AWS re:Invent

DAT205

How Verizon Media implemented push notification using Amazon DynamoDB

Jon McCamant
Sr. Solutions Delivery Manager
Amazon Web Services

Suneel Joshi
Sr. Technical Acct Manager
Amazon Web Services

Bob Leano
Sr. Principal SW Engineer
Verizon Media

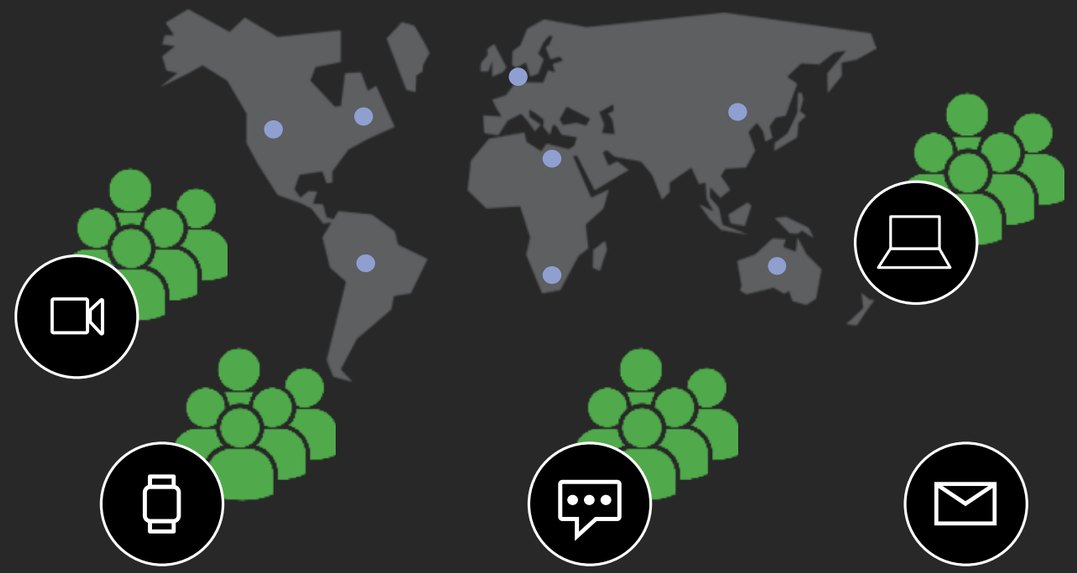
Harshil Shukla
Sr. SW Engineer
Verizon Media

Agenda

- Overview of Amazon DynamoDB
- Amazon DynamoDB global tables & Amazon DynamoDB Streams
- Verizon Media (A Verizon Company) launches Mobile Push Notifications
- Global deployment architecture & scale
- Lessons learned
- Results achieved

Overview of Amazon DynamoDB

Characteristics of Internet-scale apps

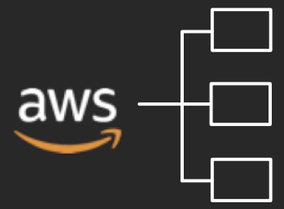


Relational | Key value | Document | Graph

Users	1 million+
Data volume	TB, PB, EB
Locality	Global
Performance	Milliseconds, microseconds
Request rate	Millions
Access	Mobile, IoT, devices
Scale	Up and down
Economics	Pay as you go
Developer access	Instant API access

Amazon DynamoDB

Fully managed nonrelational database for any scale



Fully managed

- Maintenance-free
- Serverless
- Auto scaling
- Backup and restore
- Global tables

High performance

- Fast, consistent performance
- Virtually unlimited throughput
- Virtually unlimited storage

Enterprise ready

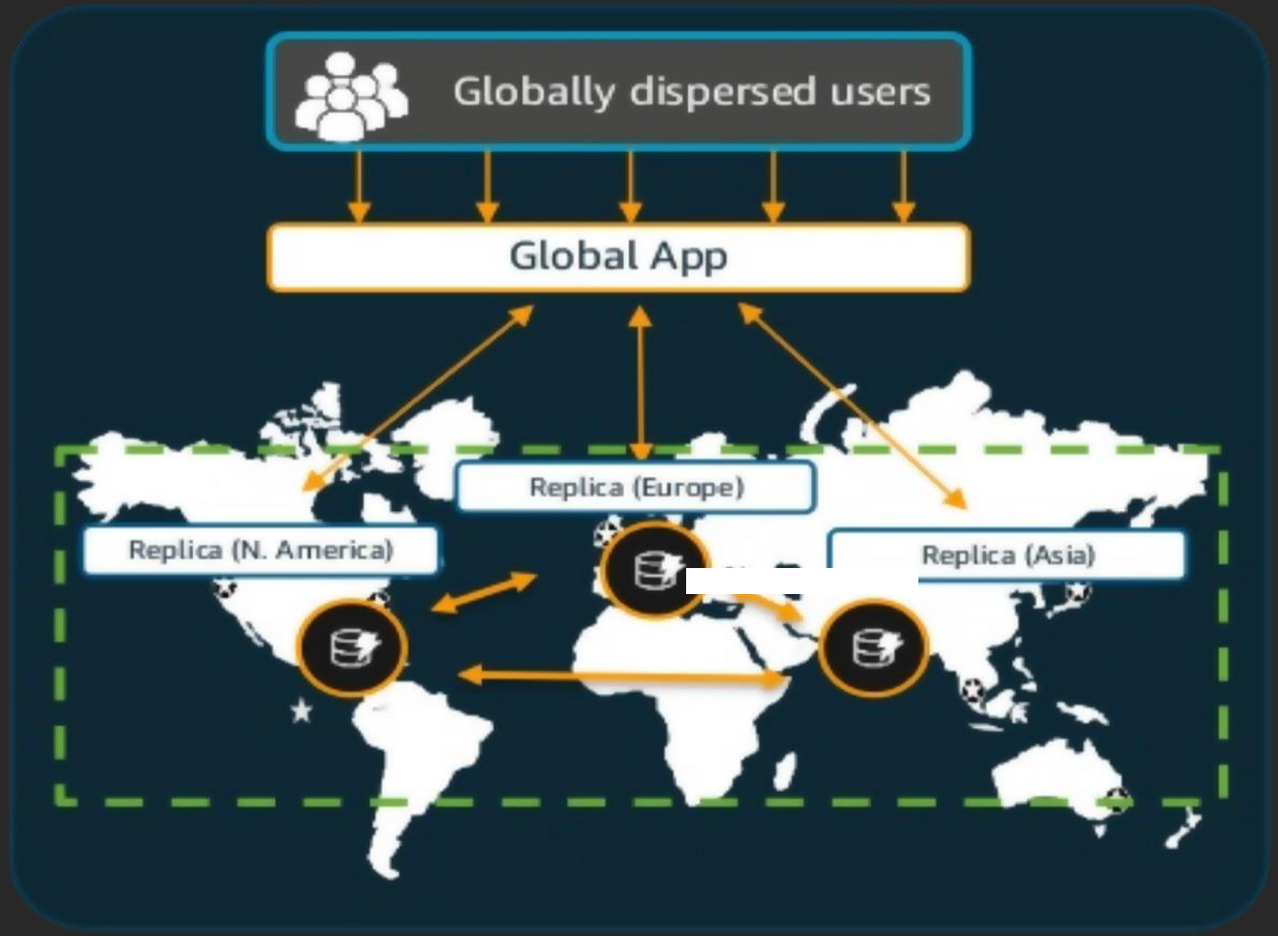
- Encryption at rest and transit
- Fine-grained access control
- PCI, HIPAA, FIPS140-2 eligible
- Service-level agreement

