AWS DynamoDB & CloudWatch

CS516 – Cloud Computing

Computer Science Department

Maharishi International University

Maharishi International University - Fairfield, Iowa

Content

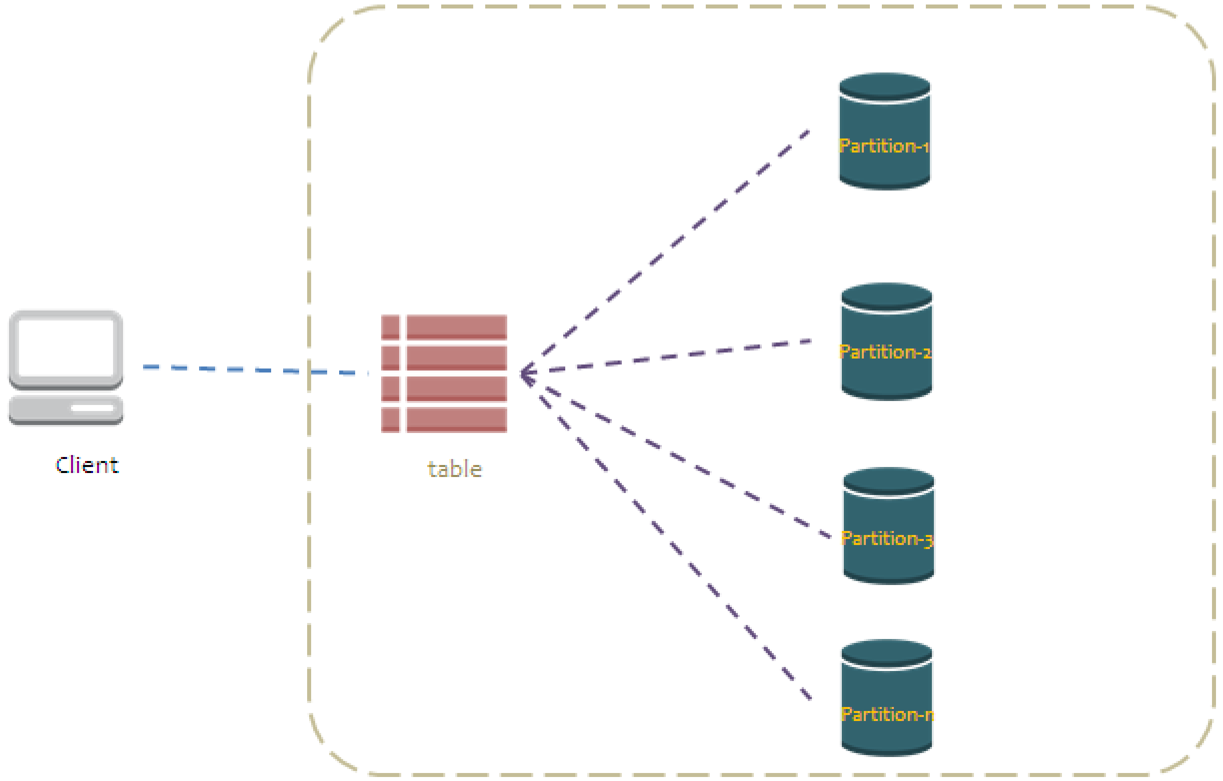
- NoSQL overview
- DynamoDB features
- Basics: Partition key, sort key, indexes
- DynamoDB streams
- Read consistency
- Read/write capacity modes and units
- Backups
- Global table
- DynamoDB TTL
- SQL vs NoSQL

Content

- Collect and store logs
- Custom metrics and how to create it
  - PutMetricData API
  - CloudWatch Agent
- CloudWatch Insights
- Dashboard
- Synthetics

What are NoSQL databases?

- NoSQL database is a mechanism for storage and retrieval of data that is modeled **other than the tabular relations** used in relational databases.
- NoSQL databases are widely recognized for their **ease of development**, functionality, and performance at scale.
- NoSQL databases break the traditional mindset of storing data at a single location. Instead, NoSQL distributes and stores data over a set of **multiple servers**.



Why should you use a NoSQL database?

- **Flexibility**: NoSQL databases generally provide flexible schemas that enable faster and more iterative development. The flexible data model makes NoSQL databases ideal for semi-structured and unstructured data.
- **Scalability**: NoSQL databases are generally designed to scale out by using distributed clusters of hardware instead of scaling up by adding expensive and robust servers. Some cloud providers handle these operations behind-the-scenes as a fully managed service.
- **High-performance**: NoSQL database are optimized for specific data models and access patterns that enable higher performance than trying to accomplish similar functionality with relational databases.

Types of NoSQL Databases

- **Key-value**: Key-value databases are highly partitionable and allow horizontal scaling at scales that other types of databases cannot achieve. Amazon **DynamoDB** is designed to provide consistent single-digit millisecond latency for any scale of workloads.
- **Document**: In application code, data is represented often as an object or JSON-like document because it is an efficient and intuitive data model for developers. Amazon DocumentDB (with MongoDB compatibility).
- **Graph**: A graph database's purpose is to make it easy to build and run applications that work with highly connected datasets. Typical use cases for a graph database include social networking, recommendation engines, fraud detection, and knowledge graphs. Amazon Neptune is a fully-managed graph database service.
- **In-memory**: Gaming and ad-tech applications have use cases such as leaderboards, session stores, and real-time analytics that require microsecond response times. Amazon ElastiCache offers Memcached and Redis, to provide low-latency, high-throughput.
- **Search**: Many applications output logs to help developers troubleshoot issues. Amazon Elasticsearch Service (Amazon ES) is purpose built for providing near-real-time visualizations and analytics of machine-generated data by indexing, aggregating, and searching semistructured logs and metrics.

Amazon DynamoDB

Amazon DynamoDB is a key-value and document database that delivers **single-digit millisecond** performance at **any scale**.

It's a fully managed, multi-region, multi-active, durable database with built-in security, backup and restore, and in-memory caching for internet-scale applications.

DynamoDB can handle more than 10 trillion requests per day and can support peaks of more than 20 million requests per second.

Many of the world's fastest growing businesses such as Lyft, Airbnb, and Redfin as well as enterprises such as Samsung, Toyota, and Capital One depend on the scale and performance of DynamoDB to support their mission-critical workloads. Snapchat's largest storage write workload, moved to DynamoDB.

