Billions Served: Processing Security Event Logs with the AWS Serverless Stack

fwd:cloudsec 2023, Josh Liburdi

Who, Me? 👋

→ 10 years of security industry experience

→ Security Engineer & Tech Lead at Brex

→ Previously: Splunk, Target, CrowdStrike

→ ❤️ making life more difficult for bad guys

The Worst Kept Secret in Security Operations...

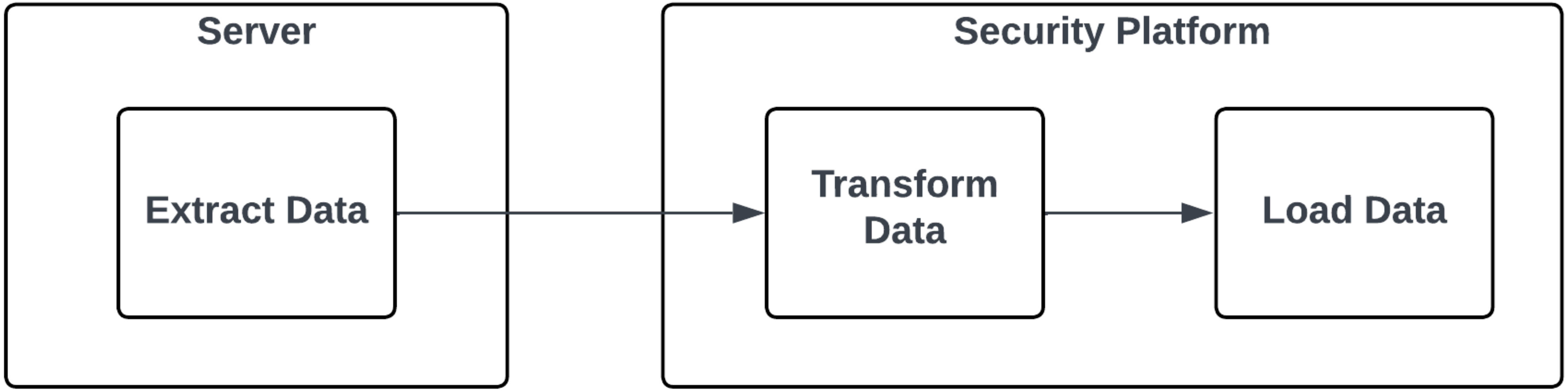
Eventually Everyone Becomes a Data Engineer

```
index=* source=auth
| eval user_name=mvindex(split(email, "@"), 0)
| eval user_domain=mvindex(split(email, "@"), -1)
| join type=inner [ search index=* source=users
    | dedup user_name
    | fields user_name, full_name, department, title ]
    on user_name
| table ts, id, ip, user_name, user_domain,
  full_name, department, title
```

```
SELECT
  a.ts,
  a.id,
  a.ip,
  SUBSTRING_INDEX(a.email, '@', 1) AS user_name,
  SUBSTRING_INDEX(a.email, '@', -1) AS user_domain,
  u.full_name,
  u.department
  u.title
FROM auth a
JOIN users u ON u.user_name =
  SUBSTRING_INDEX(a.email, '@', 1);
```

How did this happen, and can we make it better?

