

Sprint 3

This document outlines the security data ingestion pipelines for the SOC platform, covering data sources, ingestion architecture, automation and scalability, error handling, log management and normalization, and access control.

1. Security Data Ingestion Pipelines (Matthew Escalera, Pedro Gomez)

1.1 Data Source Identification

The SOC platform will ingest security data from a variety of sources to provide a comprehensive view of the security posture. Here's a list of potential data sources:

- **Cloud Service Logs:**
 - VPC Flow Logs: Records network traffic flowing in and out of VPCs.
 - GuardDuty Findings: For collecting logs.
- **Network Security Appliance Logs:**
 - Firewalls: Logs security events like blocked traffic and access attempts.
 - Intrusion Prevention/Detection Systems (IPS/IDS): Alerts on suspicious network activity.
 - Web Application Firewalls (WAF): Logs attempts to exploit web application vulnerabilities.

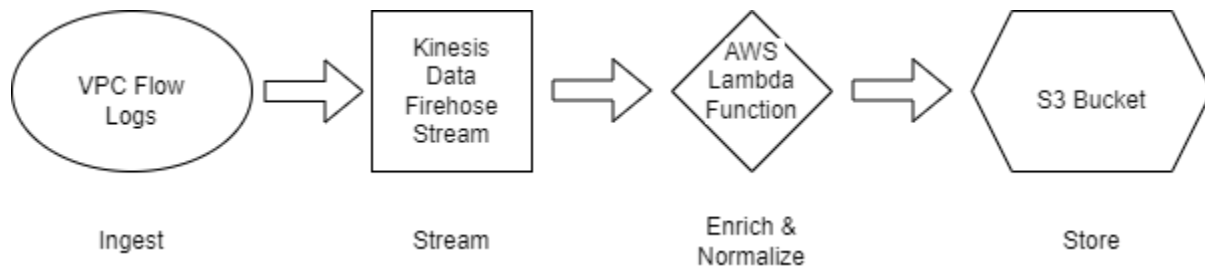
1.2 Data Ingestion Architecture

The data ingestion architecture will leverage various services to collect, transform, and deliver security data to the log management platform. Here's a breakdown of the key components:

- **Log Forwarding Mechanisms:**
 - Amazon EventBridge: A standard protocol for sending logs from devices to a central server, used for Aws config, Inspector and Guard Duty.
 - Cloud Trail made to native settings to send logs to CloudWatch, VPC flow logs has the same thing.
- **Data Ingestion Services:**
 - Amazon Kinesis Data Streams: Real-time streaming data for high-volume log ingestion.
 - AWS Lambda: Used for converting information coming from VPC flow logs and parsing and enriching it into a more readable log.
- **Data Transformation and Enrichment:**
 - AWS Lambda Functions: Serverless functions for real-time data processing and transformation tasks.

1.3 Data ingestion pipeline architecture (sample)

VPC FLOW LOGS → Kinesis Data Firehose Stream → AWS Lambda Function (data is being enriched normalized here) → S3 bucket (where the results are being stored).



1.3 Error Handling and Monitoring

```
import logging
```

```
logger = logging.getLogger()
```

```
logger.setLevel(logging.INFO)
```

```
def lambda_handler(event, context):
```

```
    logger.info('Lambda execution started...')
```

```
    try:
```

```
        # Your Lambda function logic here
```

```
        logger.info('Function logic executed successfully.')
```

```
    return {
```

```
        'statusCode': 200,
```

```
        'body': 'Function executed successfully!'
```

```
    }
```

```
except Exception as e:
```

```
    # Log the error
```

```
    logger.error(f'Error: {str(e)}')
```

```
    # Return a custom error response
```

```
    return {
```

```
        'statusCode': 500,
```

```
        'body': 'Internal Server Error'
```

```
    }
```

finally:

```
logger.info('Lambda execution completed.')
```

The code above can be of use when Lambda is acting unexpectedly. You can use detailed logging with the 'logger' module. This can be used with CloudWatch Logs to examine logs and identify the root cause of the unexpected behavior. When using detailed logging, this can provide a trail of events within the Lambda function.