Amazon DynamoDB Global Tables & Amazon DynamoDB Streams

AWS re:Invent

aws

Global apps, global users with local performance



- Customers choose the regions where they want their data to replicate to
- Decisions based on needs of the applications and the locations of the users
- Same data model as ordinary table, same API
- Applications control existing endpoints to perform read and write operations
- Tolerate regional disruption
- Achieve high performance

New version of DynamoDB Global Tables
(Version 2019.11.21)

Benefits of New Version:

Convert your existing single-region tables to Global Tables

- Add a hot standby for Disaster Recovery purposes
- Extend tables to new AWS regions for fast local performance

Add new AWS regions to existing Global Tables

Faster and less expensive replication

- Only one write will occur in each region of the global table
- Reduce Global Tables costs by up to 50% compared to earlier version

For Global Tables created before 11/21/2019, Upgrade tool will be available in few weeks

DynamoDB Streams

Build ordered, near real-time data processing capabilities

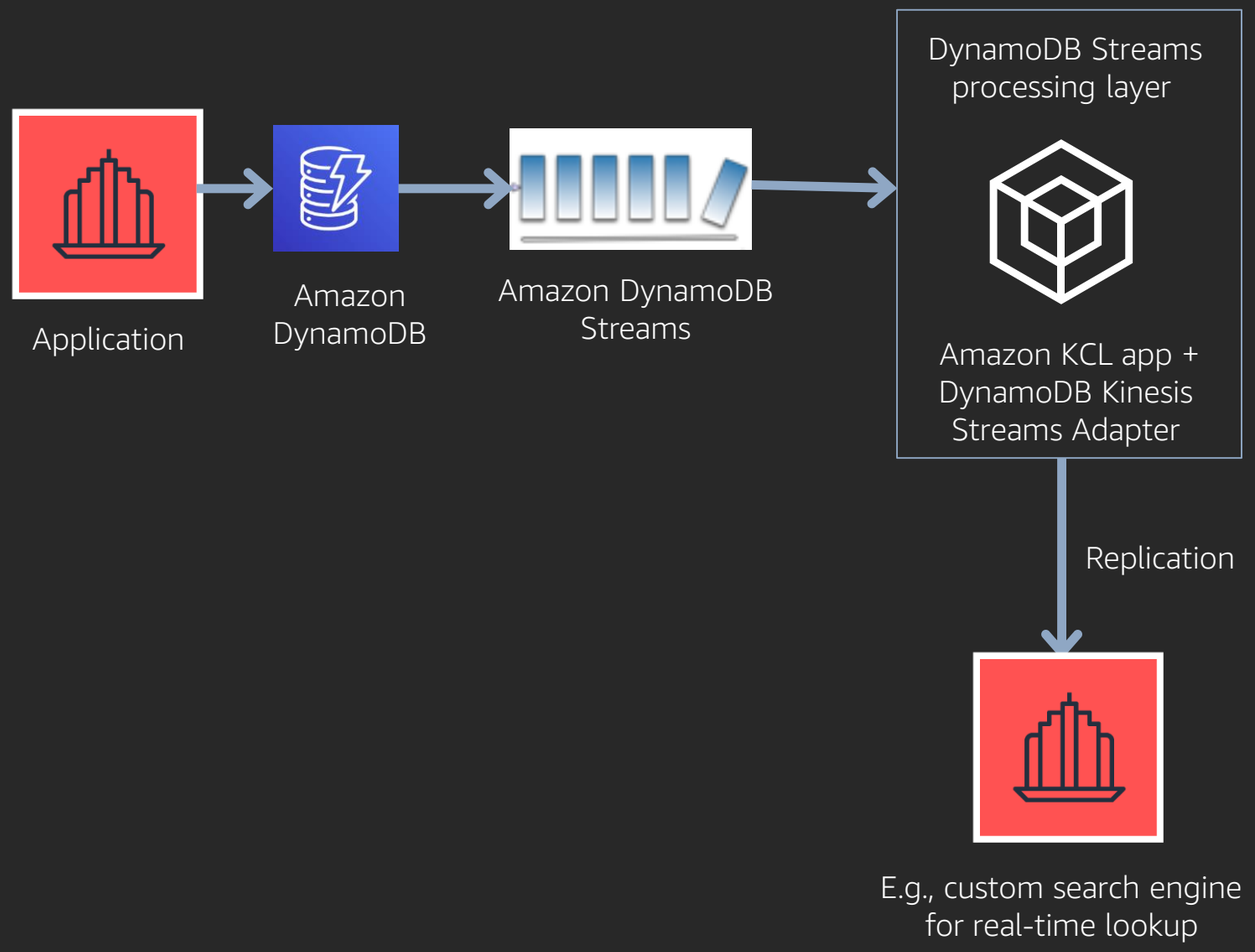
Integrate DynamoDB Streams with other AWS services to build solutions:

- Notify other systems and users about item-level inserts, updates, and deletes
- Audit or archive item-level modifications
- Replicate to other data stores in near real time
- Aggregate/filter data from item-level modifications
- Run analytical or search queries against data stored in DynamoDB

Options for stream processing:

- AWS Lambda – serverless and easier to manage
- Custom app with Amazon Kinesis Client Library (KCL) with DynamoDB Streams Adapter

Use case: DynamoDB Streams processing with Kinesis Client Library

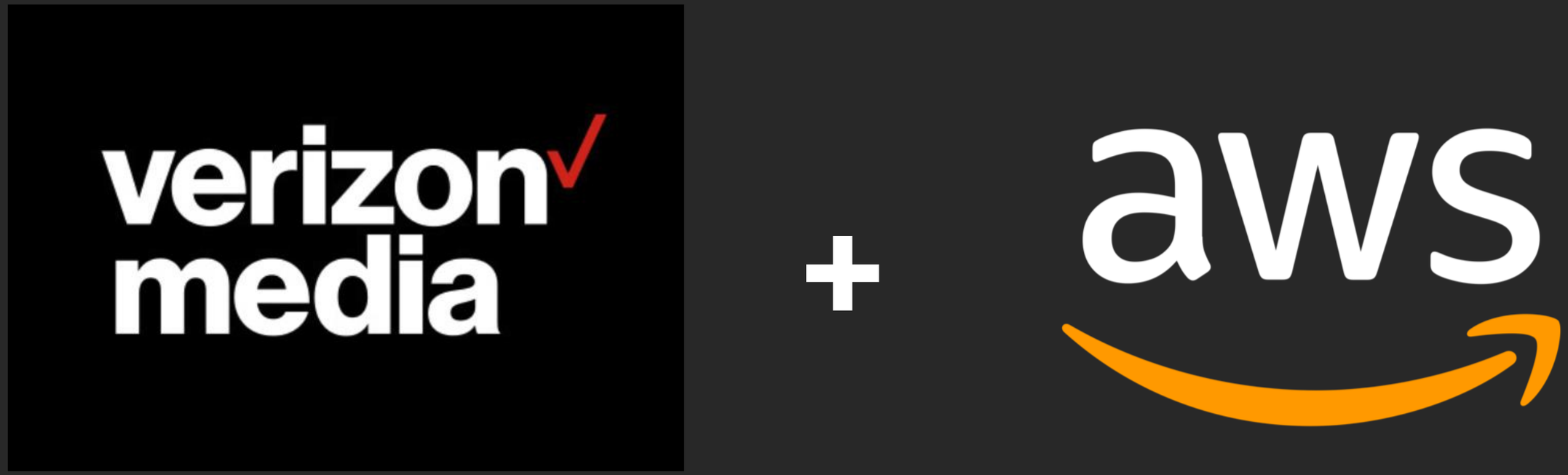


Choose custom app using KCL when:

- You need more sophisticated stream processing
- You need more control over the behavior of your application

Please note:

- You need to manage the shards, monitoring, scaling, and checkpointing process
- Host custom app in Amazon EC2 Auto Scaling group for high availability
- Tune configuration parameters of KCL worker for optimal performance


verizon
media
+
aws

Verizon Media (A Verizon Company) launches Mobile Push Notifications

AWS re:Invent

aws

Project objectives

- Create a better, stronger, and faster Push Notification System for all the iconic Verizon Media Brands (Fantasy, Sports, News, Finance, Mail, AOL, and HuffPost)
- Fulfill a push notification completion rate of up to 500,000 devices per second
- Use Amazon DynamoDB global tables, DynamoDB Streams, and Amazon Simple Queue Service (SQS) to power all the push use cases of our brands

Business requirements

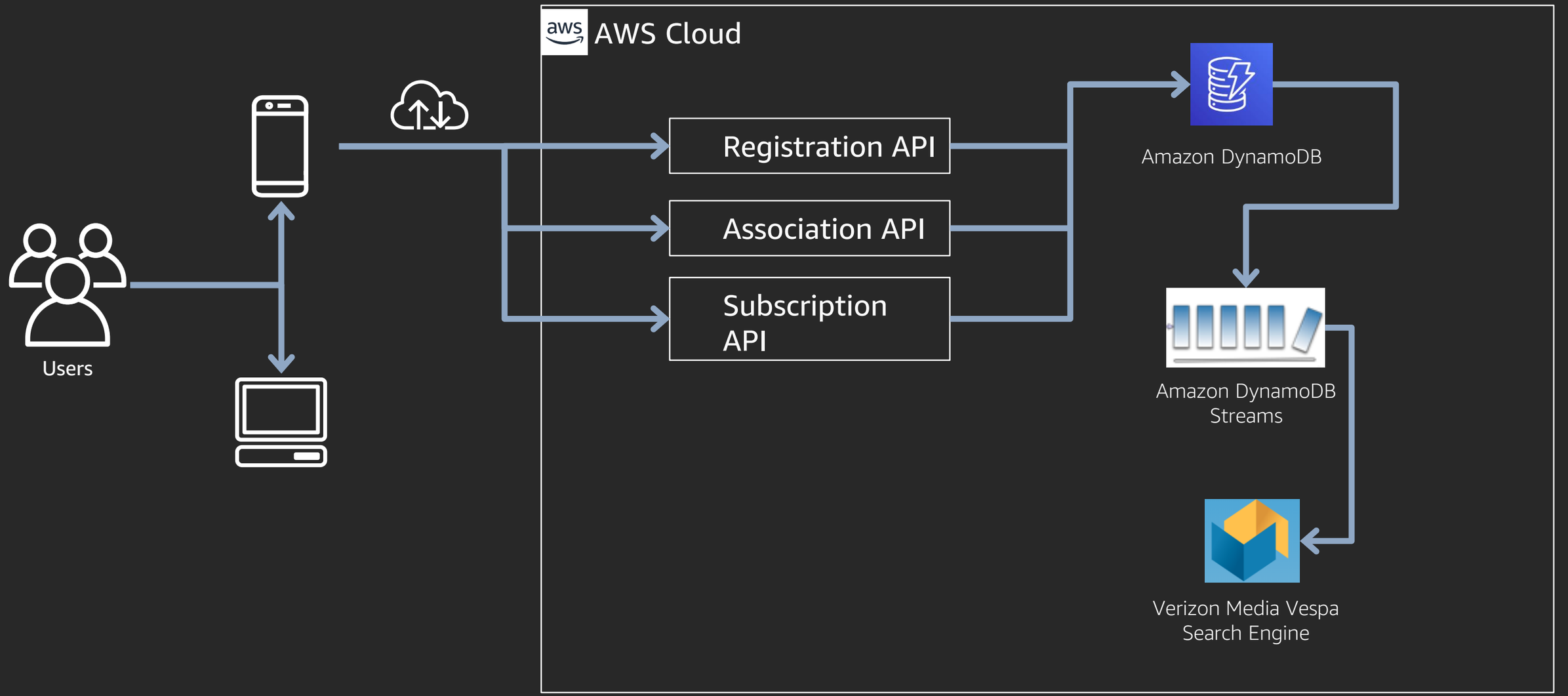
- Content-based (push on “breaking-news” or “video of the day”)
- Personalized (push on “Joe liked your comment”)
- Multi-criteria (“iOS, 13 devices, in the US-Eastern Time Zone”)
- Throttling (6 price alerts per day for a ticker symbol)

Technical requirements

- Push to 150,000 devices per second or better
- Minimize idle resources and don’t over provision
- Asynchronous event-based architecture to achieve parallelism
- 24/7/365

Architecture

Data management



Data model

Registration

```
{
  "app": {
    "id": "com.test.application",
    "version": "9.1.0"
  },
  "os": {
    "type": "android",
    "name": "Android SDK built for x86",
    "version": "11.1.2"
  },
  "pushInfo": {
    "pushToken": "eaaaJvklNYTtkE:....",
    "pushService": "fcm"
  }
}
```

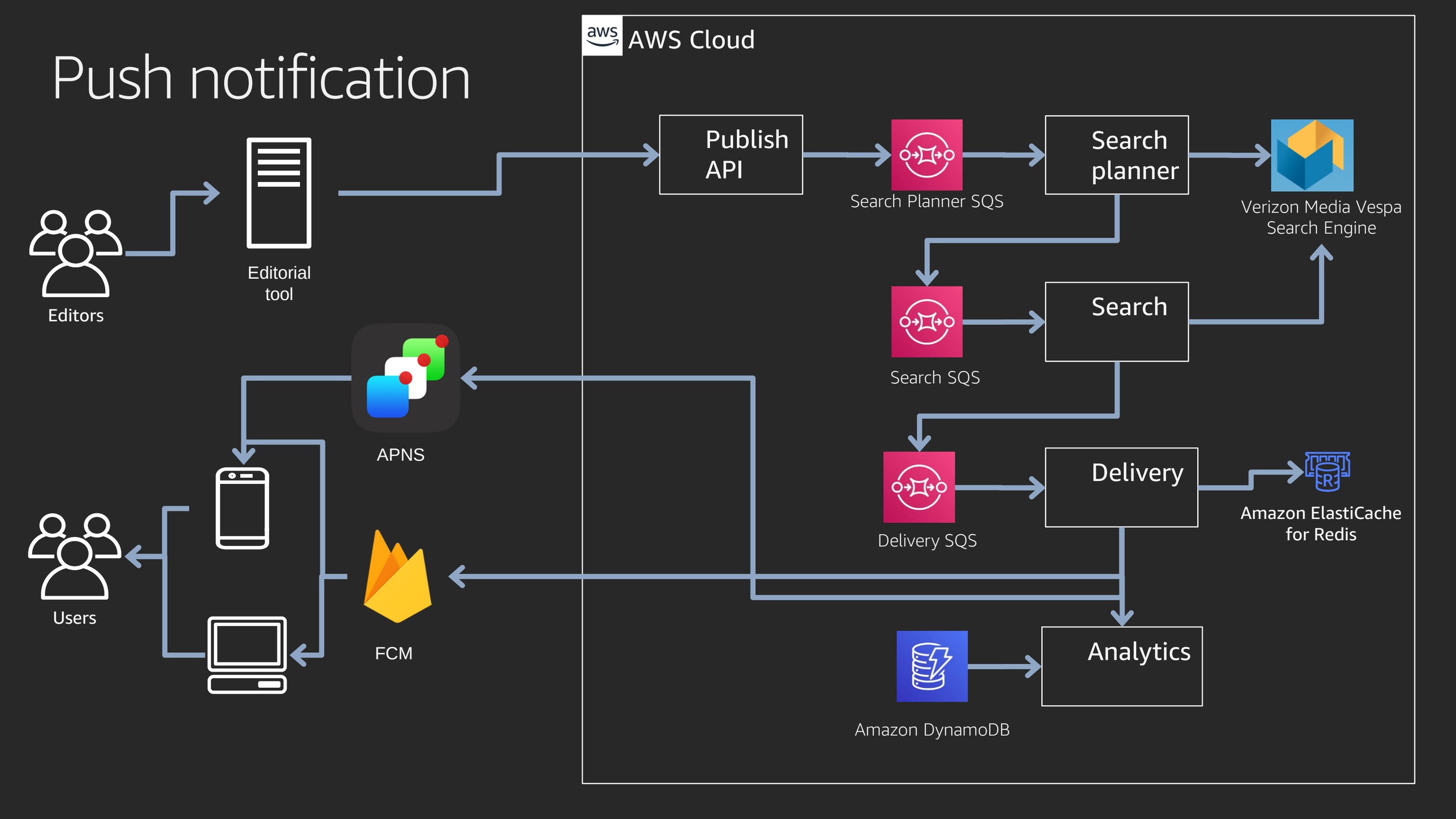
Association

```
{
  "operation": "add",
  "type": ”finance-user",
  "value": ”123456"
}
```