Data Engineering? 🥴



Server

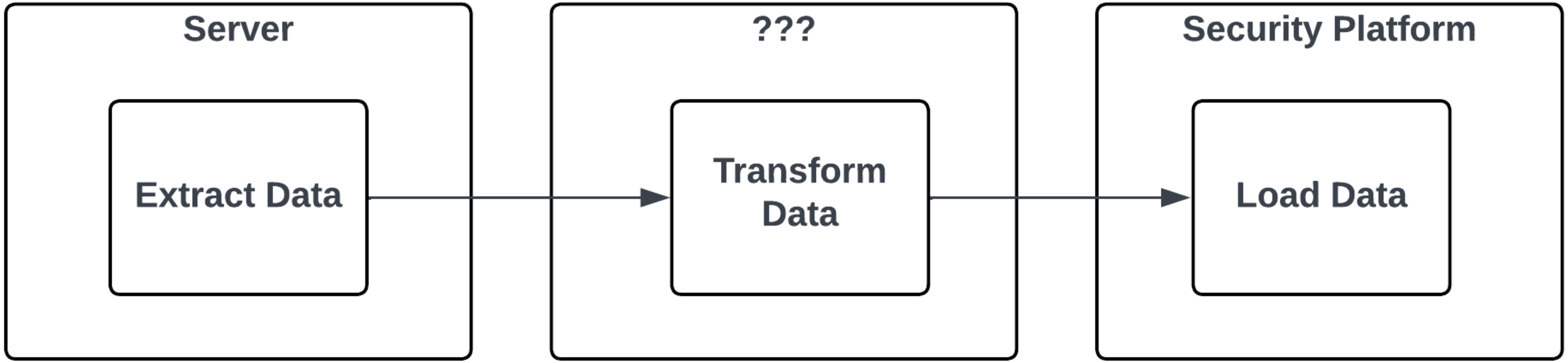
Extract Data

Security Platform

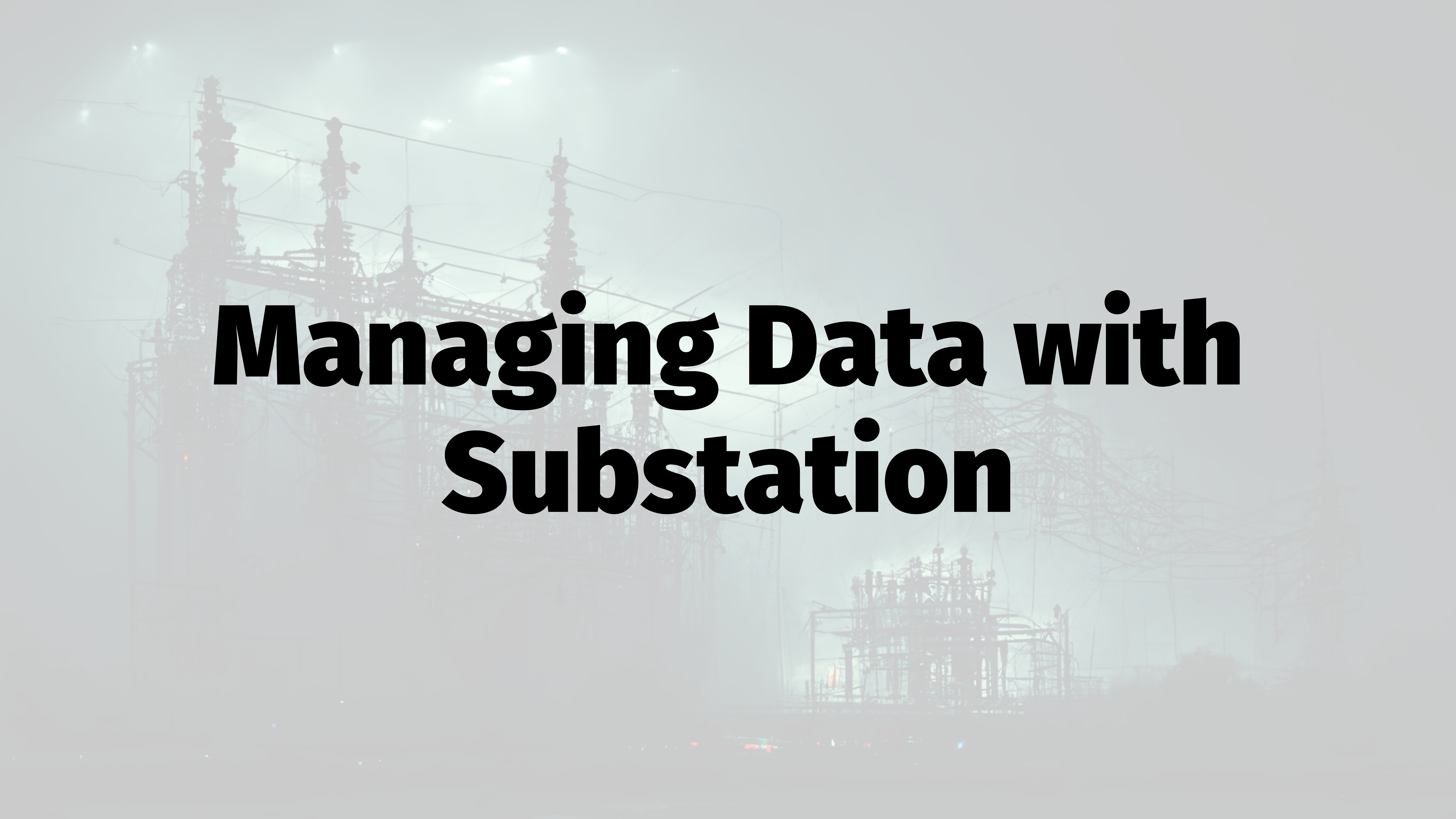
Transform Data

Load Data

Data Engineering! 🤩



Managing Data with Substation



github.com/brexhq/substation

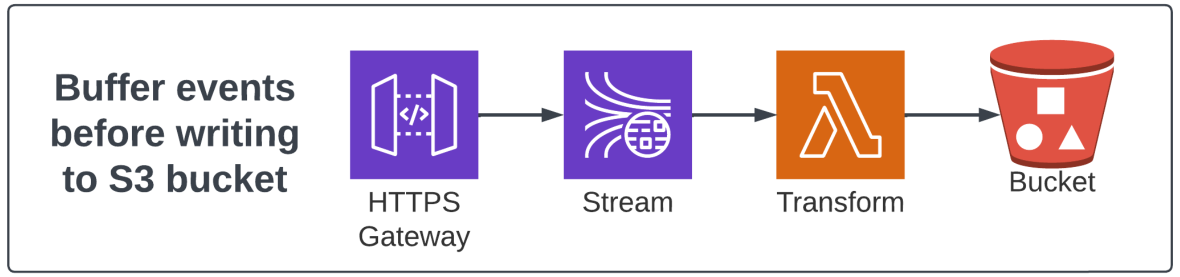
Substation is a cloud-native, event-driven data pipeline and transformation toolkit written in Go.

