# How Toyota Racing Development Makes Racing Decisions in Real Time with AWS

## The Power of DynamoDB and DynamoDB Streams

Jason Chambers, Toyota Racing Development
Philip Loh, Toyota Racing Development
Martin Sirull, AWS

November 30, 2016

# What to Expect from the Session

- How to efficiently manage and hash time-series data in Amazon DynamoDB
- How to stream data using DynamoDB streams
- How to play back DynamoDB data in real time
- Turbo mode: How to mix DynamoDB and Amazon Firehose to get the best of each technology

# Who are we?


KENSETH
DOLLAR GENERAL
20
CAMRY
HAMLIN
NEWMAN
KYLE BUSCH
KURT BUSCH
HARVICK
18
CAMRY

# Pre-race data

- Use data and models to help predict optimal adjustments for our cars prior to race start
- Provide and present data to drivers for immediate course feedback
- Need data within minutes


# Live data

- Provide real-time information and predictions to crews
- Data is used for split-second decisions about pitting, tire changes, and other strategic race strategy decisions
- Need data within seconds

# Data architecture

How we laid it out

# Our tech stack

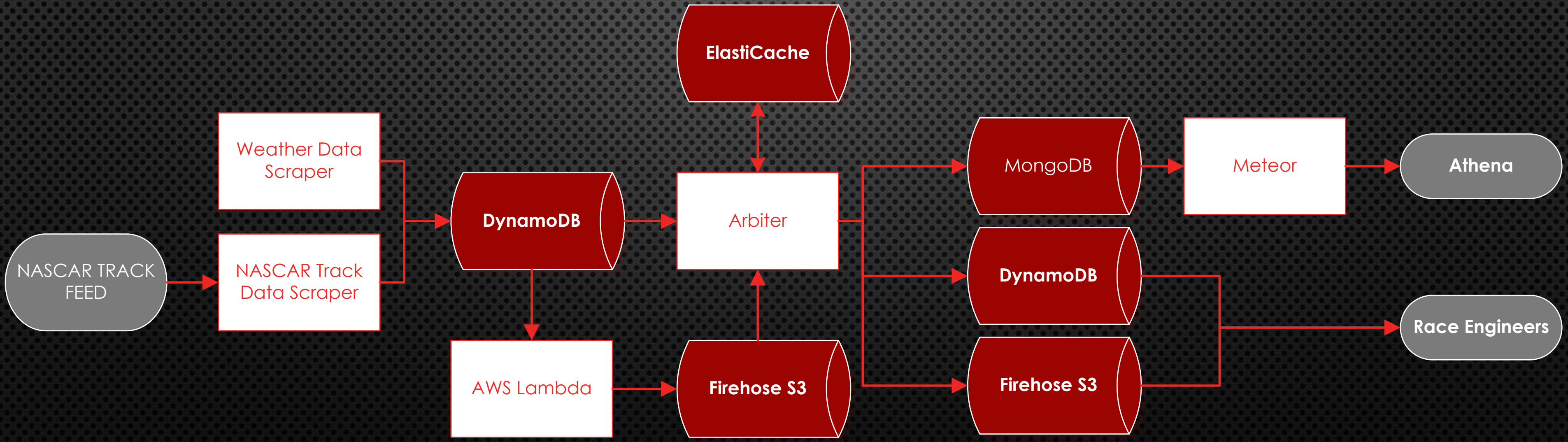
## Backend

- Built in JavaScript on Node.js
- Dynamic data – DynamoDB streams & Mongo
- Persistent data - DynamoDB and Amazon S3 services
- Cache – Amazon ElastiCache

## Frontend

- Built on JavaScript in Polymer
- Dynamic data served to clients via meteor

# Real-time data flow



# The power of DynamoDB

Our real-time workhorse

# Why we chose DynamoDB

- Decoupled analysis from data collection
- Multiple data sources can write to DynamoDB
- Data processors can listen for this data via streams in real time
- Data is persisted permanently for future queries
- Managed service means no maintenance
- Easy access control with AWS IAM