2. Log Management & Normalization (Giovanni Garcia Flores, Mario Register)

2.1 Log Management Platform

AWS Cloud Watch

We decided to use AWS Cloud Watch as our log management platform.

We currently have these log groups for these services:

AWS Guard Duty

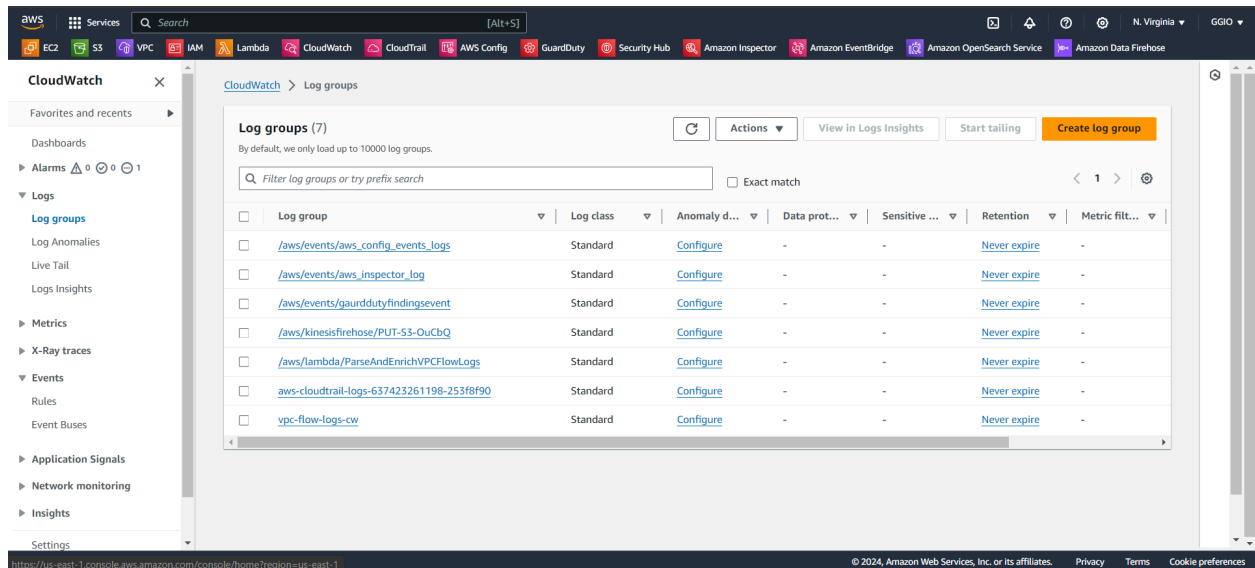
AWS Inspector

AWS Config

AWS VPC Flow Logs

AWS Cloudtrail

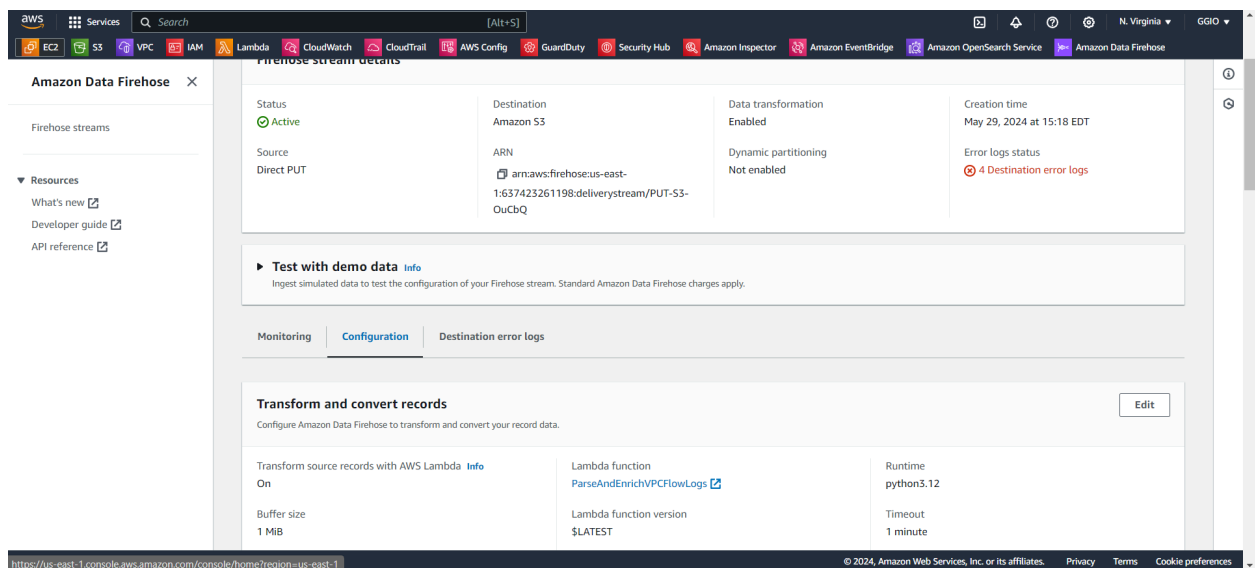
There are also logs being collected for both our Lambda and Kinesis Data Firehose services running our log data normalization and enrichment service.



