
AWS
re:Invent

DAT310

Deep dive on Amazon Timestream

Tony Gibbs

Principal Database Solutions Architect

AWS

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Overview: Deep dive on Amazon Timestream

Introducing Amazon Timestream

Architectural concepts and terminology

Data storage and ingestion

Query processing

Additional resources



AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Introducing Amazon Timestream



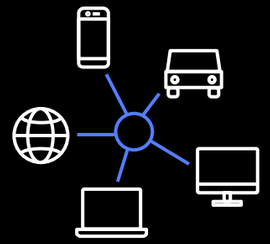
AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Are you an Amazon Timestream user?



Time-series use cases



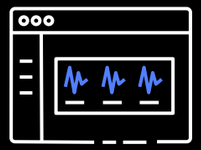
IoT applications

Collect motion or temperature data from the device sensors, interpolate to identify the time ranges without motion, or alert consumers to take actions such as turning off the lights to save energy



DevOps analysis

Collect and analyze performance and health metrics such as CPU/ memory utilization, network data, and IOPS to monitor health and optimize instance usage



App analysis

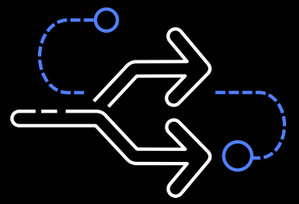
Easily store and analyze clickstream data at scale to understand the customer journey—the user activity across your applications over a period of time

Building with time-series data is challenging

Relational databases



Inefficient at processing time-series data

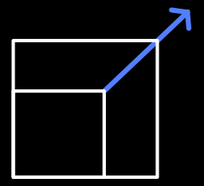


Data management issues with rigid schema

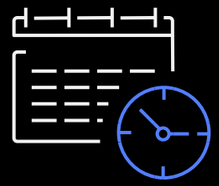


Limited integrations for ML, analytics, and data collection

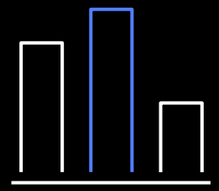
Existing time-series solutions



Difficult to scale for large volumes of data



Minimal data lifecycle management



Real-time and historical data are decoupled

Amazon Timestream

FAST, SCALABLE, AND SERVERLESS TIME-SERIES DATABASE

Serverless and easy to use



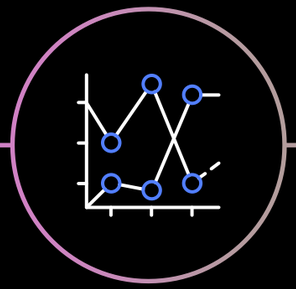
No servers to manage or instances to provision; software patches, indexes, and database optimizations are handled automatically

Performance at scale



Capable of ingesting trillions of events daily; the adaptive SQL query engine provides rapid point-in-time queries with its in-memory store, and fast analytical queries through its magnetic store

Purpose build for time series data



Built-in analytics using standard SQL with added interpolation and smoothing functions to identify trends, patterns, and anomalies

Secure from the ground up



All data is encrypted inflight, and at rest using AWS Key Management Service (AWS KMS) with customer-managed keys (CMK)

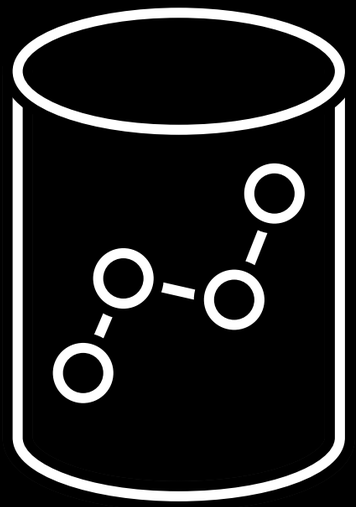
Architectural concepts and terminology



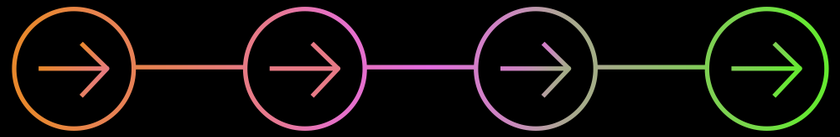
AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