# Hash & range

Choosing the proper hash & range has enormous performance implications

**Hash**

- Determines partition where data is stored
- Records with equal hashes are stored on same partition
- To query data, you must know at least the record's hash

**Range**

- Data sharing the same hash is sorted within the partition by its range
- The combination of a record's hash and range constitute its primary key and must be unique

# Hash for time-series data

| Partition key value | Uniformity |
| --- | --- |
| User ID, where the application has many users. | Good |
| Status code, where there are only a few possible status codes. | Bad |
| **Item creation date, rounded to the nearest time period (e.g. day, hour, minute)** | **Bad** |
| Device ID, where each device accesses data at relatively similar intervals | Good |
| Device ID, where even if there are a lot of devices being tracked, one is by far more popular than all the others. | Bad |

http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GuidelinesForTables.html

# Time-series data: hot key mitigation

- Separate tables
- Manual partitioning
- Composite keys (TRD solution)

# Time-series data: separate tables

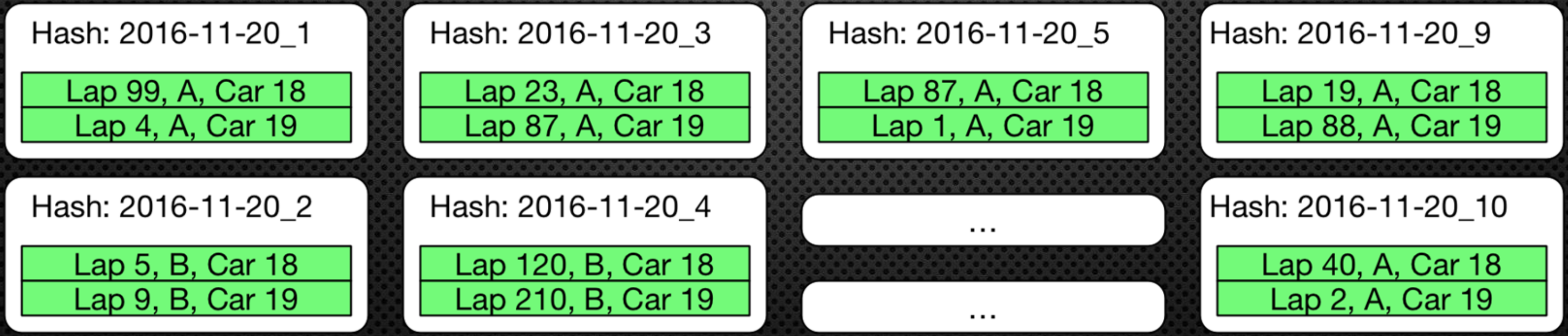
Hash: 2016-11-13

| | |
|---|---|
| Lap 1, A, Car 18 | Lap 3, A, Car 18 |
| Lap 1, B, Car 18 | Lap 3, B, Car 18 |
| Lap 1, A, Car 18 | … |
| Lap 1, B, Car 19 | … |
| Lap 2, A, Car 18 | Lap 300, A, Car 18 |
| Lap 2, B, Car 18 | Lap 300, B, Car 18 |
| Lap 2, A, Car 19 | Lap 300, A, Car 19 |
| Lap 2, B, Car 19 | Lap 300, B, Car 19 |