We used a mix of Amazon Eventbridge rules and native AWS logging services within the application to send their logs to Cloudwatch.

2.2 Log Normalization and Enrichment

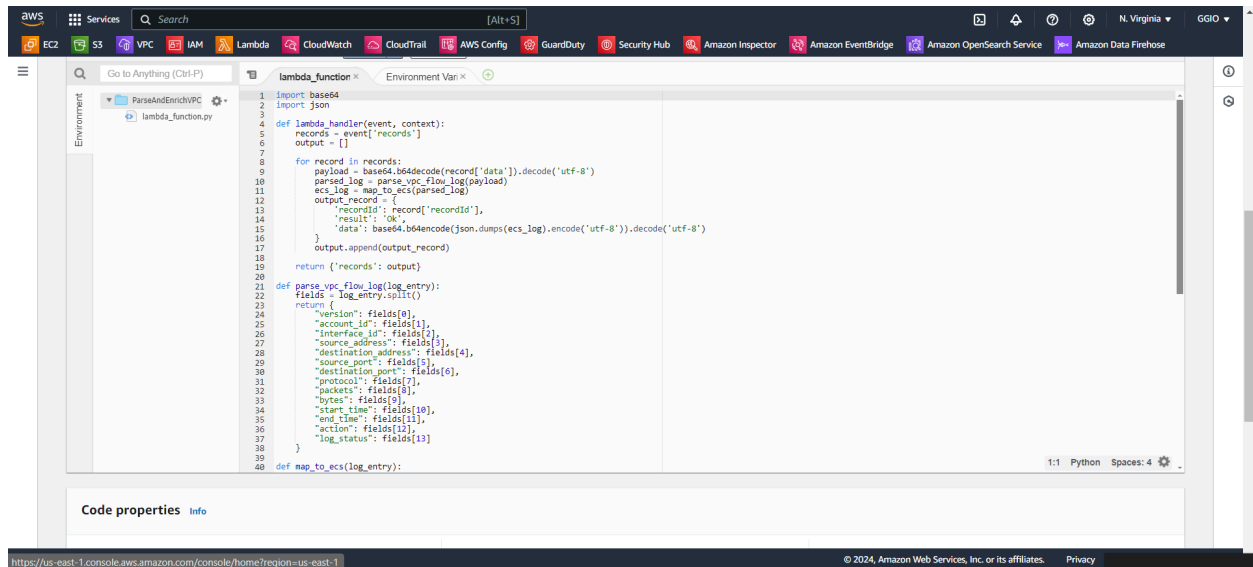
We created a Kinesis Data Firehose stream that would intake logs which would then be transformed by AWS Lambda to further normalize and enrich the data into a more readable and accessible type of data.



