SPARKPOST

Getting More Sleep
One SQS Message at a Time

Bob Evans & Jason Sorensen

Who we are?

Bob Evans

Director, Software Engineering

Jason Sorensen

Lead Data Scientist,
Architect for the Omni-ETL project

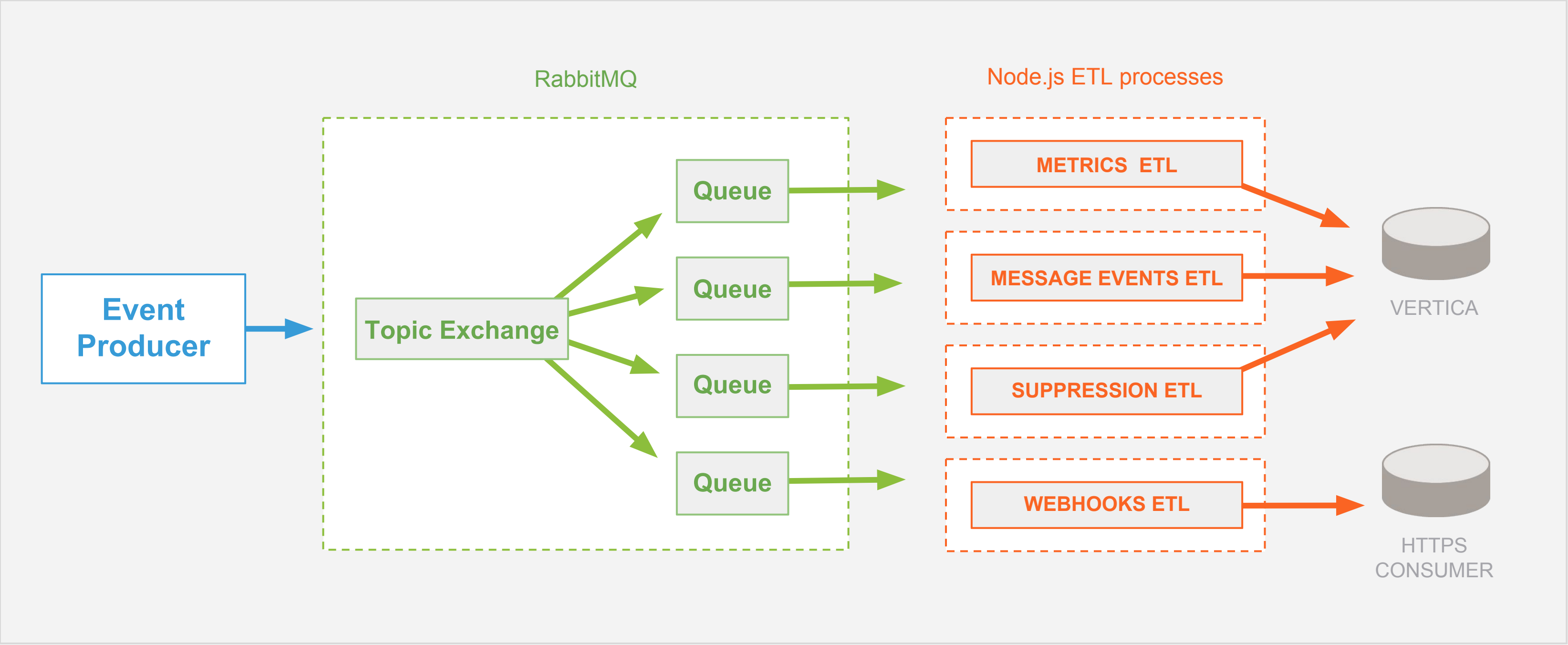
What we do?

- fastest-growing cloud email API service.
- delivers over 25% of the world’s non-spam emails.
- only independent email service build natively for the cloud on AWS.

Event Processing

- Core email platform generates JSON events for anything related to an email
  - injection, delivery, bounce, spam complaint, open, click, etc
  - Streams to RMQ exchange via the event hose
- Metrics ETL/API
  - Strip down data for long term aggregate and time series reporting
  - Stored in Vertica
- Message-Events ETL/API
  - Enriches, batches and stores raw JSON data
  - Stored in Vertica
- Webhooks ETL
  - Enriches, batches and transmits data
  - POST to customer's HTTPS endpoints
- Suppression ETL/API
  - Transforms certain bounces, spam complaints
  - Stored in Vertica (now uses DynamoDB)

