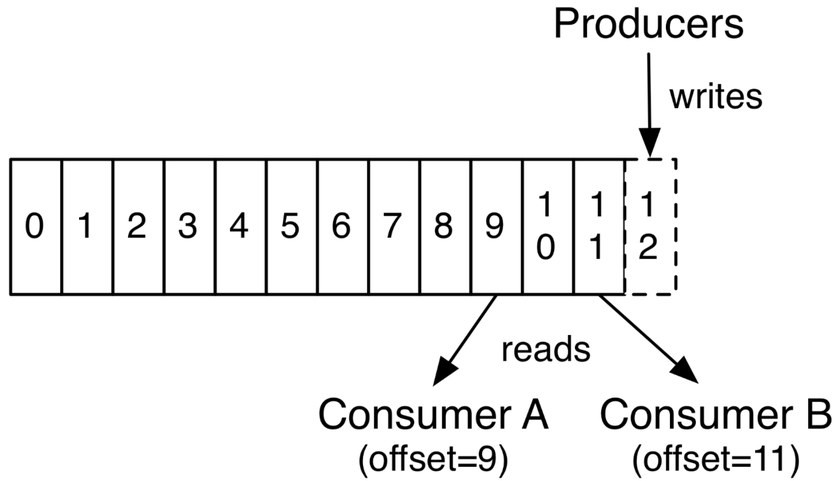
- Created at LinkedIn in 2010
- Donated to Apache as a Top-level F/OSS project in 2012
- Designed to:
  - Simplify data pipelines
  - Handle large amounts of data in a streaming pattern
  - Support real time and batch systems
  - Massively scale horizontally
- Used by these companies:
  - Insurance - 8 out of the top 10
  - Banks - 7 of the top 10
  - Telecom - 9 of the top 10
  - More than 35% of Fortune 500 companies
- Now primarily developed by Confluent
  - Confluent was formed in 2014 by the team that built Kafka

cedrus

Apache Kafka - Event Stores != Another Messaging Platform

Aspect	Traditional Queue	Traditional Topic	Kafka Topic
Cardinality	One get per put	N gets per put	Gets determined by consumer(s)
Persistence	Transient - From put until get	Transient - from put until all subscribers get (if durable)	Permanent, Immutable (Like blockchain)
Message Format	Open, with headers	Open, with headers	Open, with partition key
Performance	Implementation Specific. Generally moderate.	Implementation Specific, Generally low.	Massively Scalable

- **Key differences**
  - Kafka is basically a robust filesystem, with an **array** of messages
  - Messages assigned an ascending integer **index** as they arrive
  - Messages are never deleted or modified **(Write Once)**
  - The **consumer** specifies the index that is wanted
    - Zero (beginning of time)
    - Highest (real time)
    - Last (for subscriber group, if known by the broker)
    - Other (0-highest is valid)
- **Key Terms**
  - **Topic -** A logical collection of events, organized into one or more partitions
  - **Partition -** An ordered collection of events within a topic, identified by a key
  - **Offset -** A number that defines an event in a topic by its distance from the first event (0)



cedrus

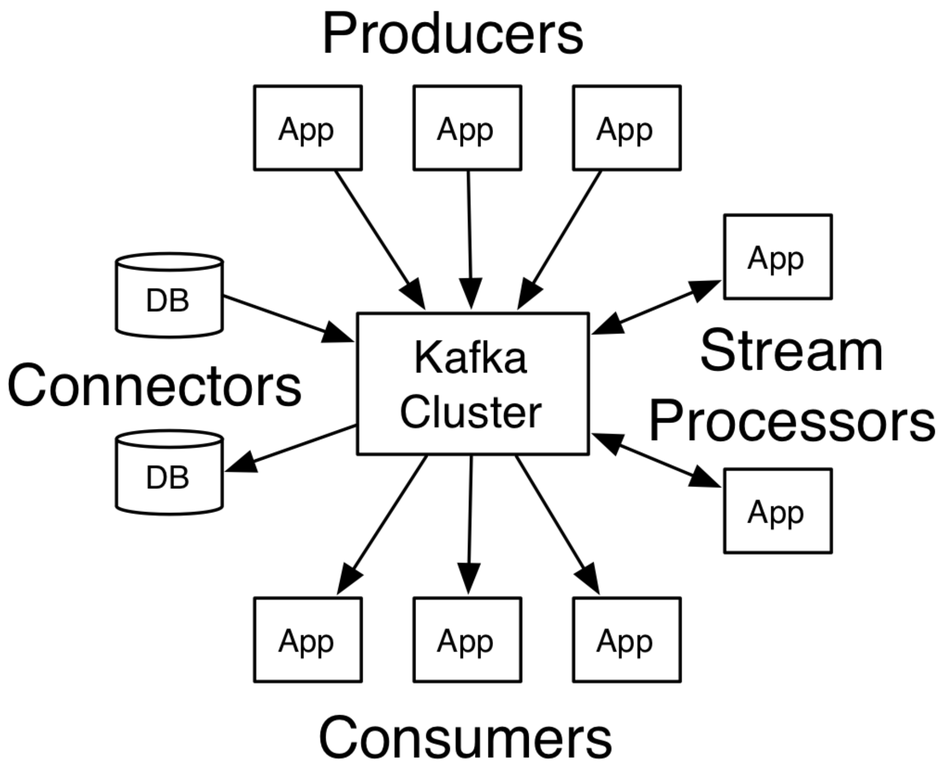
What is Event Streams?

