Running a Massively Parallel Self-serve Distributed Data System At Scale

Zhenzhong Xu | @ZhenzhongXu
Real-time Data Infrastructure
Running a Massively Parallel Self-serve Distributed Data System at Scale

- What is Netflix Real-time Data Infrastructure
- Challenges
- Solutions and Principles
What is ...

... Real-time Data Infrastructure at Netflix
Business/Product Driven Analytics

- Recommendations / Personalization Algorithms
- Customer Experience
- Content Operation
- A/B Testing
- Marketing
- etc
Rise of Event Driven Architecture

- Notification
- Event Sourcing
- CQRS
- etc
We need a data platform that’s both scalable and real-time.
Data-driven Culture
So what exactly is Keystone Streaming Platform?
Publish, Collect, Move & Compute

event data in near real time @ Cloud Scale
Keystone is ...

... a collection of microservices & components

- Self Service UI
- Control Plane
- Producer API
- Pub/Sub Queuing Service
- Stream Processing Service
- Consumer API
Keystone is ...

... a single self-contained logical PaaS
Keystone is ... … a multi-tenants, self-serving tool
Keystone is ...

... a self healing, cloud failure tolerant service, guarantees at-least-once delivery semantics

... adaptable to changing environments
Putting together ...
... the decisions we made to build scalable, reliable and maintainable systems.
Challenge 1: Scale.
A Single Stream

How much data (in MB/sec) do you expect to produce at peak in this region?
380
Anatomy of a Single Stream

Event Producer → Queuing Service → Stream Processing Job → Sink
Separation of Concerns

- Separation of Messaging and Stream Processing services.
- Each service scales individually.
- Each service manages its own states.
- Independently manage service dependencies.
  - Kafka brokers on EC2
  - Streaming Service job on Titus Container Runtime
Titus Container Runtime

- Resource Provisioning
- Scheduling and bin-packing
- Capacity Guarantees
- Resource isolation
- Per container IP address (underlay via VPN)
Delivery/Processing Semantics

- At-most once
- At-least once
- Exactly once
At-least Once Processing Checkpointing

- Synchronous checkpointing through event loop
Exactly Once* Processing Semantics

- Lightweight Asynchronous Snapshot (Async Barrier Checkpointing)
Per Stream Monitoring & Alerting
Streams with Fanout
Logical Isolation
Logical Isolation

- Streams Level
- Deployments Level
- What about regional Island Isolation?
Total Bytes Out = (num Of Consumers + replication factor - 1) * Bytes In
Solve Consumer Fanout with Hierarchies
Total Infrastructure Scale

- 500+ Billion events generated per day
- 1+ Trillion events processed per day
- ~800 Topics
- ~1,800 Streams
- 4000+ Kafka Instances
- ~9,000 Stream Processing Containers
### Kafka Clusters

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Env</th>
<th>Region</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>wardney</td>
<td>prod</td>
<td>us-east-1</td>
<td>NORMAL</td>
</tr>
<tr>
<td>rooky</td>
<td>prod</td>
<td>us-east-1</td>
<td>NORMAL</td>
</tr>
<tr>
<td>robson</td>
<td>prod</td>
<td>us-east-1</td>
<td>NORMAL</td>
</tr>
<tr>
<td>himlayas</td>
<td>prod</td>
<td>us-east-1</td>
<td>NORMAL</td>
</tr>
<tr>
<td>everest</td>
<td>prod</td>
<td>us-east-1</td>
<td>NORMAL</td>
</tr>
<tr>
<td>helbert</td>
<td>prod</td>
<td>us-east-1</td>
<td>NORMAL</td>
</tr>
<tr>
<td>senali</td>
<td>prod</td>
<td>us-east-1</td>
<td>NORMAL</td>
</tr>
<tr>
<td>blanc</td>
<td>prod</td>
<td>us-east-1</td>
<td>NORMAL</td>
</tr>
</tbody>
</table>
Kafka Cluster Failover

Producer → Kafka Cluster → Router
Failback

Producer → Kafka Cluster → Router

Decommissioned Failover Cluster
Pet vs Cattle
Immutable Kafka Clusters?
Principles:
Failure as a First Class Citizen
Separation of Concerns
Embrace Immutability
Challenge 2: Self-serv & Multi-tenants
Diverse Customer Requirements

- Diverse combination of features
- Diverse platform tradeoffs
“Change is the Only Constant”
Change Is the Only Constant

- New streams deployed in a few mins
- Customer changing needs
- Scaling activity
- Infrastructure Upgrades
Failure Modes

- Infrastructure Disaster
- Any component can become temporarily unavailable
Provide Building Blocks
Declarative Reconciliation
Declarative Reconciliation

- “Declarative” is a communication pattern
- “Reconciliation” to drive the entire system towards goal
Declarative Reconciliation

- Goal States
- Current States
Layered Reconciliation
Declarative Reconciliation

- Goal States
- Current States
- State Machine Driven Reconciliation
State Machine
Goal State Driver
Single Source of Truth

Allows Eventual Consistency

Convergence on Goal State
Single Source of Truth

Allows Eventual Consistency

Convergence on Goal State
Principle:
Leverage Reusable Building Blocks
Declarative Reconciliation
Single Source of Truth
Thought Experiment:

Kitchen Management vs Distributed Architecture?
Thank you.

References:
https://medium.com/netflix-techblog


We’re hiring - http://bit.ly/NetflixSPaaS

@ZhenzhongXu (tweet me questions!)