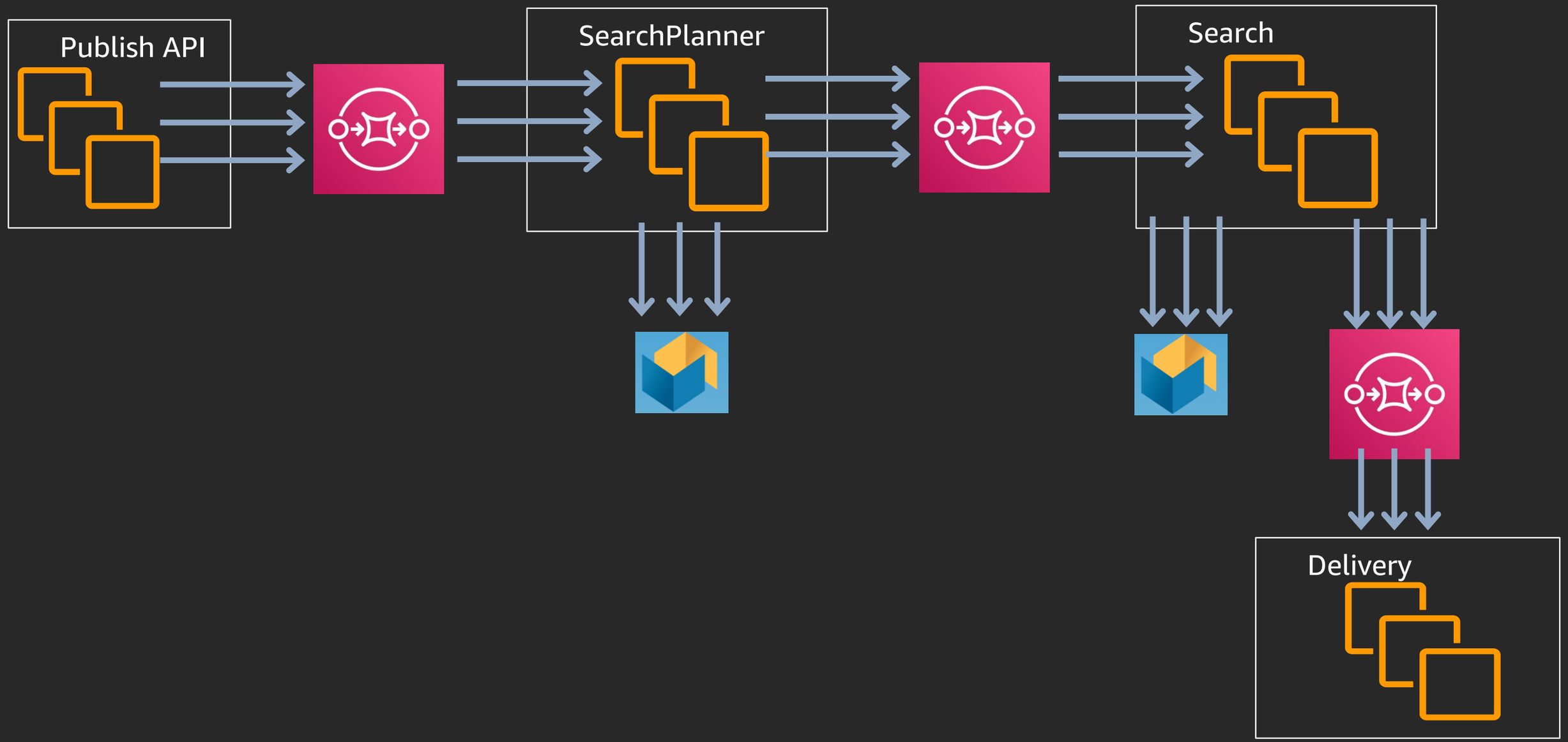
Subscription

```
{
  "subscribe": [
    {
      "tag": "ticker.vz",
      "displayName": "Subscription for
Verizon's ticker symbol"
    },
    {
      "tag": "ticker.amzn",
      "displayName": "Subscription for
Amazon's ticker symbol"
    }
  ]
}
```

Push notification



Horizontal scaling



Amazon SQS: AmazonSQS.sendMessage settings

- Did not use sendMessageBatch()
- Max size of Amazon SQS message capped at 100KB
- Average number of push objects per SQS message = 204
- Enabled customRetryPolicy