When creating the stream, an appropriate IAM role with the proper permissions to collect the logs and to be able to use the Lambda function is needed.

An IAM role also needs to be created for the Lambda function itself to be able to intake and change the logs but to also store them in an S3 bucket.

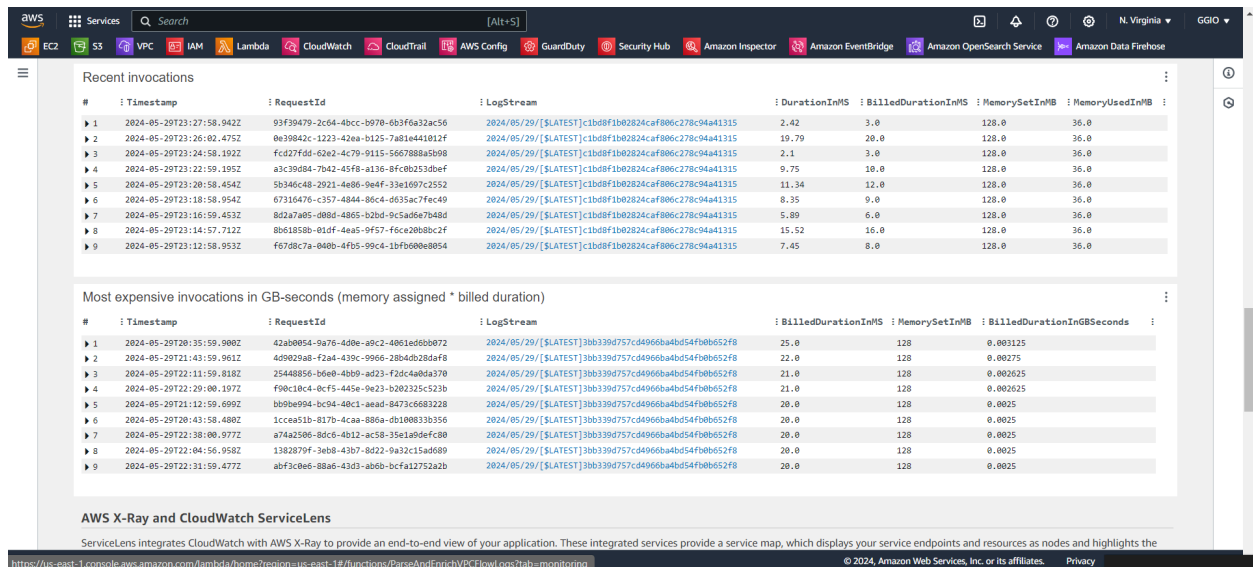
The Lambda function itself would also need to have a custom Python code to be able to do the data enrichment and normalization itself.



```
1 import base64
2 import json
3
4 def lambda_handler(event, context):
5     records = event['records']
6     output = []
7
8     for record in records:
9         payload = base64.b64decode(record['data']).decode('utf-8')
10        parsed_log = parse_vpc_flow_log(payload)
11        ecs_log = map_to_ecs(parsed_log)
12        output_record = {
13            'recordId': record['recordId'],
14            'result': 'Ok',
15            'data': base64.b64encode(json.dumps(ecs_log).encode('utf-8')).decode('utf-8')
16        }
17        output.append(output_record)
18
19    return {'records': output}
20
21 def parse_vpc_flow_log(log_entry):
22     fields = log_entry.split()
23     return {
24         "version": fields[0],
25         "account_id": fields[1],
26         "interface_id": fields[2],
27         "source_address": fields[3],
28         "destination_address": fields[4],
29         "source_port": fields[5],
30         "destination_port": fields[6],
31         "protocol": fields[7],
32         "packets": fields[8],
33         "bytes": fields[9],
34         "start_time": fields[10],
35         "end_time": fields[11],
36         "action": fields[12],
37         "log_status": fields[13]
38     }
39
40 def map_to_ecs(log_entry):
```

For demonstration, we used VPC Flow logs as an example. We created an EC2 to be able to simulate network traffic for new logs to be created and monitored.