September 2020

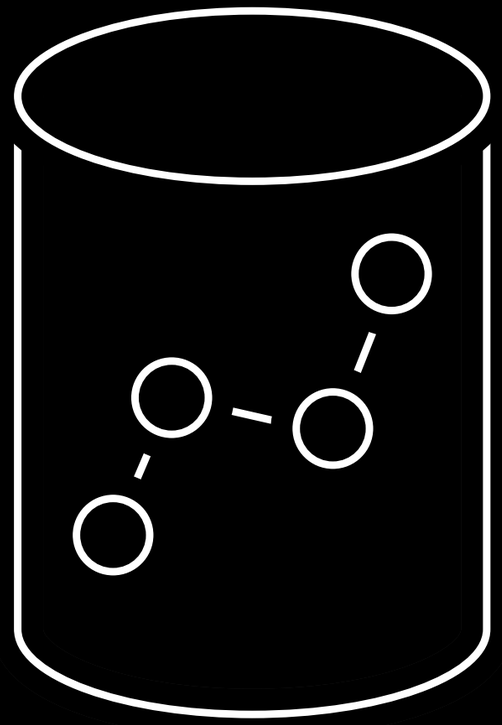


Continuous releases



No maintenance or downtime
Serverless architecture

December 2020



Terminology and concepts: Tables

Encrypted container that holds **records**

No data definition or columns are specified at creation

Time-based data retention policies for controlling data lifecycle within **storage tiers**

Terminology and concepts: Storage tiers

TWO STORAGE TIERS: IN-MEMORY AND MAGNETIC

Retention periods are required for both tiers and at table creation

Retention periods can be modified after table creation

In-memory store can range from 1 hour to 1 year, and magnetic store from 1 day to 200 years

Amazon Timestream architecture

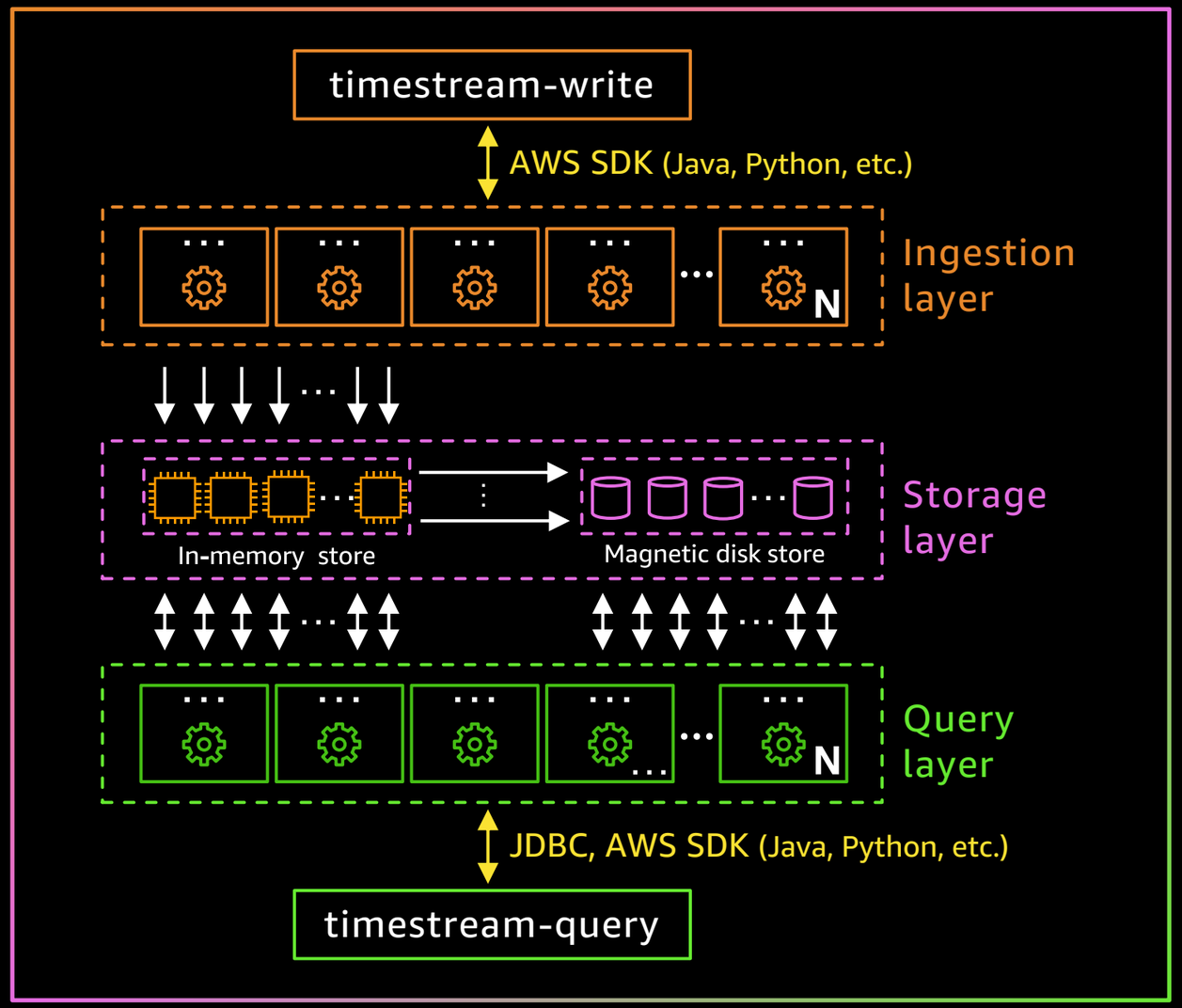
SERVERLESS CLOUD NATIVE AWS BUILT TECHNOLOGY

