BigQ

Manejando 3500 millones de mensajes por día

Agosto 2018

¿Por qué construir un servicio de mensajería in house?

- Estandarización
- Simplificar tareas de administración
- Evitar Lock In
- Features a medida (rate limit, filtros, calendarizado)

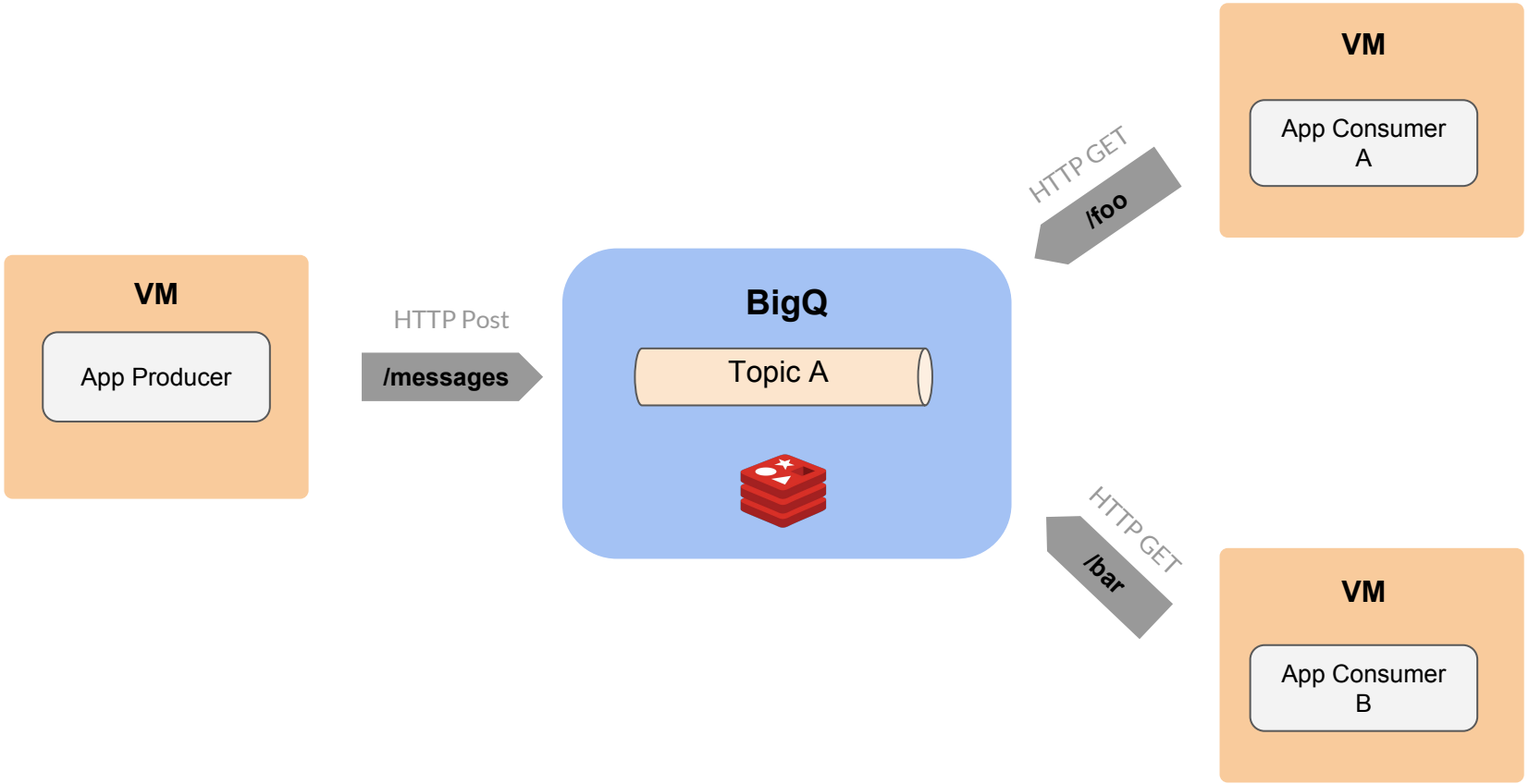
Entonces …. ¿Qué es BigQ?

- Un sistema de mensajería distribuido para comunicación asíncrona entre apps
- Basado en los conceptos de Tópicos, Producers y Consumers
- Alta disponibilidad
- Alto throughput
- Construido actualmente sobre Apache Pulsar (R) y con posibilidad de ser migrado a futuro de forma transparente