Architecture Per Server



Headaches

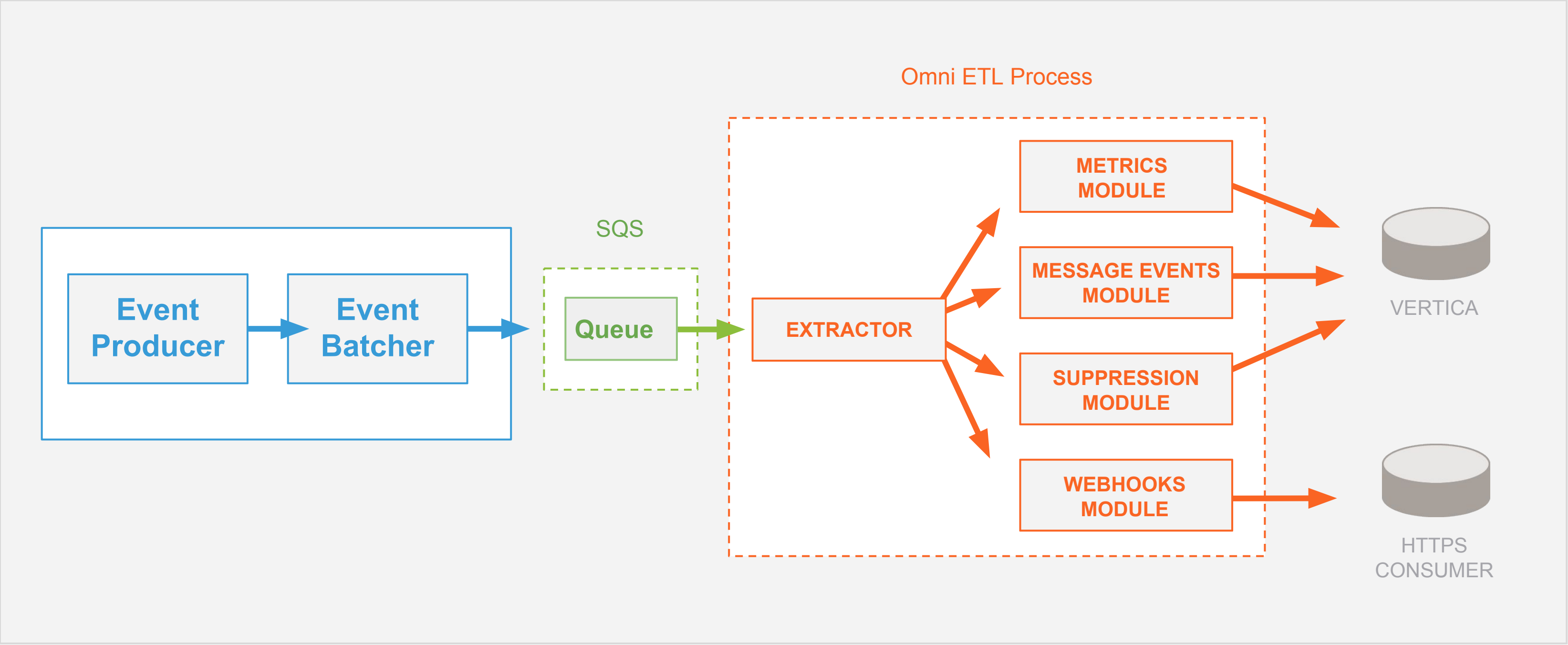
- Architecture was aligned with our on-premise product Momentum
- Under utilized node.js processes during non-peak
- Too many node.js processes on too many servers
    - Hard to troubleshoot and fix problems fast
- Expensive EBS disk volumes needed for RabbitMQ
- Fire drills during queue backups

What were the problem constraints?

