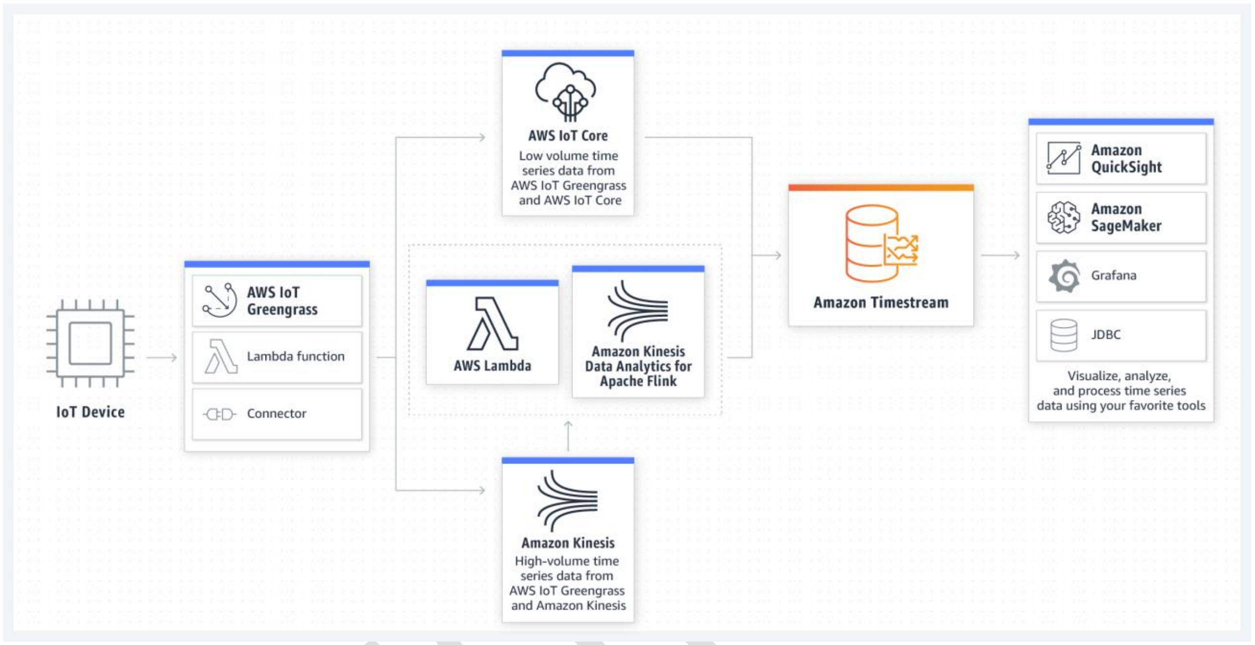
AWS new Time Stream demo

Docs, Videos, Demos:

https://aws.amazon.com/timestream/?nc=sn&loc=0


IoT Device
AWS IoT Greengrass
Lambda function
Connector
AWS IoT Core
Low volume time series data from AWS IoT Greengrass and AWS IoT Core
AWS Lambda
Amazon Kinesis Data Analytics for Apache Flink
Amazon Kinesis
High-volume time series data from AWS IoT Greengrass and Amazon Kinesis
Amazon Timestream
Amazon QuickSight
Amazon SageMaker
Grafana
JDBC
Visualize, analyze, and process time series data using your favorite tools

Features:

https://aws.amazon.com/timestream/features/?nc=sn&loc=2

Pricing:

https://aws.amazon.com/timestream/pricing/?nc=sn&loc=3

Videos, Best Practices, SDKs:

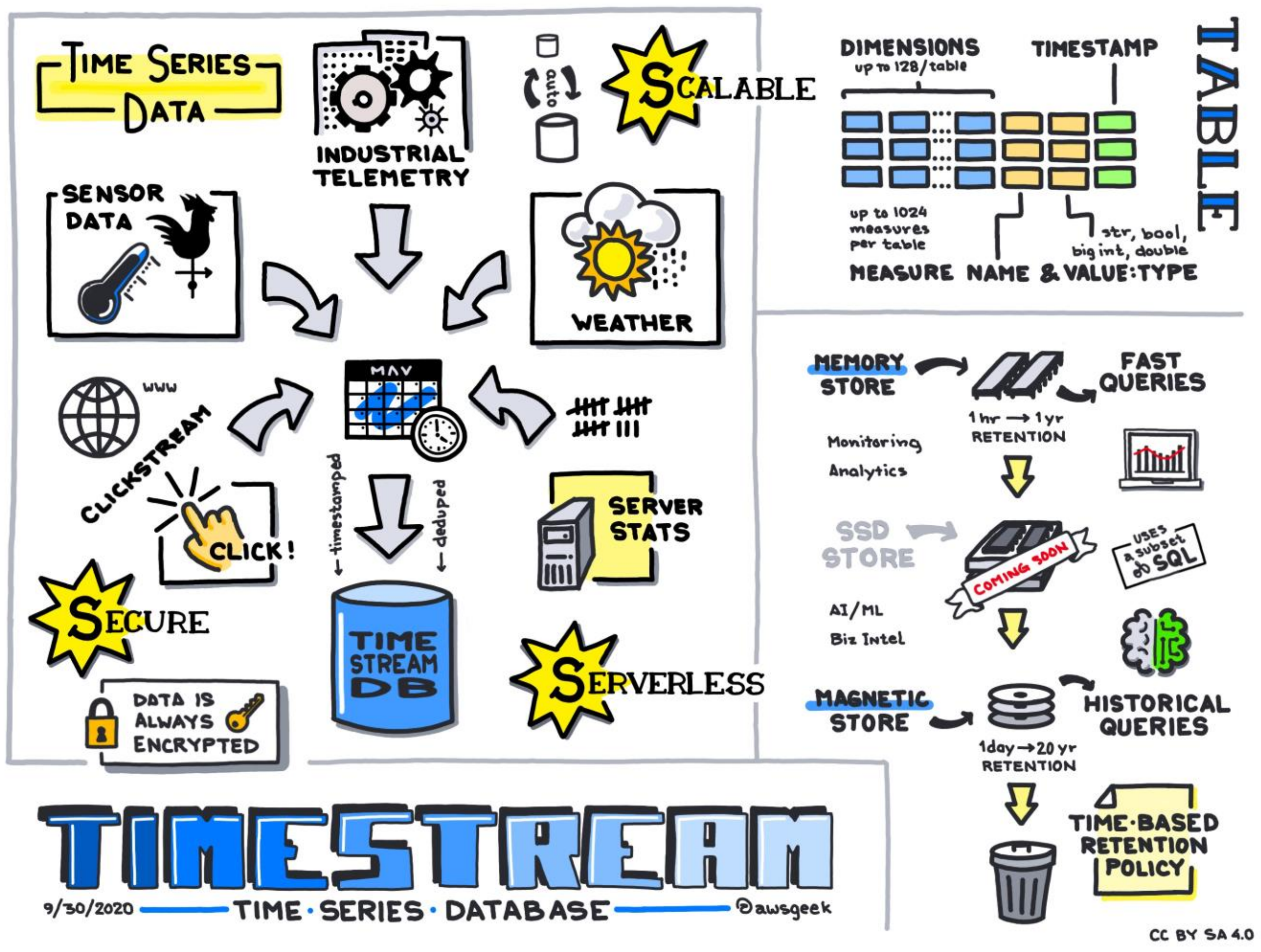
https://aws.amazon.com/timestream/getting-started/?nc=sn&loc=4

Tutorials:

https://us-east-2.console.aws.amazon.com/timestream/home?region=us-east-2#tutorials

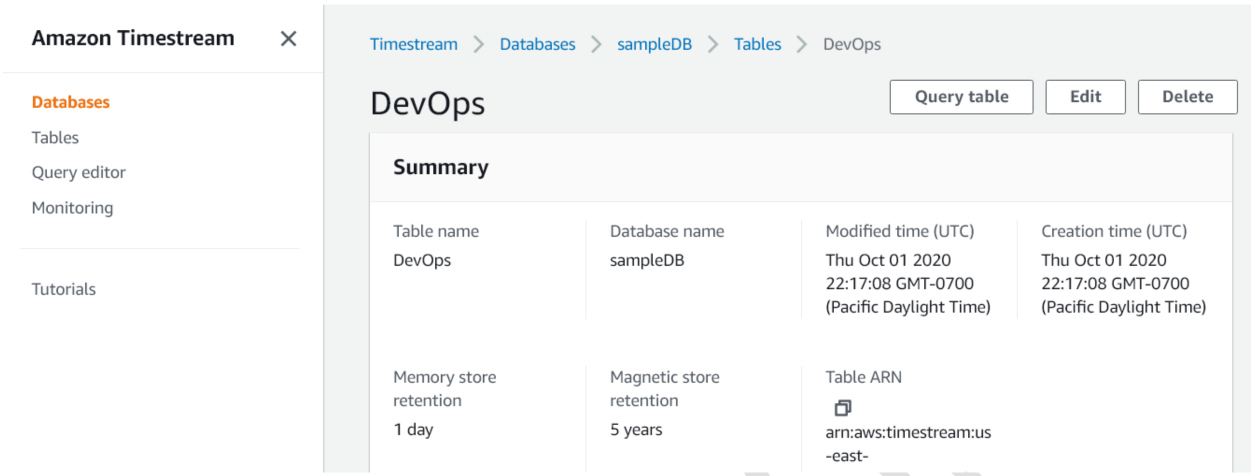
Git - https://github.com/awslabs/amazon-timestream-tools