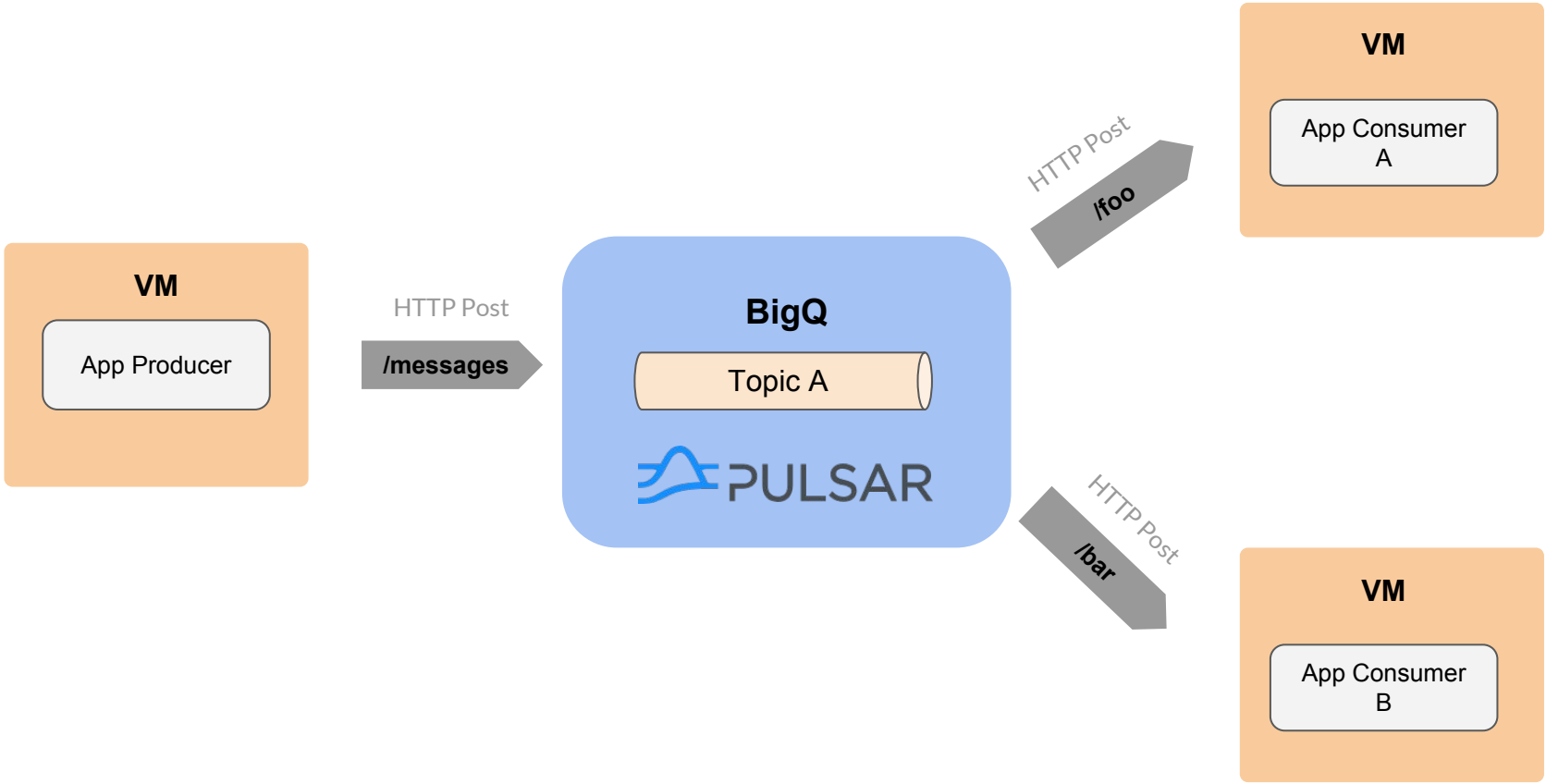
Algunos números

- +3500M de mensajes producidos al día, con picos de 3M por minuto
- Tiempos promedio de producción de 3 milisegundos
- 1000 tópicos
- 2600 consumers
- 7 Clusters
- 350 VM’s