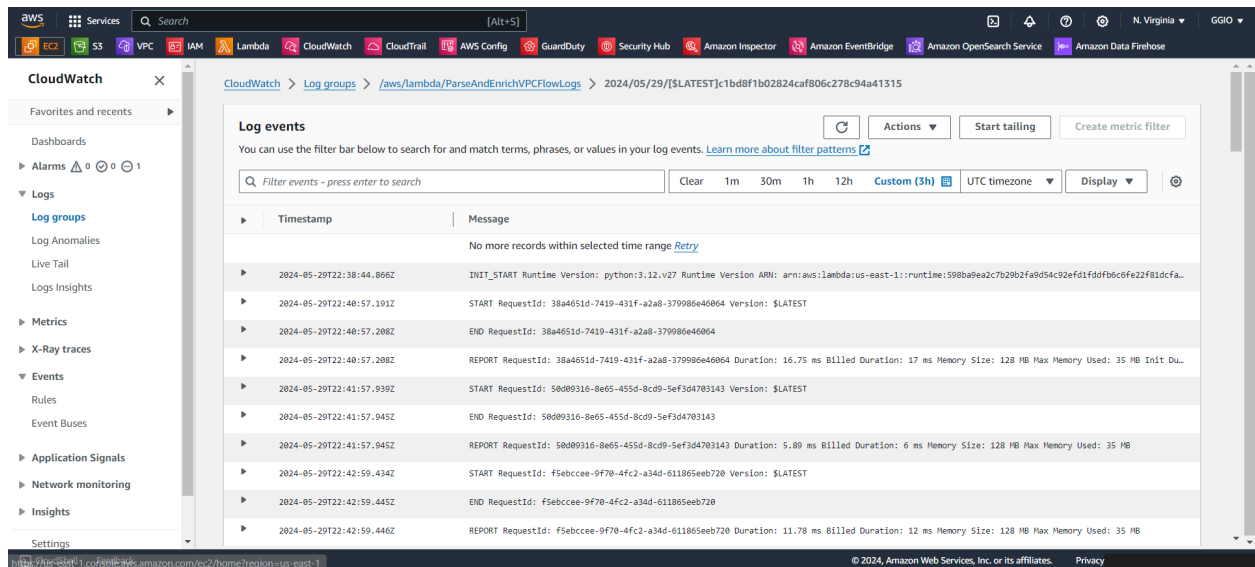
Checking the monitor tab will tell us if the code has been executed or not.