Hash: 2016-11-20

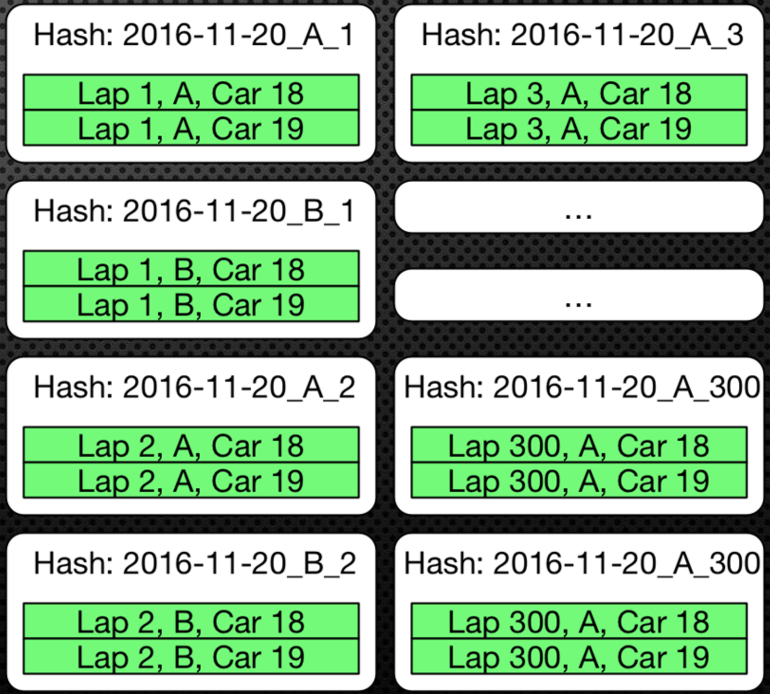
| | |
|---|---|
| Lap 1, A, Car 18 | Lap 3, A, Car 18 |
| Lap 1, B, Car 18 | Lap 3, B, Car 18 |
| Lap 1, A, Car 18 | … |
| Lap 1, B, Car 19 | … |
| Lap 2, A, Car 18 | Lap 300, A, Car 18 |
| Lap 2, B, Car 18 | Lap 300, B, Car 18 |
| Lap 2, A, Car 19 | Lap 300, A, Car 19 |
| Lap 2, B, Car 19 | Lap 300, B, Car 19 |

# Time-series data: manual partitioning



# Time-series data: composite keys

- Our solution to Hot Hashes
- Date, Type, and Lap as hash
- Data potentially in up to 10,000 partitions
- Can query in less than a second
- Could further hash by car number if needed
- Fetching data for entire race can be in parallel



# DynamoDB streams

Real-time data feed

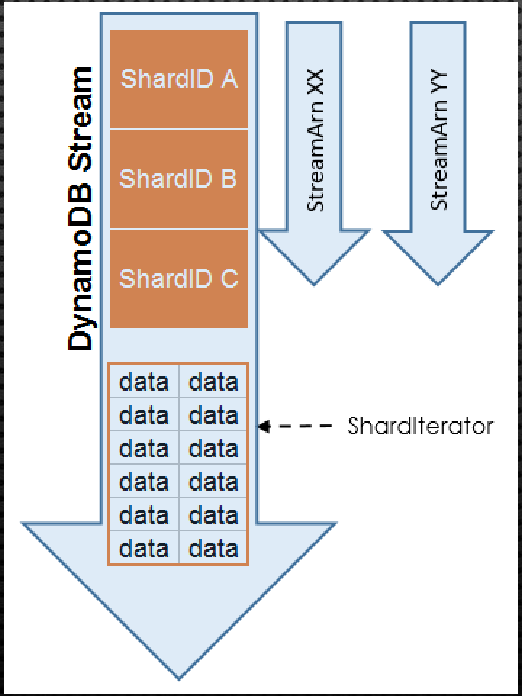
# Data streaming

- The Arbiter is set up to monitor DynamoDB tables as a stream
- AWS API calls can be used to “tail” the stream for updates
- TRD has written a JavaScript wrapper to automate the handling of these calls and is shared across multiple services
- For native Java users, AWS has made available the Amazon Kinesis Client Library, which does the same thing

# Data streaming

AWS API Calls:

1. ListStream
2. Describe Stream
3. GetShardIterator
4. GetRecords

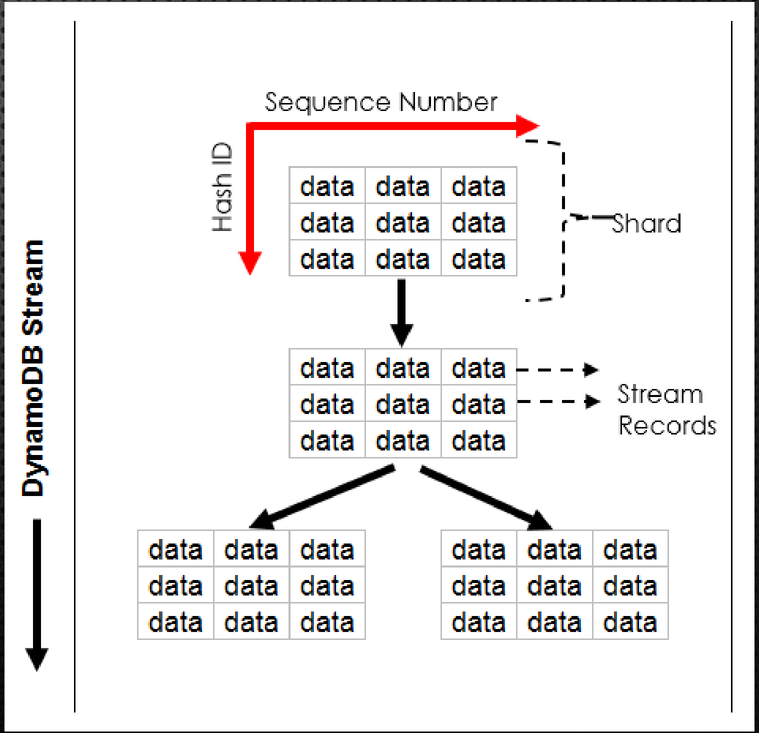



GetShardIterator Options:

- AFTER/AT_SEQUENCE NUMBER
  - Recover from a system crash
- TRIM_HORIZON
  - Resume from the beginning of the race
- LATEST
  - Start of the race

# Sharding

- Tables have partitions and streams have shards
- Shards are bounded by:
  - Min/max hash id
  - Min/max sequence number
- New shards will be created when:
  - Boundaries of the current shard have been exceeded
  - Increased throughput requirements



# Sharding

**Problems:**

- When streaming, we had to actively monitor shard status to determine when it expired and to scraper for new active iterators to stream data continuously

**Gotchas:**

- Sharding is not instantaneous, hence new shards aren't available immediately upon the active shard expiring
- Need to constantly hunt for new active shards

# Testing DynamoDB streams

- Can be challenging for functional testing because we cannot control when a stream re-shards
- API is the same as Kinesis. Can use Kinesis to simulate streams and force re-shards during functional tests
- For node.js we can use Kinesalite for functional testing
- Our most devastating bugs were due to not handling stream re-shards correctly

# DynamoDB streams vs. Kinesis streams

| | DynamoDB Stream | Kinesis Stream |
|---|---|---|
| Persistence | Stream: 24 hours<br>Table: As long as required | 24 Hours – 7 days |
| Sharding | Automatic | Manual |
| Querying | Stream: As series<br>Table: Ad Hoc | As series |
| Stream Latency* | ~ 1 second | ~200-400 ms |
| Billing | Per Read | Per Shard & Write |

*Unofficial. Derived from basic benchmarking, and could change

# DynamoDB + Firehose

The perfect match

# Turbo mode

**Scenario**

- Change of prediction model mid-race
- Need to re-process entire data set as fast as possible, then resume from stream where we left off