RetryPolicy.RetryCondition

Property	value
shouldRetry	true

RetryPolicy.BackoffStrategy

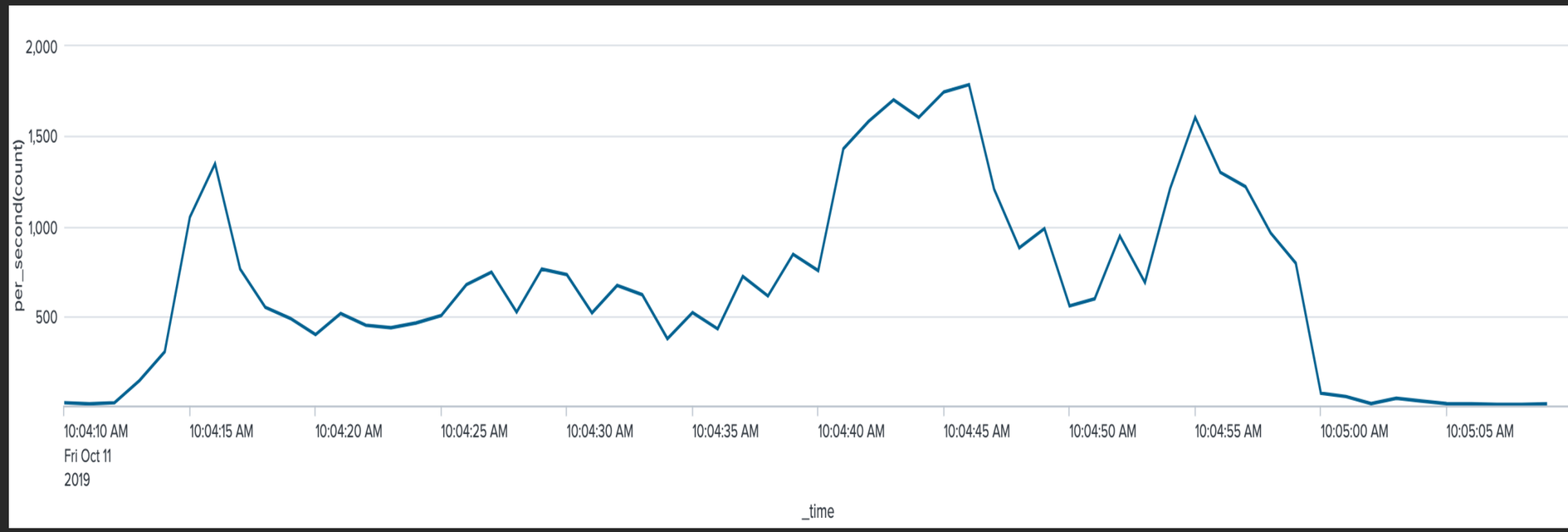
Property	value
delayBeforeNextRetry	0

ClientConfiguration

Property	value
setUseGzip	True
setMaxConnections	10,000
setConnectionTimeout	100 msec.
SqsClientCustomRetryPolicy Enabled	True
setRequestTimeout	120 sec. for internal services, 300 ms for api
maxRetry	3

Amazon SQS: AmazonSQS.sendMessage() performance

- Up to 1.8K TPS
- p95: 684 msec
- Size of Amazon SQS message affects sendMessage latency



	queue	count	min(latency)	max(latency)	avg(latency)	p50(latency)	p90(latency)	p95(latency)	p99(latency)
1	https://sqs.us-east-1.amazonaws.com/[redacted]/delivery-prod	39873	5	4988	205.46334612394352	113.59710616478753	428.22002490020134	684.7946468935188	1258.9972554899

Amazon SQS: AmazonSQS.receiveMessage() settings

Number of SQS reader threads per Java Virtual Machine = 150 (c5.4xlarge)

Sweet spot:

- Long poll of 10 seconds
- Up to 10 messages per receiveMessage()

Don’t be surprised if SQS returns 1 message (for us, it was 93%)

ClientConfiguration

Property	value
setUseGzip	True
setMaxConnections	200
setConnectionTimeout	10 seconds
SqsClientCustomRetryPolicy Enabled	False
setRequestTimeout	0

AmazonSQS

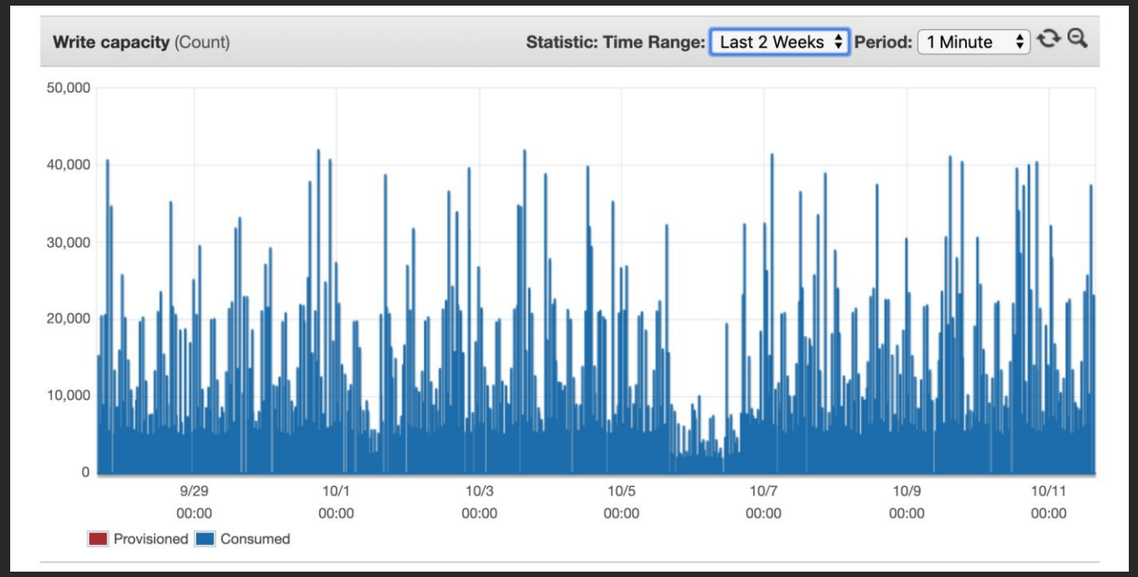
Property	value
withWaitTimeSeconds	10
withMaxNumberOfMessages	10

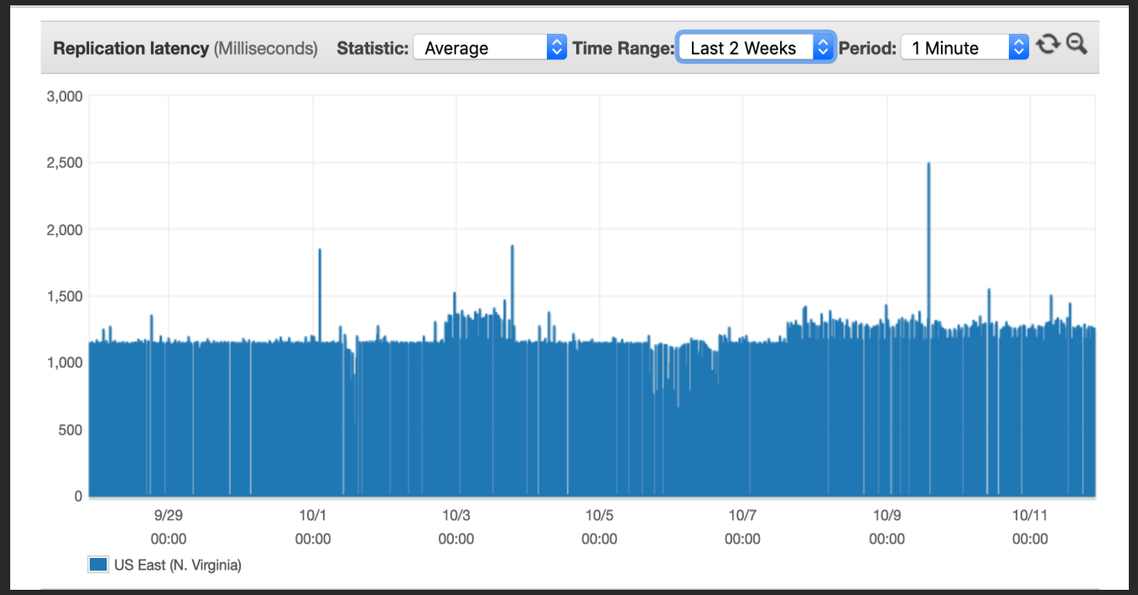
Number of Messages per receiveMessage(…)