Decoupled architecture
- Highly available—99.99% SLA
- Independently scalable ingestion, storage, and SQL processing

High throughput auto-scaling ingestion
- Data is replicated across multiple Availability Zones
- Automatic data deduplication handling
- No need to provision or configure write I/O

Multiple tiers of storage
- Scalable to petabytes and beyond
- In-memory store is designed for fast point-in-time queries
- Magnetic store is designed for high performance analytics queries and low cost long-term storage

Scalable SQL query engine
- Adaptive query engine is capable of querying data across multiple data tiers
- No indexes to configure and no provisioning required



Terminology and concepts: Dimensions

ARE A SET OF ATTRIBUTES THAT UNIQUELY DESCRIBE A MEASUREMENT

Each table allows up to 128 unique dimensions

All dimensions are represented as varchars

Dimensions are dynamically added to the table during ingestion

Terminology and concepts: Measures

EACH AMAZON TIMESTREAM RECORD CONTAINS A SINGLE MEASUREMENT COMPRISED OF A NAME AND VALUE

Each table supports up to 1,024 unique measure names

Measurement values support boolean, bigint, double, and varchar

Measures are dynamically added to the table during ingestion

Terminology and concepts: Time series

SEQUENCE OF RECORDS THAT ARE REPRESENTED AS DATA POINTS OVER A TIME INTERVAL FOR GIVEN MEASUREMENT

A time series is a set of timestamp and measure value pairs that have the same dimension name, value, and measure name

Example: Time series in Amazon Timestream

time	region	az	vpc	hostname	measure_name	measure_value::double	measure_value::bigint
2020-06-17 19:00:00.000000000	us-east-1	1d	vpc-1a2b3c4d	host-24Gju	cpu_utilization	35.0	null
2020-06-17 19:00:01.000000000	us-east-1	1d	vpc-1a2b3c4d	host-24Gju	cpu_utilization	38.2	null
2020-06-17 19:00:02.000000000	us-east-1	1d	vpc-1a2b3c4d	host-24Gju	cpu_utilization	45.3	null
2020-06-17 19:00:00.000000000	us-east-1	1d	vpc-1a2b3c4d	host-24Gju	mem_utilization	54.9	null
2020-06-17 19:00:01.000000000	us-east-1	1d	vpc-1a2b3c4d	host-24Gju	mem_utilization	42.6	null
2020-06-17 19:00:02.000000000	us-east-1	1d	vpc-1a2b3c4d	host-24Gju	mem_utilization	33.3	null
2020-06-17 19:00:00.000000000	us-east-1	1d	vpc-1a2b3c4d	host-24Gju	networks_bytes	null	30000
2020-06-17 19:00:01.000000000	us-east-1	1d	vpc-1a2b3c4d	host-24Gju	networks_bytes	null	15200
2020-06-17 19:00:02.000000000	us-east-1	1d	vpc-1a2b3c4d	host-24Gju	networks_bytes	null	15200