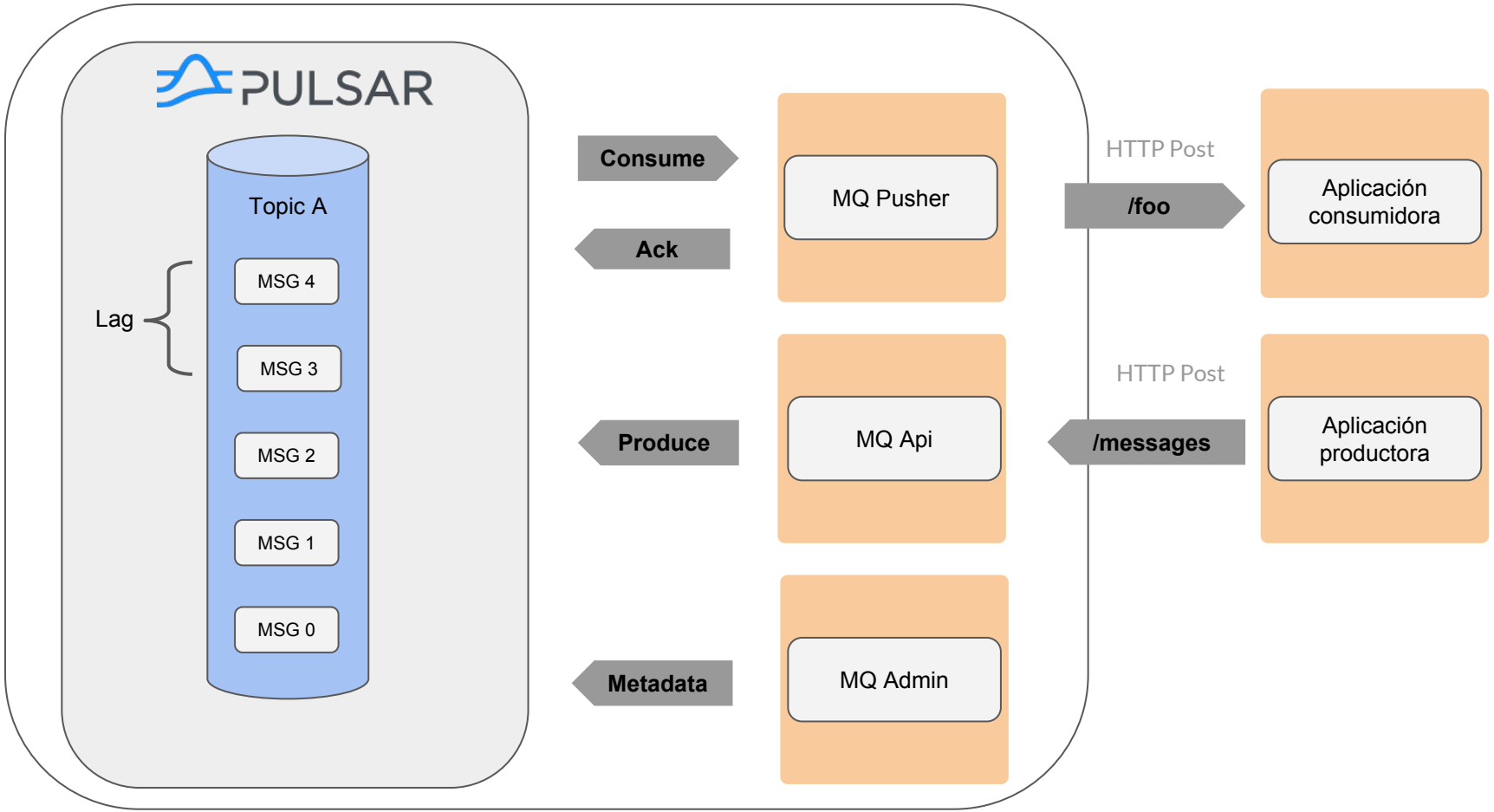
Orígenes de BigQ



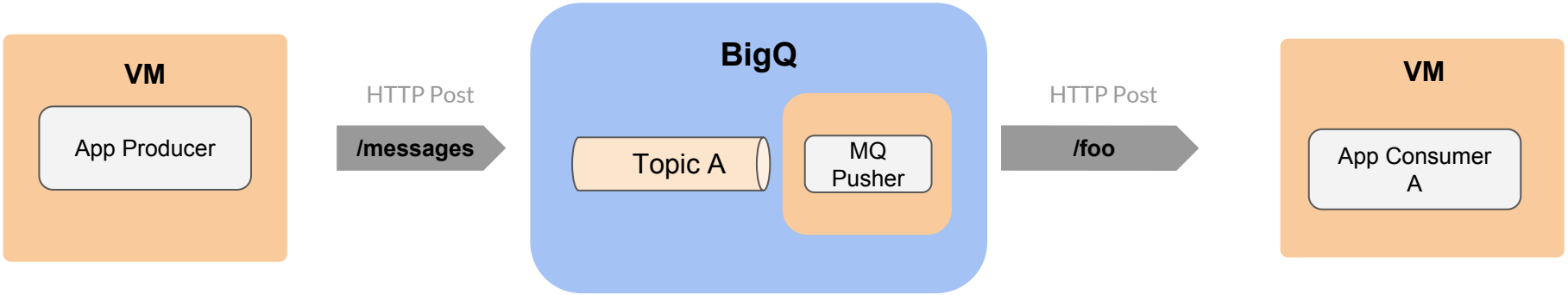
Nuevo modelo de Push de mensajes



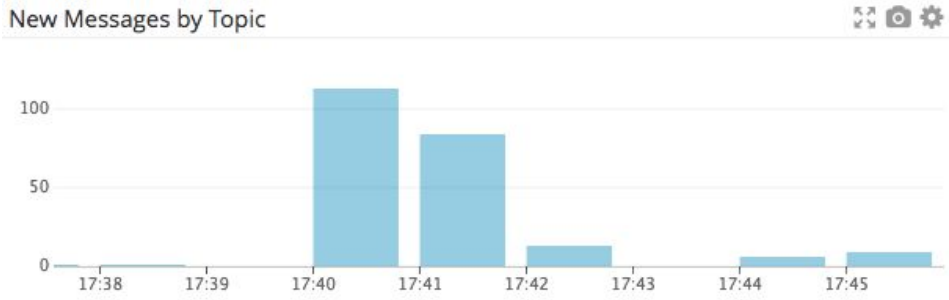
Arquitectura Alto Nivel

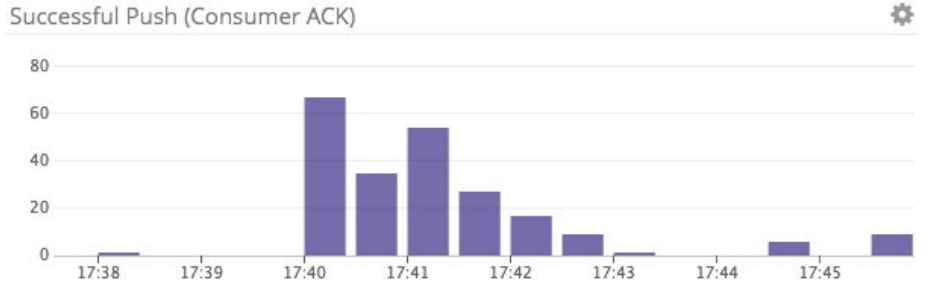


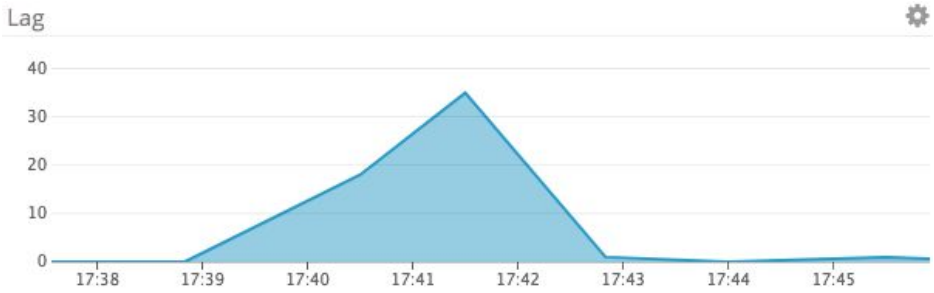
Gestionando el Lag de 2600 consumers

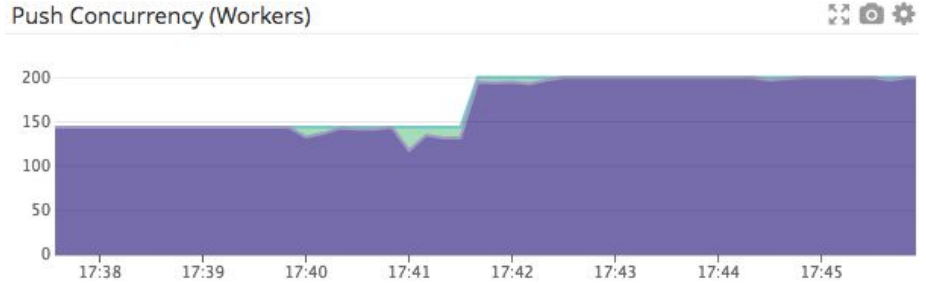


Gestionando el Lag de 2600 consumers

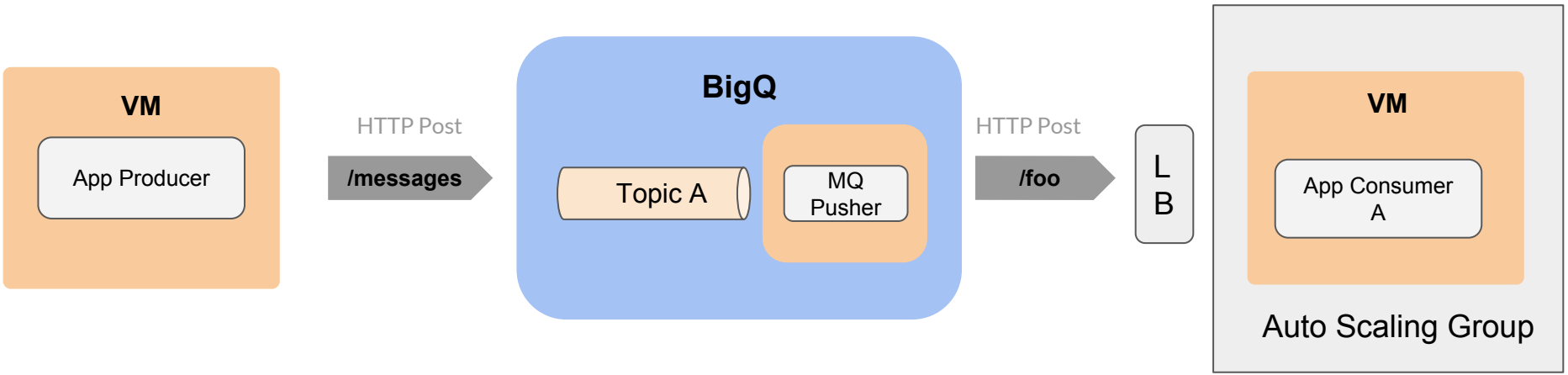




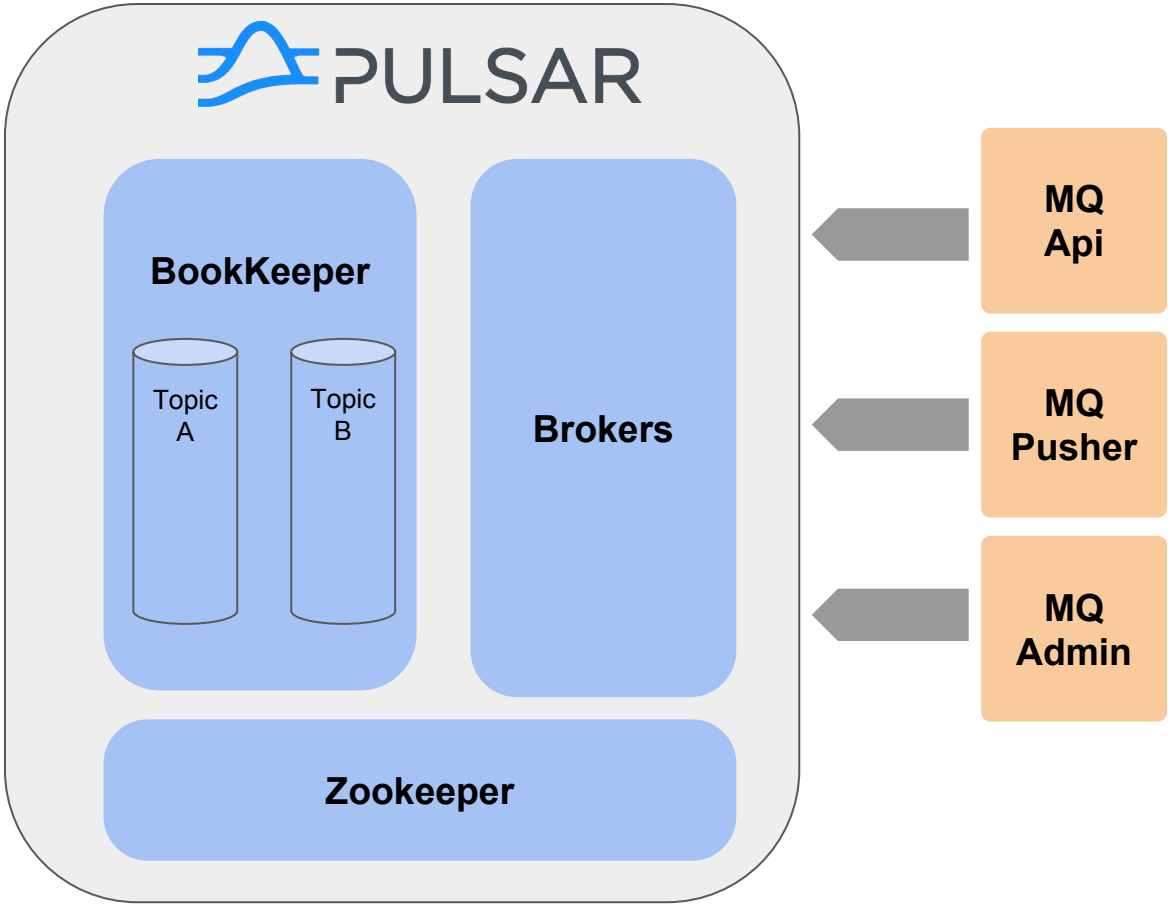