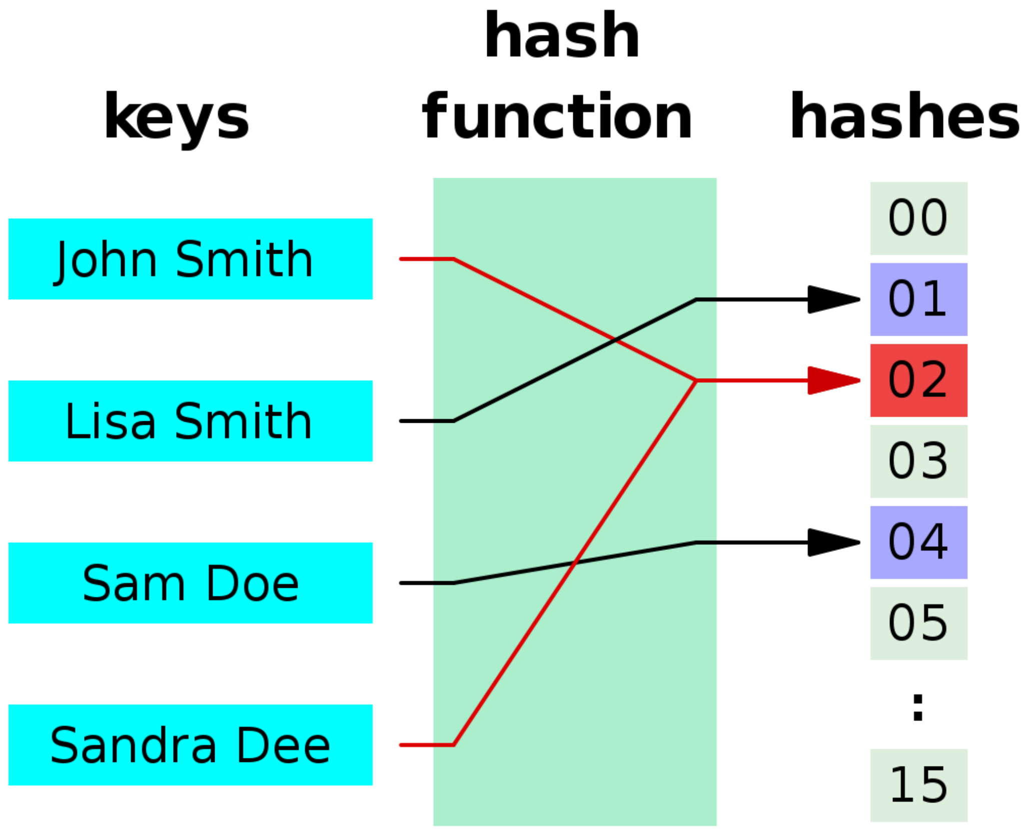
DynamoDB features

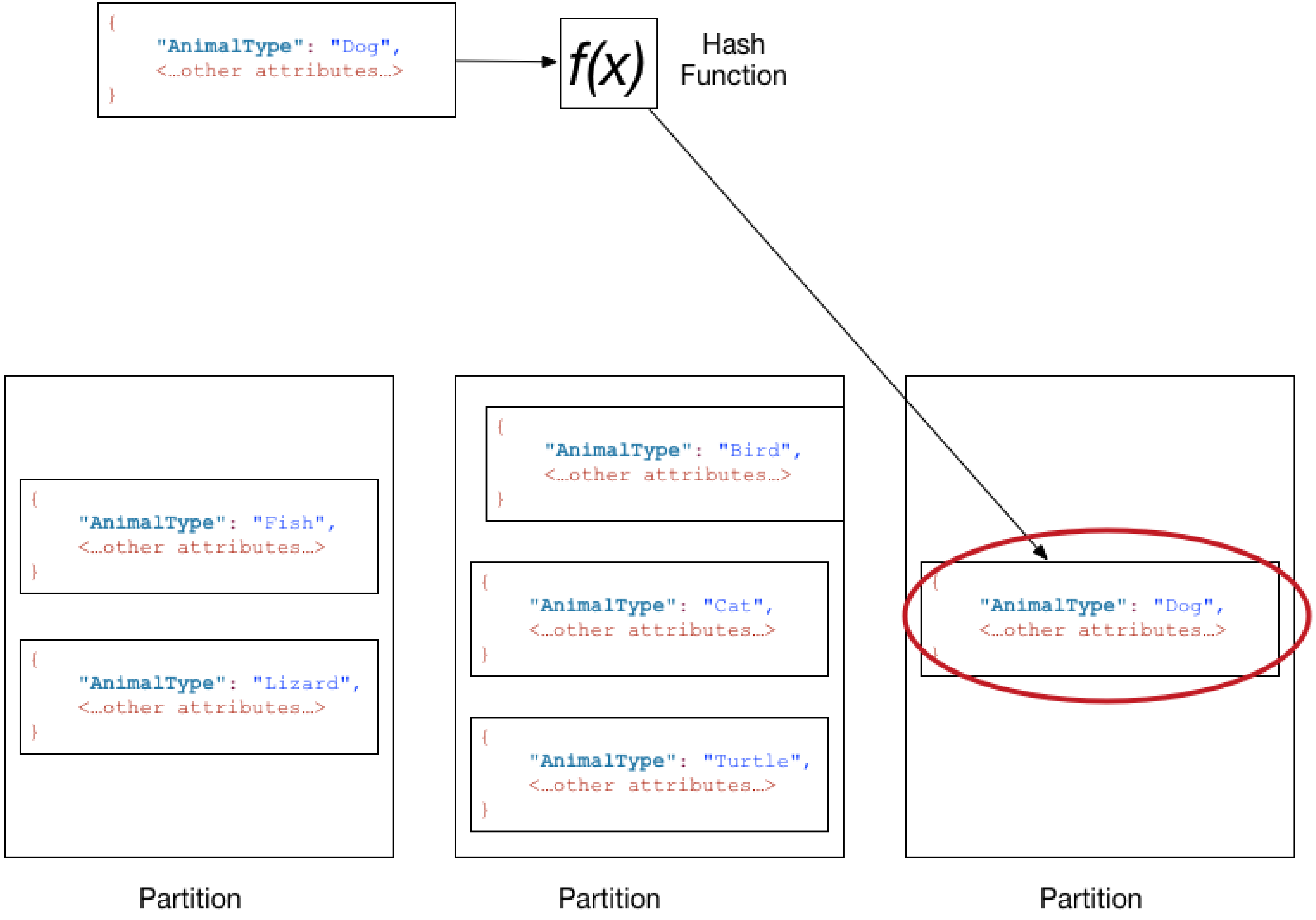
- DynamoDB supports both **key-value** and **document** data models. This enables DynamoDB to have a **flexible schema**, so each row can have any number of columns at any point in time.
- **Microsecond latency** with DynamoDB Accelerator (DAX).
- Automated **global replication** with global tables.
- Amazon Kinesis Data Streams for DynamoDB **captures item-level changes** in your DynamoDB tables as a Kinesis data stream. This feature enables you to build advanced streaming applications such as real-time log aggregation, real-time business analytics, and Internet of Things data capture.

Partition key

It is a simple primary key.

DynamoDB uses the partition key's value as input to an internal **hash function**. The output from the hash function determines the partition (physical storage internal to DynamoDB) in which the item will be stored.



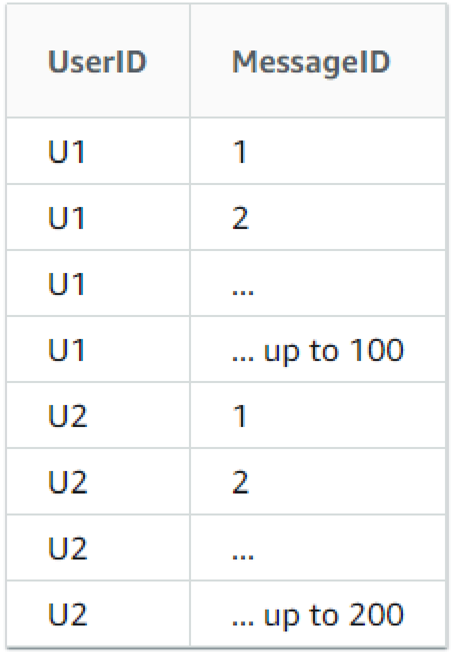

{
"AnimalType": "Dog",
<...other attributes...>
}
f(x)
Hash Function
{
"AnimalType": "Fish",
<...other attributes...>
}
{
"AnimalType": "Lizard",
<...other attributes...>
}
{
"AnimalType": "Bird",
<...other attributes...>
}
{
"AnimalType": "Cat",
<...other attributes...>
}
{
"AnimalType": "Turtle",
<...other attributes...>
}
{
"AnimalType": "Dog",
<...other attributes...>
}
Partition
Partition
Partition

Sort (Range) key

- There can be duplicate partition keys. We can differentiate the item with a sort key and select one item.
- Also, the sort key is used to represent the one-to-many relationship.
- With the sort key, we can improve the speed of writing data and effectively query.

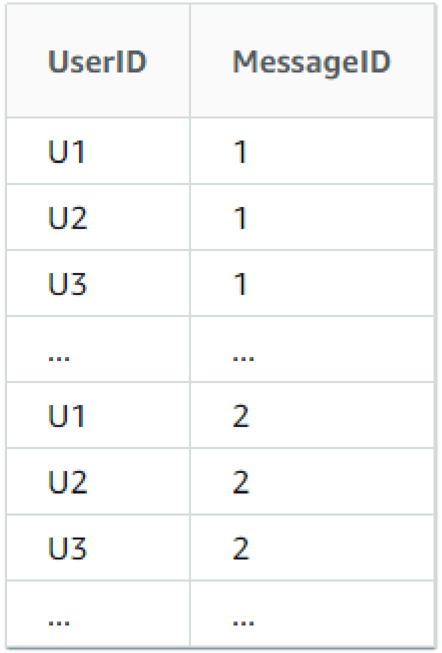
In this example, there is no throttling when inserting data for the user 2 (U2) with the sort key.

Without sort key



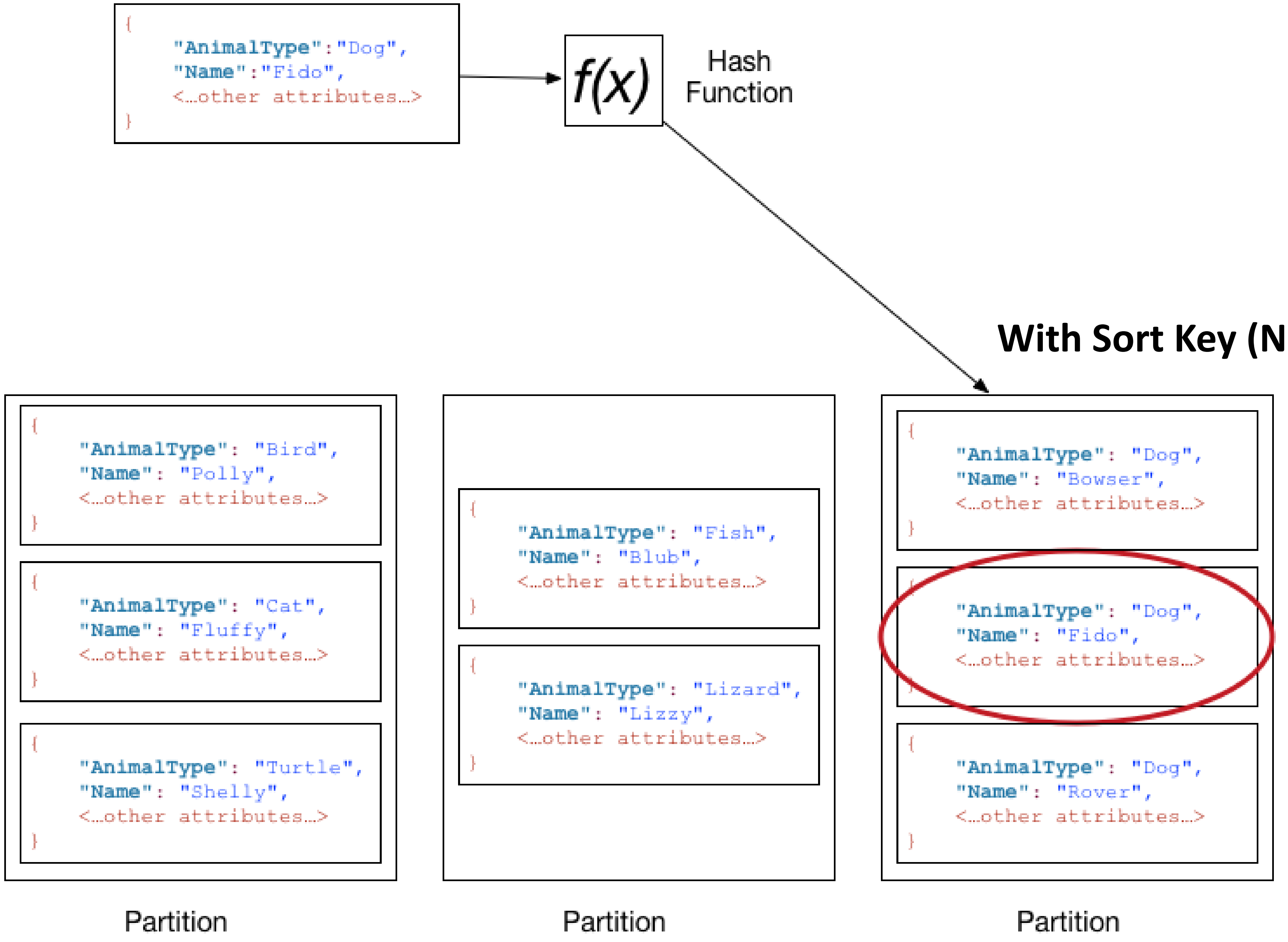
UserID	MessageID
U1	1
U1	2
U1	...
U1	... up to 100
U2	1
U2	2
U2	...
U2	... up to 200

With sort key



UserID	MessageID
U1	1
U2	1
U3	1
...	...
U1	2
U2	2
U3	2
...	...