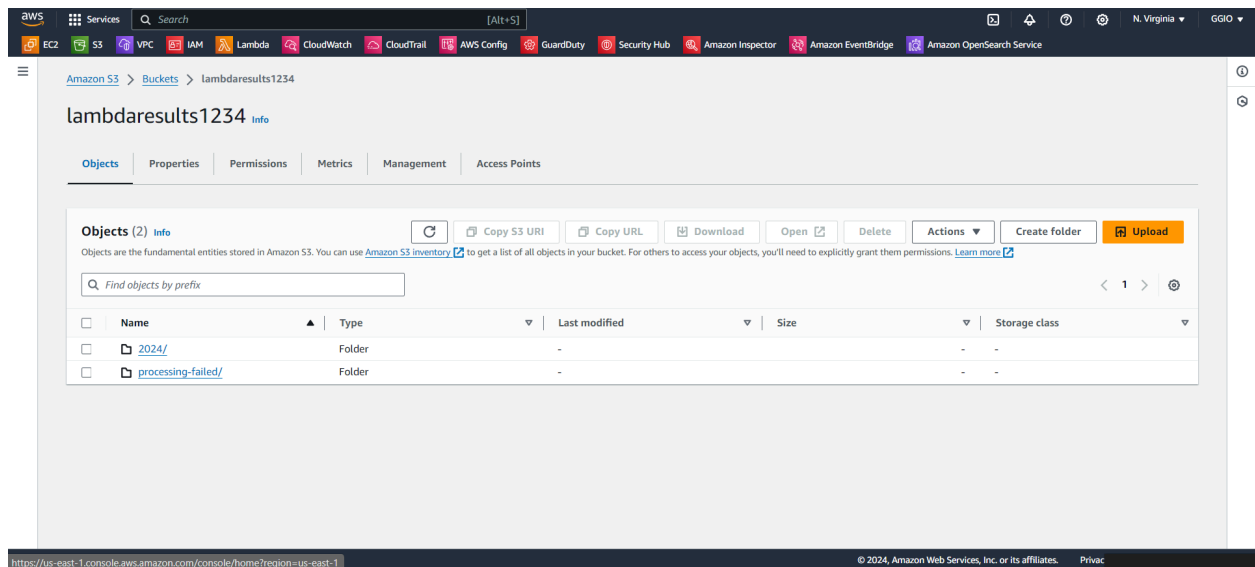
#	Timestamp	RequestId	LogStream	DurationInMS	BilledDurationInMS	MemorySetInMB	MemoryUsedInMB
1	2024-05-29T23:27:58.942Z	91f39479-2c04-4b0c-b070-6b3f6a12ac56	2024/05/29/[SLATEST]c1bd8f1b02824caf806c278c94a41315	2.42	3.0	128.0	36.0
2	2024-05-29T23:26:02.475Z	0e39842c-1223-420a-b125-7a01e441012f	2024/05/29/[SLATEST]c1bd8f1b02824caf806c278c94a41315	19.79	20.0	128.0	36.0
3	2024-05-29T23:24:58.192Z	fc227f6d-62e2-4c79-9115-5667888a5098	2024/05/29/[SLATEST]c1bd8f1b02824caf806c278c94a41315	2.1	3.0	128.0	36.0
4	2024-05-29T23:22:59.195Z	a1c39d94-7042-45f8-a136-8f0b0253dbef	2024/05/29/[SLATEST]c1bd8f1b02824caf806c278c94a41315	9.75	10.0	128.0	36.0
5	2024-05-29T23:20:58.454Z	5b346c4e-2921-4e86-9e4f-3fa1697c2552	2024/05/29/[SLATEST]c1bd8f1b02824caf806c278c94a41315	11.34	12.0	128.0	36.0
6	2024-05-29T23:18:58.197Z	67316476-c357-4844-80c4-d635ac7fec49	2024/05/29/[SLATEST]c1bd8f1b02824caf806c278c94a41315	8.35	9.0	128.0	36.0
7	2024-05-29T23:14:57.712Z	8d2a7a05-080d-4865-b2bd-9c5ad67b48d8	2024/05/29/[SLATEST]c1bd8f1b02824caf806c278c94a41315	5.89	6.0	128.0	36.0
8	2024-05-29T23:12:58.953Z	8b618508-01df-4e45-9f57-f6ce20b0bc2f	2024/05/29/[SLATEST]c1bd8f1b02824caf806c278c94a41315	15.52	16.0	128.0	36.0
9	2024-05-29T23:12:58.953Z	f6708c7a-0406-4f05-99c4-1bf600e08054	2024/05/29/[SLATEST]c1bd8f1b02824caf806c278c94a41315	7.45	8.0	128.0	36.0

