Nsync Collection Statistics

Intern Project Final Demo

David Munechika
SDE Intern, EC2 Voyager
Summer 2022
dmune@amazon.com

Contents

Background

Nsync is a distribution service which stores Virtual Private Cloud (VPC) data in the form of Nsync records. Nsync records are grouped into sets called collections.

Project Objective

The goal of this project is to gather statistics for each collection in the form of CloudWatch metrics. These metrics will be used to construct live dashboards that will allow us to predict scaling cliffs and help debug customer issues.

Required Goals

NCC in NAWS

Status: COMPLETED

Set up an Nsync Consumer Client (NCC) in Native AWS

Compute Statistics

Status: COMPLETED

Gather collections, add update listeners, and compute statistics for Nsync collections from Zeta

Emit Metrics

Status: COMPLETED

Emit statistics from Fargate instance to CloudWatch metric namespace

Dashboards

Status: COMPLETED

Create dashboards in CloudWatch from collection statistics metric data

Stretch Goals

New Collections

Status: COMPLETED

Use Nsync service API `DescribeCollections()` function to automatically add new collections

Anomaly Detection

Status: COMPLETED

Train anomaly detection models for metric dashboards and setup custom alarms

Pipeline & CDK

Status: COMPLETED

Create a pipeline for code and redefine architecture in CDK code

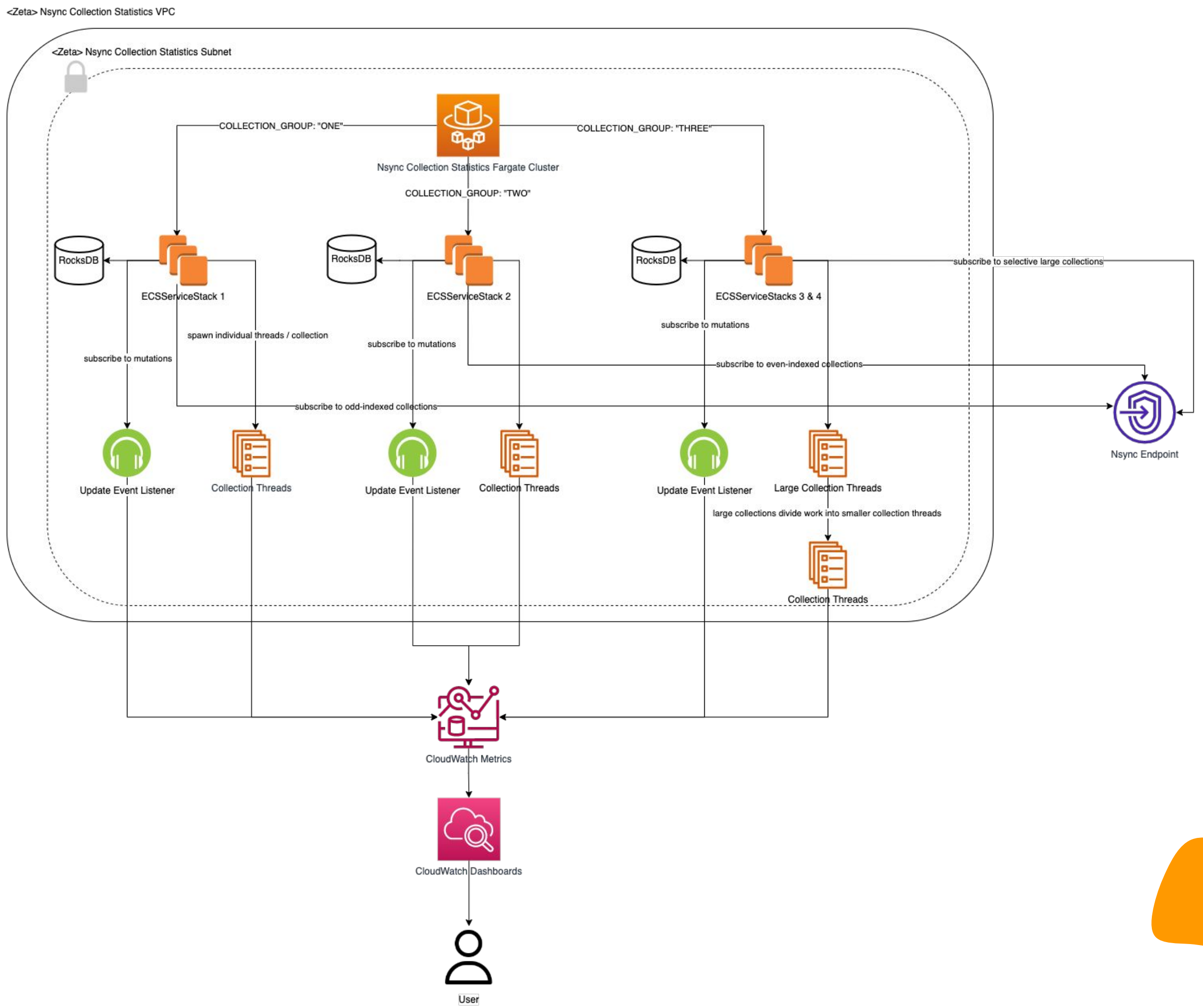
Optimize & Scale

Status: COMPLETED

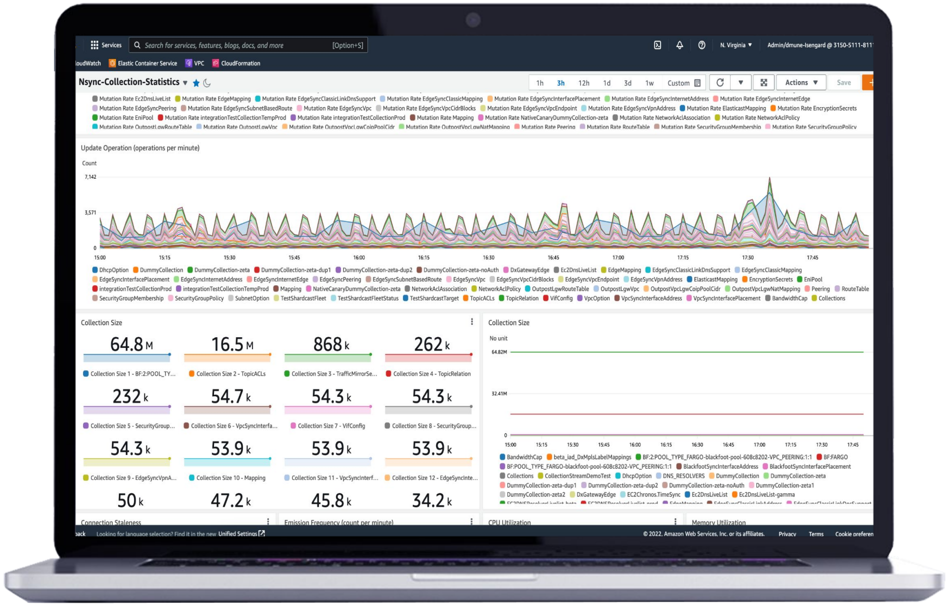
Optimize statistical analysis using distributed computing and horizontal scaling

Architecture

- Fargate cluster connects to the Nsync endpoint through an Nsync Service Client and Nsync Consumer Client (NCC)
- Service Client retrieves all available collections using `DescribeCollections()`
- NCC subscribes to a group of collections
- NCC streams updates and computes statistics for each collection, emitting metrics to CloudWatch
- CloudWatch dashboards are generated from metric data



Demo



Link to dashboard:
https://us-east-1.console.aws.amazon.com/cloudwatch/home?region=us-east-1#dashboards:name=Nsync-Collection-Statistics

Challenges and Obstacles

Connection Staleness Monitoring

Problem: Need to ensure the Nsync Consumer Client remains up-to-date with the current collection data.