With Sort Key (Name)



```
{
    "AnimalType":"Dog",
    "Name":"Fido",
    <…other attributes…>
}
```

```
{
    "AnimalType": "Bird",
    "Name": "Polly",
    <…other attributes…>
}
{
    "AnimalType": "Cat",
    "Name": "Fluffy",
    <…other attributes…>
}
{
    "AnimalType": "Turtle",
    "Name": "Shelly",
    <…other attributes…>
}
```

```
{
    "AnimalType": "Fish",
    "Name": "Blub",
    <…other attributes…>
}
{
    "AnimalType": "Lizard",
    "Name": "Lizzy",
    <…other attributes…>
}
```

```
{
    "AnimalType": "Dog",
    "Name": "Bowser",
    <…other attributes…>
}
{
    "AnimalType": "Dog",
    "Name": "Fido",
    <…other attributes…>
}
{
    "AnimalType": "Dog",
    "Name": "Rover",
    <…other attributes…>
}
```

Composite primary key

A composite primary key is **composed of two attributes**. The first attribute is the **partition key**, and the second attribute is the **sort key**.

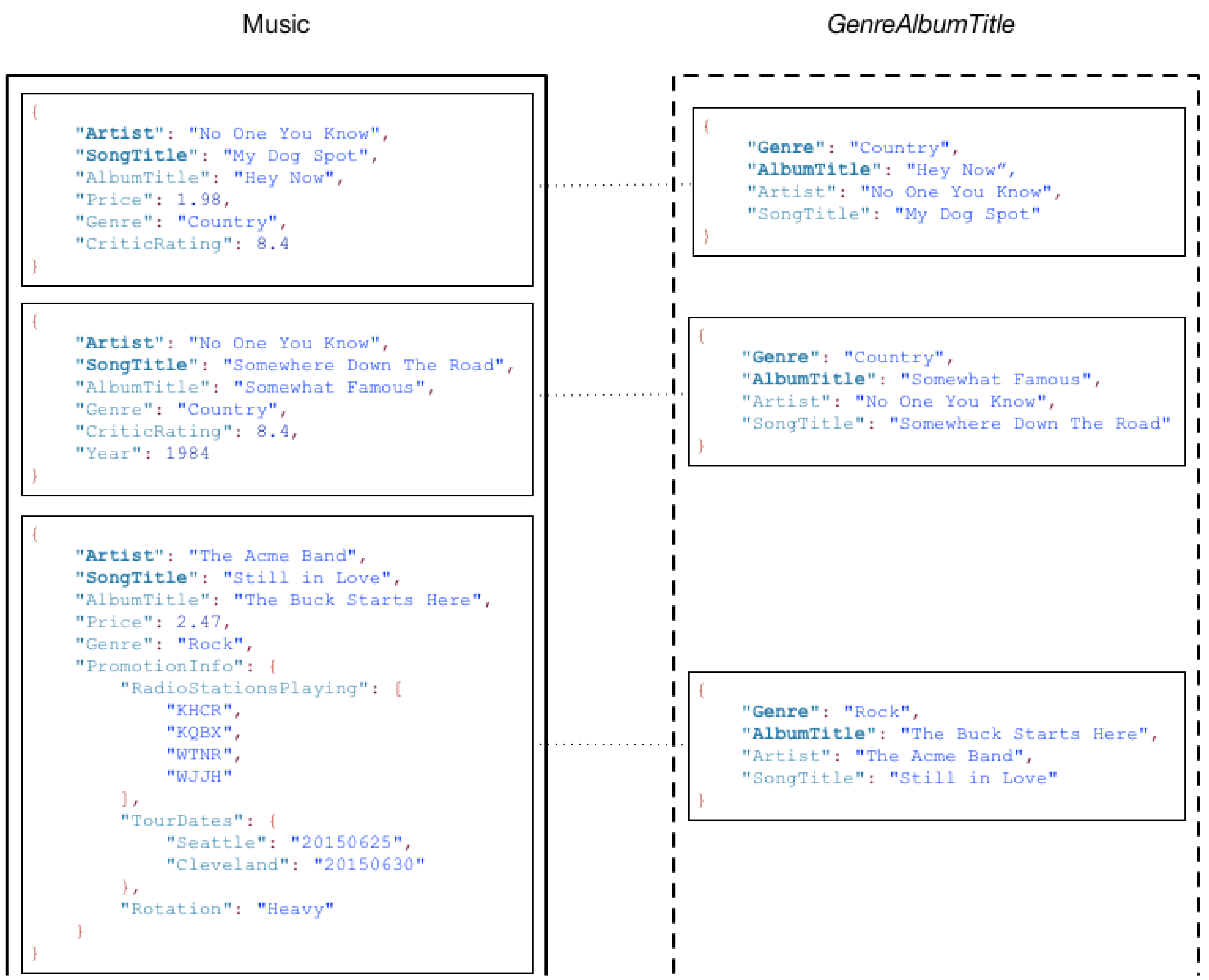
DynamoDB uses the partition key value as input to an internal hash function. The output from the hash function determines the partition (physical storage internal to DynamoDB) in which the item will be stored. All items with the same partition key value are stored together, **in sorted order** by sort key value.

Partition key – Artist

Sort key – SongTitle

What if you also wanted to query the data by **Genre** and **AlbumTitle**?

You could create an **index** on Genre and AlbumTitle, and then query the index



```
{
    "Artist": "No One You Know",
    "SongTitle": "My Dog Spot",
    "AlbumTitle": "Hey Now",
    "Price": 1.98,
    "Genre": "Country",
    "CriticRating": 8.4
}
```

```
{
    "Artist": "No One You Know",
    "SongTitle": "Somewhere Down The Road",
    "AlbumTitle": "Somewhat Famous",
    "Genre": "Country",
    "CriticRating": 8.4,
    "Year": 1984
}
```

```
{
    "Artist": "The Acme Band",
    "SongTitle": "Still in Love",
    "AlbumTitle": "The Buck Starts Here",
    "Price": 2.47,
    "Genre": "Rock",
    "PromotionInfo": {
        "RadioStationsPlaying": [
            "KHCR",
            "KQBX",
            "WTNR",
            "WJJH"
        ],
        "TourDates": {
            "Seattle": "20150625",
            "Cleveland": "20150630"
        },
        "Rotation": "Heavy"
    }
}
```

```
{
    "Genre": "Country",
    "AlbumTitle": "Hey Now",
    "Artist": "No One You Know",
    "SongTitle": "My Dog Spot"
}
```

```
{
    "Genre": "Country",
    "AlbumTitle": "Somewhat Famous",
    "Artist": "No One You Know",
    "SongTitle": "Somewhere Down The Road"
}
```

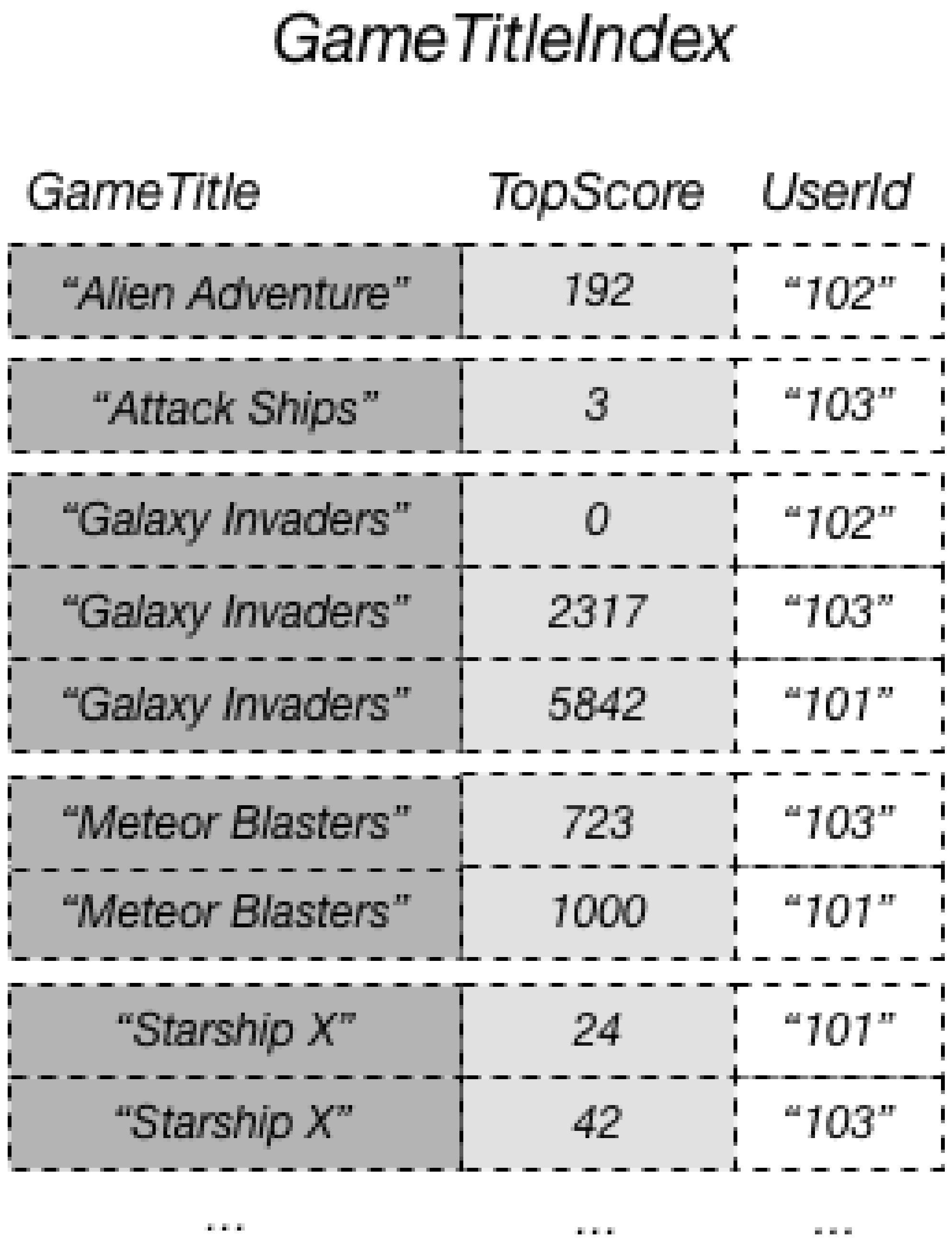
```
{
    "Genre": "Rock",
    "AlbumTitle": "The Buck Starts Here",
    "Artist": "The Acme Band",
    "SongTitle": "Still in Love"
}
```