Nice drawing by https://www.awsgeek.com/Amazon-Timestream/



Demo:

1/ Easy quick start with sample DB in 1 min



Amazon Timestream

- Databases
- Tables
- Query editor
- Monitoring
- Tutorials

Timestream > Databases > sampleDB > Tables > DevOps

DevOps

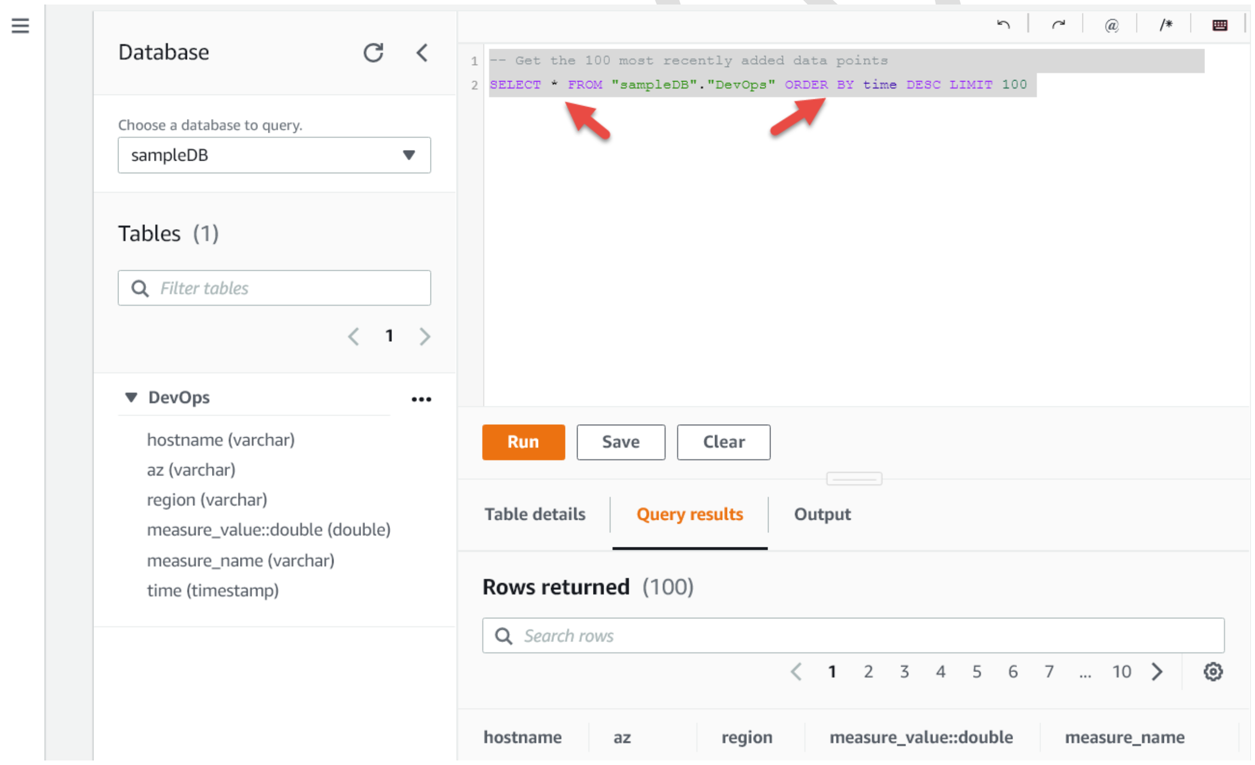
Query table | Edit | Delete

Summary

Table name	Database name	Modified time (UTC)	Creation time (UTC)
DevOps	sampleDB	Thu Oct 01 2020 22:17:08 GMT-0700 (Pacific Daylight Time)	Thu Oct 01 2020 22:17:08 GMT-0700 (Pacific Daylight Time)