Example: Time series in Amazon Timestream

time	region	az	vpc	hostname	measure_name	measure_value::double	measure_value::bigint
2020-06-17 19:00:00.000000000	us-east-1	1d	vpc-1a2b3c4d	host-24Gju	cpu_utilization	35.0	null
2020-06-17 19:00:01.000000000	us-east-1	1d	vpc-1a2b3c4d	host-24Gju	cpu_utilization	38.2	null
2020-06-17 19:00:02.000000000	us-east-1	1d	vpc-1a2b3c4d	host-24Gju	cpu_utilization	45.3	null
2020-06-17 19:00:00.000000000	us-east-1	1d	vpc-1a2b3c4d	host-24Gju	mem_utilization	54.9	null
2020-06-17 19:00:01.000000000	us-east-1	1d	vpc-1a2b3c4d	host-24Gju	mem_utilization	42.6	null
2020-06-17 19:00:02.000000000	us-east-1	1d	vpc-1a2b3c4d	host-24Gju	mem_utilization	33.3	null
2020-06-17 19:00:00.000000000	us-east-1	1d	vpc-1a2b3c4d	host-24Gju	networks_bytes	null	30000
2020-06-17 19:00:01.000000000	us-east-1	1d	vpc-1a2b3c4d	host-24Gju	networks_bytes	null	15200
2020-06-17 19:00:02.000000000	us-east-1	1d	vpc-1a2b3c4d	host-24Gju	networks_bytes	null	15200

Terminology and concepts: Data modeling

In a traditional relational database we would create a wide table or use dimension and fact tables to model the data

time	region	az	vpc	hostname	cpu_utilization	mem_utilization	networks_bytes
2020-12-01 19:00:00.000000000	us-east-1	1d	vpc-1a2b3c4d	host-24Gju	35.0	54.9	30000

Amazon timestream represents data where there is a single measure per record

time	region	az	vpc	hostname	measure_name	measure_value::double	measure_value::bigint
2020-12-01 19:00:00.000000000	us-east-1	1d	vpc-1a2b3c4d	host-24Gju	cpu_utilization	35.0	null
2020-12-01 19:00:00.000000000	us-east-1	1d	vpc-1a2b3c4d	host-24Gju	mem_utilization	54.9	null
2020-12-01 19:00:00.000000000	us-east-1	1d	vpc-1a2b3c4d	host-24Gju	networks_bytes	null	30000

Characteristics of Amazon Timestream data

All records require a timestamp, one or more dimensions, a measurement name, and a measurement value

Records cannot be deleted or updated

- Records are only removed when they reach the retention limit within the magnetic tier
- Choice of first or last writer wins semantics for handling duplicates

Multiple measures are logically represented as multiple individual records

- One measure per record

Automatically scales to handle high throughput, real-time data ingestion

Data storage and ingestion



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Data ingestion: Connectivity

Data is written using the AWS SDK

Java, Python, Golang, Node.js, .NET, etc.

AWS Command Line Interface (AWS CLI)

Adapters and plugins

AWS IoT Core

Amazon Kinesis Data Analytics for Apache Flink connector (GitHub)

Telegraf connector (GitHub)

Data ingestion: Pricing

$0.50* for 1 million writes of 1KB

time	region	az	vpc	hostname	measure_name	measure_value::double	measure_value::bigint
1602983435238563000	us-east-1	1d	vpc-1a2b3c4d	host-24Gju	cpu_utilization	35.0	null

```
timestamp:                      8  bytes
first dimension:                15 bytes  ("region" + "us-east-1")
second dimension:               4 bytes   ("az + 1d")
third dimension:                15 bytes  ("vpc + "vpc-1a2b3c4d")
fourth dimension:               18 bytes  ("hostname + "host-24Gju")
name of the measure:            15 bytes  ("cpu_utilization")
value measured:                 4  bytes  ("35.0")
                                ---------
total bytes for the record:     79 bytes
```