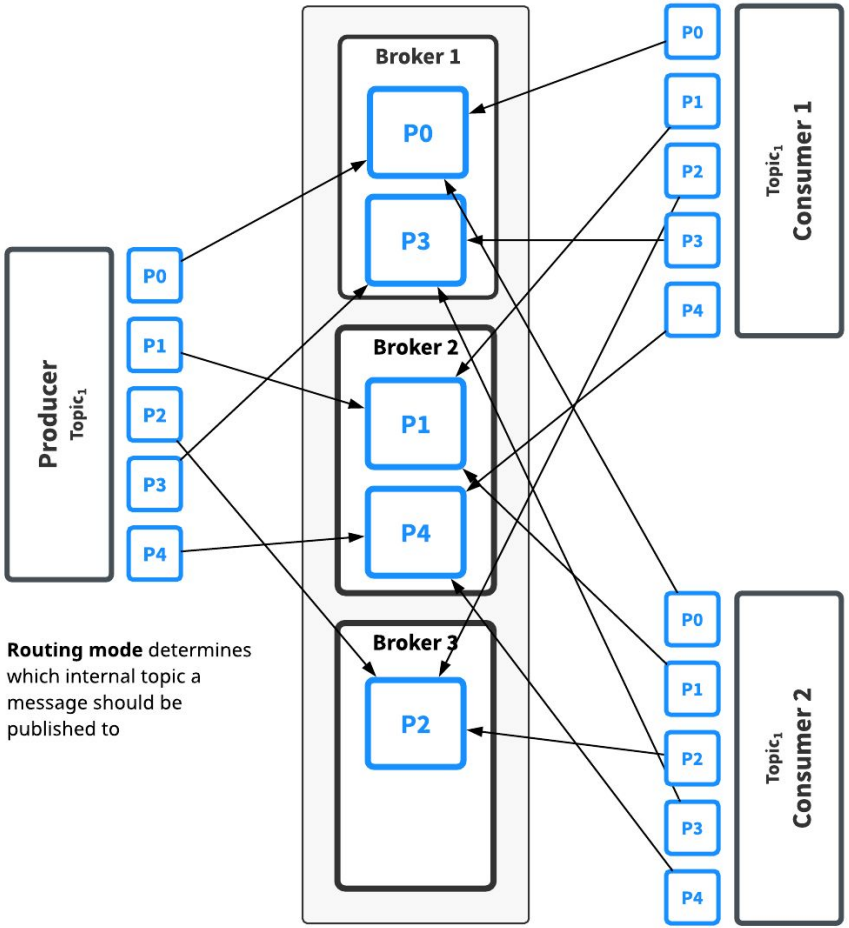
Gestionando el Lag de 2600 consumers



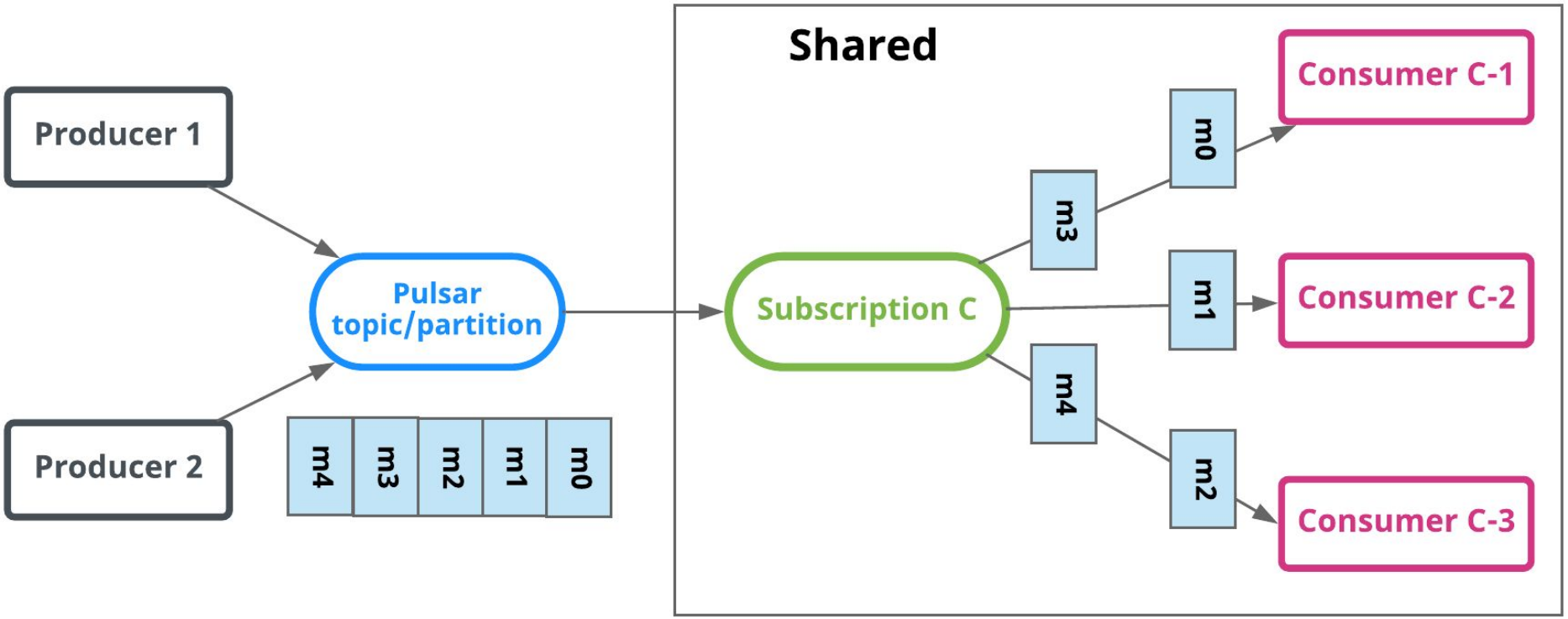
Apache Pulsar



Partitioned topics



Shared subscriptions



Ventajas de Pulsar respecto a Kafka

- Shared subscription: reparte el consumo entre varios clientes
- Kafka requiere balancear data si agregamos brokers
- Si un broker de Kafka muere, es necesario replicar sus particiones por completo, leyendo solo de sus réplicas
- Selective ACKs (no solo offset)