**Problem**

- Read limits on DynamoDB streams

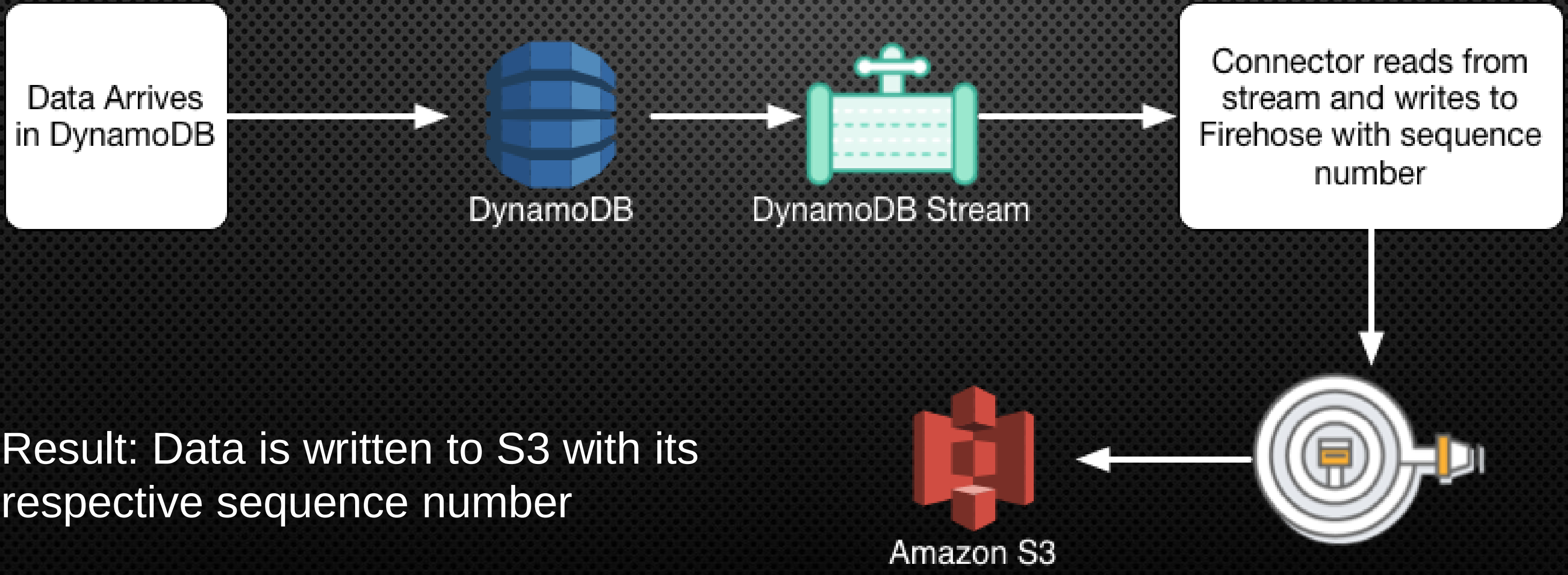
**Solution**

- Read from S3, then resume from stream

**Prerequisite**

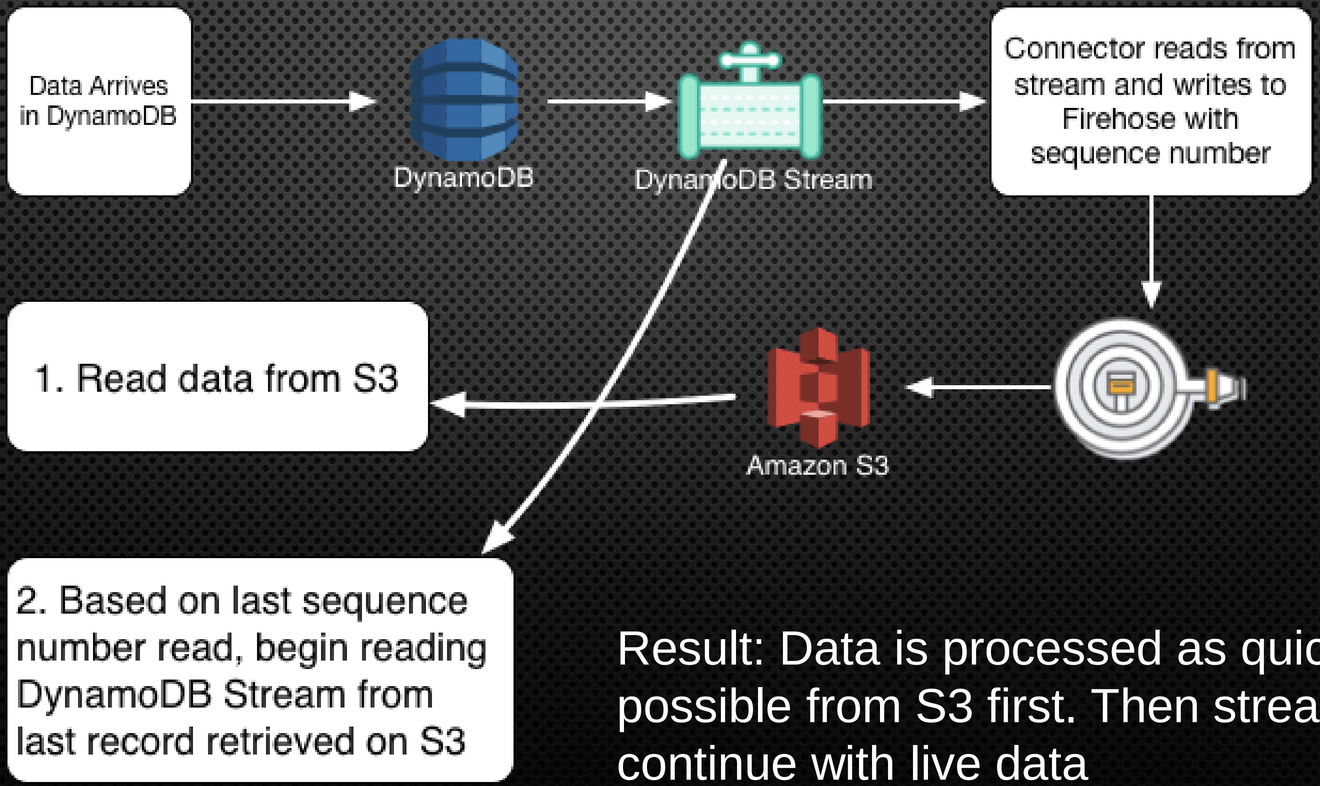
- Data on S3 must contain sequence number for each record for use in resumption of stream