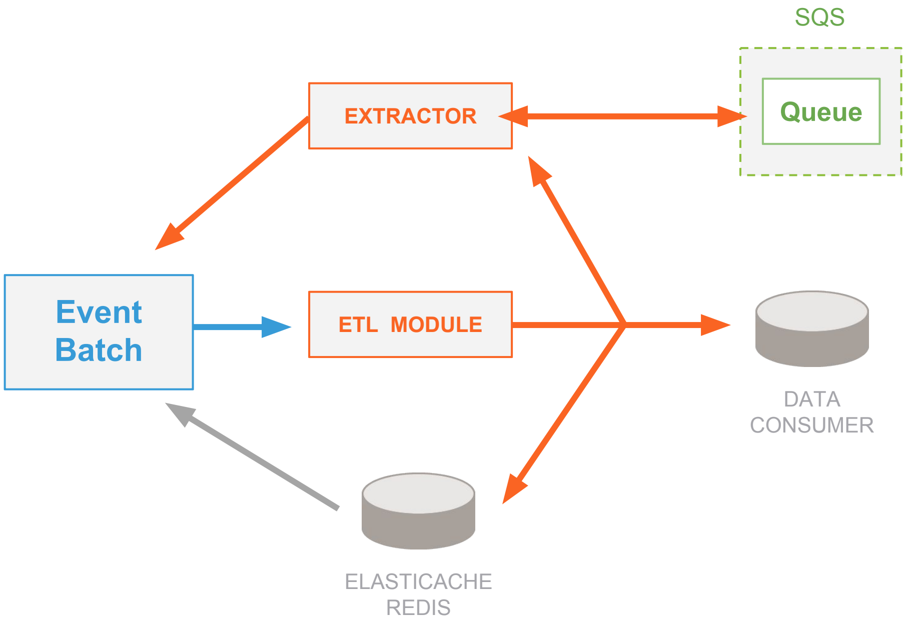
- Easier to manage
- Cost effective
- Auto-scalable
- Reduce risks during queue backups
- Fault tolerant to any service outage
- Near Real-Time visibility of data
- Backwards compatible

Omni-ETL

Omni-ETL Shared Architecture



ETL Module Coordination



RabbitMQ vs SQS

- **RabbitMQ**
  - Single uncompressed raw “event” per message published to exchange
  - One queue per data consumer
    - Requires persistent storage per queue for reliability
    - Data is copied onto disk
    - Queues are FIFO
    - Analogous to TCP
  - Pushes data to consumer
- **SQS**
  - Compressed batches of 750 events published to queue
  - Single queue for all consumers
    - Not guaranteed FIFO
    - At least once delivery
    - Analogous to UDP
  - Consumer must poll for data

SQS Event Batching

- SQS Publishes Limited to 256 KB
- Speed Testing 28,000 Events Published to RabbitMQ vs SQS

Publish Method	Time
RabbitMQ Single Events Sync	8.8s
SQS Single Events Async	143s
SQS 10-Event Batch Sync	58.4s
SQS 10-Event Batch Async	23.5s
SQS 10x100 Compressed Batch Async	7.6s
SQS 1000 Compressed Batch Async	4.0s

- SQS Cost Per For One Email’s Events: $.0000000095 ($9.50 per Billion)

Transforming ETL Code into Omni

- Created unified “etl-base” module
- Extractor for SQS. Extractor for legacy RabbitMQ service
- Extractor feeds into consumer modules
  - One stand alone module for legacy ETL processes
  - SQS extractor feeds into many consumer modules
  - Consumer calls the transformer then batches
  - After batch consumer calls loader
  - Loader calls callback function triggers extractor acknowledgement
- Shared Loader Types for writing to Vertica, HTTP, SQS, etc
- Each module has a unique transformer

Architecture Choices

- **SQS**
  - 750 raw events per batch = faster, cheaper
  - Much cheaper than persistent (“safe”) RabbitMQ
  - Also useful as a delay queue
  - Abstractably analogous to RabbitMQ operation for ETLs
- **Redis for state management**
  - Previously tried node child process based coordination
  - Used LevelDB for some old architecture
  - ElastiCache Redis is cost-effective and broadly applicable

Rollout

- 24/7/365 service, 0 downtime
- Nothing beats testing in production….sensibly
  - dual load events to RMQ and SQS
  - standup test schema
  - blackholed webhooks
  - verify counts between live and test schema are aligned
- Ensured no lost or duplicated event data
  - cutover times that load/drop data on relevant architectures
- Monitoring and alerts in place well before cutover
- War room collaboration during rollout for a customer

Savings

- OLD: 1099 node.js processes across 157 servers
- NEW: 161 node.js process across 5 servers
- $7,000 month on EBS disks
- reduced servers and sizes of EC2 instances

Brand new world

Before

- Downstream backups cause queues to back up, resulting in delayed event data and severely impacting timely email delivery

After

- Downstream backups cause queues to back up , resulting in delayed event data, but other data keeps flowing

Takeaways

- Service based queues take stress off your infrastructure
- Running features dark in production best performance test
- Try different models to reduce costs on SQS
- Re-evaluate your stack quarterly
- Be incremental in your changes
- Have a well defined rollout plan
  - involve all relevant teams
  - explain the impacts of monitoring and alerting

Questions?