* Pricing in us-east-1, us-east-2, us-west-2 regions

Data ingestion: Pricing

$0.50* for 1 million writes of 1KB

time	region	az	vpc	hostname	measure_name	measure_value::double	measure_value::bigint
1602983435238563000	us-east-1	1d	vpc-1a2b3c4d	host-24Gju	cpu_utilization	35.0	null
1602983435238563000	us-east-1	1d	vpc-1a2b3c4d	host-24Gju	mem_utilization	54.9	null

Scenario

- Send 100 different measurements
- Assume that measurements are sent every 5 seconds
- Assume on average each record is 110 bytes

* Pricing in us-east-1, us-east-2, us-west-2 regions

Example: Data ingestion using Python

```
record = {
    'Time': '1602983435238563000',
    'TimeUnit': 'NANOSECONDS',
    'Dimensions': [{'Name': 'region', 'Value': 'us-east-1'},
                   {'Name': 'az', 'Value': '1d'},
                   {'Name': 'hostnsame', 'Value': 'host-24Gju'},
                   {'Name': 'vpc', 'Value': 'vpc-1a2b3c4d'}],
    'MeasureValueType': 'DOUBLE',
    'MeasureName': 'cpu_utilization',
    'MeasureValue': '35.0'         # CPU utilization
}
write_client.write_records(DatabaseName=DATABASE_NAME, TableName=TABLE_NAME, Records=[record], CommonAttributes={})
…
record = {
    'Time': '1602983435238563000',
    'TimeUnit': 'NANOSECONDS',
    'Dimensions': [{'Name': 'region', 'Value': 'us-east-1'},
                   {'Name': 'az', 'Value': '1d'},
                   {'Name': 'hostname', 'Value': 'host-24Gju'},
                   {'Name': 'vpc', 'Value': 'vpc-1a2b3c4d'}],
    'MeasureValueType': 'DOUBLE',
    'MeasureName': 'mem_utilization',
    'MeasureValue': '54.9'         # memory utilization
}
write_client.write_records(DatabaseName=DATABASE_NAME, TableName=TABLE_NAME, Records=[record], CommonAttributes={})
```

Data ingestion: Calculating pricing

100 measures—**100** writes
(1 write/measure)

x12 records/minute—**1,200** writes

x60 minutes/hour—**72,000** writes

x24 hours/day—**1,728,000** writes

x30 days/month—**51,840,000** writes

$0.50* for 1 million writes of 1KB
51,840,000 / 1,000,000 x $0.5 = **$25.92**

* Pricing in us-east-1, us-east-2, us-west-2 regions

Example: Data ingestion using Python

```
record = {
    'Time': '1602983435238563000',
    'TimeUnit': 'NANOSECONDS',
    'Dimensions': [{'Name': 'region', 'Value': 'us-east-1'},
                   {'Name': 'az', 'Value': '1d'},
                   {'Name': 'hostname', 'Value': 'host-24Gju'},
                   {'Name': 'vpc', 'Value': 'vpc-1a2b3c4d'}],
    'MeasureValueType': 'DOUBLE',
    'MeasureName': 'cpu_utilization',
    'MeasureValue': '35.0'         # CPU utilization
}
write_client.write_records(DatabaseName=DATABASE_NAME, TableName=TABLE_NAME, Records=[record], CommonAttributes={})
…
record = {
    'Time': '1602983435238563000',
    'TimeUnit': 'NANOSECONDS',
    'Dimensions': [{'Name': 'region', 'Value': 'us-east-1'},
                   {'Name': 'az', 'Value': '1d'},
                   {'Name': 'hostname', 'Value': 'host-24Gju'},
                   {'Name': 'vpc', 'Value': 'vpc-1a2b3c4d'}],
    'MeasureValueType': 'DOUBLE',
    'MeasureName': 'mem_utilization',
    'MeasureValue': '54.9'         # memory utilization
}
write_client.write_records(DatabaseName=DATABASE_NAME, TableName=TABLE_NAME, Records=[record], CommonAttributes={})
```