Indexes

A secondary index provides **fast access** and lets you query the data in the table using an **alternate key**, in addition to queries against the primary key.

GameScores

UserId	GameTitle	TopScore	TopScoreDateTime	Wins	Losses	
"101"	"Galaxy Invaders"	5842	"2015-09-15:17:24:31"	21	72	...
"101"	"Meteor Blasters"	1000	"2015-10-22:23:18:01"	12	3	...
"101"	"Starship X"	24	"2015-08-31:13:14:21"	4	9	...
"102"	"Alien Adventure"	192	"2015-07-12:11:07:56"	32	192	...
"102"	"Galaxy Invaders"	0	"2015-09-18:07:33:42"	0	5	...
"103"	"Attack Ships"	3	"2015-10-19:01:13:24"	1	8	...
"103"	"Galaxy Invaders"	2317	"2015-09-11:06:53:00"	40	3	...
"103"	"Meteor Blasters"	723	"2015-10-19:01:13:24"	22	12	...
"103"	"Starship X"	42	"2015-07-11:06:53:00"	4	19	...
...	...	...	...	...	...	



GameTitleIndex

GameTitle	TopScore	UserId
"Alien Adventure"	192	"102"
"Attack Ships"	3	"103"
"Galaxy Invaders"	0	"102"
"Galaxy Invaders"	2317	"103"
"Galaxy Invaders"	5842	"101"
"Meteor Blasters"	723	"103"
"Meteor Blasters"	1000	"101"
"Starship X"	24	"101"
"Starship X"	42	"103"
...	...	...

If you need an item (game score) by GameTitle, you would need to **scan** every single record that is inefficient as your table gets bigger.

To speed up, you can create an index with GameTitle. The table's **primary key attributes** (partition key) are always projected into an index.

Read Consistency

DynamoDB supports eventually consistent and strongly consistent reads.

Eventually Consistent Reads - When you read data from a DynamoDB table, the response might not reflect the results of a recently completed write operation. The response might include some stale data. If you repeat your read request after a short time, the response should return the latest data.

Strongly Consistent Reads - When you request a strongly consistent read, DynamoDB returns a response with the most up-to-date data, reflecting the updates from all prior write operations that were successful in all nodes.

However, this consistency comes with some disadvantages:

- Strongly consistent reads may have higher latency
- Strongly consistent reads use more throughput capacity

Read/write capacity modes

Amazon DynamoDB has two read/write capacity modes for processing reads and writes on your tables.

On-Demand Mode – it is a flexible billing option capable of serving thousands of requests per second without capacity planning. On-demand mode is a good option if there is unknown workloads, unpredictable traffic.

Provisioned Mode – You specify the number of reads and writes per second that you require for your application. You can use auto scaling to adjust your table's provisioned capacity automatically in response to traffic changes.

Read/write capacity units

With one **read capacity unit** (RCU), you can read 4 KB data per second.

With one **write capacity unit** (WCU), you can write 1 KB data per second.

ACID transactions

Transactions provide atomicity, consistency, isolation, and durability (ACID) in DynamoDB, helping you to maintain **data correctness** in your applications.

Amazon DynamoDB transactions simplify the developer experience of making coordinated, **all-or-nothing** changes to multiple items both within and across tables.

- **TransactWriteItems** - group multiple *Put, Update, Delete*, and *ConditionCheck* actions
- **TransactGetItems** – multiple *Get* actions

DynamoDB Backups

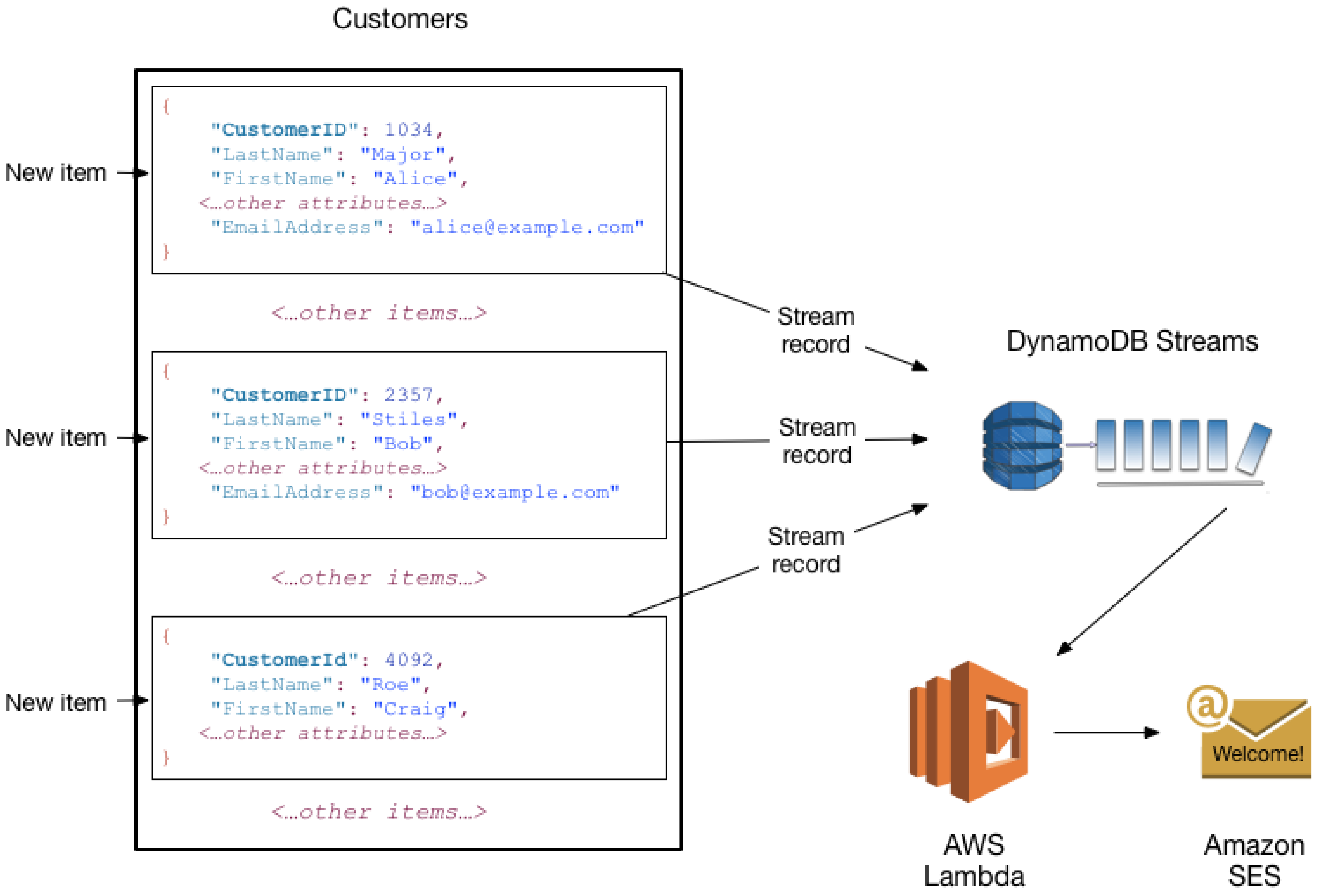
- Create on-demand backups - You can use the DynamoDB on-demand backup capability to create full backups of your tables for **long-term retention** and **archival** for regulatory compliance needs.
- Enable continuous backups using **point-in-time** recovery - You can restore that table to any point in time during the last 35 days. DynamoDB maintains incremental backups of your table.

DynamoDB Streams

DynamoDB Streams is an optional feature that **captures data modification** events in DynamoDB tables. Near-real time, and events are in order. Events are:

1. A new item is **added**: The stream captures an image of the entire item, including all of its attributes.
2. An item is **updated**: The stream captures the "before" and "after" image of any attributes that were modified in the item.
3. An item is **deleted**: The stream captures an image of the entire item before it was deleted.