- **IBM - Supported Kafka, ready for Kubernetes**
  - An expert-built offering of Kafka, with rational and secure configuration
    - Similar to an opinionated framework
    - Built on IBM Message Hub experience
  - Turn-key deployment of the application - ready to go in a few clicks
  - Enterprise features (ex. Geo-Replication for DR)
- **Value-Add additions**
  - Intuitive UI for:
    - Administration - Creation and configuration of topics
    - Testing - Injection of messages is simple and repeatable
    - Monitoring - Dashboards show data flow metrics and messages
  - Connectors to other systems
    - MQ
    - Z/OS

cedrus

Use Cases

- **Data Pipelines**
  - Movement and processing of large amounts of data in real time
    - Massively parallel and horizontally scalable allows for automatic scaling to achieve business goals in real time
  - ETL replacement - real time instead of batch
  - Populating data lakes
    - Batch operations that might overwhelm infrastructure with data or require backpressure are smoothed by Kafka
- **Real-Time APIs, Analytics and Alerting**
  - Expose constantly changing data as services
  - Visualize data as it is flowing - not tomorrow in a report
  - Raise awareness of issues as they are occuring
- **Evolving Applications**
  - Strong isolation and decoupling allow great flexibility to add/remove components
  - Replayability of past events allows new stateful components to ‘catch up’
  - Business details/patterns overlooked in early iterations can be run on past data to realize new insights