→ Designed for Security Operations teams

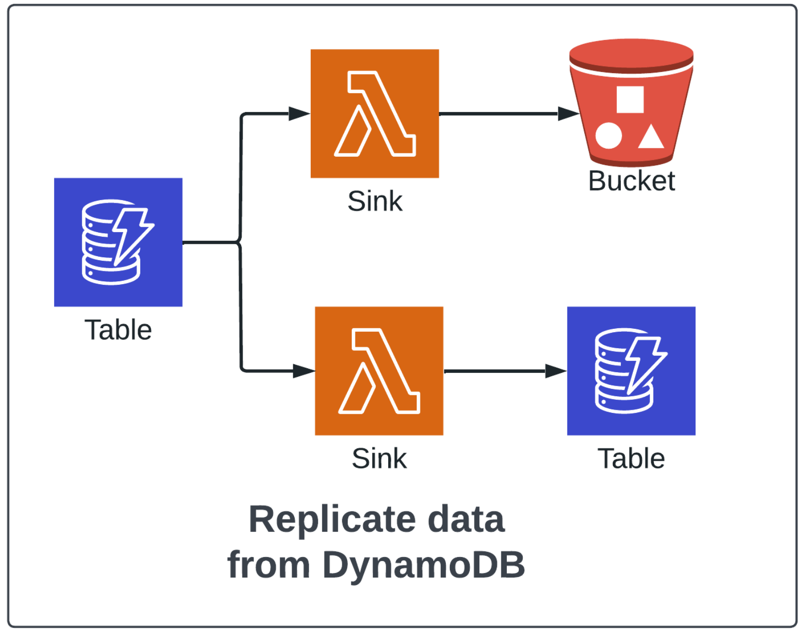
→ Built by Detection and Response at Brex

```json
{
  "eventVersion": "1.08",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AROAS5AFBLNG2RLOZNWEQ:anonymous@example.com",
    "arn": "arn:aws:sts::987654321012:assumed-role/AWSReservedSSO_ACCOUNT_87654321/anonymous@example.com",
    "accountId": "987654321012",
    "sessionContext": { ... }
  },
  "eventTime": "2023-05-16T21:47:25Z",
  "eventSource": "signin.amazonaws.com",
  "eventName": "ConsoleLogin",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.0.2.0",
  "userAgent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36",
  "requestParameters": null,
  "responseElements": {
    "ConsoleLogin": "Success"
  },
  "additionalEventData": {
    "MobileVersion": "No",
    "MFAUsed": "No"
  },
  "eventID": "130e6e1b-4753-4080-a398-07dc9fb53cb0",
  "readOnly": false,
  "eventType": "AwsConsoleSignIn",
  "managementEvent": true,
  "recipientAccountId": "987654321012",
  "eventCategory": "Management",
  "tlsDetails": {
    "tlsVersion": "TLSv1.3",
    "cipherSuite": "TLS_AES_128_GCM_SHA256",
    "clientProvidedHostHeader": "us-east-1.signin.aws.amazon.com"
  }
}
```

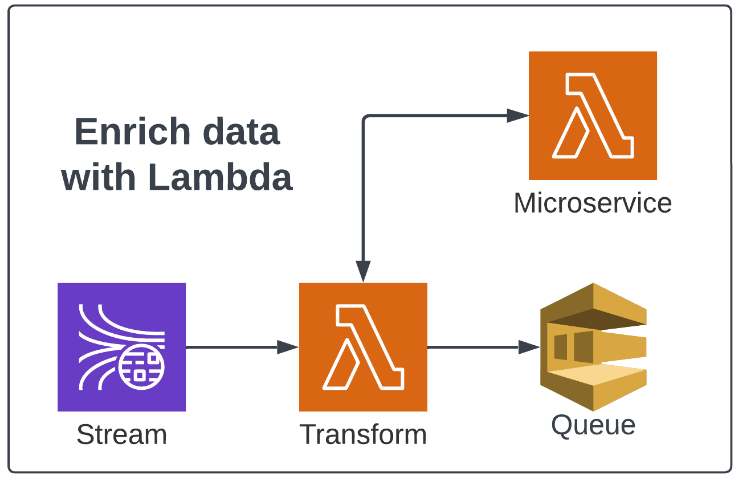
```
{
  "@timestamp": "2023-05-16T21:47:25Z",
  "event": {
    "action": "ConsoleLogin",
    "id": "130e6e1b-4753-4080-a398-07dc9fb53cb0",
    "hash": "2c0866a6957af6d6d3836b740b0a6d7b43a2f57398e74f26c7a2ef1e1718f972",
    "original": { ... },
    "outcome": "success"
  },
  "cloud": {
    "account": {
      "id": "987654321012",
      "name": "Development"
    },
    "provider": "aws",
    "region": "us-east-1",
    "service": {
      "name": "signin"
    }
  },
  "source": {
    "ip": "192.0.2.0",
    "domain": "c-192-0-2-0.hsd1.ca.comcast.net",
    "as": {
      "organization": {
        "name": "Comcast Cable Communications, LLC"
      },
      "number": 7922
    }
  },
  "tls": {
    "cipher": "TLS_AES_128_GCM_SHA256"
  },
  "user": {
    "email": "anonymous@example.com",
    "full_name": "Jane Doe",
    "roles": [ "admin", "security" ]
  },
  "user_agent": {
    "original": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36",
  }
```



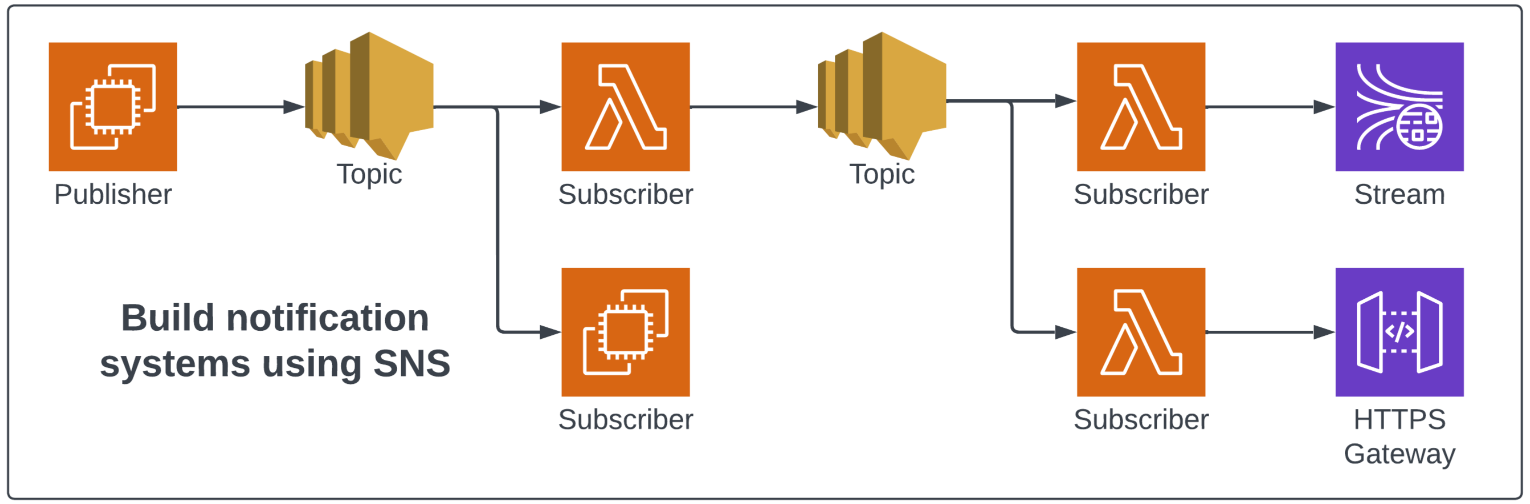
Buffer events before writing to S3 bucket