Each stream record also contains the name of the table, the event timestamp, and other metadata. Stream records have a lifetime of 24 hours; after that, they are automatically removed from the stream.



```
{
    "CustomerID": 1034,
    "LastName": "Major",
    "FirstName": "Alice",
   <…other attributes…>
    "EmailAddress": "alice@example.com"
}
```

<…other items…>

```
{
    "CustomerID": 2357,
    "LastName": "Stiles",
    "FirstName": "Bob",
   <…other attributes…>
    "EmailAddress": "bob@example.com"
}
```

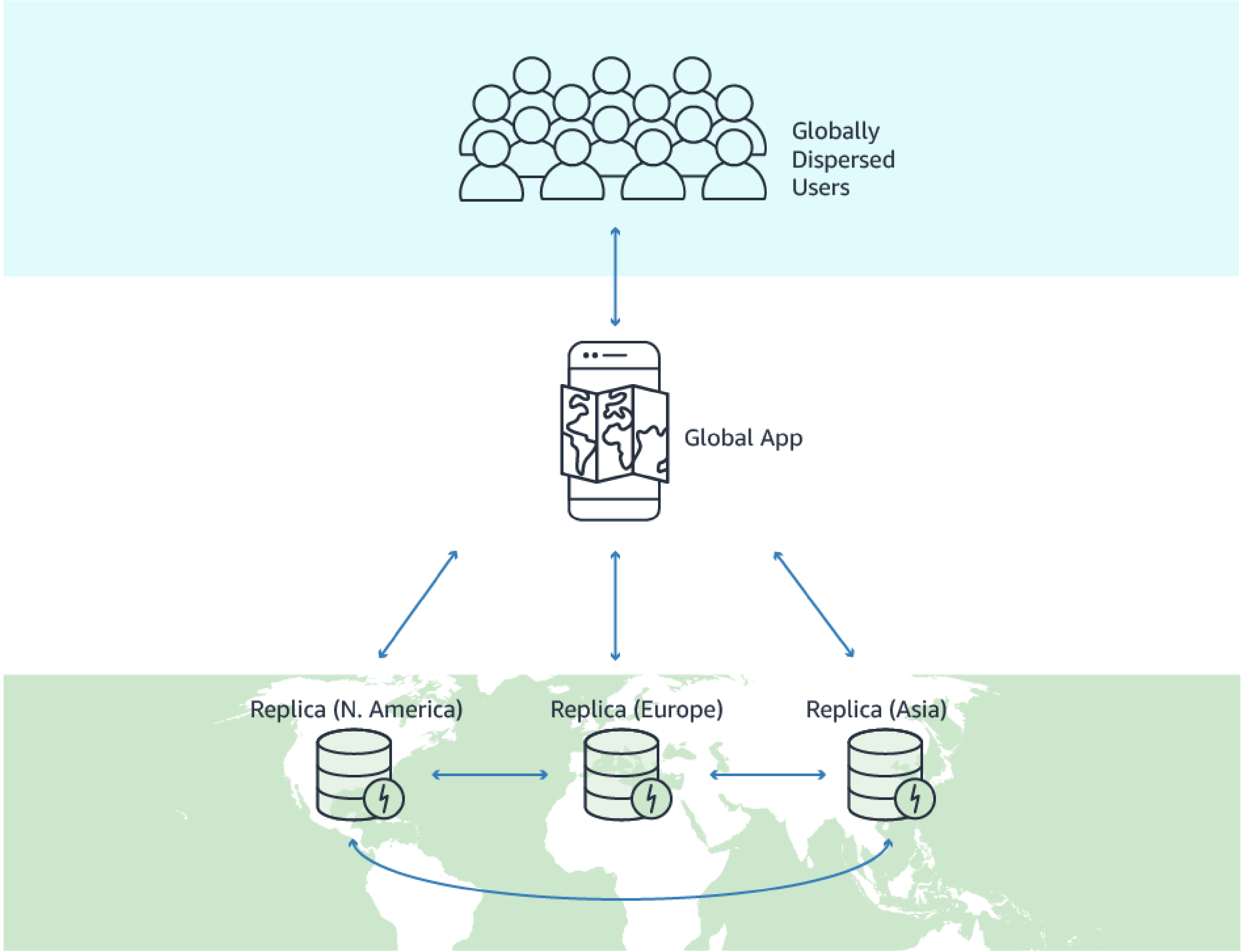
<…other items…>

```
{
    "CustomerId": 4092,
    "LastName": "Roe",
    "FirstName": "Craig",
   <…other attributes…>
}
```

<…other items…>

DynamoDB Global Table

Global tables build on the global Amazon DynamoDB footprint to provide you with a fully managed, **multi-region**, and multi-active database that delivers fast, local, read and write performance for massively scaled, global applications. Global tables **replicate** your DynamoDB tables **automatically** across your choice of AWS Regions.

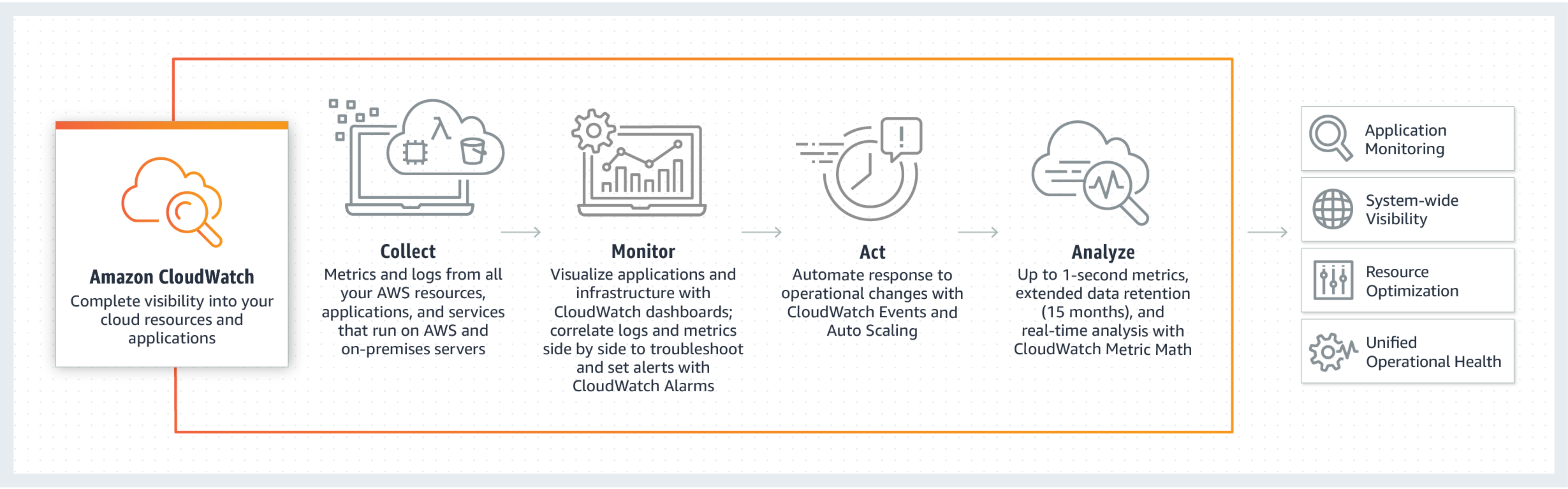

Globally Dispersed Users
Global App
Replica (N. America)
Replica (Europe)
Replica (Asia)

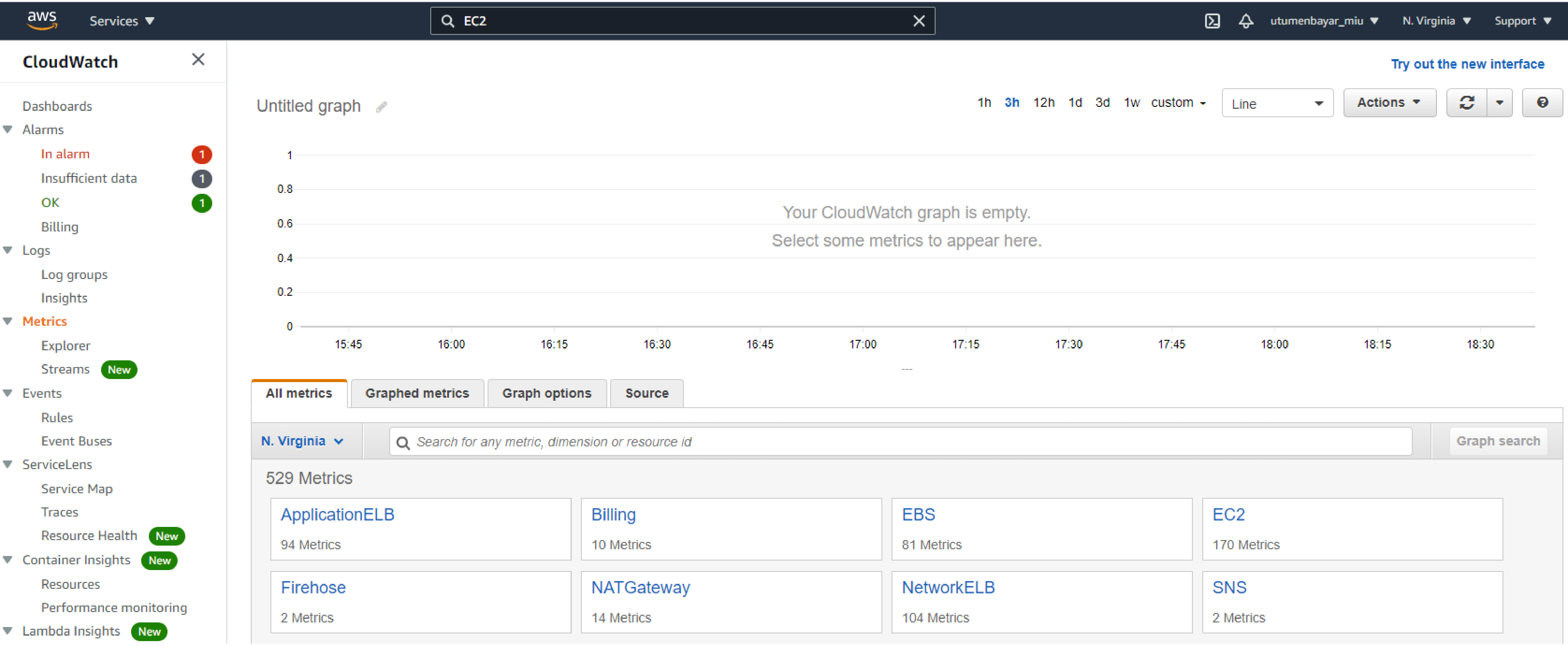
DynamoDB TTL

Amazon DynamoDB Time to Live (TTL) allows you to define a per-item timestamp to determine when an item is no longer needed. Shortly after the specified timestamp, DynamoDB deletes the item from your table without consuming any write throughput.