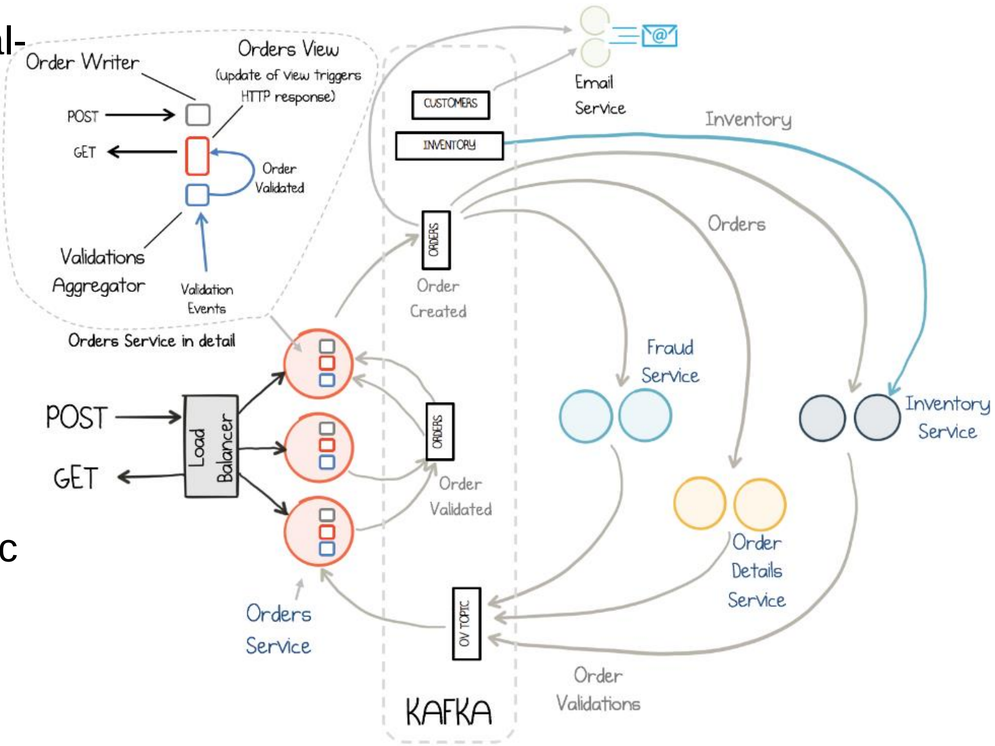


cedrus

Kafka/Event Stream Technical Deep Dive

Kafka Enables Event Source Architectures

- **An event source architecture is the only architecture that can never lose data**
  - A permanent log of all business interactions are automatically kept in an archival-quality log
- **We can (re)play history as we wish**
  - Recovery point objective (RPO) is achieved at per-message granularity
  - Subscribers need not be interested in the same position
  - New applications can be introduced years into production, but can fully absorb history, as it happened, at full system speed
  - Strange transient error based on unusual system state? Replay production topic into lower instrumented environment at your leisure!
- **Rich Ecosystem (for Java and many other popular languages)**
  - KStreams - A native, rich system for multi-node stream processing
  - KTables - A native system for building temporal database tables off the streams
  - KSQL - A runtime for building SQL like expressions and serving the data as REST



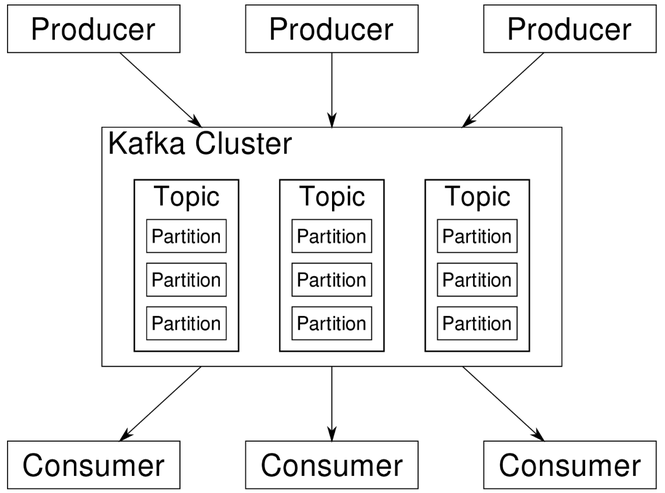
cedrus

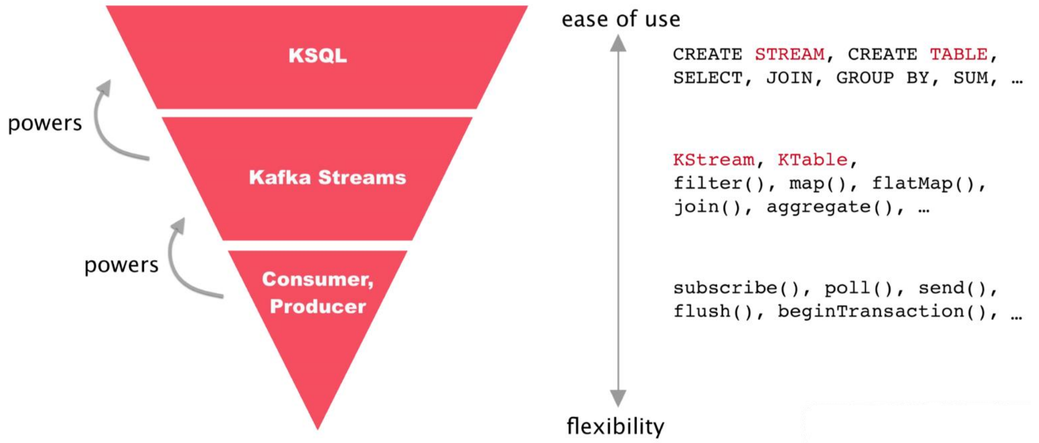
Ecosystem and Kafka

- **Kafka broker clusters do two main things, and does them well:**
  - Persist events in a reliable, fully replicated environment, even in containers
  - Securely serve events given a request with a partition and offset with failover and load balancing

- **Simplicity breeds speed and horizontal scalability**
  - By not putting much logic into the broker, it's easily replicated and scaled over many nodes for speed and failover

- **Everything else is the ecosystem!**
  - All high level functionality runs on the application side
  - Streams and Tables are an abstraction on topics' events
  - KTables can re-persist themselves via Kafka topics
  - Kafka libraries continue to improve in function and be written in more languages