# Capture sequence number




Result: Data is written to S3 with its respective sequence number

# Turbo read




Result: Data is processed as quickly as possible from S3 first. Then stream can continue with live data

# In action

DynamoDB to Firehose

# In action

Athena demo

Athena

PHILIP LOH

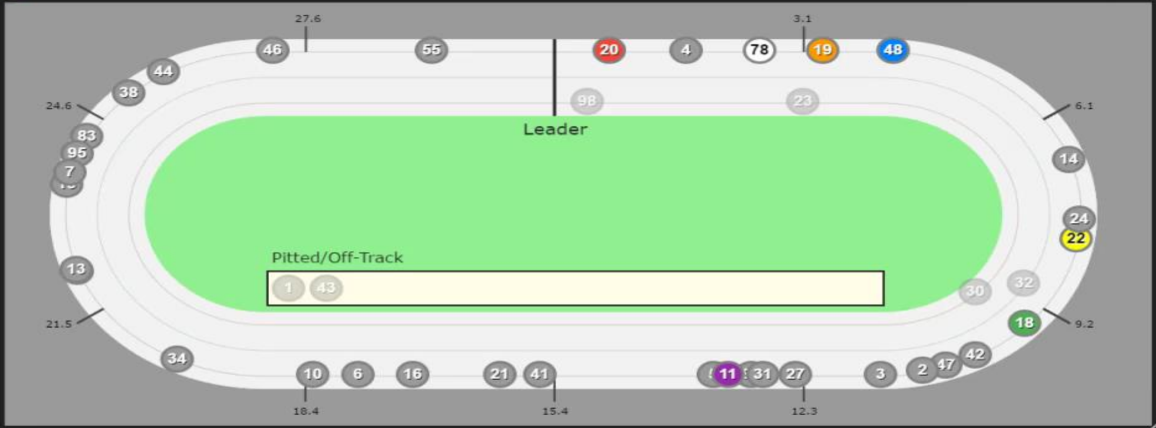
## Leaderboard

Lap: **67** of **267** | **200** to go | Track: **Kansas Speedway**

| Pos | | Car | Driver | Lap | LD | LT | R | FLT | O Split | L Split | TTL | LS | Stops | Pos +/- |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | 20 | Kenseth (C) | 67 | | 30.710 | 4 | 29.484 (2) | 0.46 | 0.46 | 0.46 | | | 0 |
| 2 | | 4 | Harvick (C) | 67 | | 30.855 | 11 | 29.686 (50) | 1.41 | 1.41 | 1.41 | | | +2 |
| 3 | | 78 | Truex Jr. (C) | 67 | | 30.780 | 7 | 29.876 (43) | 0.91 | 2.32 | 2.32 | | | -1 |
| 4 | | 19 | Edwards (C) | 67 | | 30.686 | 3 | 29.641 (2) | 0.78 | 3.10 | 3.10 | | | -1 |
| 5 | | 48 | Johnson (C) | 67 | | 30.611 | 1 | 30.095 (49) | 0.87 | 3.97 | 3.97 | | | +5 |
| 6 | | 14 | Stewart | 67 | | 30.938 | 14 | 30.136 (3) | 2.89 | 6.86 | 6.86 | | | +2 |
| 7 | | 24 | Elliott # (C) | 67 | | 30.767 | 6 | 30.270 (49) | 0.93 | 7.78 | 7.78 | | | 0 |
| 8 | | 22 | Logano (C) | 67 | | 30.722 | 5 | 30.048 (49) | 0.30 | 8.08 | 8.08 | | | -3 |
| 9 | | 18 | Busch (C) | 67 | | 31.007 | 19 | 29.678 (2) | 1.46 | 9.54 | 9.54 | | | -3 |
| 10 | | 42 | Larson | 66 | | 30.983 | 16 | 30.136 (50) | 1.07 | 10.32 | 10.32 | | | +3 |
| 11 | | 47 | Allmendinger | 66 | | 30.940 | 15 | 30.324 (3) | 0.40 | 10.71 | 10.71 | | | 0 |
| 12 | | 2 | Keselowski (C) | 66 | | 30.841 | 10 | 30.427 (50) | 0.30 | 11.01 | 11.01 | | | +2 |
| 13 | | 3 | Dillon (C) | 66 | | 30.870 | 12 | 30.225 (49) | 0.53 | 11.54 | 11.54 | | | -4 |
| 14 | | 27 | Menard | 66 | | 30.784 | 8 | 30.473 (51) | 1.06 | 12.60 | 12.60 | | | +3 |
| 15 | | 31 | Newman | 66 | | 30.897 | 13 | 30.370 (5) | 0.39 | 12.99 | 12.99 | | | -3 |
| 16 | | 88 | Bowman(i) | 66 | | 30.838 | 9 | 30.193 (3) | 0.14 | 13.13 | 13.13 | | | +18 |
| 17 | ▲ | 11 | Hamlin (C) | 66 | | 30.668 | 2 | 29.944 (3) | 0.29 | 13.42 | 13.42 | | | +7 |