- Remove user or sensor data after one year of inactivity in an application.
- Archive expired items to an Amazon S3 data lake via Amazon DynamoDB Streams and AWS Lambda.
- Retain sensitive data for a certain amount of time according to contractual or regulatory obligations.

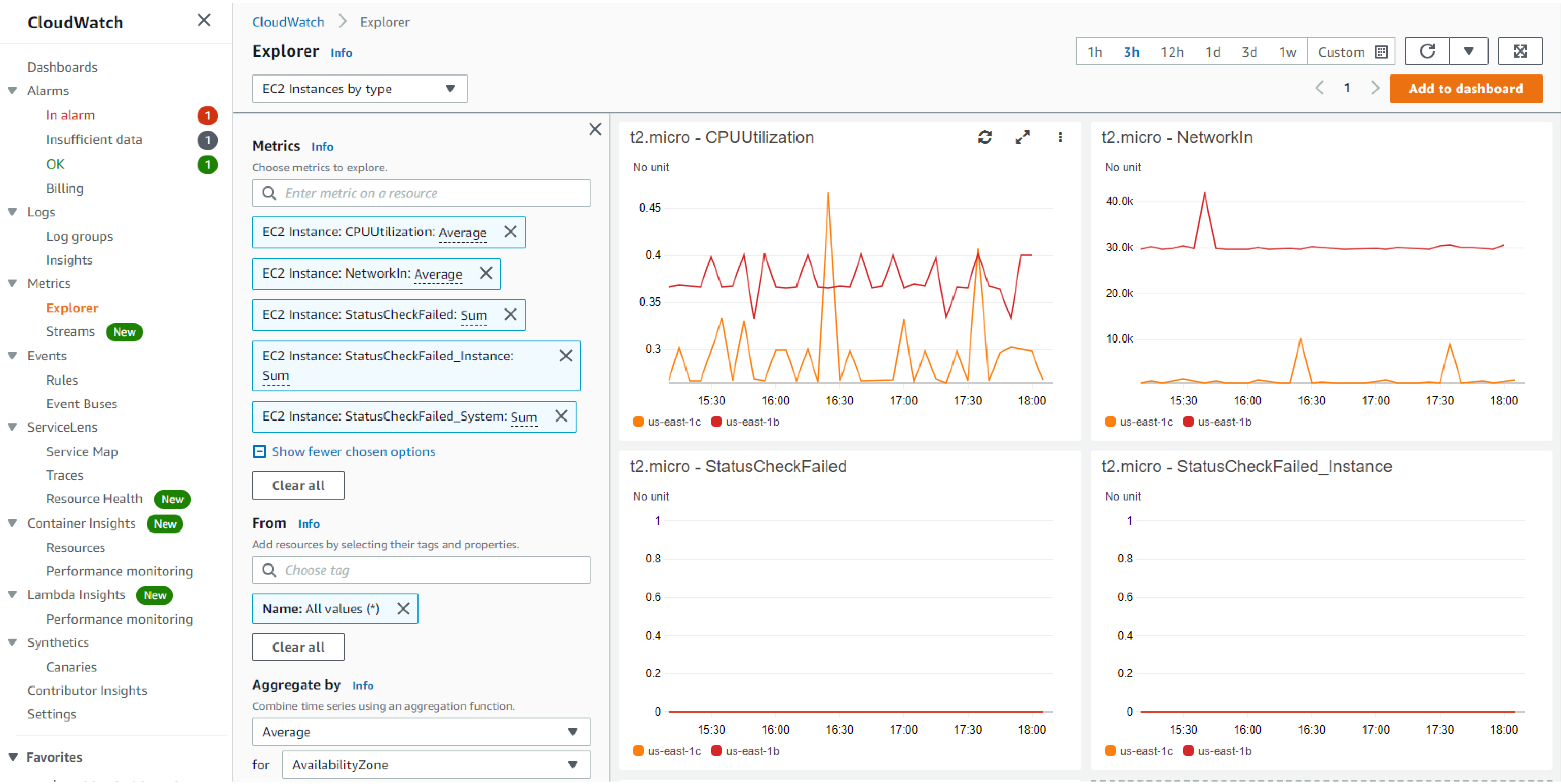
CloudWatch


Amazon CloudWatch
Complete visibility into your cloud resources and applications
Collect
Metrics and logs from all your AWS resources, applications, and services that run on AWS and on-premises servers
Monitor
Visualize applications and infrastructure with CloudWatch dashboards; correlate logs and metrics side by side to troubleshoot and set alerts with CloudWatch Alarms
Act
Automate response to operational changes with CloudWatch Events and Auto Scaling
Analyze
Up to 1-second metrics, extended data retention (15 months), and real-time analysis with CloudWatch Metric Math
Application Monitoring
System-wide Visibility
Resource Optimization
Unified Operational Health



529 Metrics

ApplicationELB 94 Metrics	Billing 10 Metrics	EBS 81 Metrics	EC2 170 Metrics
Firehose 2 Metrics	NATGateway 14 Metrics	NetworkELB 104 Metrics	SNS 2 Metrics



Explorer Info

EC2 Instances by type

Metrics Info

Choose metrics to explore.

Enter metric on a resource

- EC2 Instance: CPUUtilization: Average
- EC2 Instance: NetworkIn: Average
- EC2 Instance: StatusCheckFailed: Sum
- EC2 Instance: StatusCheckFailed_Instance: Sum
- EC2 Instance: StatusCheckFailed_System: Sum

From Info

Add resources by selecting their tags and properties.

Choose tag

Name: All values (*)

Aggregate by Info

Combine time series using an aggregation function.

Average

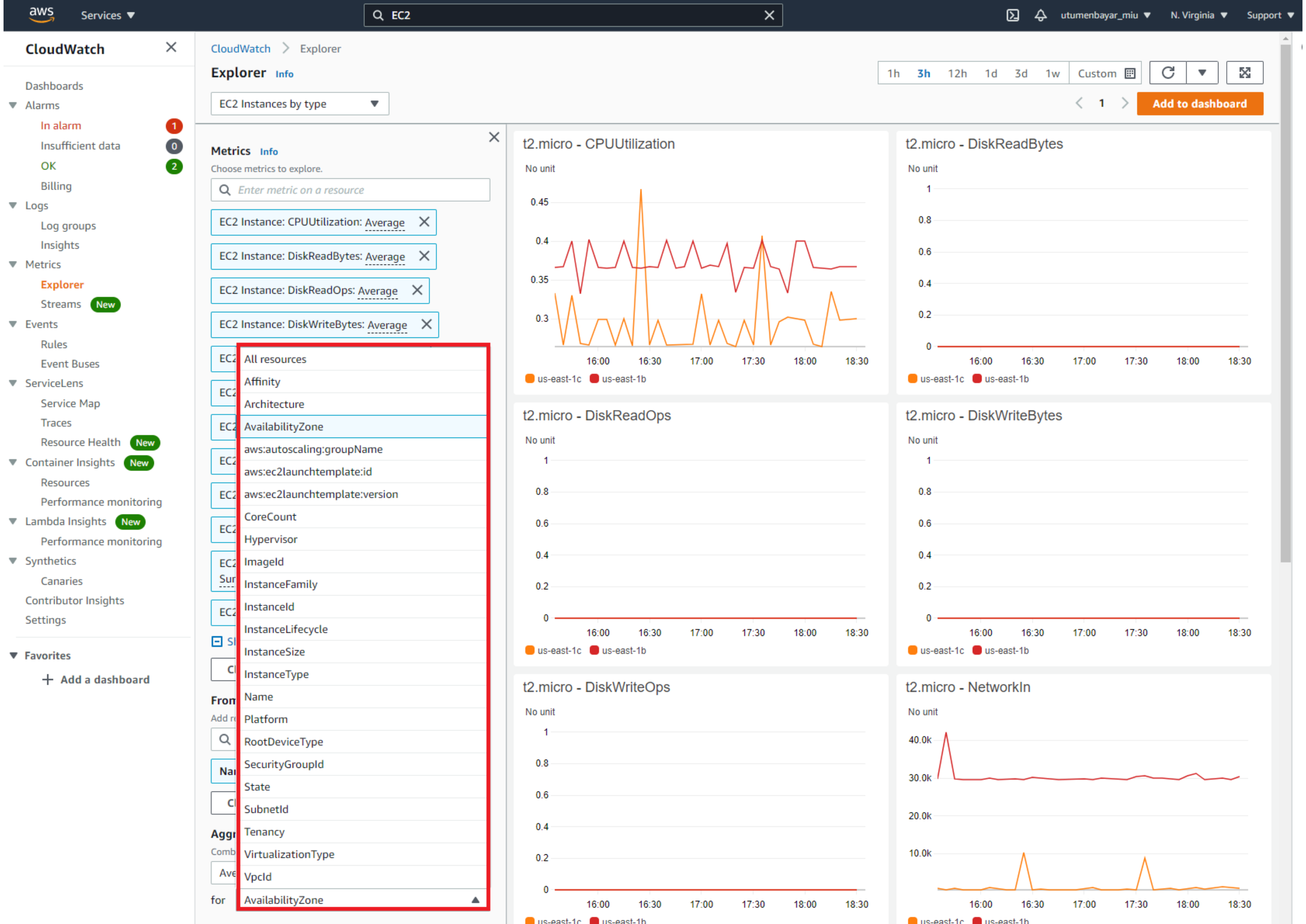
for AvailabilityZone

t2.micro - CPUUtilization

t2.micro - NetworkIn

t2.micro - StatusCheckFailed

t2.micro - StatusCheckFailed_Instance



aws Services ▼ | EC2 | utumenbayar_miu ▼ | N. Virginia ▼ | Support ▼

CloudWatch

- Dashboards
- Alarms
  - In alarm 1
  - Insufficient data 0
  - OK 2
  - Billing
- Logs
  - Log groups
  - Insights
- Metrics
  - Explorer
  - Streams New
- Events
  - Rules
  - Event Buses
- ServiceLens
  - Service Map
  - Traces
  - Resource Health New
- Container Insights New
  - Resources
  - Performance monitoring
- Lambda Insights New
  - Performance monitoring
- Synthetics
  - Canaries
- Contributor Insights
- Settings
- Favorites
  - Add a dashboard