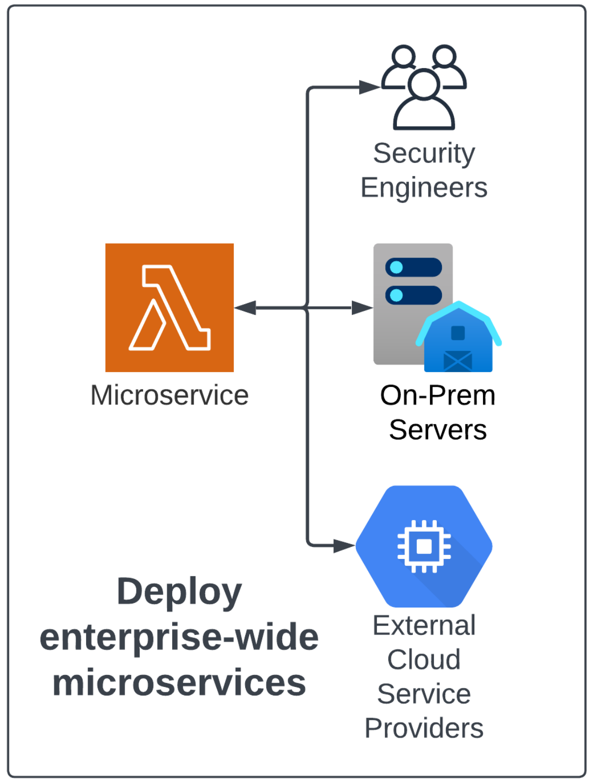
Replicate data from DynamoDB



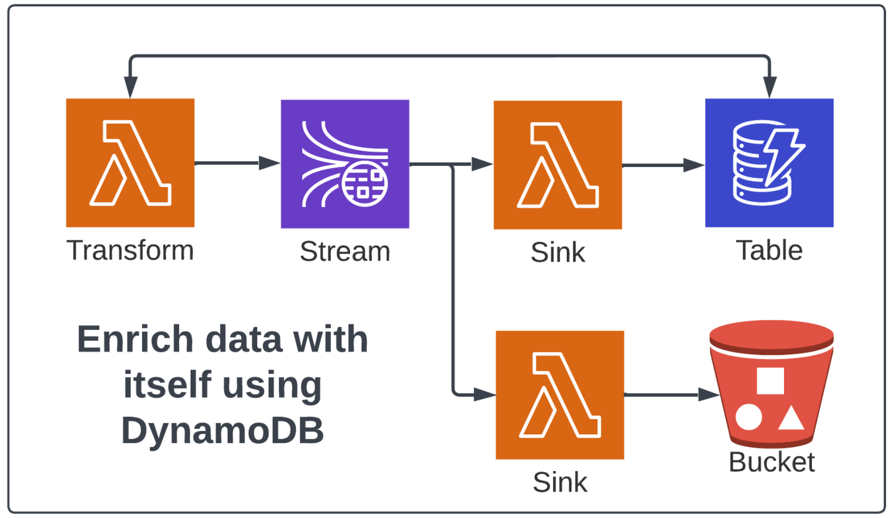
Enrich data with Lambda



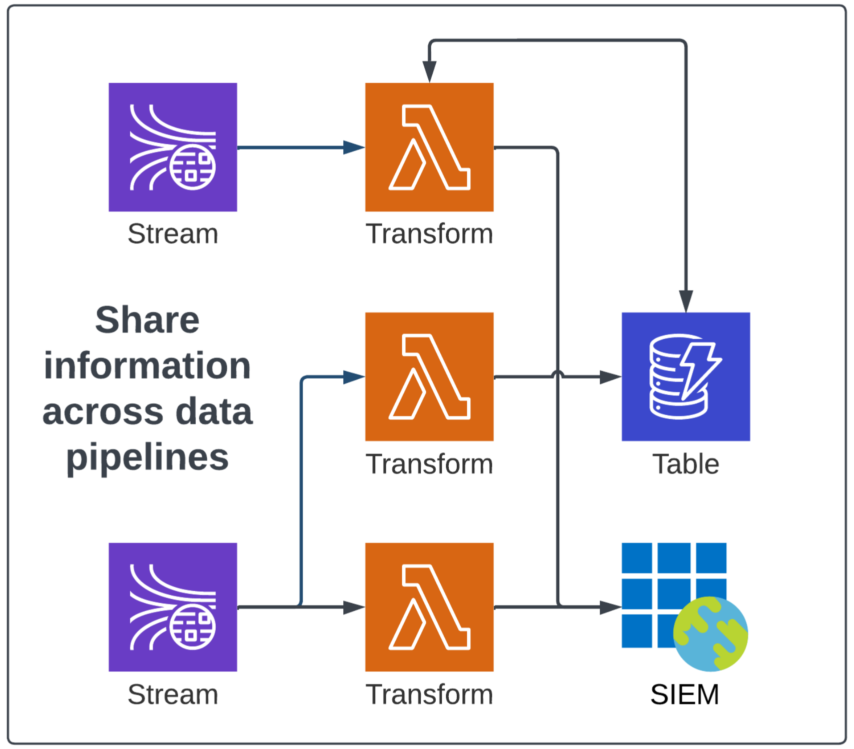
Build notification systems using SNS



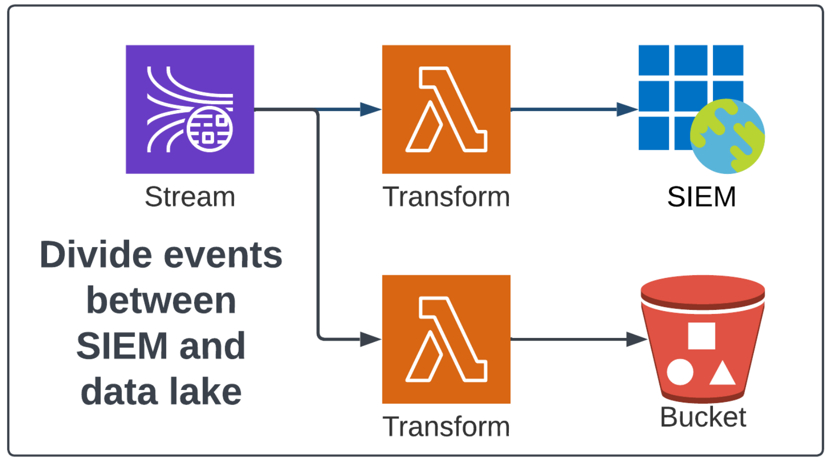
Deploy enterprise-wide microservices



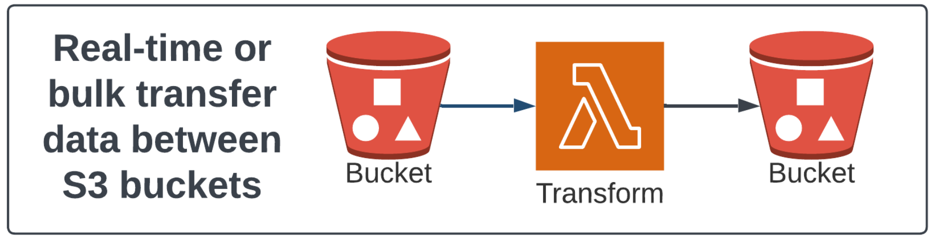
Enrich data with itself using DynamoDB



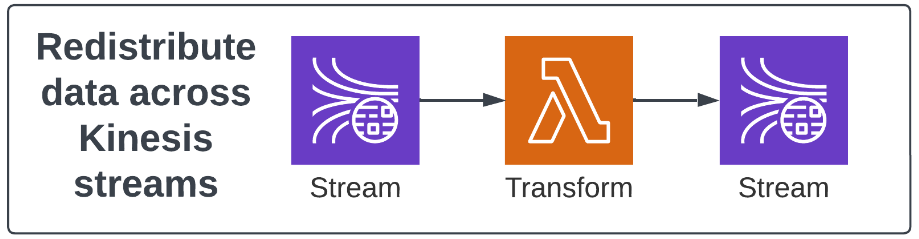