| 18 | ▼ | 5 | Kahne | 66 | | 30.986 | 17 | 30.547 (51) | 0.18 | 13.61 | 13.61 | | | -2 |
| 19 | | 41 | Busch (C) | 66 | | 31.069 | 22 | 30.636 (52) | 2.13 | 15.74 | 15.74 | | | 0 |
| 20 | | 21 | Blaney # | 66 | | 31.023 | 20 | 30.444 (2) | 0.50 | 16.24 | 16.24 | | | -5 |
| 21 | | 16 | Biffle | 66 | | 31.000 | 18 | 30.677 (51) | 1.08 | 17.32 | 17.32 | | | -3 |
| 22 | | 6 | Bayne | 66 | | 31.043 | 21 | 30.484 (2) | 0.66 | 17.99 | 17.99 | | | -1 |
| 23 | | 10 | Patrick | 66 | | 31.201 | 24 | 30.784 (52) | 0.56 | 18.54 | 18.54 | | | 0 |
| 24 | | 34 | Buescher # | 66 | | 31.186 | 23 | 30.720 (3) | 1.71 | 20.25 | 20.25 | | | +2 |
| 25 | | 17 | Stenhouse Jr. | 66 | | 31.591 | 33 | 30.593 (4) | 1.86 | 22.11 | 22.11 | | | -3 |
| 26 | | 13 | Mears | 66 | | 31.460 | 30 | 30.534 (2) | 0.03 | 22.14 | 22.14 | | | -6 |
| 27 | | 15 | Bowyer | 66 | | 31.404 | 29 | 30.938 (3) | 1.32 | 23.47 | 23.47 | | | +2 |
| 28 | | 7 | Smith | 66 | | 31.355 | 27 | 31.091 (53) | 0.18 | 23.65 | 23.65 | | | +5 |
| 29 | | 95 | McDowell | 66 | | 31.283 | 25 | 30.975 (2) | 0.30 | 23.95 | 23.95 | | | +2 |
| 30 | | 83 | DiBenedetto | 66 | | 31.288 | 26 | 30.580 (2) | 0.27 | 24.23 | 24.23 | | | +2 |
| 31 | | 38 | Cassill | 66 | | 31.389 | 28 | 31.056 (5) | 0.85 | 25.08 | 25.08 | | | -6 |
| 32 | | 44 | Scott # | 66 | | 31.470 | 31 | 30.759 (2) | 0.54 | 25.62 | 25.62 | | | -4 |
| 33 | | 46 | Annett | 66 | | 31.478 | 32 | 31.087 (2) | 1.40 | 27.02 | 27.02 | | | -6 |
| 34 | | 55 | Whitt | 66 | | 31.669 | 34 | 31.232 (3) | 1.96 | 28.98 | 28.98 | | | +1 |
| 35 | | 32 | Gase(i) | 66 | -1 | 32.625 | 37 | 31.324 (49) | 10.63 | 8.91 | 39.62 | | | -5 |
| 36 | | 98 | Sorenson | 65 | -2 | 31.718 | 35 | 31.196 (2) | 23.94 | 0.22 | 61.46 | | | +1 |
| 37 | | 23 | Ragan | 64 | -3 | 32.329 | 36 | 31.306 (47) | 3.17 | 3.39 | 95.25 | | | +2 |
| 38 | | 30 | Wise | 63 | -3 | 32.640 | 38 | 31.444 (3) | 7.51 | 9.28 | 100.84 | | | 0 |
| 39 | | 1 | McMurray | 45 | -21 | - | - | 30.626 (2) | - | - | - | - | | - |
| 40 | | 43 | Almirola | 37 | -29 | - | - | 31.060 (2) | - | - | - | - | | - |