Memory store retention	Magnetic store retention	Table ARN
1 day	5 years	arn:aws:timestream:us -east-

2/ Easy data preview SQL syntax



Database

Choose a database to query.

sampleDB

Tables (1)

Filter tables

DevOps

- hostname (varchar)
- az (varchar)
- region (varchar)
- measure_value::double (double)
- measure_name (varchar)
- time (timestamp)

```
-- Get the 100 most recently added data points
SELECT * FROM "sampleDB"."DevOps" ORDER BY time DESC LIMIT 100
```

Run | Save | Clear

Table details | Query results | Output

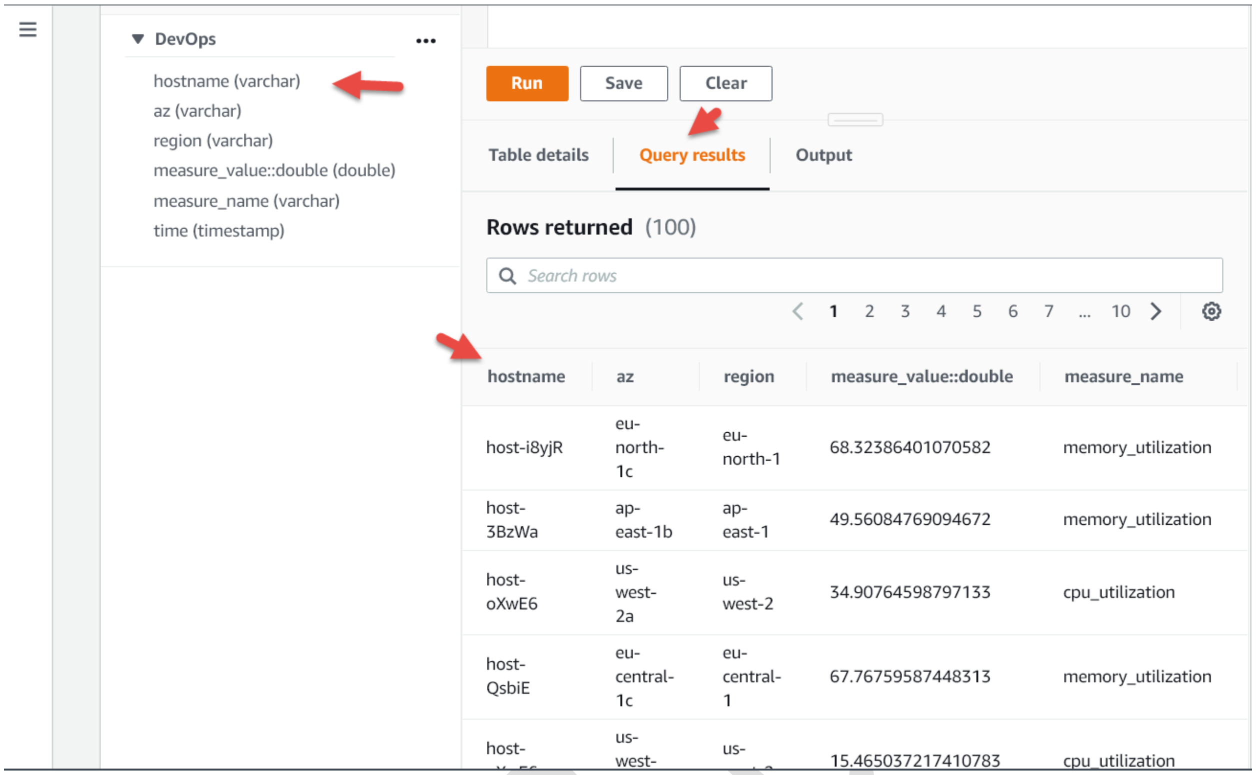
Rows returned (100)