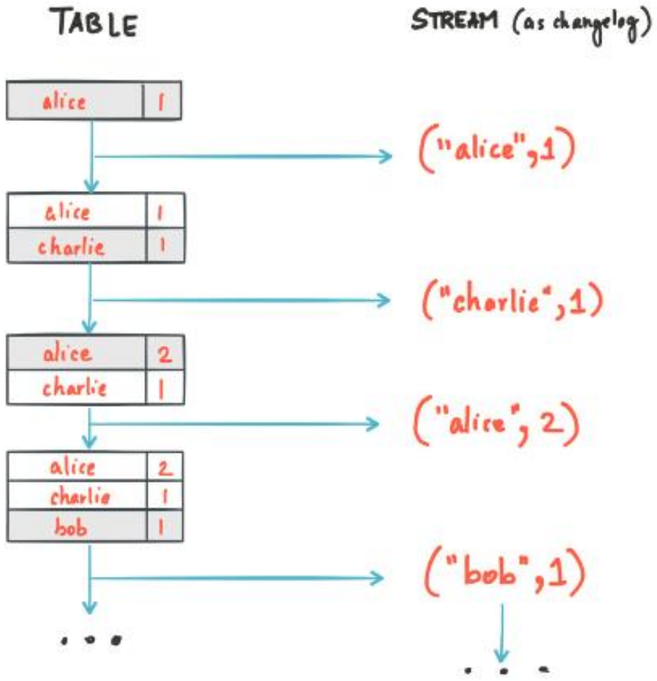


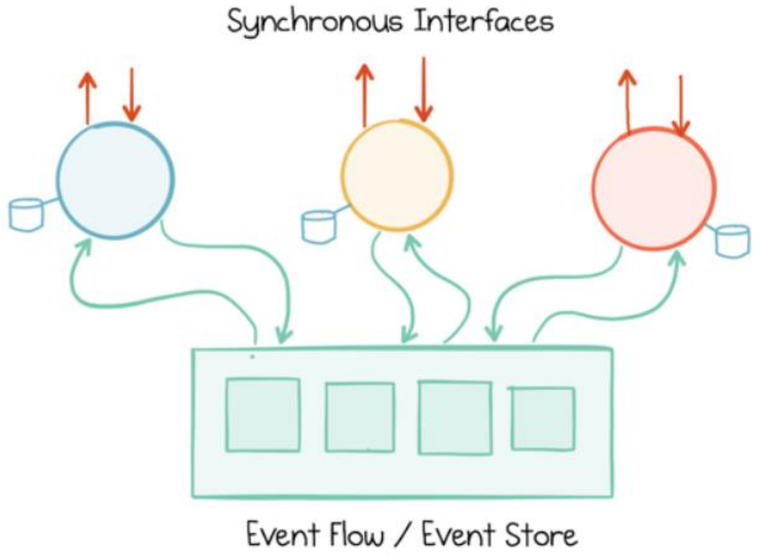


cedrus

KStreams / KTables - Streaming Microservices

- **Functional Aspects**
  - Microservices can utilize topic(s) as a stateful intermediary for event handling logic
  - Microservices advance the state of the data by appending new messages to topic(s)
  - KTables can be used to build real-time APIs based on streaming data
  - Message format independent (Avro, JSON, XML...)
  - Rich libraries for functional design of streaming applications
- **Advantages**
  - Stateful - Services may fail, but upon start, can continue processing backlog
  - Completely Decoupled - Applications do not know about each other
  - Asynchronous - Robust during load spikes, scales easily when needed
  - More extensible - Topics allow for seamless additions
- **Disadvantages**
  - Can’t handle any transactionality between stages
  - Use of stateful stream or table elements requires microservice state (Non 12-Factor)
  - Application latency can be varied





