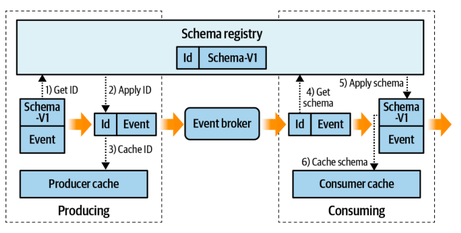
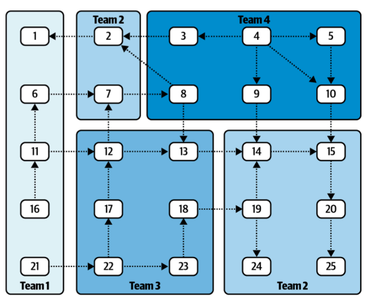
Adam Bellemare - Building Event-Driven Microservices

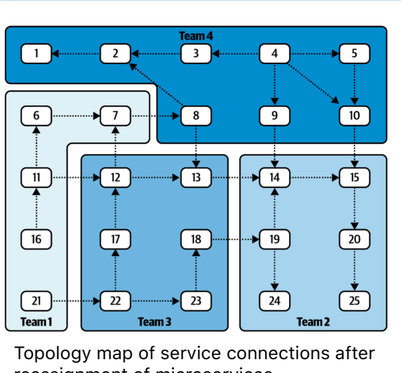
14. Supportive Tooling

- Microservice-to-Team Assignment System
  - It is important to explicitly track ownership of microservice implementations and event streams
  - ❓ Is it just a an information/documentation or some working settings in Kafka?
- Event Stream Creation and Modification
  - Teams will need the ability to create new event streams and modify them accordingly
- Event Stream Metadata Tagging
  - Stream owner (service)
  - Personally identifiable information (PII)
  - Financial information
  - Namespace
  - Deprecation
  - Custom tags
- Quotas
  - Quotas are generally established by the event broker at a universal level
  - You want to ensure at the very least that your entire cluster won't be saturated by one service's I/O requests
- Schema Registry
  - 

    Schema registry workflow for producing and consuming an event
  - Apache Kafka supports schemas in Apache Avro, Protobuf, and JSON formats
- Schema Creation and Modification Notifications
  - are delivered via specific schema stream and clients can subscribe to it
- Offset Management
  - Application reset: Resetting the offset
    - When you want to start from the beginning
  - Application reset: Advancing the offset
    - Sometimes your service just need the newest data
  - Application recovery: Specifying the offset
    - You can also start from specific point in time
- Permissions and Access Control Lists (ACL) for Event Streams
  - Can be READ, WRITE, CREATE, DELETE, MODIFY, and DESCRIBE
  - Typical event stream permissions for a given microservice

Component	Permissions for microservice
Input event streams	READ
Output event streams	CREATE, WRITE (and maybe READ, if used internally)
Internal and changelog event streams	CREATE, WRITE, READ
  - ⚠️ Discovering Orphaned Streams and Microservices
    - If a stream has no consumers, it may be marked for deletion
- State Management and Application Reset
  - Steps to reset the app
    - Delete a microservice's internal streams and changelog streams
    - Delete any external state store materializations (if applicable)
    - Reset the consumer group offsets to the beginning for each input stream!
  - ❓ Q: What if the stream is very big? Moreover if we cannot keep all the events? Shouldn't we start not from the beginning then, but from some snapshot?
- Consumer Offset Lag Monitoring
  - 👍 Consumer lag - the best indicator that microservice needs to scale up
    - We can scale up / scale down automatically
  - Monitoring systems
    - Burrow for Apache Kafka
- Streamlined Microservice Creation Process
  1. Create a repository
  2. Create any necessary integrations with the CI pipeline
  3. Configure any webhooks or other dependencies
  4. Assign ownership to a team using the microservice-to-team assignment system.
  5. Register for access permissions from input streams.
  6. Create any output streams and apply ownership permissions.
  7. Provide the option for applying a template or code generator to create the skeleton of the microservice.
- Container Management Controls
  - Container management service (CMS)
  - Teams should be able to DevOps (business decides what should be done by developers and what by Ops team)
    - Setting environment variables for their microservices
    - Indicating which cluster to run a microservice on (e.g., testing, integration, production)
    - Managing CPU, memory, and disk resources, depending on the needs of their microservices
    - Increasing and decreasing service count manually, or depending on service-level agreements and processing lag
    - Autoscaling on CPU, memory, disk or lag metrics
- Cluster Creation and Management
  - Programmatic Bringup of Event Brokers
  - Programmatic Bringup of Compute Resources
  - Cross-Cluster Event Data Replication
    - important for scaling up
  - Programmatic Bringup of Tooling
- Dependency Tracking and Topology Visualization
  - Can use self-reporting system
    - consumers and producers report on their own consumption and production patterns
      - 👎 it is effectively voluntary
  - 👍 Leveraging the permissions structure
    - microservice cannot operate without permissions
    - any changes in permissions update topology
  - uses of such a tool
    - Determine data lineage
    - Overlay team boundaries
    - Discover data sources
    - Measure interconnectedness and complexity
    - Map business requirements to microservices
  - Topology Example
    - Before change
      - 

        Topology map of service connections
      - Topology graph measure of interconnectedness

	Incoming streams	Outgoing streams	Incoming team connections	Outgoing team connections	Services owned
Team 1	1	3	1 (Team 2)	2 (Teams 2,3)	5
Team 2	8	2	3 (Teams 1,3,4)	2 (Teams 1,4)	8
Team 3	3	3	2 (Teams 1,4)	1 (Team 2)	6
Team 4	1	5	1 (Team 1)	2 (Teams 2,3)	8
    - After change
      - 

        Topology map of service connections after reassignment of microservices
      - New measure of interconnectedness; differences are shown in brackets

	Incoming streams	Outgoing streams	Incoming team connections	Outgoing team connections	Services owned
Team 1	1	3	1 (Team 3)	1 (Team 3) (−1)	5
Team 2	4 (−4)	0 (−2)	2 (Teams 3,4) (−1)	0 (−2)	6 (−2)
Team 3	3	3	2 (Teams 1,4)	2 (Teams 1,2) (+1)	6
Team 4	1	3 (−2)	1 (Team 1)	2 (Teams 3,4)	8 (+2)