- Provee una Reader API para consumir como log distribuido

Ventajas de Pulsar respecto a RabbitMQ

- Soporta múltiples suscriptores para un mismo topic. RabbitMQ mantiene una queue por cada subscriber

- Mayor throughput (+100k/sec)

- Replay messages

Garantías del servicio

- Garantía de delivery (At least once)
  - ACK de mensajes
  - Timeout de 30 segundos
  - Re-delivery en caso de error o timeout
- Pueden existir mensajes duplicados
- No se mantiene el orden de los mensajes
- TTL de los mensajes configurable.

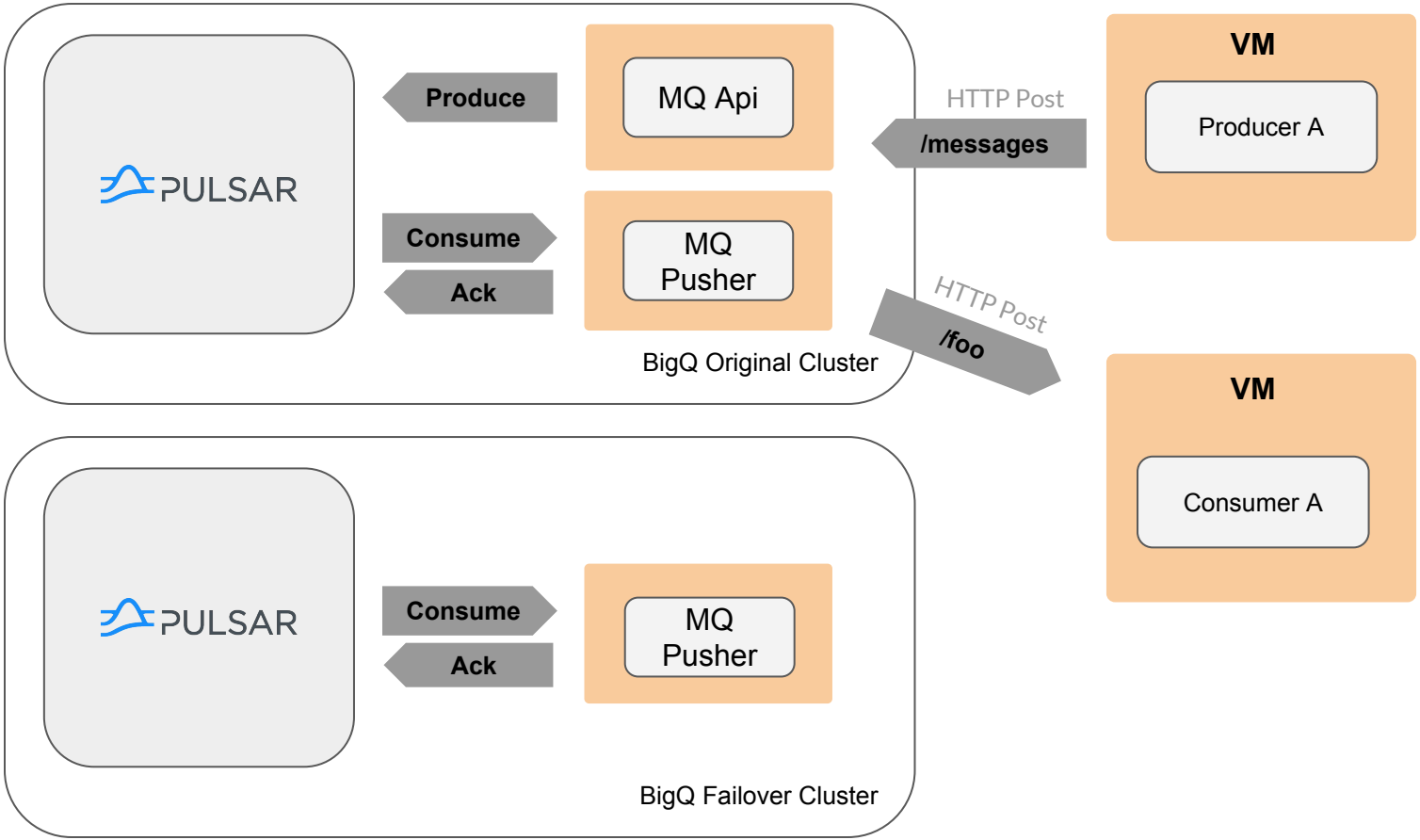
Resiliencia

- Atomicidad en la escritura de un mensaje (3 réplicas)
- Segmentos (ledgers) distribuidos en todos los bookies
- Si un broker falla, otro toma el ownership de sus particiones casi al instante
- Si un bookie falla,
  - Los brokers dejan de elegirlo para lecturas y escrituras
  - Un proceso de replicación mantiene las 3 copias
- Zookeeper para coordinación: 5 servers (toleran hasta 2 fallas)