Share information across data pipelines



Divide events between SIEM and data lake



Real-time or bulk transfer data between S3 buckets



Redistribute data across Kinesis streams

github.com/brexhq/substation

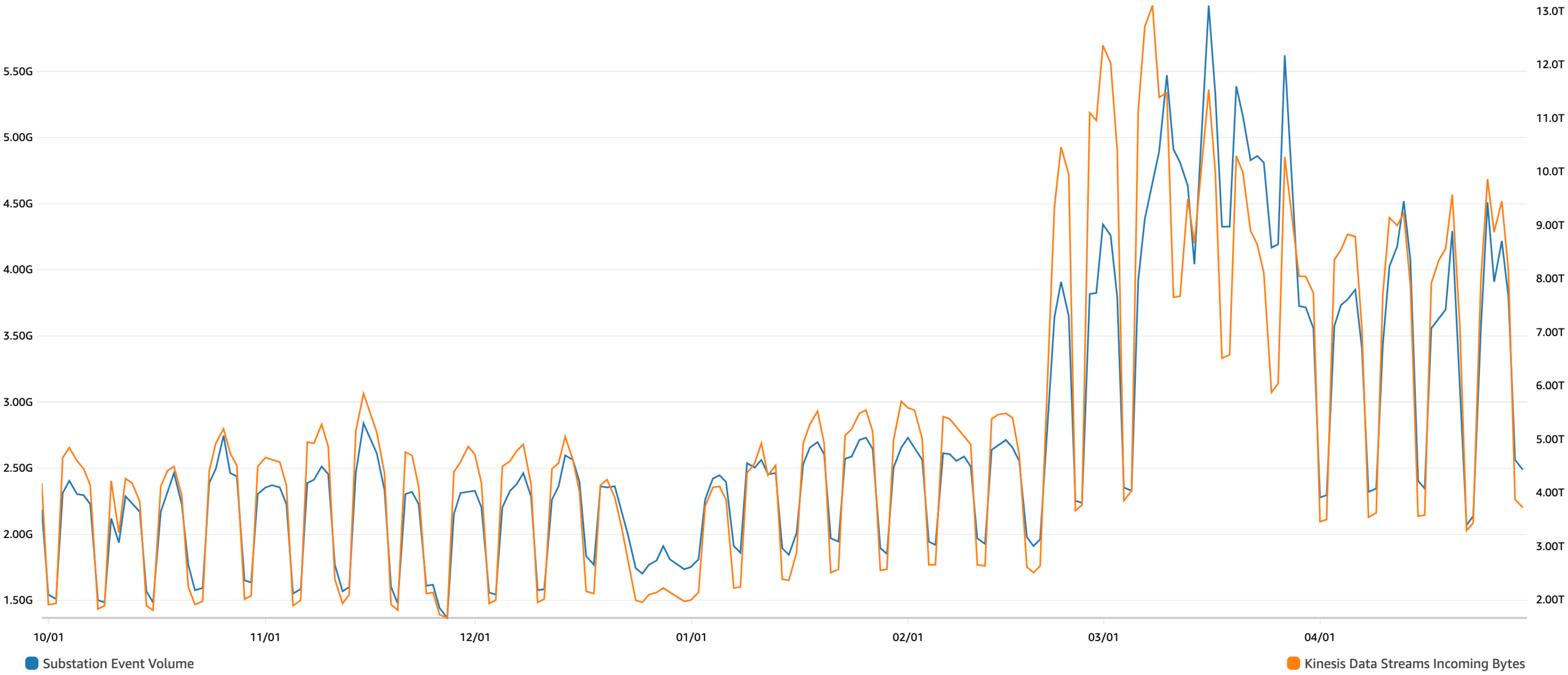
→ Used in production for 2+ years

→ 1,000,000,000s of events processed per day

→ 1,000,000s of transforms executed every second

→ $0.01/GB to $0.05/GB (all-in cost)

→ <1 hour maintenance each week


5.50G