Example: Data ingestion using Python

```
records = [{
    'Time': '1602983435238563000',
    'TimeUnit': 'NANOSECONDS',
    'Dimensions': [{'Name': 'region', 'Value': 'us-east-1'},
                   {'Name': 'az', 'Value': '1d'},
                   {'Name': 'hostname', 'Value': 'host-24Gju'},
                   {'Name': 'vpc', 'Value': 'vpc-1a2b3c4d'}],
    'MeasureValueType': 'DOUBLE',
    'MeasureName': 'cpu_utilization',
    'MeasureValue': '35.0'         # CPU utilization
},
{
    'Time': '1602983435238563000',
    'TimeUnit': 'NANOSECONDS',
    'Dimensions': [{'Name': 'region', 'Value': 'us-east-1'},
                   {'Name': 'az', 'Value': '1d'},
                   {'Name': 'hostname', 'Value': 'host-24Gju'},
                   {'Name': 'vpc', 'Value': 'vpc-1a2b3c4d'}],
    'MeasureValueType': 'DOUBLE',
    'MeasureName': 'mem_utilization',
    'MeasureValue': '54.9'         # memory utilization
},
…
]
write_client.write_records(DatabaseName=DATABASE_NAME, TableName=TABLE_NAME, Records=records, CommonAttributes={})
```

Data ingestion: Calculating pricing

100 measures—**11** writes
(1 batch of 11KB)

x12 records/minute—**132** writes

x60 minutes/hour—**7,920** writes

x24 hours/day—**190,080** writes

x30 days/month—**5,702,400** writes

$0.50* for 1 million writes of 1KB
5,702,400 / 1,000,000 x $0.5 = **$2.85**

* Pricing in us-east-1, us-east-2, us-west-2 regions

Example: Data ingestion using Python

```
records = [{
    'Time': '1602983435238563000',
    'TimeUnit': 'NANOSECONDS',
    'Dimensions': [{'Name': 'region', 'Value': 'us-east-1'},
                   {'Name': 'az', 'Value': '1d'},
                   {'Name': 'hostname', 'Value': 'host-24Gju'},
                   {'Name': 'vpc', 'Value': 'vpc-1a2b3c4d'}],
    'MeasureValueType': 'DOUBLE',
    'MeasureName': 'cpu_utilization',
    'MeasureValue': '35.0'          # CPU utilization
},
{
    'Time': '1602983435238563000',
    'TimeUnit': 'NANOSECONDS',
    'Dimensions': [{'Name': 'region', 'Value': 'us-east-1'},
                   {'Name': 'az', 'Value': '1d'},
                   {'Name': 'hostname', 'Value': 'host-24Gju'},
                   {'Name': 'vpc', 'Value': 'vpc-1a2b3c4d'}],
    'MeasureValueType': 'DOUBLE',
    'MeasureName': 'mem_utilization',
    'MeasureValue': '54.9'          # memory utilization
},
…
]
write_client.write_records(DatabaseName=DATABASE_NAME, TableName=TABLE_NAME, Records=records, CommonAttributes={})
```

Example: Data ingestion using Python

```
common_attributes = {
    'Time': '1602983435238563000',
    'TimeUnit': 'NANOSECONDS',
    'Dimensions': [{'Name': 'region', 'Value': 'us-east-1'},
                   {'Name': 'availability_zone', 'Value': '1d'},
                   {'Name': 'hostname', 'Value': 'host-24Gju'},
                   {'Name': 'vpc', 'Value': 'vpc-1a2b3c4d'}],
}
records = [{
    'MeasureValueType': 'DOUBLE',
    'MeasureName': 'cpu_utilization',
    'MeasureValue': '35.0'          # CPU utilization
},
{
    'MeasureValueType': 'DOUBLE',
    'MeasureName': ‘mem_utilization',
    'MeasureValue': '54.9'          # memory utilization
},
…
]
write_client.write_records(DatabaseName=DATABASE_NAME, TableName=TABLE_NAME, Records=records,
CommonAttributes=common_attributes)
```