CloudWatch > Explorer

Explorer Info

EC2 Instances by type ▼

1h 3h 12h 1d 3d 1w Custom

1 | Add to dashboard

Metrics Info

Choose metrics to explore.

Enter metric on a resource

- EC2 Instance: CPUUtilization: Average
- EC2 Instance: DiskReadBytes: Average
- EC2 Instance: DiskReadOps: Average
- EC2 Instance: DiskWriteBytes: Average

- All resources
- Affinity
- Architecture
- AvailabilityZone
- aws:autoscaling:groupName
- aws:ec2launchtemplate:id
- aws:ec2launchtemplate:version
- CoreCount
- Hypervisor
- ImageId
- InstanceFamily
- InstanceId
- InstanceLifecycle
- InstanceSize
- InstanceType
- Name
- Platform
- RootDeviceType
- SecurityGroupId
- State
- SubnetId
- Tenancy
- VirtualizationType
- VpcId

for AvailabilityZone ▲

Collect and store logs

- Logs that are published by AWS services. For example, API Gateway, Lambda.
- Custom logs - These are logs from your own application and on-premises resources via a CloudWatch agent or the PutLogData API.

Build-in metrics

Collecting metrics from distributed applications (such as those built using microservices architectures) is time consuming. Amazon CloudWatch allows you to collect default metrics from more than 70 AWS services.

For example, EC2 instances automatically publish CPU utilization, data transfer, and disk usage metrics to help you understand changes in state.

Custom metrics

Amazon CloudWatch allows you to collect custom metrics from your own applications to monitor operational performance, troubleshoot issues, and spot trends.

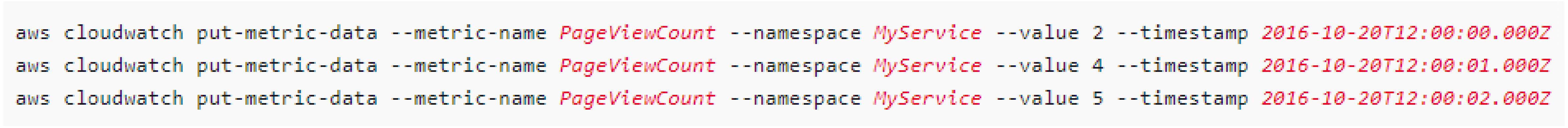
You can use **CloudWatch Agent** or the **PutMetricData API** action to publish these metrics to CloudWatch.

All the same CloudWatch functionality will be available at up to one-second frequency for your own custom metrics data, including statistics, graphs, and alarms.

PutMetricData API

Publishes metric data points to Amazon CloudWatch.

Every PutMetricData call for a custom metric is charged, so calling PutMetricData more often on a high-resolution metric can lead to higher charges.



```
aws cloudwatch put-metric-data --metric-name PageViewCount --namespace MyService --value 2 --timestamp 2016-10-20T12:00:00.000Z
aws cloudwatch put-metric-data --metric-name PageViewCount --namespace MyService --value 4 --timestamp 2016-10-20T12:00:01.000Z
aws cloudwatch put-metric-data --metric-name PageViewCount --namespace MyService --value 5 --timestamp 2016-10-20T12:00:02.000Z
```

CloudWatch Agent

It is a library helps you:

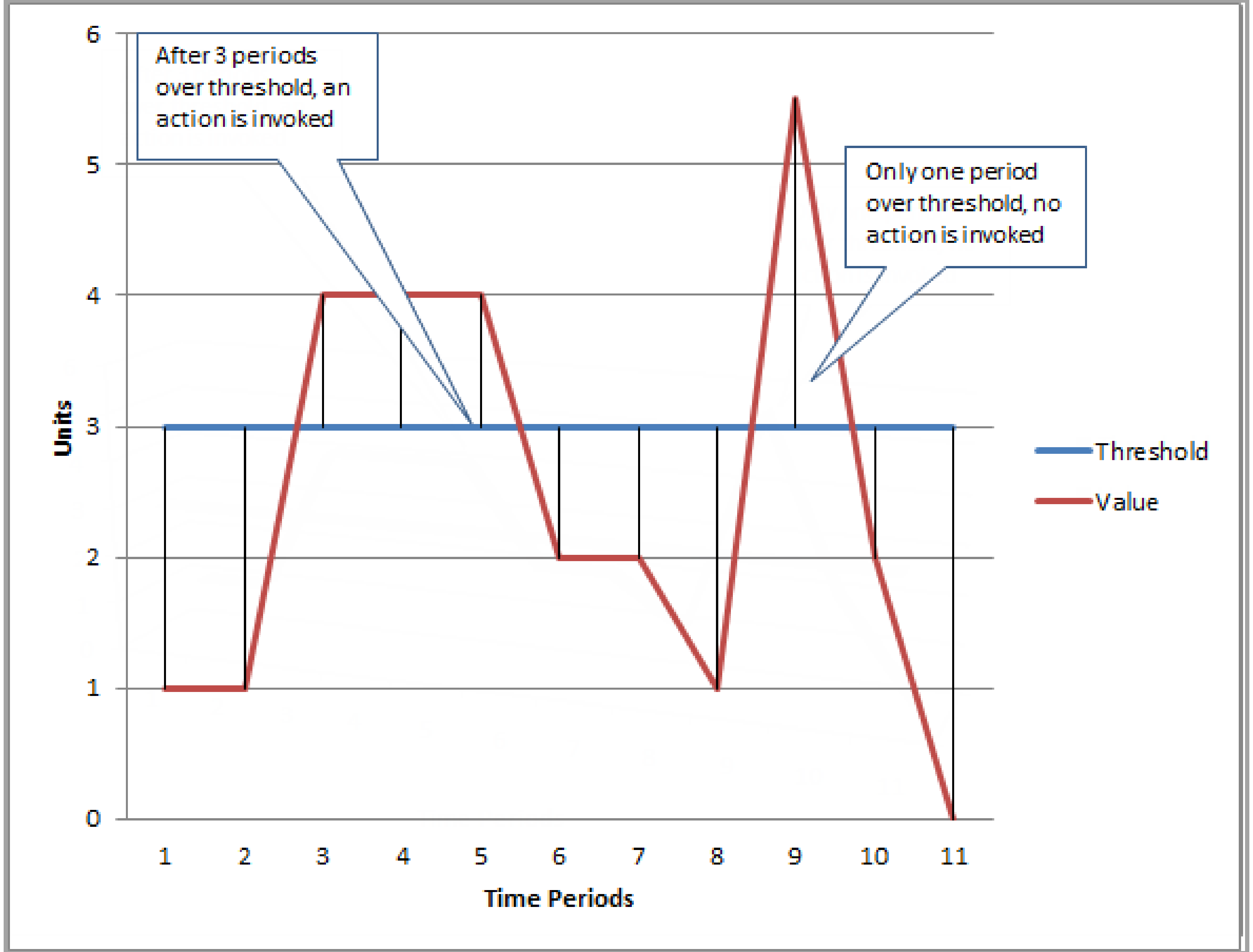
- Collect internal system-level metrics from Amazon EC2 instances or on-premises servers. For example, getting memory utilization.
- Retrieve custom metrics from your applications or services using the StatsD and collectd protocols. StatsD is supported on both Linux and Windows Server. collectd is supported only on Linux servers.

Evaluating an alarm

An alarm invokes actions only when the alarm changes state.

- **OK** – The metric or expression is within the defined threshold.
- **ALARM** – The metric or expression is outside of the defined threshold.
- **INSUFFICIENT_DATA** – The alarm has just started, the metric is not available, or not enough data is available for the metric to determine the alarm state.

The action can be sending a notification to an Amazon SNS topic, performing an Amazon EC2 action or an Auto Scaling action.


After 3 periods over threshold, an action is invoked
Only one period over threshold, no action is invoked
Units
Time Periods
Threshold
Value
0
1
2
3
4
5
6
1
2
3
4
5
6
7
8
9
10
11

Log group and log stream

A log **stream** is a sequence of log events that share the same source. Each separate source of logs in CloudWatch Logs makes up a separate log stream.

A log **group** is a group of log streams that share the same retention, monitoring, and access control settings. You can define log groups and specify which streams to put into each group. There is no limit on the number of log streams that can belong to one log group.

CloudWatch Logs Insights

CloudWatch Logs Insights enables you to interactively **search** and **analyze** your log data in Amazon CloudWatch Logs. You can perform **queries** to help you more efficiently and effectively respond to operational issues. If an issue occurs, you can use CloudWatch Logs Insights to **identify potential causes** and **validate deployed fixes**.