5.00G
4.50G
4.00G
3.50G
3.00G
2.50G
2.00G
1.50G
13.0T
12.0T
11.0T
10.0T
9.00T
8.00T
7.00T
6.00T
5.00T
4.00T
3.00T
2.00T
10/01
11/01
12/01
01/01
02/01
03/01
04/01
🟦 Substation Event Volume
🟧 Kinesis Data Streams Incoming Bytes

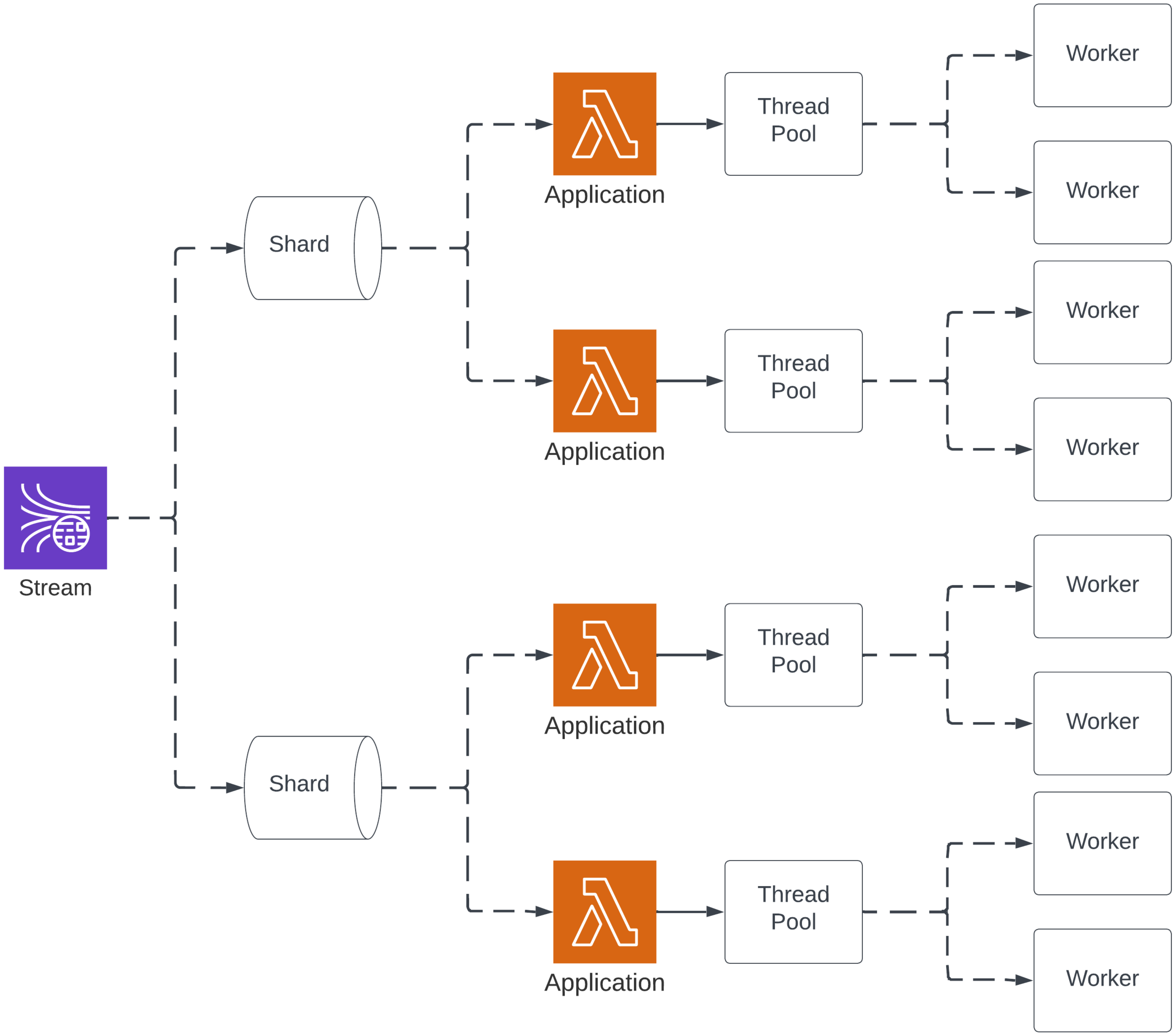
Optimizing the AWS Serverless Stack

Optimizing Lambda - Parallelism

- → Event source determines concurrency
- → 1-1: S3, SNS, API Gateway
- → N-1: SQS, Kinesis, DynamoDB