Search rows

1 2 3 4 5 6 7 ... 10

hostname	az	region	measure_value::double	measure_name

3/ Quick SQL results

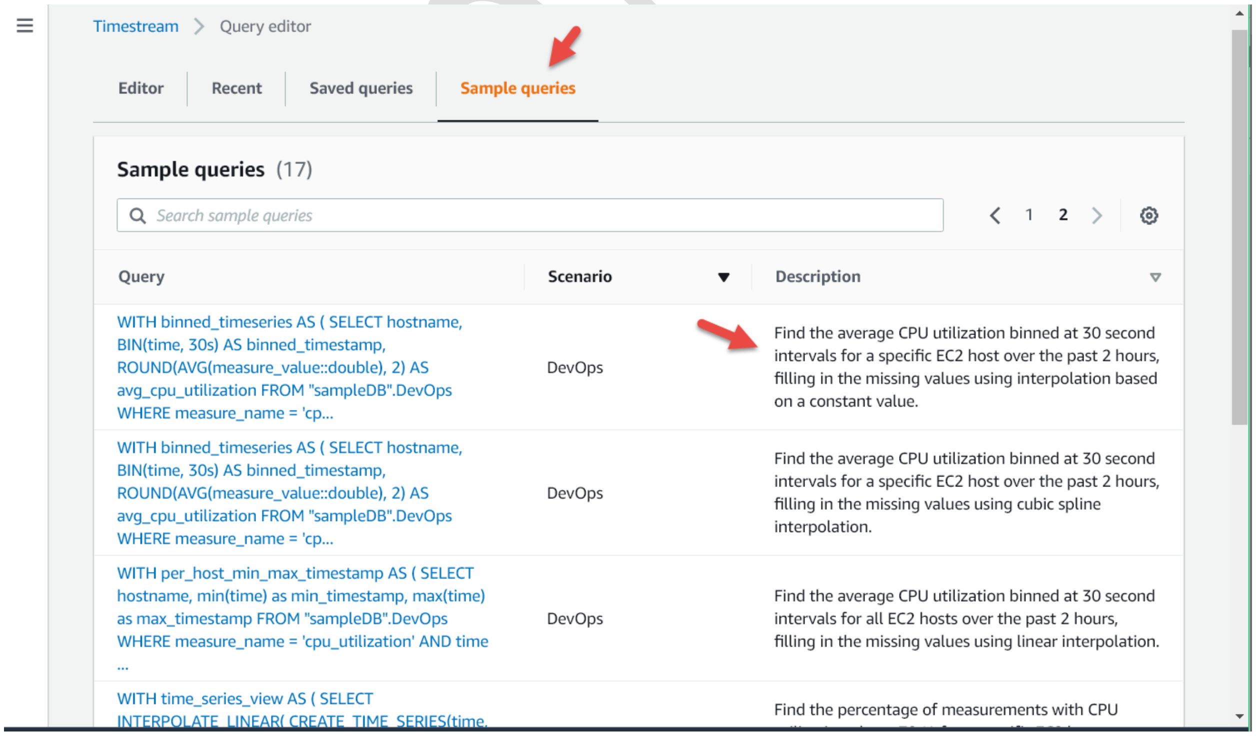


hostname (varchar)
az (varchar)
region (varchar)
measure_value::double (double)
measure_name (varchar)
time (timestamp)

Rows returned (100)

hostname	az	region	measure_value::double	measure_name
host-i8yjR	eu-north-1c	eu-north-1	68.32386401070582	memory_utilization
host-3BzWa	ap-east-1b	ap-east-1	49.56084769094672	memory_utilization
host-oXwE6	us-west-2a	us-west-2	34.90764598797133	cpu_utilization
host-QsbiE	eu-central-1c	eu-central-1	67.76759587448313	memory_utilization
host-	us-west-	us-	15.465037217410783	cpu_utilization

