# Unlocking Serverless Observability

A Serverless Integration Journey

# Who am I?

- Serverless Architect, Data & AI at Accenture in Melbourne
- AWS Community Builder (Serverless)
- Co-Host AWS Programming and Tools Meetup (Melbourne)
- Witnessed the birth of the Internet as an Adult

## What are we Talking about?

- What is Observability?
- Meet Alice
- What did Alice Build?
- Unlock Serverless Observability
- Wrap-up and References

# What is Observability?

- Able to ask questions and infer answers

# What is Observability?

- Able to ask questions and infer answers
- Logs must “Tell a Story"

# What is Observability?

- Able to ask questions and infer answers
- Logs must “Tell a Story”
- Logs must be structured

# What is Observability?

- Able to ask questions and infer answers
- Logs must “Tell a Story”
- Logs must be structured
- Metrics should Track Business KPIs

# What is Observability?

- Able to ask questions and infer answers
- Logs must “Tell a Story”
- Logs must be structured
- Metrics should Track Business KPIs
- Traces to give you performance data

# Meet Alice

- Cloud engineer at Widgets Inc.
- Keeps the lights on for the proprietary e-commerce website
- Responsible for running the Weekly export Sales reports
- Export runs weekly because the EC2 is at capacity all other times of the week

# Max From Sales and Marketing

- Needs a view of Orders as they happen rather than just weekly
- A New delivery partner is also coming on board who has an API for scheduling pickups

# What Can Alice do?

- Alice worries about adding more processing to the e-Commerce Server since its at capacity and Widgets Inc doesn't have a large budget
- She notices that the e-commerce suite has an existing web hook allowing notifications to be sent out

Serverless Land

Content Learn Code EDA Search Search

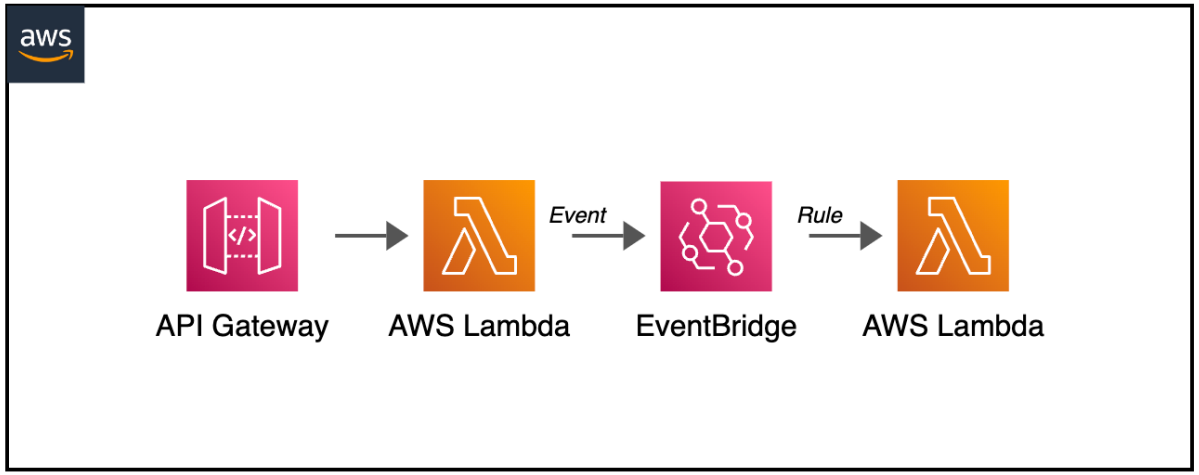
# Welcome to *Serverless* Land

This site brings together the latest information, blogs, videos, code, and learning resources for AWS Serverless. Learn to use and build apps that scale automatically on low-cost, fully-managed serverless architecture.

## Building **Event Driven** Architectures

Event-driven architectures are an architecture style that can help you boost agility and build reliable, scalable applications. Serverless services like EventBridge, Step Functions, SQS, SNS, and Lambda have a natural affinity with event-driven architectures - they are invoked by events, emit events, and have

# API Gateway to Lambda to EventBridge




## Make a request to API Gateway that sends an event to EventBridge via Lambda.

This pattern creates an Amazon API Gateway HTTP API, a AWS Lambda function (publisher), a custom EventBridge bus and event rule, and another