- → Parallelization Factor invokes up to 10x functions per batch
- → Kinesis & DynamoDB only

- → Multi-threaded functions can have perf. boost depending on use case


Stream
Shard
Shard
Application
Application
Application
Application
Thread Pool
Thread Pool
Thread Pool
Thread Pool
Worker
Worker
Worker
Worker
Worker
Worker
Worker
Worker

Optimizing Lambda - Parallelism

Tip: Use multi-threading for I/O bound tasks

→ Data transformation is *usually* CPU bound, but becomes I/O bound when enriching data

→ Thread pool can improve application performance and reduce overall runtime

Optimizing Lambda - Parallelism

Use Case: Enrich event logs with external services

→ DNS

→ IP<>Domain

→ TXT records

→ HTTP

→ Location

→ Reputation

→ Intelligence

→ Lambda

→ Internal APIs

→ Custom data processing

Optimizing Lambda - More Tips

- More memory, more vCPU (1770MB = 2 vCPU)
- Keep local enrichment data in memory
- Lazy load external resources once
- Monitor API calls and performance with X-Ray
- Use AppConfig to continuously retrieve configurations and avoid cold starts

Optimizing Kinesis - Aggregation

Tip: Use the Kinesis Producer & Consumer Libraries

→ Aggregate many events into a single record to increase throughput and *significantly* reduce cost

→ Formats: Protobuf (KPL, KCL), JSON arrays, compression … nearly anything works!

Optimizing Kinesis - Aggregation

Size x Events Per Second (EPS)	Kinesis Data Streams (Provisioned)	Kinesis Firehose	Kinesis Data Streams (On-Demand)	Managed Streaming Kafka (MSK)[1]
1KB x 10k (10 MB/s)	$17/day	$119/day	$100/day	$48/day
1KB x 100k (100 MB/s)	$174/day	$1094/day	$987/day	$373/day
5KB x 20k (100 MB/s)	$77/day	$238/day	$987/day	$373/day
25KB x 4k (100 MB/s)	$58/day	$238/day	$987/day	$373/day

[1] Cluster settings: m5.large, 3 replicas & AZs, 24 hours of retention

Optimizing Kinesis - Additional Costs

Kinesis Data Streams

→ 3+ Consumers: ~10% increase each cons.

→ Enhanced Consumer: $0.013/GB + $0.015/sh

→ Extended (7-Day) Retention: $0.0068/GB

Kinesis Firehose

→ Dynamic Partitioning: $0.02/GB

→ Data Conversion: $0.018/GB

→ VPC (PrivateLink): $0.01/GB

Optimizing Kinesis - More Tips

→ Batch size and window affects Lambda cost

→ Avoid hot shards with random partition keys

→ Use auto-scaling; scale up quick and down slow

→ Bursts of records will cause errors on write, increase retries and exponential backoff