Number Of Messages	Frequency
10	573
9	7
8	14
7	85…
1	36824

Registration table

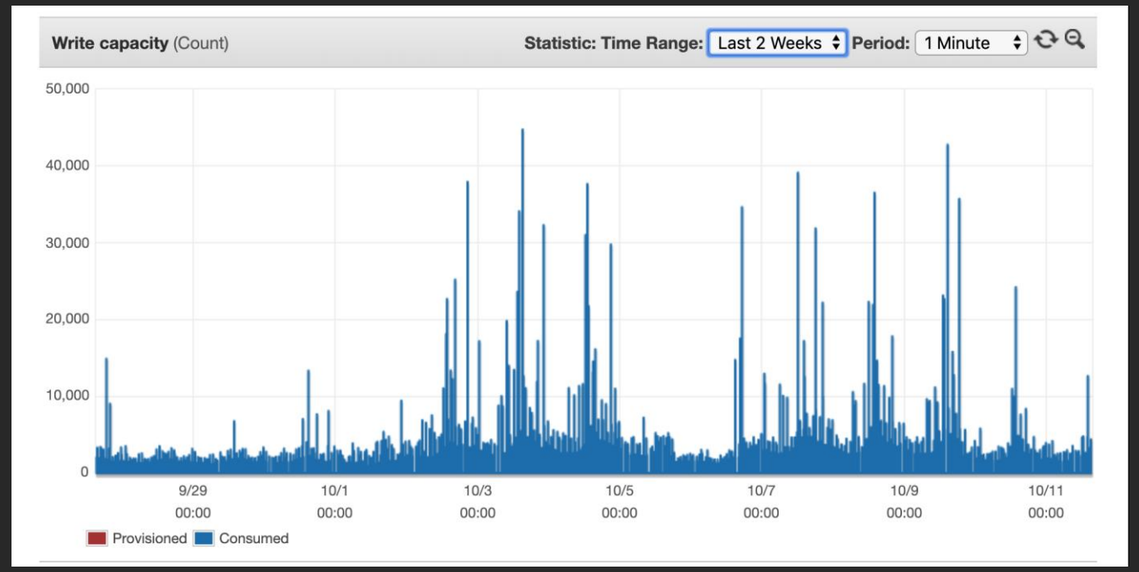
- On demand, WCU: 5K-40K
- Global table: US-East-1 & US-West-2
- Storage: 200 GB
- Item count: 70M
- Replication latency: 1.2 sec avg
- DynamoDB Streams: enabled
- NEW_AND_OLD_IMAGES

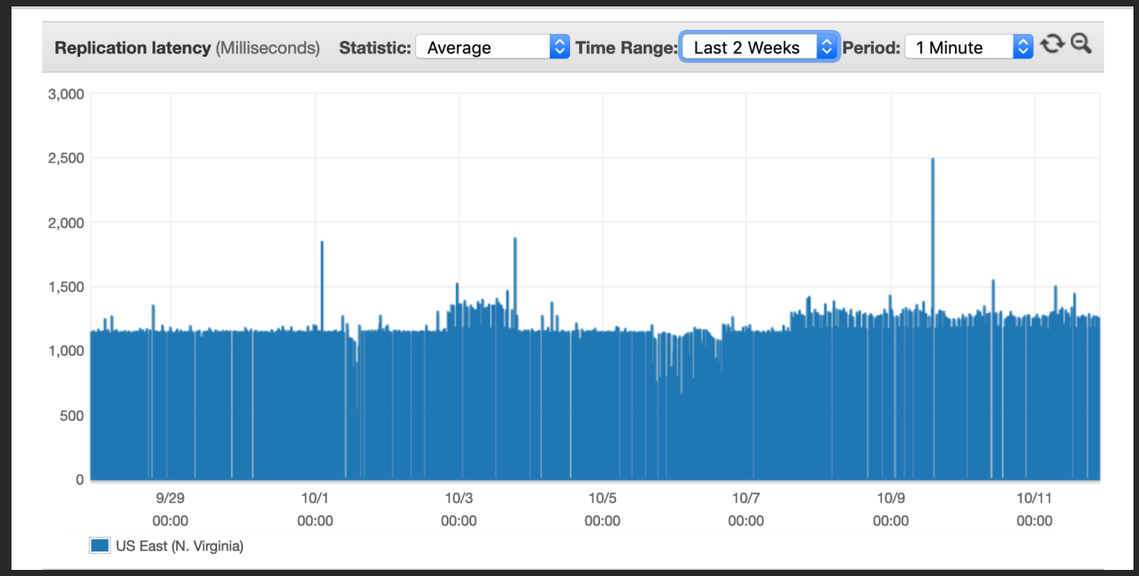




Subscription table

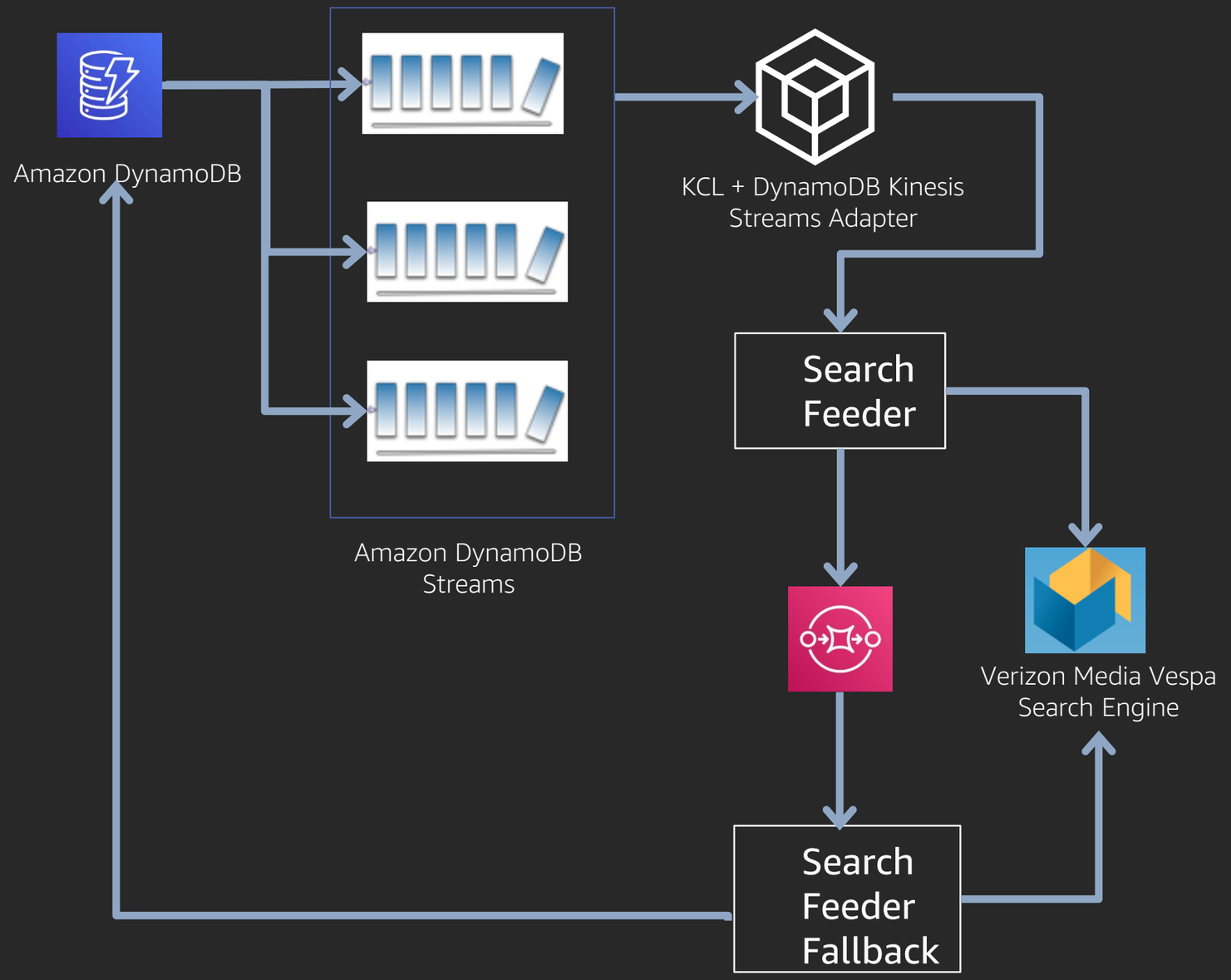
- Storage: 80 GB
- Item count: 300M
- Global table: US-East-1, US-West-2
- On demand, WCU: 3K-45K
- Replication latency: 1.2 sec avg
- DynamoDB Streams: enabled
- NEW_AND_OLD_IMAGES