Data ingestion: Calculating pricing

100 measures—**3** writes
(25 bytes/record, 1 batch~2.5KB)

x12 records/minute—**36** writes

x60 minutes/hour—**2,160** writes

x24 hours/day—**51,840** writes

x30 days/month—**1,555,200** writes

$0.50* for 1 million writes of 1KB
1,555,200 / 1,000,000 x $0.5 = **$0.78**

* Pricing in us-east-1, us-east-2, us-west-2 regions

Data ingestion: Pricing recap

Single measure/write

$0.50* for 1 million writes of 1KB

51,840,000 / 1,000,000 x $0.5 = **$25.92**

---

100 measures/write

$0.50* for 1 million writes of 1KB

5,702,400 / 1,000,000 x $0.5 = **$2.85**

Common attributes + 100 measure/write

$0.50* for 1 million writes of 1KB

1,555,200 / 1,000,000 x $0.5 = **$0.78**

* Pricing in us-east-1, us-east-2, us-west-2 regions

Storage: Memory and magnetic stores

In-memory tier

Handles the ingestion of all data

Timestamp associated with the record must land in the in-memory tier

Automatically handles data deduplication

Optimized for latency sensitive point-in-time queries

$0.036*/GB/hour

Magnetic disk tier

Optimized for high performance analytical queries

Cost effective for long-term storage

$0.03*/GB/month

* Pricing in us-east-1, us-east-2, us-west-2 regions

Best practices: Data storage and ingestion

Use record batching

- A single write_records request can write a batch of up to 100 records
- Each write request has a minimum charge of 1KB

Use common_attributes

- This removes the need to redundantly send dimensional data for each record

Make measure and dimension names only as long as necessary

- There are ingestion and storage costs for user-defined dimension and measure names

Configure the in-memory tier as long as necessary to accommodate late arriving data

- The in-memory tier is optimized for queries that access narrow windows of time

The magnetic tier is better optimized for analytics queries

- The magnetic tier is cost-optimized to store data indefinitely

Query processing



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Query processing: SQL and connectivity

(Mostly) ANSI-2003 SQL for querying

Time-series, interpolation, and gap-filling functions

250+ scalar, aggregate, and windowing functions

Pricing is $0.01*/GB of data scanned

Data is queried using the AWS SDK or AWS CLI

Java, Python, Node.js, .NET, etc.

JDBC Driver

Amazon QuickSight support

Grafana (Open Source Edition)

* Pricing in us-east-1, us-east-2, us-west-2 regions

Example: Amazon Timestream SQL

```
SELECT region, az, hostname, bin(time, 15s) AS binned_timestamp,
  round(avg(measure_value::double), 2) AS avg_cpu_utilization,
  round(approx_percentile(measure_value::double, 0.9), 2) AS p90_cpu_utilization,
  round(approx_percentile(measure_value::double, 0.95), 2) AS p95_cpu_utilization,
  round(approx_percentile(measure_value::double, 0.99), 2) AS p99_cpu_utilization
FROM devops.host_metrics
WHERE measure_name = 'cpu_utilization’          -- Predicate on measure_name
AND time > ago(2h)                              -- Predicate on time
AND hostname = 'host-24Gju'                     -- Optional predicates on other dimensions
GROUP BY region, hostname, az, bin(time, 15s)   -- bin and GROUP BY time
ORDER BY binned_timestamp ASC
```

Best practices: Query processing

Queries should have a predicate on the measure_name

Queries should have a predicate on time

Most queries should have a predicate on one or more dimensions

Predicates on time, measure_name, and dimensions can reduce data scan charges by leveraging range-restricted scans

Queries using a GROUP BY clause will perform faster if the first grouping dimension has a high cardinality