## Race Positions

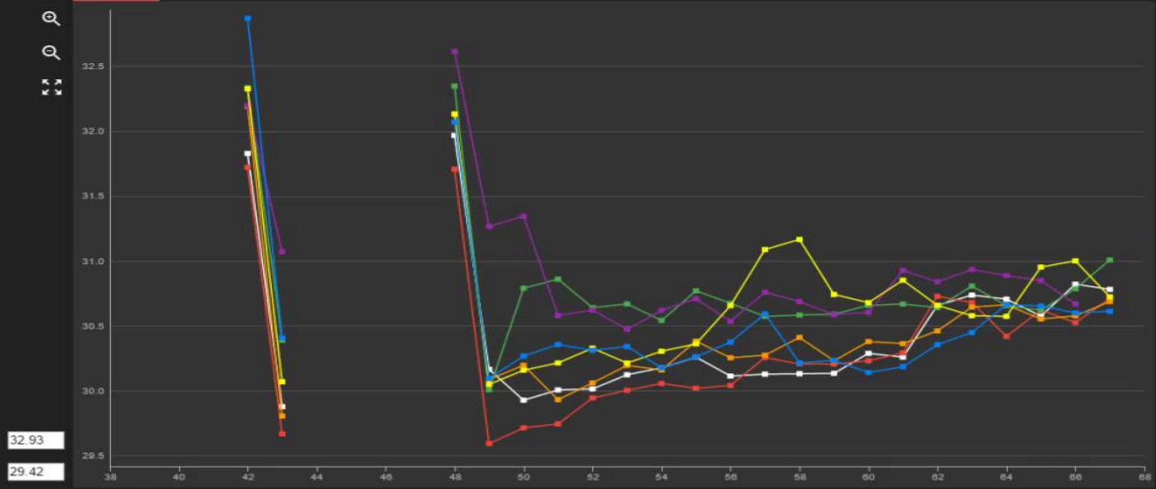
Lap: **67** | Leader: **20** | Leader Laptime: **9.830**




## Graphs

Lap Time | Rank | Cumulative Delta

PAN GROW PAUSE




Last 30 laps ▾ | 38 | 68

AWS re:Invent

# Thank you!

# Remember to complete your evaluations!