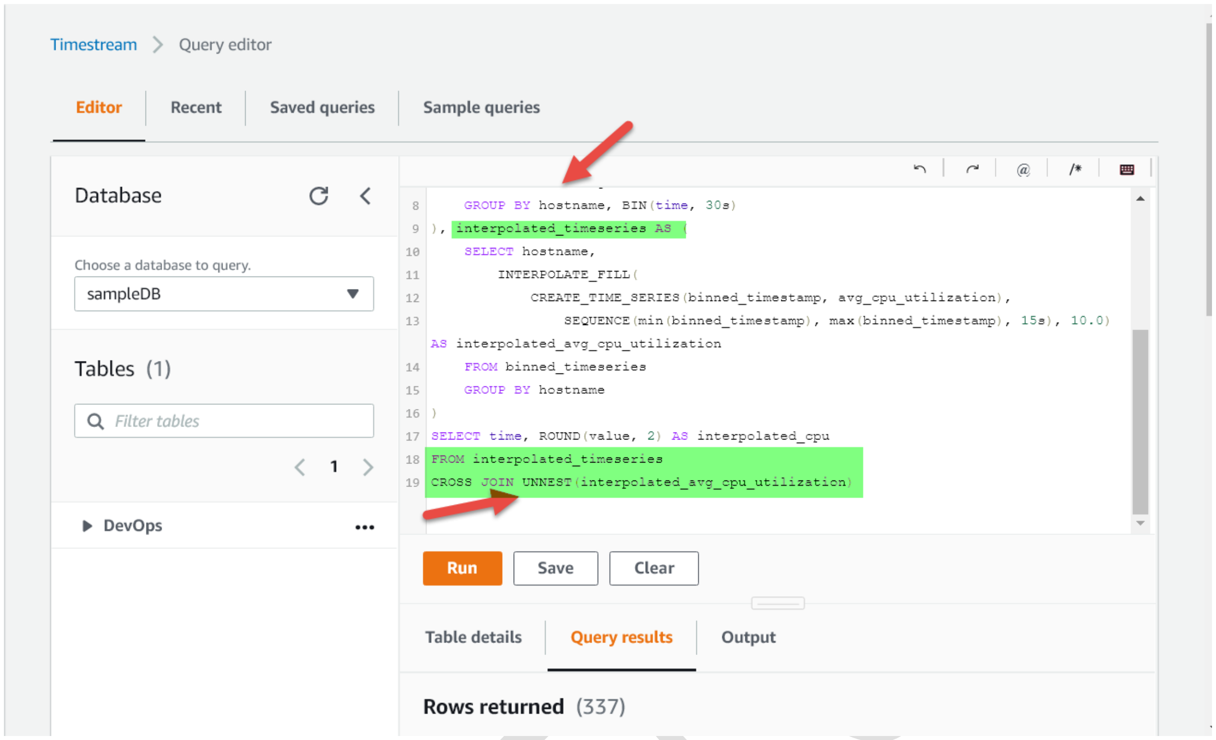
4/ Many Sample Queries



Sample queries (17)

Query	Scenario	Description
WITH binned_timeseries AS (SELECT hostname, BIN(time, 30s) AS binned_timestamp, ROUND(AVG(measure_value::double), 2) AS avg_cpu_utilization FROM "sampleDB".DevOps WHERE measure_name = 'cp...	DevOps	Find the average CPU utilization binned at 30 second intervals for a specific EC2 host over the past 2 hours, filling in the missing values using interpolation based on a constant value.
WITH binned_timeseries AS (SELECT hostname, BIN(time, 30s) AS binned_timestamp, ROUND(AVG(measure_value::double), 2) AS avg_cpu_utilization FROM "sampleDB".DevOps WHERE measure_name = 'cp...	DevOps	Find the average CPU utilization binned at 30 second intervals for a specific EC2 host over the past 2 hours, filling in the missing values using cubic spline interpolation.
WITH per_host_min_max_timestamp AS (SELECT hostname, min(time) as min_timestamp, max(time) as max_timestamp FROM "sampleDB".DevOps WHERE measure_name = 'cpu_utilization' AND time ...	DevOps	Find the average CPU utilization binned at 30 second intervals for all EC2 hosts over the past 2 hours, filling in the missing values using linear interpolation.
WITH time_series_view AS (SELECT INTERPOLATE_LINEAR(CREATE_TIME_SERIES(time,		Find the percentage of measurements with CPU