Solution: Emit a connection staleness metric on each iteration of statistical analysis (for each consumer client that is initialized) and implement health checks and CloudWatch alarms to trigger if the staleness exceeds a certain threshold.

Challenges and Obstacles

Number of Collections

Problem: One Fargate instance doesn’t have enough compute resources to handle all 93 collections in Zeta.
Solution: Spawn multiple instances with increased CPU and memory resources and different environment variables indicating which group of collections to subscribe to.

Challenges and Obstacles

Large Collection Sizes

Problem: The largest collections had upwards of 65 million records which made it impossible to achieve a dashboard metric granularity of 1 minute when iterating over the records.
Solution: For these collections, individual child threads with batch CloudWatch emission was used to divide up the statistical computation into groups of 1 million records.

Challenges and Obstacles

Cold Starts

Problem: Nsync Consumer Client can take up to 15 minutes to cold start which can lead to gaps in dashboard metric data.
Solution: Redefine service health checks to provide a timeout that is sufficient to allow the service the cold start without being interrupted by a task restarting. Also, setup persistence configuration in the form of RocksDB databases for each Nsync Consumer Client.

Limitations and Future Work

Horizontal Scaling

Limitation: Environment variables are used to define instance collection groups. An auto-scaling capability would allow this service to more easily scale to other regions and availability zones.

Future Work: Implement an orchestrator which decides which tasks should subscribe to which collections. Eliminate the need for using environment variables and allow tasks to be auto-scaled rather than manually created.

Extremely Large Collections

Limitation: Since each Fargate instance subscribes to at least one total collection, the service still struggles to support the desired metric granularity for extremely large collections (concretely, collections with more than 50 million records).

Future Work: It would be useful to be able to divide up statistical computation of a single collection across multiple instances. Alternatively, different compute instance options could be tested which may have the resources capable for handling such large collections.

Thanks!