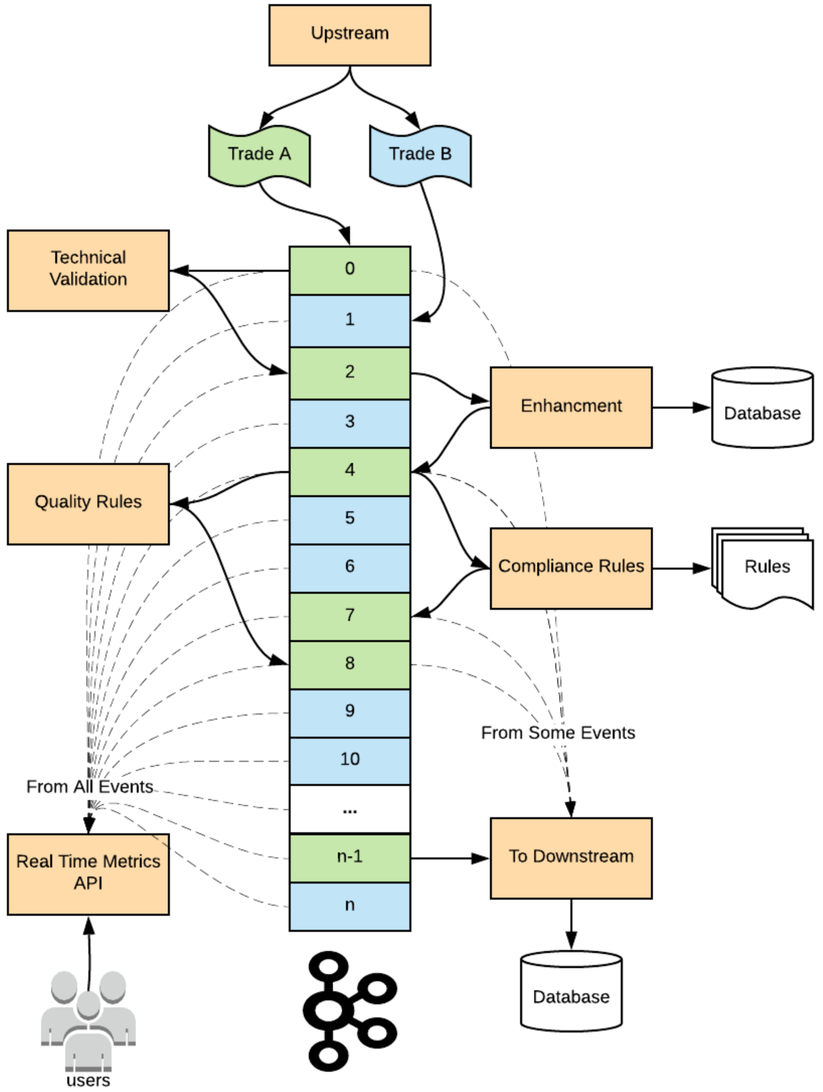
cedrus

Financial Institution Use Case

cedrus

Putting it all together - Major Financial Institution

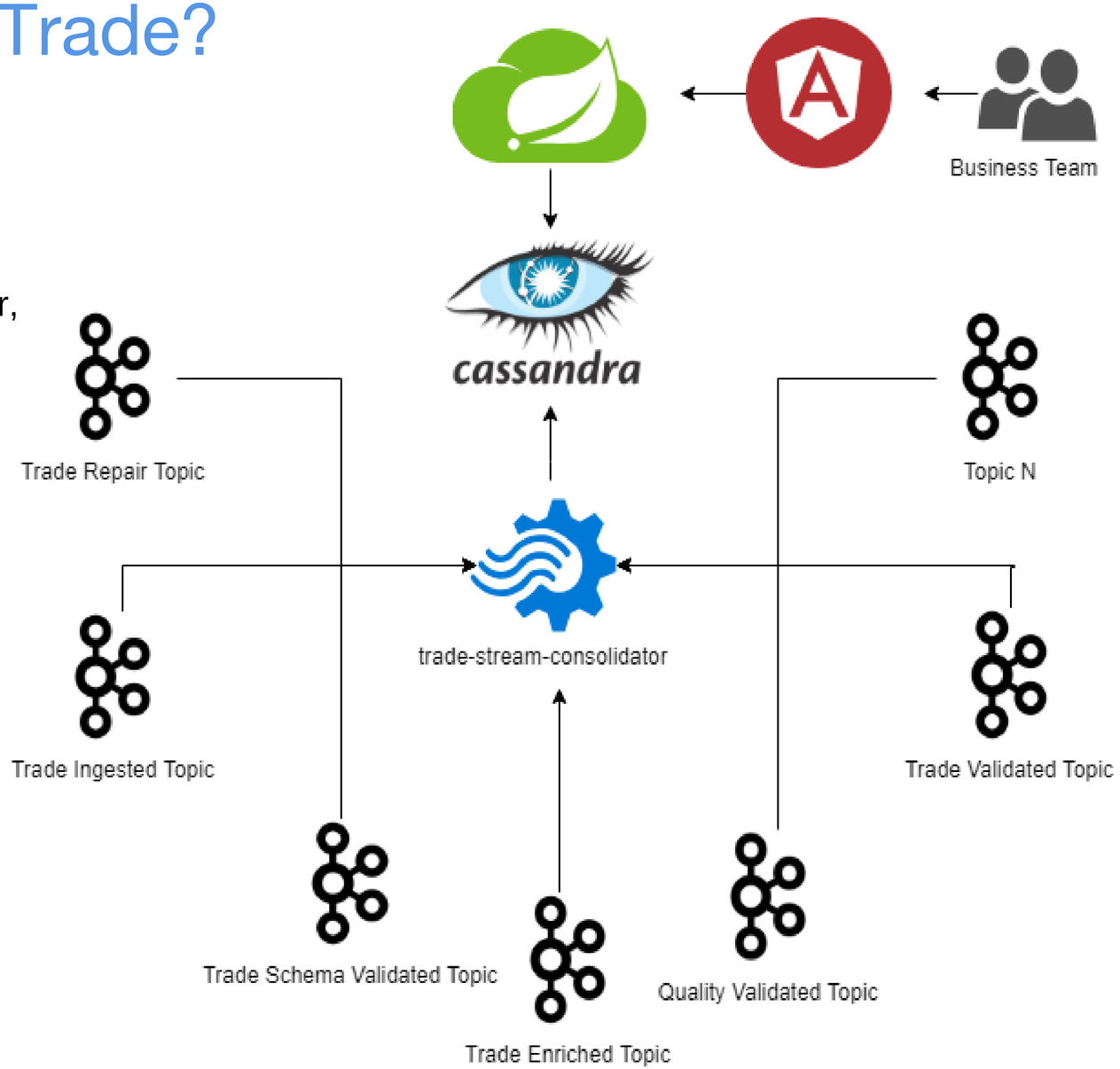
- **Company Profile**
  - Large investment bank
  - Security and Trust is paramount - Private cloud only
  - Trades billions of USD in assets regularly in over >10k of trades daily
- **Requirements**
  - Trade processing must be fully auditable
  - Trade processing stages have interdependencies
  - New stages must be able to be added without disruption
- **Solution**
  - Spring Boot, on IBM Cloud Private (Kubernetes)
  - Kafka events represent trades reaching statuses
  - Spring Boot microservices run KStreams to interact with events
  - Microservices call dependent microservices via Rest
  - Business rules exposed as their own microservices
  - Full state of a trade is the sum of all related events to date
  - Upon action, each microservice produces a new trade event on the topic