Data ingestion using DynamoDB Streams

- We use KCL + DynamoDB Stream Adapter
- 3 DynamoDB Streams, processed independently
- Handle duplicate MODIFY events properly
- Fallback mechanism for slower consumer
- Ingests data into search engine within 1-2 secs



Scale DynamoDB Streams consumer using KCL

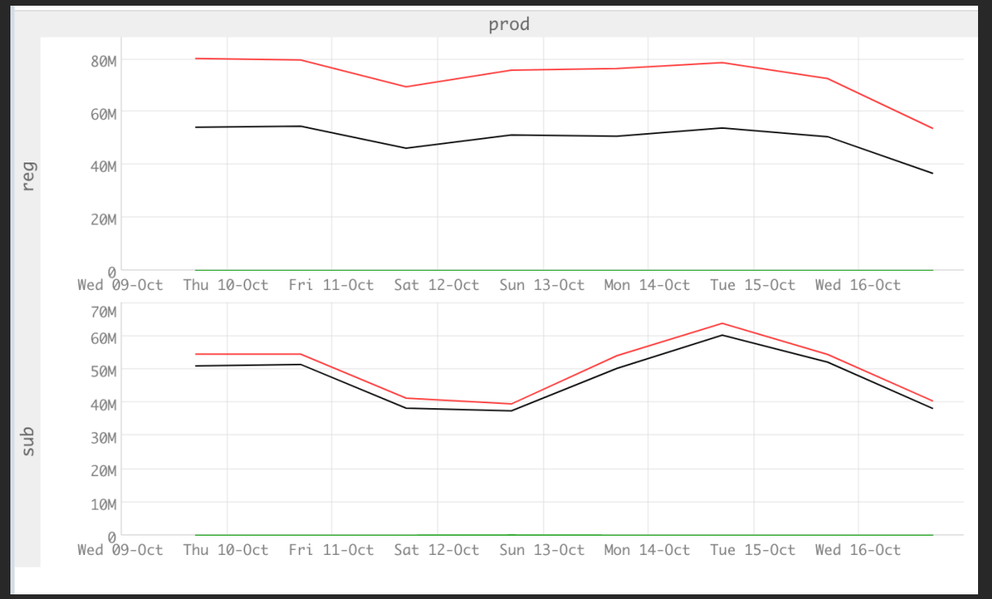
- Use different worker name to get parallelism
- withMaxRecords to keep up with producer
- withFailoverTimeMillis to avoid zombie lease, default is 10 secs, we use 20 secs, do not go over 50-60 secs
- Calling checkpoint too often can slow down the application - we update at every 100 records

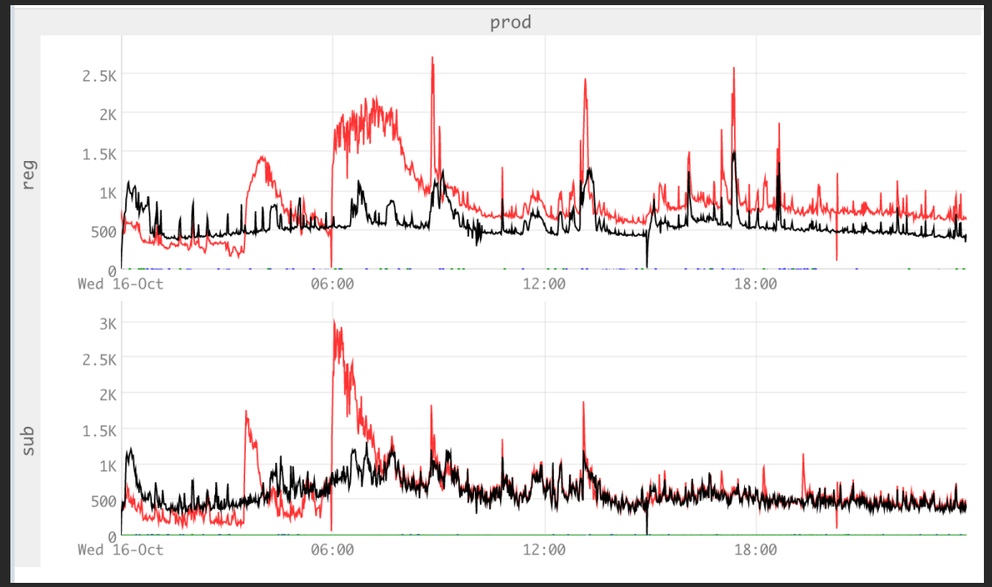
```
KinesisClientLibConfiguration workerConfig = new
KinesisClientLibConfiguration(streamAppName, streamARN, streamsCredentials,
    streamAppName + "-worker-" + System.nanoTime())
    .withFailoverTimeMillis(20000)
    .withMaxRecords(500)
    .withIdleTimeBetweenReadsInMillis(200)
    .withInitialPositionInStream(InitialPositionInStream.TRIM_HORIZON);
```

```
private void updateCheckpoint(IRecordProcessorCheckpointer checkpointer, Record record) {
 checkpointCounter += 1;
 // we update checkpoint at every 100 records, calling checkpoint too often can slow down the
application
 if (checkpointCounter % 100 == 0) {
  try {
   checkpointer.checkpoint(record);
  } catch (Exception e) {
   LOGGER.error("failed to update checkpoint", e);
  }
 }
}
```

Feeder ingestion rate

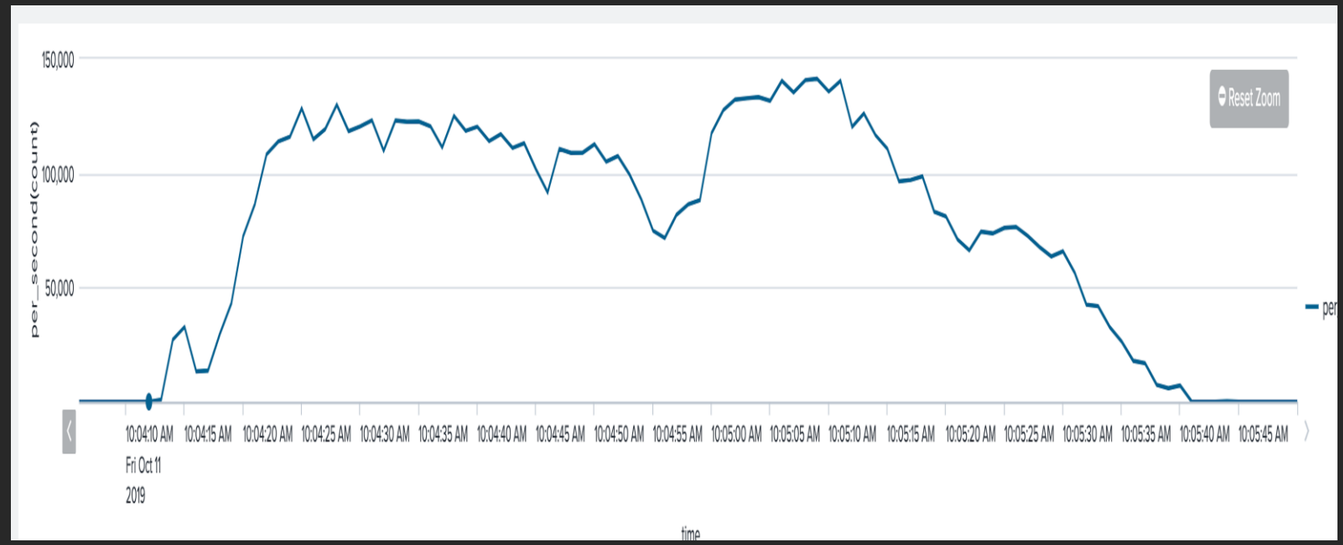
- 55M registration events processed per day
- 60M subscriptions events processed per day
- 2.5K-3K events per second
- Subscription will be available in our system within 1-2 secs (user -> DynamoDB -> DynamoDB Streams -> search engine)