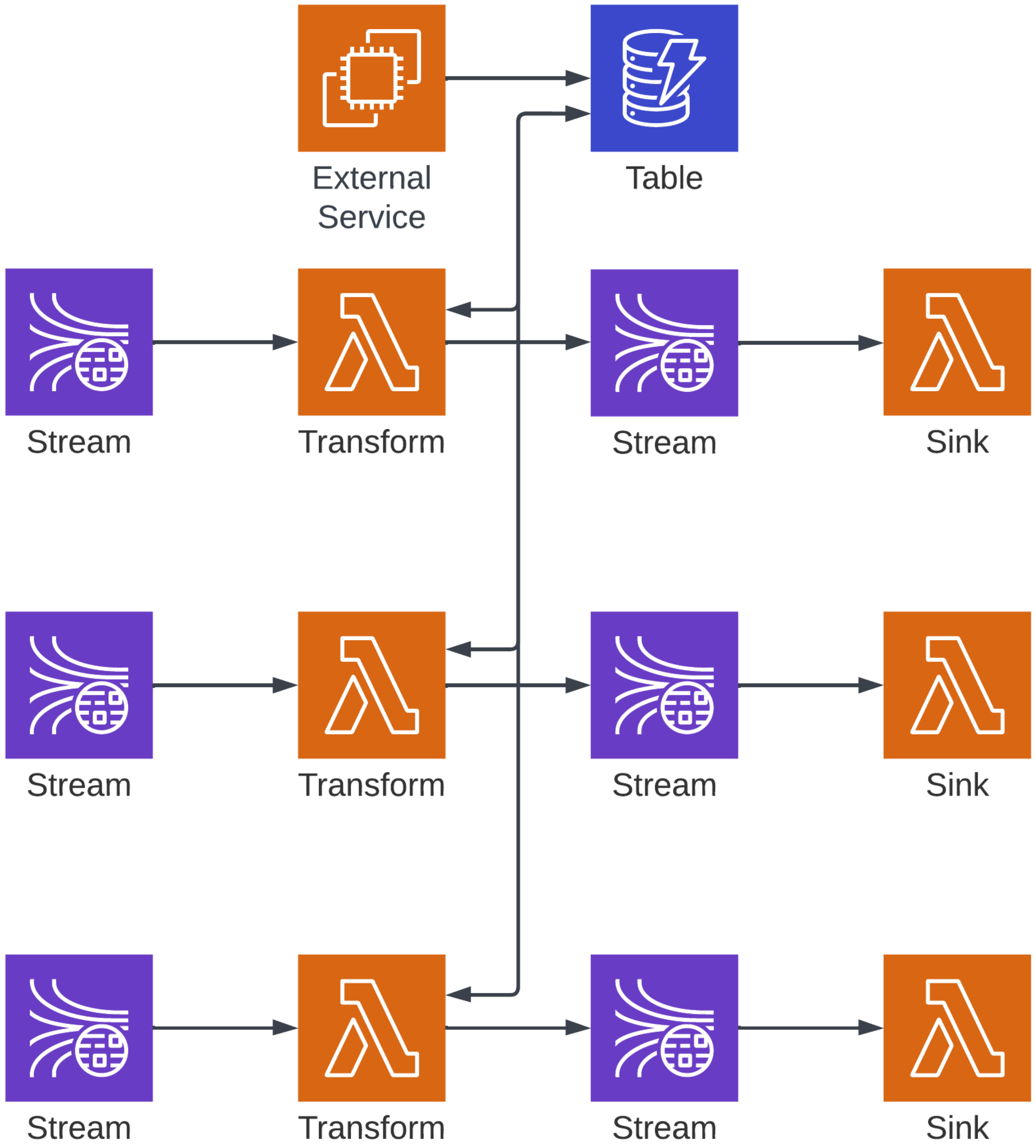
Optimizing DynamoDB - Distributed Cache

Tip: Use DynamoDB as a distributed cache for enrichment data

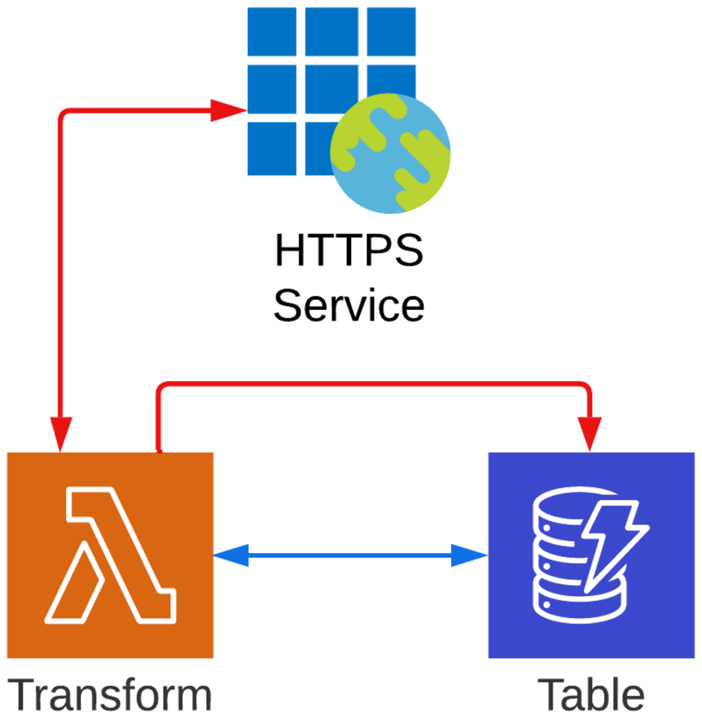
→ Use cache aside pattern to improve performance

→ Keep data fresh with configurable time-to-live

Distributed Cache



Cache Aside



Optimizing DynamoDB - Distributed Cache

Use Case: Any event log can become context

→ Data-Driven Inventories

→ Indicators of Compromise

→ Cache API Responses

→ Curate Biz & Threat Intel

→ Share Data Between Services

→ Share Info Across Teams

Optimizing DynamoDB - More Tips

- → Practice single-table design
- → Use Provisioned capacity with auto-scaling
- → Retrieve all data for an entity in one query
- → Use in-memory cache to reduce query volume
- → Use hash functions on large partition keys and sort keys, store large items in S3

Serverless Gotchas - Continuous Retries

Tip: Use Lambda's continuous retries carefully

→ Polling event sources retry until data expires

→ Duplicates data and costs will 🚀

→ Use CloudWatch to alert on errors or use dead letter queues

Serverless Gotchas - Backpressure

Tip: Don't under-provision downstream services

→ Security event logs will burst: backpressure and delayed processing is a risk

→ Use auto-scaling features or deploy custom auto-scaling applications

Serverless Gotchas - Bottlenecks

Tip: 📈 Lambda Duration and IteratorAge metrics can identify bottlenecks

→ Lambda: increase memory or p. factor

→ Kinesis: increase shard count

→ DynamoDB: increase read or write capacity

Thanks for Listening!

→ Reach out on LinkedIn **linkedin.com/in/joshliburdi**

→ Read on for resources that can help you optimize Lambda, Kinesis, and DynamoDB!

Resources - Lambda

- → *Operating Lambda: Performance optimization (Parts 1, 2, 3)* by James Beswick
- → *Optimizing your AWS Lambda costs (Parts 1 & 2)* by Chris Williams & Thomas Moore
- → *Caching data and configuration settings with AWS Lambda extensions* by Hari Ohm Prasath Rajagopal & Vamsi Vikash Ankam

Resources - Kinesis

→ *Kinesis vs. Kafka: Which Stream Processor Comes Out on Top?* by Alex Chan

→ *Mastering AWS Kinesis Data Streams (Parts 1 & 2)* by Anahit Pogosova

→ *Amazon Kinesis Data Streams: Auto-scaling the number of shards* by Brandon Stanley

Resources - DynamoDB

→ *Best practices for designing and architecting with DynamoDB* (AWS docs)

→ *The What, Why, and When of Single-Table Design with DynamoDB* by Alex DeBrie

→ *Maximize cost savings and scalability with an optimized DynamoDB secondary index* by Pete Naylor