#	Timestamp	RequestId	LogStream	BilledDurationInMS	MemorySetInMB	BilledDurationInGBSeconds
1	2024-05-29T20:35:59.900Z	42ab0054-9a76-4d0e-a9c2-4061ed6bb072	2024/05/29/[SLATEST]3bb339d757cd4966ba4bd54fb0b652f8	25.0	128	0.003125
2	2024-05-29T21:43:59.961Z	409029a8-f2a4-439c-9966-28b4db28daf8	2024/05/29/[SLATEST]3bb339d757cd4966ba4bd54fb0b652f8	22.0	128	0.00275
3	2024-05-29T22:11:59.818Z	25448856-b6e0-4b09-ad23-f2dc4a0da370	2024/05/29/[SLATEST]3bb339d757cd4966ba4bd54fb0b652f8	21.0	128	0.002625
4	2024-05-29T22:29:00.197Z	f90c18c4-0cf5-445e-9a23-b202325c523b	2024/05/29/[SLATEST]3bb339d757cd4966ba4bd54fb0b652f8	21.0	128	0.002625
5	2024-05-29T21:12:59.699Z	bb9be994-bc94-48c1-aeaa-8473c6683228	2024/05/29/[SLATEST]3bb339d757cd4966ba4bd54fb0b652f8	20.0	128	0.0025
6	2024-05-29T20:43:58.480Z	1ccea510-8170-4caa-886a-d100833b3556	2024/05/29/[SLATEST]3bb339d757cd4966ba4bd54fb0b652f8	20.0	128	0.0025
7	2024-05-29T22:38:00.977Z	a74a2506-80c6-4b12-ac58-35e1a0defc80	2024/05/29/[SLATEST]3bb339d757cd4966ba4bd54fb0b652f8	20.0	128	0.0025
8	2024-05-29T22:04:56.958Z	1382879f-3a08-4307-b022-9a32c15ad089	2024/05/29/[SLATEST]3bb339d757cd4966ba4bd54fb0b652f8	20.0	128	0.0025
9	2024-05-29T22:31:59.477Z	abf3c0e6-88a6-43d3-ab0b-bcfaf12752a2b	2024/05/29/[SLATEST]3bb339d757cd4966ba4bd54fb0b652f8	20.0	128	0.0025

We can check on the logs and we can see there was a successful run,



We then go into the bucket where the data has been transformed and delivered to.