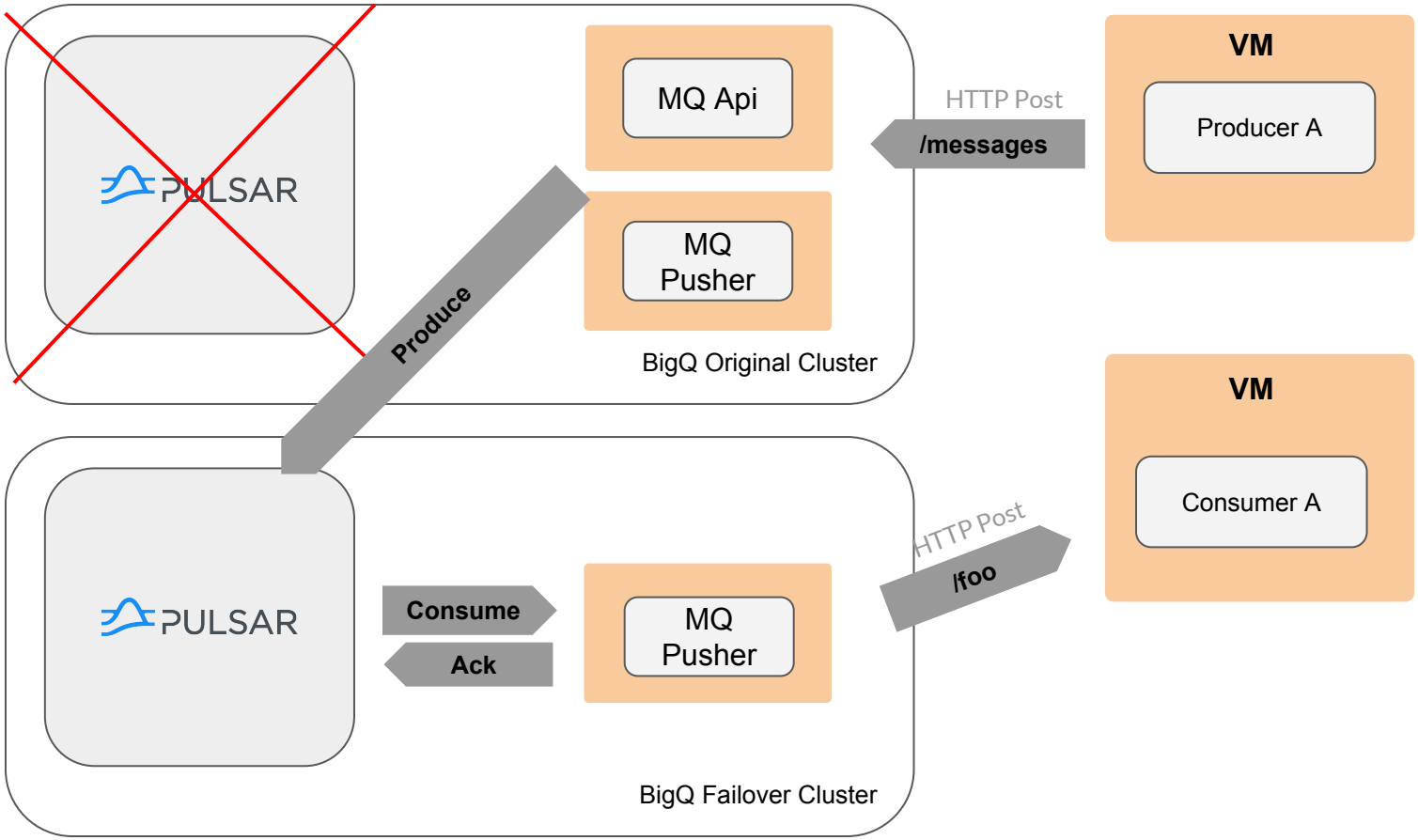
Resiliencia

- Geo-replicación
- AWS autoscaling groups para los MQ services y las apps consumidoras
- Rate limiting en la producción
  - Evita que un producer degrade al cluster
- Rate limiting en el consumo
  - Protege a recursos externos de las apps consumidoras
- Cluster failover (a continuación)