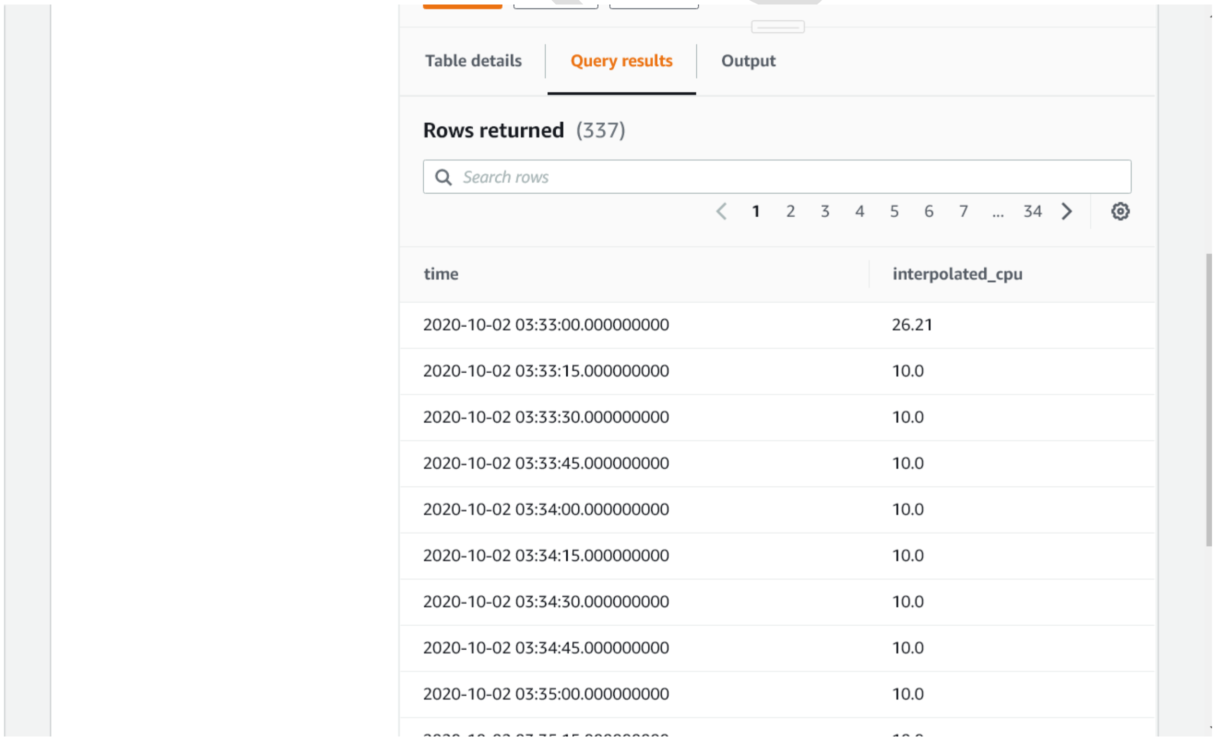
5/ SQL for sample query to get interpolated CPU average



```
    GROUP BY hostname, BIN(time, 30s)
), interpolated_timeseries AS (
    SELECT hostname,
        INTERPOLATE_FILL(
            CREATE_TIME_SERIES(binned_timestamp, avg_cpu_utilization),
                SEQUENCE(min(binned_timestamp), max(binned_timestamp), 15s), 10.0)
AS interpolated_avg_cpu_utilization
    FROM binned_timeseries
    GROUP BY hostname
)
SELECT time, ROUND(value, 2) AS interpolated_cpu
FROM interpolated_timeseries
CROSS JOIN UNNEST(interpolated_avg_cpu_utilization)
```

Rows returned (337)

6/ output interpolated CPU average in UTC time:



Rows returned (337)

time	interpolated_cpu
2020-10-02 03:33:00.000000000	26.21
2020-10-02 03:33:15.000000000	10.0
2020-10-02 03:33:30.000000000	10.0
2020-10-02 03:33:45.000000000	10.0
2020-10-02 03:34:00.000000000	10.0
2020-10-02 03:34:15.000000000	10.0
2020-10-02 03:34:30.000000000	10.0
2020-10-02 03:34:45.000000000	10.0
2020-10-02 03:35:00.000000000	10.0