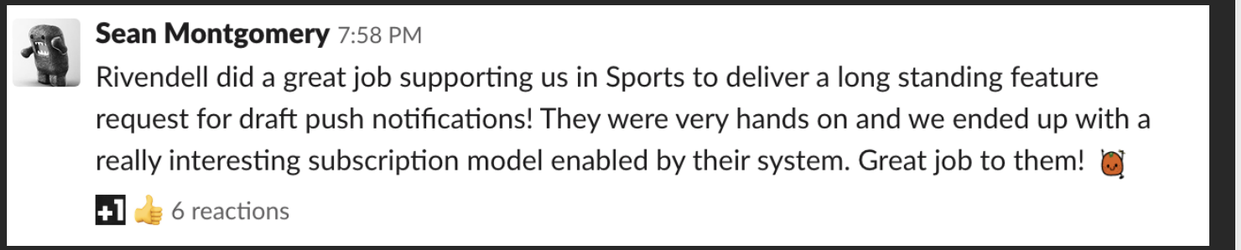


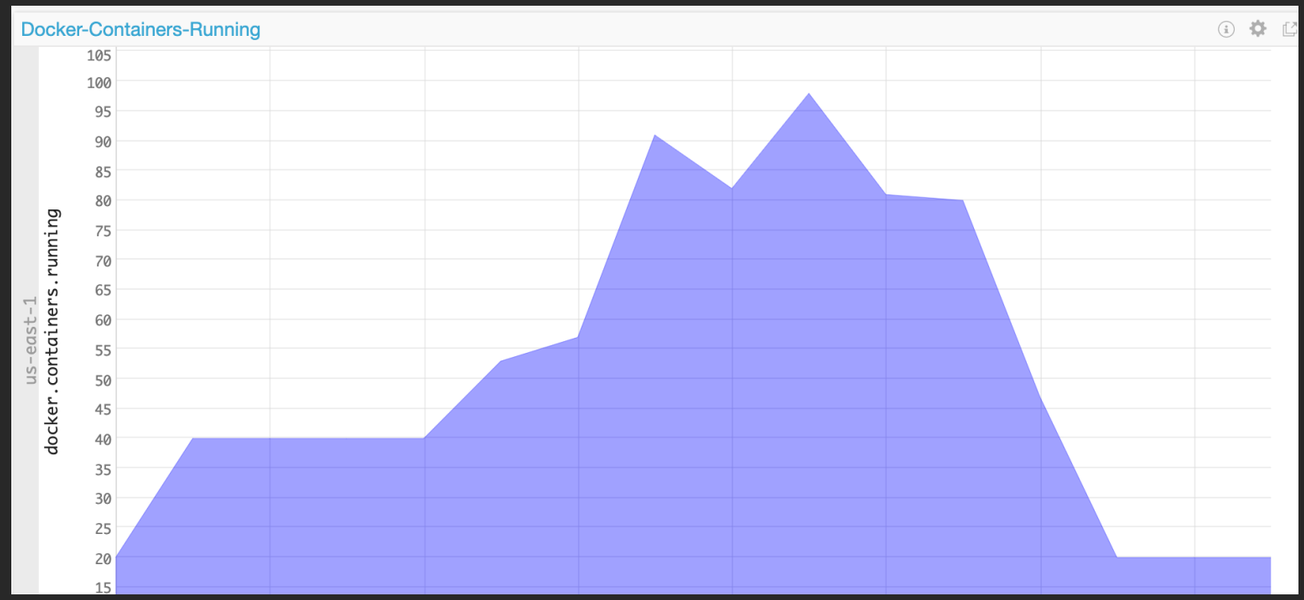


Results

- Up to 150K devices per second, and it’s easy to scale up to 500K devices per second
- Auto scale of delivery service during the burst







Lessons learned: DynamoDB Streams

- 24-hours data retention
  - DynamoDB Streams by design will keep data for 24 hours
  - Make sure your consumer is fast enough to process the load and has fallback logic in place in case of any failure
- Update checkpoint
  - Do not call it too frequently—it’s expensive
  - Use record object instance while updating checkpoint
- Handle exception properly
  - Catch exception properly and write failed record to another queue for later process before updating checkpoint
- Set withFailoverTimeMillis & withMaxRecords

Lessons learned (continued)

- Global Tables
  - Select your regions for Global Tables at time of creation
  - Eventual global consistency—make sure you handle this properly in consumer of DynamoDB Streams
  - Do not reprocess redundant MODIFY event
- DO NOT delete checkpoint table
  - Keep eyes on shards that are at TRIM_HORIZON—you can increase workers to add parallelism
- Partner closely with AWS teams
  - Do joint architecture reviews
  - Coordinate with Infrastructure Event Management (IEM) team prior and during launch. IEM is a value-add service available with Enterprise Support

"Verizon Media needed a horizontally scalable system that could deliver low-latency push notifications to our customers. We evaluated several technologies and AWS—specifically DynamoDB—was the best solution for us."

Kelly Hirano

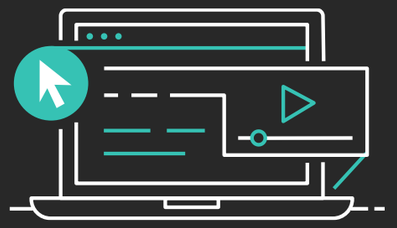
Vice President, Engineering
Verizon Media

AWS re:Invent

aws

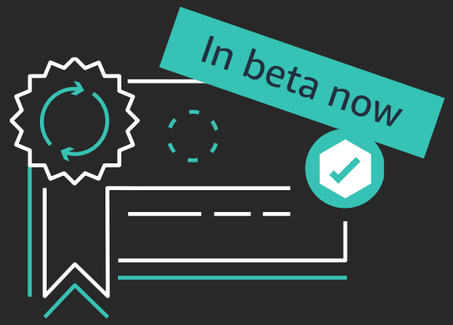
Learn databases with AWS Training and Certification

Resources created by the experts at AWS to help you build and validate database skills



25+ free digital training courses cover topics and services related to databases, including:

- Amazon Aurora
- Amazon Neptune
- Amazon DocumentDB
- Amazon DynamoDB
- Amazon ElastiCache
- Amazon Redshift
- Amazon RDS



Validate expertise with the new **AWS Certified Database - Specialty** beta exam

Visit aws.training

aws training and certification

Thank you!

AWS re:Invent

aws

Please complete the session survey in the mobile app.