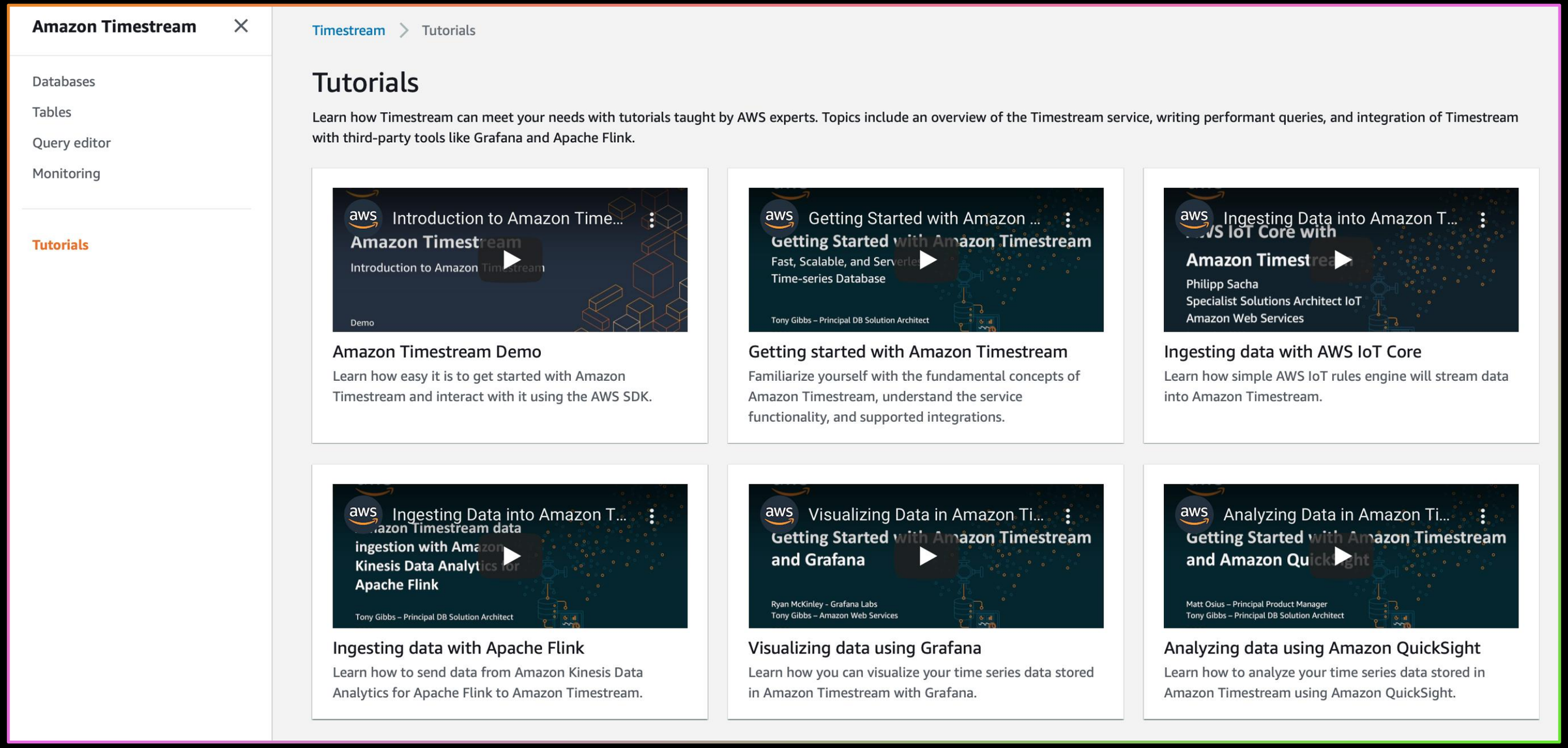
Only select the dimensions that are necessary; unnecessary columns read can impact both performance and cost

Additional resources



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Video tutorials: Amazon Timestream console



AWS Labs on GitHub: amazon-timestream-tools

Sample applications (available in various languages)
https://github.com/awslabs/amazon-timestream-tools/tree/master/sample_apps

Amazon Kinesis Data Analytics for Apache Flink connector example
https://github.com/awslabs/amazon-timestream-tools/tree/master/integrations/flink_connector

Ingesting data with AWS IoT core
https://github.com/awslabs/amazon-timestream-tools/tree/master/integrations/iot_core

Amazon SageMaker example
https://github.com/awslabs/amazon-timestream-tools/tree/master/integrations/sagemaker

Continuous data-generator tools
https://github.com/awslabs/amazon-timestream-tools/tree/master/tools/continuous-ingestor

Thank you!



AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.



Please complete the session survey

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.