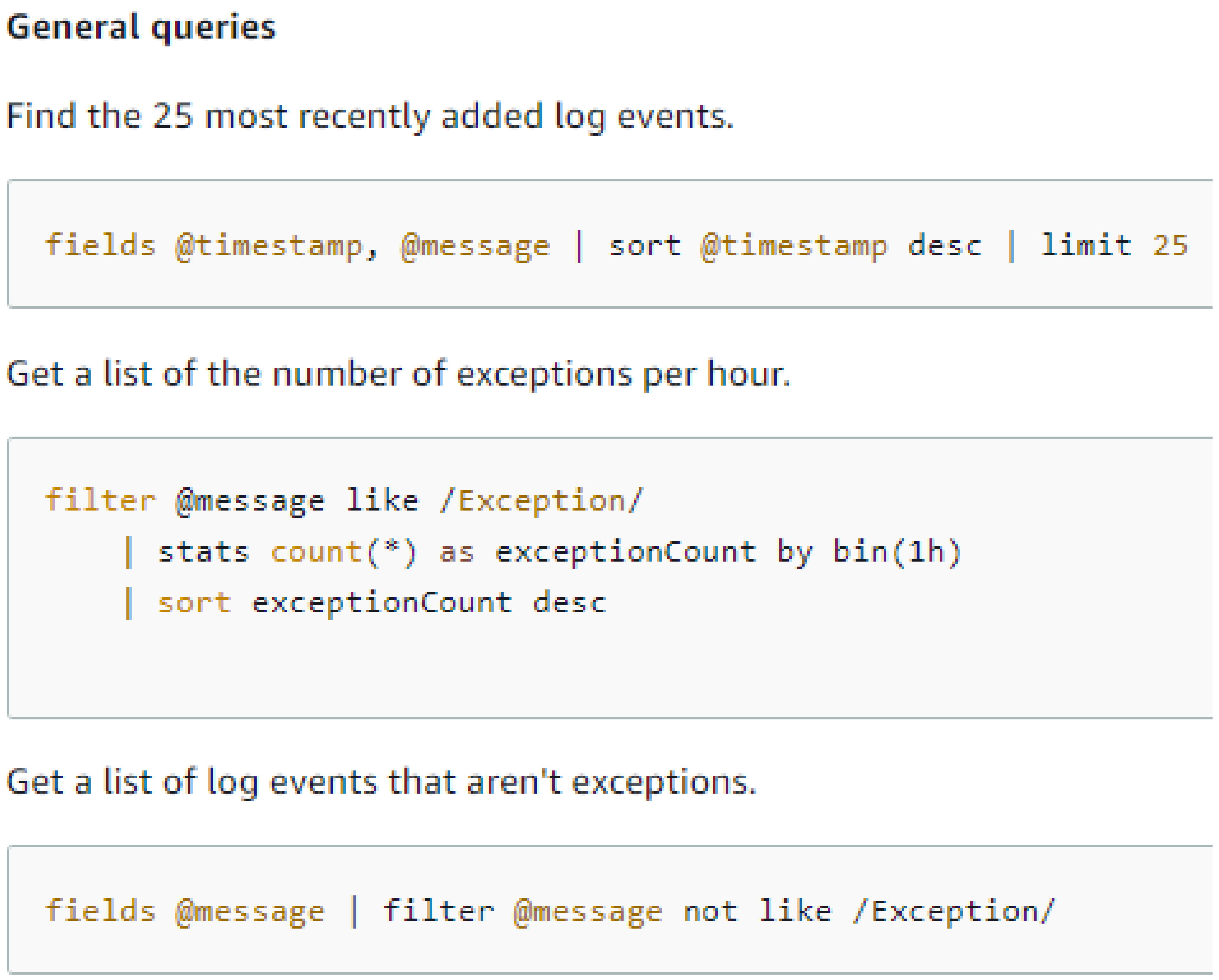
CloudWatch Logs Insights includes a purpose-built query language with a few simple but powerful commands.

Supported logs and discovered fields

CloudWatch Logs Insights supports all types of logs. For every log sent to CloudWatch Logs, five system fields are automatically generated:

- **@message** contains the raw unparsed log event. This is equivalent to the message field in InputLogevent.
- **@timestamp** contains the event timestamp contained in the log event's timestamp field. This is equivalent to the timestamp field in InputLogevent.
- **@ingestionTime** contains the time when the log event was received by CloudWatch Logs.
- **@logStream** contains the name of the log stream that the log event was added to. Log streams are used to group logs by the same process that generated them.
- **@log** is a log group identifier in the form of account-id:log-group-name. This can be useful in queries of multiple log groups, to identify which log group a particular event belongs to.

Sample CloudWatch Insights queries



General queries

Find the 25 most recently added log events.

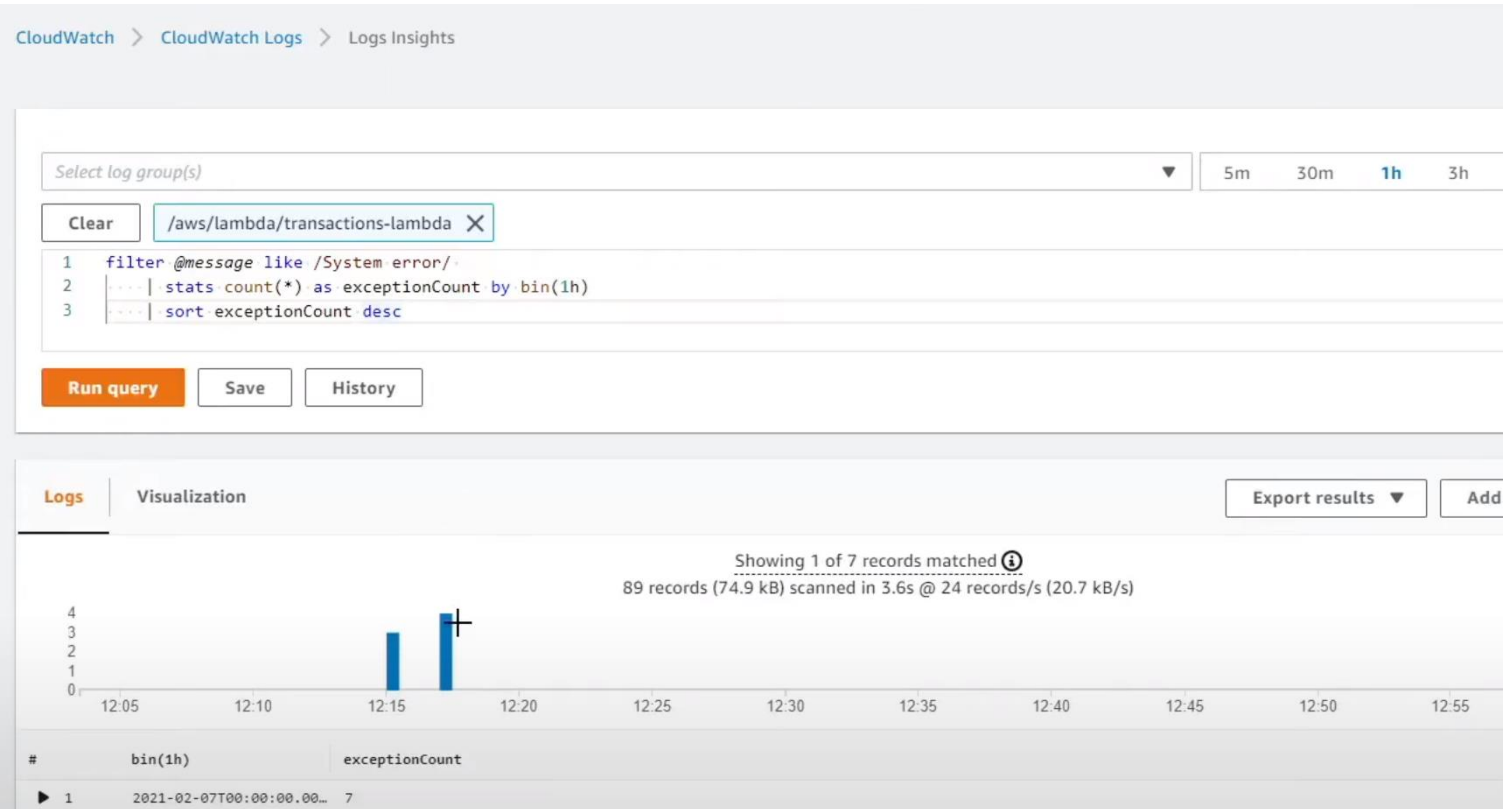
```
fields @timestamp, @message | sort @timestamp desc | limit 25
```

Get a list of the number of exceptions per hour.

```
filter @message like /Exception/
    | stats count(*) as exceptionCount by bin(1h)
    | sort exceptionCount desc
```

Get a list of log events that aren't exceptions.

```
fields @message | filter @message not like /Exception/
```



```
filter @message like /System error/
    | stats count(*) as exceptionCount by bin(1h)
    | sort exceptionCount desc
```

Showing 1 of 7 records matched

89 records (74.9 kB) scanned in 3.6s @ 24 records/s (20.7 kB/s)

#	bin(1h)	exceptionCount
▶ 1	2021-02-07T00:00:00.00…	7

Dashboard

Amazon CloudWatch dashboards are customizable home pages in the CloudWatch console that you can use to monitor your resources in a single view, even those resources that are spread across different Regions. You can use CloudWatch dashboards to create customized views of the metrics and alarms for your AWS resources.

$3.00 per dashboard per month.

Anomaly Detection

Amazon CloudWatch Anomaly Detection applies machine-learning algorithms to continuously analyze data of a metric and identify anomalous behavior.