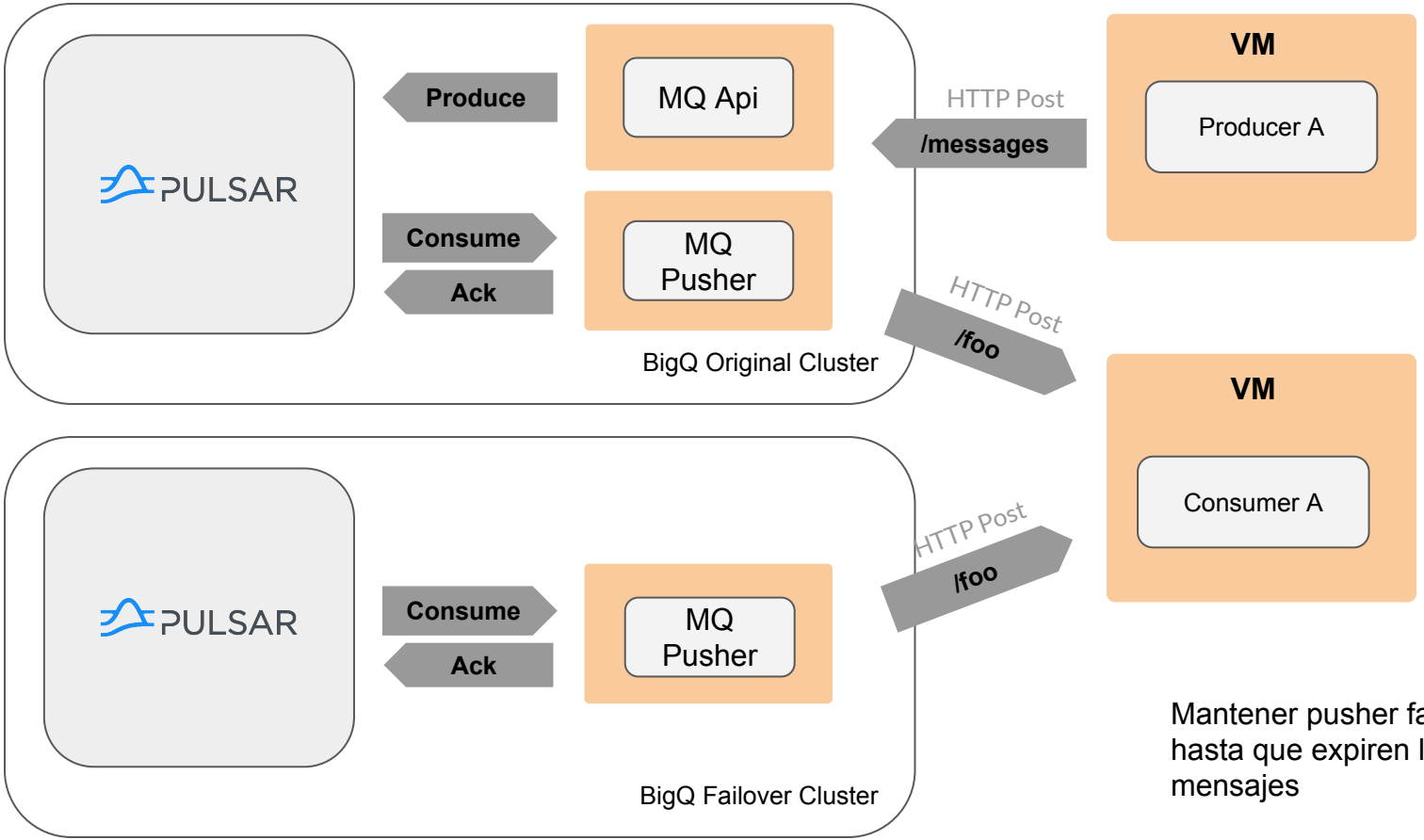
Failover: inactive



Failover: Pulsar cluster down



Failover: Pulsar cluster recovered



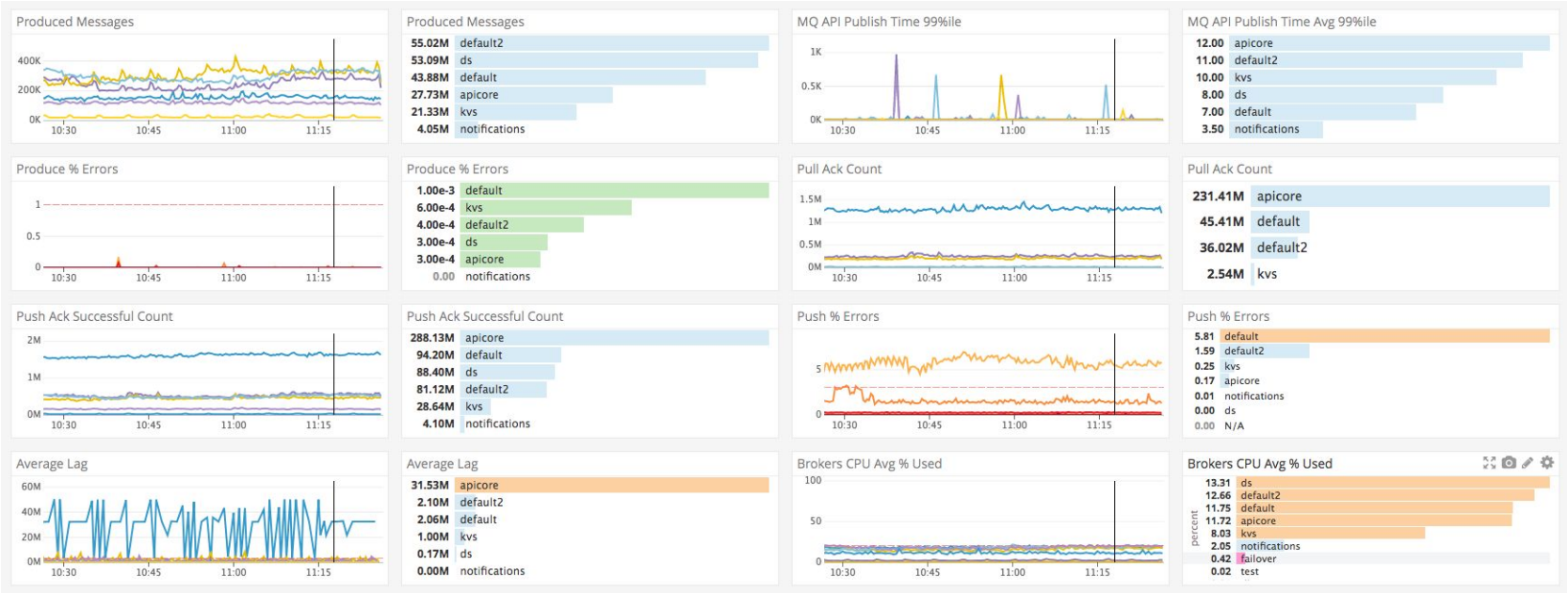
Mantener pusher failover hasta que expiren los mensajes

Monitoring

- Logs
  - Collector en cada máquina
  - Elasticsearch + Kibana

Monitoring

- Metrics
  - Datadog dashboards
  - Datadog monitors + OpsGenie alerts



Muchas gracias