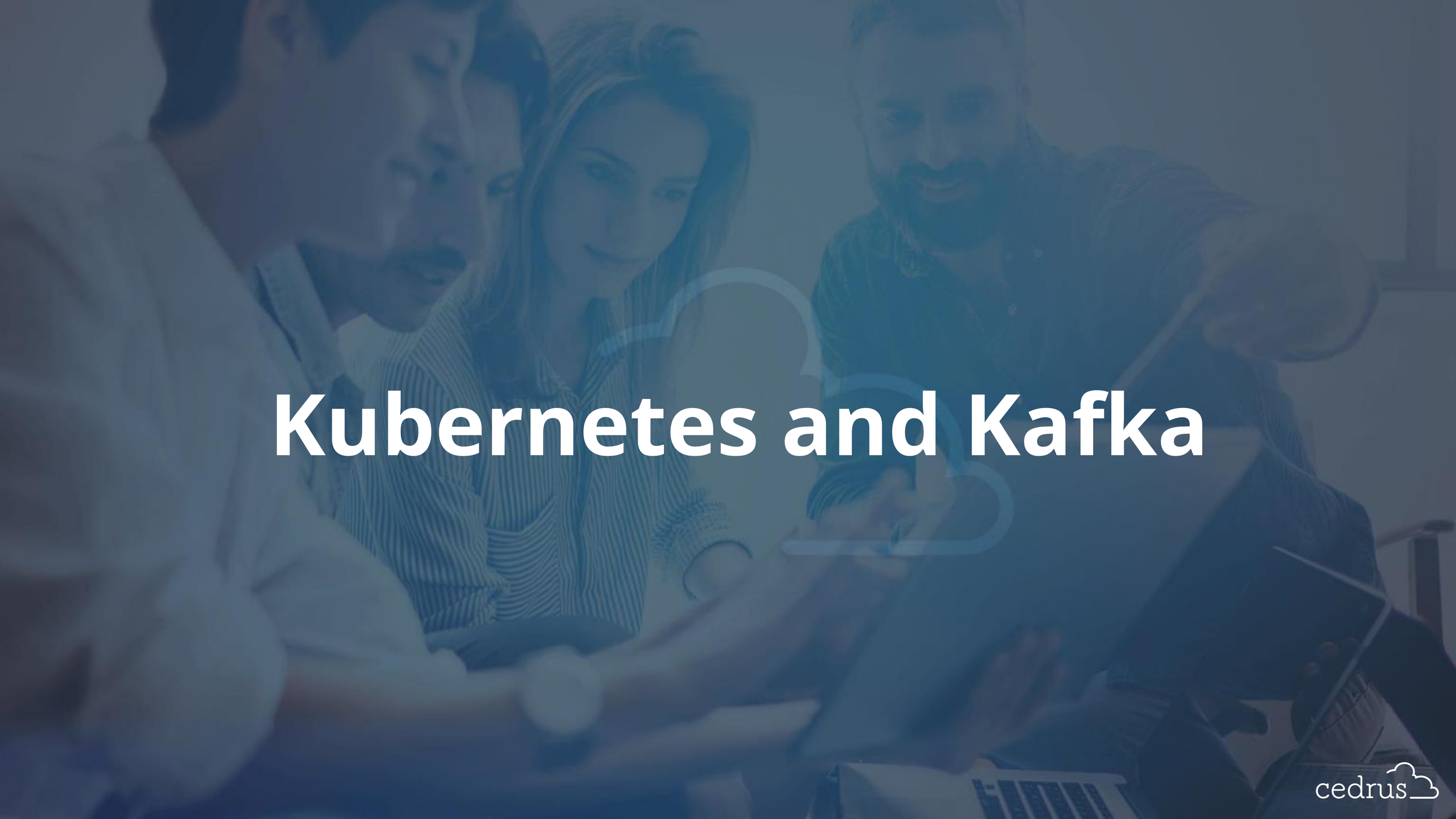
cedrus

Hybrid Architecture - Where's My Trade?

- **"Fedex Tracking Portal for Trades"**
  - Business team wants to see real-time status and history of trades
  - Lookup can be by trade ID or other aspects, such as the customer, product or time
  - >10k trades a day = ~100k order status records / day
- **Initial Solution**
  - Focused on Kafka-centric storage
  - Use KTables to introspect Kafka records, build state
  - Orders must be read into memory
  - Core application changes required, limited time window
- **Cedrus Solution**
  - Embrace Kafka as *topics*, Single Responsibility Principle
  - New subscriber to all topics feed Cassandra real-time
  - Angular front end queries Cassandra via Spring Microservice
  - Performance is excellent (~25ms),
  - New services / search criteria can be added very easily