7/ detail output for above query

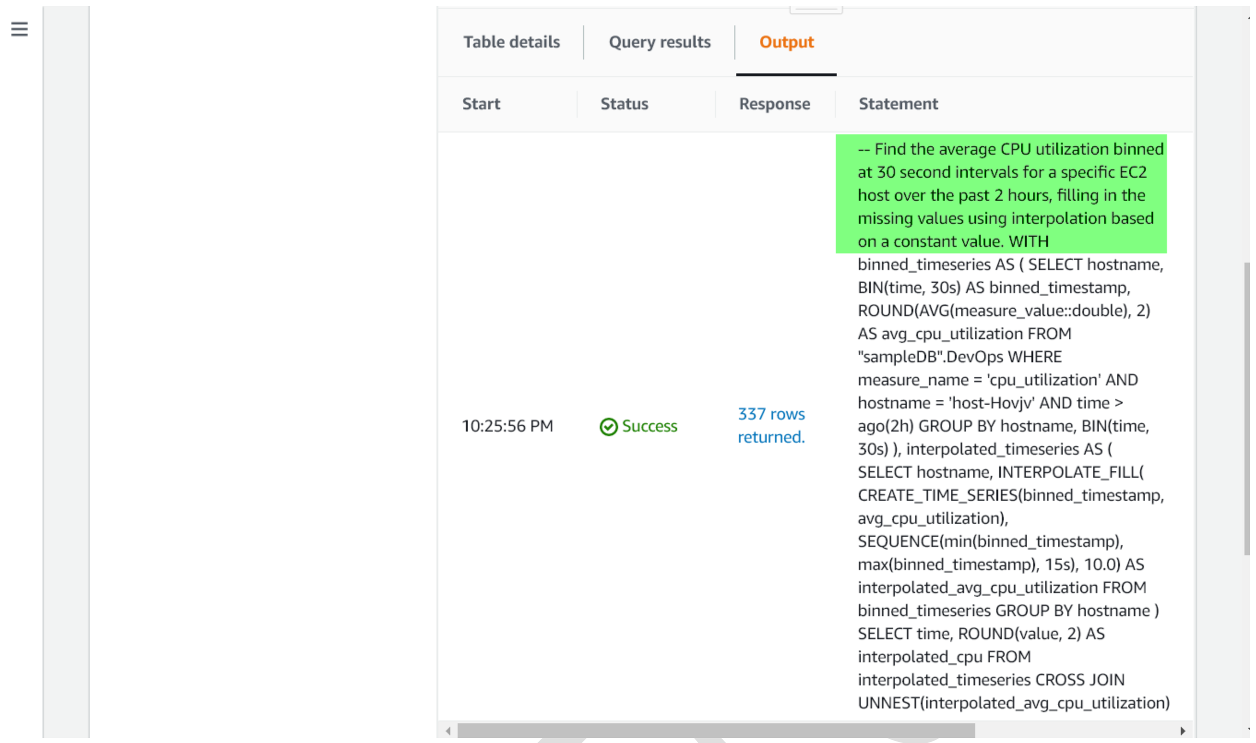


Table details | Query results | Output

Start	Status	Response	Statement
10:25:56 PM	Success	337 rows returned.	-- Find the average CPU utilization binned at 30 second intervals for a specific EC2 host over the past 2 hours, filling in the missing values using interpolation based on a constant value. WITH binned_timeseries AS (SELECT hostname, BIN(time, 30s) AS binned_timestamp, ROUND(AVG(measure_value::double), 2) AS avg_cpu_utilization FROM "sampleDB".DevOps WHERE measure_name = 'cpu_utilization' AND hostname = 'host-Hovjv' AND time > ago(2h) GROUP BY hostname, BIN(time, 30s)), interpolated_timeseries AS (SELECT hostname, INTERPOLATE_FILL(CREATE_TIME_SERIES(binned_timestamp, avg_cpu_utilization), SEQUENCE(min(binned_timestamp), max(binned_timestamp), 15s), 10.0) AS interpolated_avg_cpu_utilization FROM binned_timeseries GROUP BY hostname) SELECT time, ROUND(value, 2) AS interpolated_cpu FROM interpolated_timeseries CROSS JOIN UNNEST(interpolated_avg_cpu_utilization)

8/ billing for this demo:

Timestream

$0.00

US East (Ohio)

Amazon Timestream USE2-DataIngestion-Bytes

$0.00

price per GB ingested in Timestream - US East (Ohio)

0.000163 GB

$0.00

Amazon Timestream USE2-DataScanned-Bytes

$0.00

price per GB scanned by Timestream queries - US East (Ohio)

0.028 GB

$0.00

Usage and recurring charges for this statement period will be charged on your next billing date. Estimated charges shown on this page, or shown on any notifications that we send to you, may differ from your actual charges for this statement period. This is because estimated charges presented on this page do not include usage charges accrued during this statement period after the date you view this page. Similarly, information about estimated charges sent to you in a notification do not include usage charges accrued during this statement period after the date we send you the notification. One-time fees and subscription charges are assessed separately from usage and reoccurring charges, on the date that they occur.