We then look out for the most recent file and download it, then open it. We personally used Visual Studio Code to view the file, here is the result.

```
File Edit Selection View Go Run ... Search
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

PUT-S3-OuCbQ-1-2024-05-29-20-33-56-c2d6762d-4b92-4482-8f73-adbcf3524634 1 X
C: > Users > Giova > Downloads > PUT-S3-OuCbQ-1-2024-05-29-20-33-56-c2d6762d-4b92-4482-8f73-adbcf3524634 > ...
1 [{"ecs": {"version": "1.0.0"}, "network": {"protocol": "tcp", "bytes": 44, "packets": 1, "transport": "tcp"}, "source": {"ip": "162.216.149.191", "port": 49236}, "destination": {"ip": "19.0.1.193", "port": 23306}, "event": {"outcome": "failure", "category": "network"}, "cloud": {"account": {"id": "637423261198"}, "observer": {"id": "eni-05232e29b9f1cdb2a"}, "log": {"status": "OK\\\\"}}], {"ecs": {"version": "1.0.0"}, "network": {"protocol": "tcp", "bytes": 44, "packets": 1, "transport": "tcp"}, "source": {"ip": "193.163.125.226", "port": 51318}, "destination": {"ip": "19.0.1.193", "port": 49083}, "event": {"outcome": "failure", "category": "network"}, "cloud": {"account": {"id": "637423261198"}, "observer": {"id": "eni-05232e29b9f1cdb2a"}, "log": {"status": "OK\\\\"}}], {"ecs": {"version": "1.0.0"}, "network": {"protocol": "tcp", "bytes": 60, "packets": 1, "transport": "tcp"}, "source": {"ip": "165.154.182.223", "port": 38150}, "destination": {"ip": "19.0.1.193", "port": 1024}, "event": {"outcome": "failure", "category": "network"}, "cloud": {"account": {"id": "637423261198"}, "observer": {"id": "eni-05232e29b9f1cdb2a"}, "log": {"status": "OK\\\\"}}], {"ecs": {"version": "1.0.0"}, "network": {"protocol": "tcp", "bytes": 40, "packets": 1, "transport": "tcp"}, "source": {"ip": "192.241.211.22", "port": 51410}, "destination": {"ip": "19.0.1.193", "port": 10250}, "event": {"outcome": "failure", "category": "network"}, "cloud": {"account": {"id": "637423261198"}, "observer": {"id": "eni-05232e29b9f1cdb2a"}, "log": {"status": "OK\\\\"}}], {"ecs": {"version": "1.0.0"}, "network": {"protocol": "tcp", "bytes": 44, "packets": 1, "transport": "tcp"}, "source": {"ip": "205.210.31.154", "port": 53233}, "destination": {"ip": "19.0.1.193", "port": 61337}, "event": {"outcome": "failure", "category": "network"}, "cloud": {"account": {"id": "637423261198"}, "observer": {"id": "eni-05232e29b9f1cdb2a"}, "log": {"status": "OK\\\\"}}], {"ecs": {"version": "1.0.0"}, "network": {"protocol": "tcp", "bytes": 120, "packets": 2, "transport": "tcp"}, "source": {"ip": "117.241.199.208", "port": 60157}, "destination": {"ip": "19.0.1.193", "port": 23}, "event": {"outcome": "failure", "category": "network"}, "cloud": {"account": {"id": "637423261198"}, "observer": {"id": "eni-05232e29b9f1cdb2a"}, "log": {"status": "OK\\\\"}}], {"ecs": {"version": "1.0.0"}, "network": {"protocol": "tcp", "bytes": 44, "packets": 1, "transport": "tcp"}, "source": {"ip": "198.235.24.231", "port": 54158}, "destination": {"ip": "19.0.1.193", "port": 6002}, "event": {"outcome": "failure", "category": "network"}, "cloud": {"account": {"id": "637423261198"}, "observer": {"id": "eni-05232e29b9f1cdb2a"}, "log": {"status": "OK\\\\"}}], {"ecs": {"version": "1.0.0"}, "network": {"protocol": "tcp", "bytes": 44, "packets": 1, "transport": "tcp"}, "source": {"ip": "35.203.210.44", "port": 56133}, "destination": {"ip": "19.0.1.193", "port": 15498}, "event": {"outcome": "failure", "category": "network"}, "cloud": {"account": {"id": "637423261198"}, "observer": {"id": "eni-05232e29b9f1cdb2a"}, "log": {"status": "OK\\\\"}}], {"ecs": {"version": "1.0.0"}, "network": {"protocol": "tcp", "bytes": 44, "packets": 1, "transport": "tcp"}, "source": {"ip": "162.216.149.46", "port": 49314}, "destination": {"ip": "19.0.1.193", "port": 9195}, "event": {"outcome": "failure", "category": "network"}, "cloud": {"account": {"id": "637423261198"}, "observer": {"id": "eni-05232e29b9f1cdb2a"}, "log": {"status": "OK\\\\"}}], {"ecs": {"version": "1.0.0"}, "network": {"protocol": "tcp", "bytes": 40, "packets": 1, "transport": "tcp"}, "source": {"ip": "79.110.62.69", "port": 48486}, "destination": {"ip": "19.0.1.193", "port": 48301}, "event": {"outcome": "failure", "category": "network"}, "cloud": {"account": {"id": "637423261198"}, "observer": {"id": "eni-05232e29b9f1cdb2a"}, "log": {"status": "OK\\\\"}}], {"ecs": {"version": "1.0.0"}, "network": {"protocol": "tcp", "bytes": 44, "packets": 1, "transport": "tcp"}, "source": {"ip": "35.203.211.23", "port": 56299}, "destination": {"ip": "19.0.1.193", "port": 9603}, "event": {"outcome": "failure", "category": "network"}, "cloud": {"account": {"id": "637423261198"}, "observer": {"id": "eni-05232e29b9f1cdb2a"}, "log": {"status": "OK\\\\"}}], {"ecs": {"version": "1.0.0"}, "network": {"protocol": "tcp", "bytes": 44, "packets": 1, "transport": "tcp"}, "source": {"ip": "35.203.210.166", "port": 52261}, "destination": {"ip": "19.0.1.193", "port": 4221}, "event": {"outcome": "failure", "category": "network"}, "cloud": {"account": {"id": "637423261198"}, "observer": {"id": "eni-05232e29b9f1cdb2a"}, "log": {"status": "OK\\\\"}}}
Ln 1, Col 1 Spaces: 4 UTF-8 CRLF {} JSON
```