cedrus

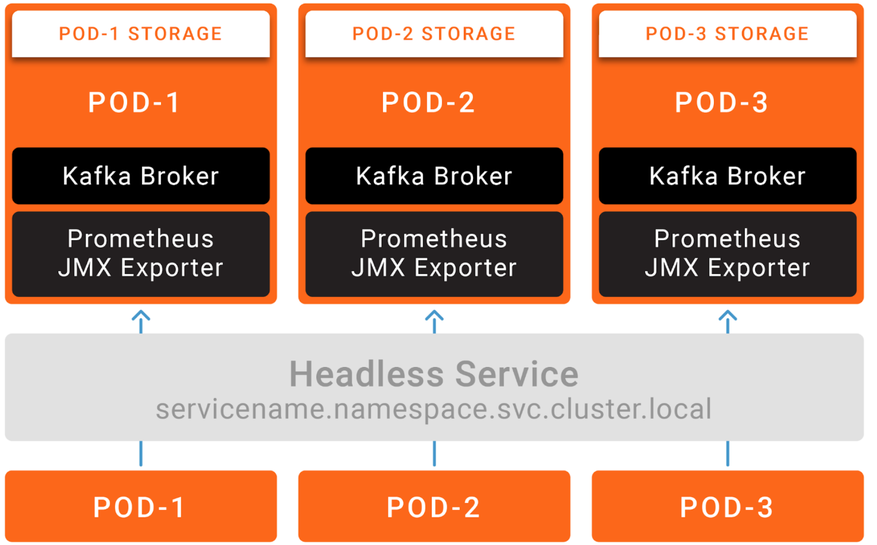


Kubernetes and Kafka

cedrus

Kafka on Kubernetes

- **Kafka's native replication benefits from K8s health checks**
  - Quick failure discovery and replacement
- **Kafka's filesystem is fully replicated and distributed**
  - Often companies are hesitant to put stateful apps on Kubernetes
  - Kafka broker clusters assign different partitions (shards of a topic) to different nodes
  - All other nodes have the events and know the consumer details
    - In the case of failure, an alternative node can take over instantly
  - This native replication allows for fast and reliable replacement
- **Brokers close to the producers/consumers**
  - Kafka serves events with a very low latency and high speed
    - It's best to be as close as possible to the microservices running in K8s to keep latency low and network bandwidth high
- **Infrastructure teams better at supporting a more homogeneous runtime**



cedrus

Scaling Best Practices - Infrastructure

- **Kafka applications are network IO intensive**
  - Reduce network hops as much as possible
  - Aim for a 10G or better network between all brokers and application nodes
  - No use of overlay networks like Flannel - only zero-overhead ones like Calico

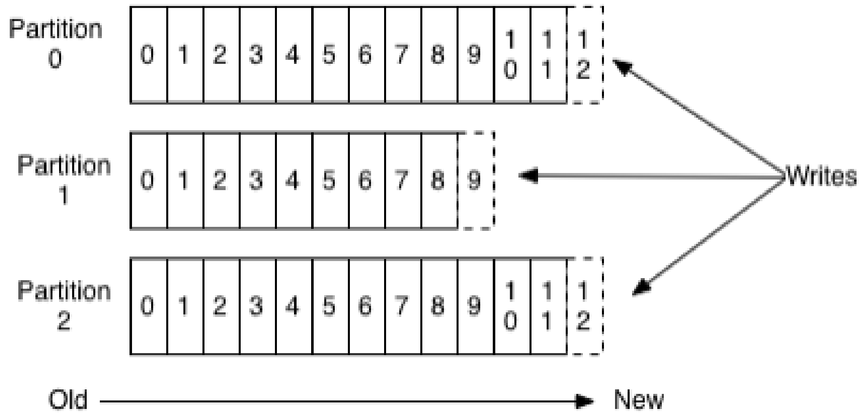
- **Kafka applications are disk IO intensive**
  - Every publish writes to multiple nodes, according to replication factor
  - Publish replication is limited by the slowest node
  - IO pattern is rather random - use appropriate storage
    - High random IO bandwidth, low latency

- **Use balanced replication settings**
  - Highly replicated environments will slow on writes and use more internal bandwidth
  - If you are highly replicated, consider a higher maximum lag factor
  - Monitor your application in production, and adjust accordingly

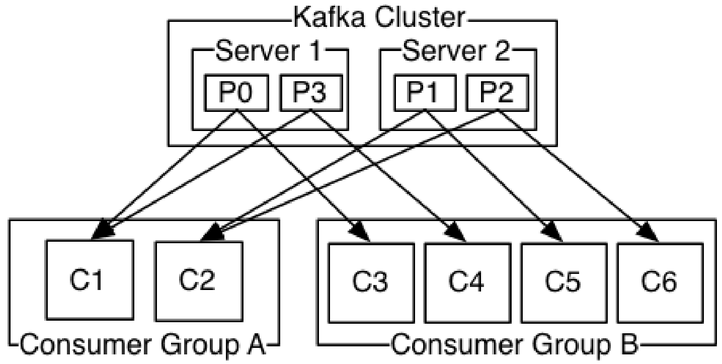
cedrus

Scaling Best Practices - Applications

- **Partition your data wisely***
  - Partitions shard your data, making it easier to parallelize activities
  - Smaller partitions allow for faster and simpler scaling and failover
  - KStream libraries will automatically parallelize on partitions
  - Too many partitions will slow the cluster and decrease reliability



- **Think about partitions when designing KStreams/KTables**
  - Keep in mind:
    - Producers determine partitions
    - Consumers get rebalanced when a consumer group's members change
  - Do you need stateful aggregation, if so, is it over partitions?
  - Try to isolate partitions, use stateless stream actions and avoid re-keying



- **Don't view Kafka as a panacea**
  - There is still a place for traditional databases and application patterns
  - Kafka is another tool in the IT toolbox

* https://www.confluent.io/blog/how-choose-number-topics-partitions-kafka-cluster

cedrus