< Back to patterns

**Download**

```
> git clone https://github.com/aws-sa
  cd serverless-patterns/apigw-lambda
```

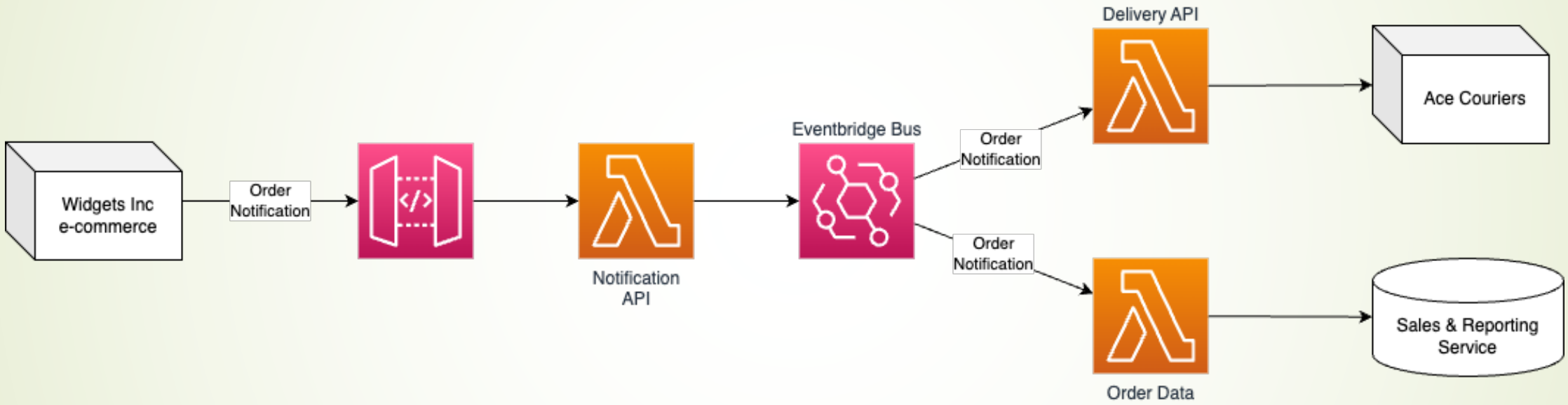
**Deploy**

```
> sam deploy --guided
```

**Testing**

See the GitHub repo for detailed testing instructions.

# What Alice Built



# How Alice Built it …

- Used retries with Variable Wait time + random Jitter

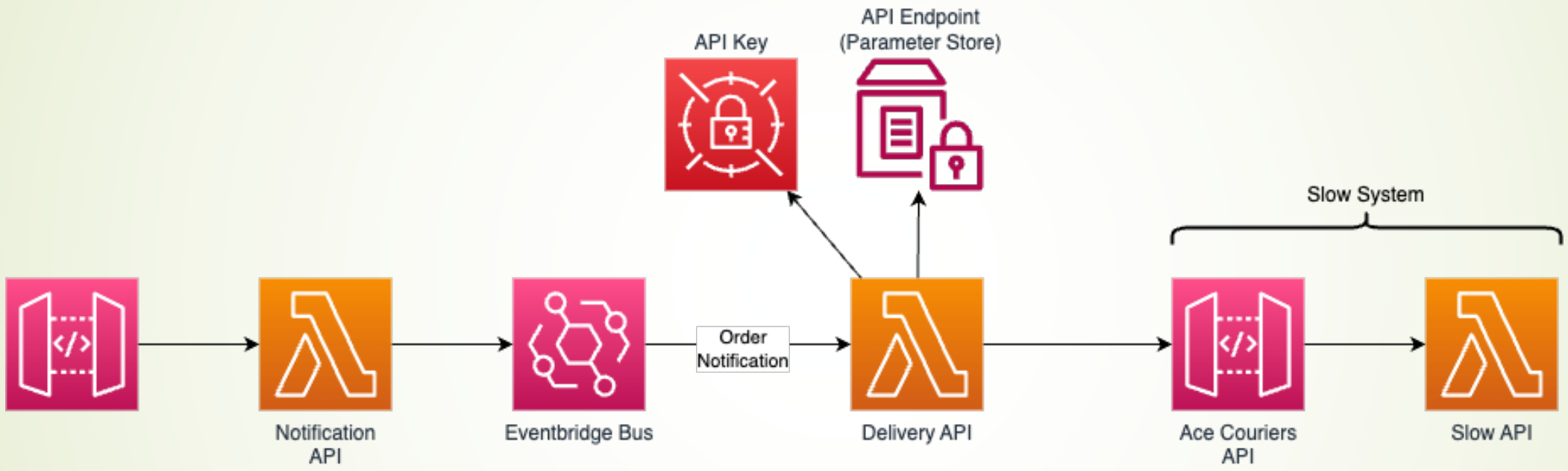
```python
# retry up to 5 times and wait 3 seconds + random time f
@retry(wait=wait_fixed(3) + wait_random(0, 5), stop=stop
def try_api_delivery(
    endpoint: str, api_key: str, correlation_id: str, bo
) -> str:
    # do stuff
    logger.info("trying API delivery")
    http_headers: Dict[str, str] = {
        "x-api-key": api_key,
        "x-correlation-id": correlation_id,
    }

    response = requests.post(url=endpoint, json=body, he

    # Log the status code
    logger.info({"message": response.status_code})

    # raise exception to enable tenacity retry
    response.raise_for_status()
    return response.json()

```

# How Alice Built it …

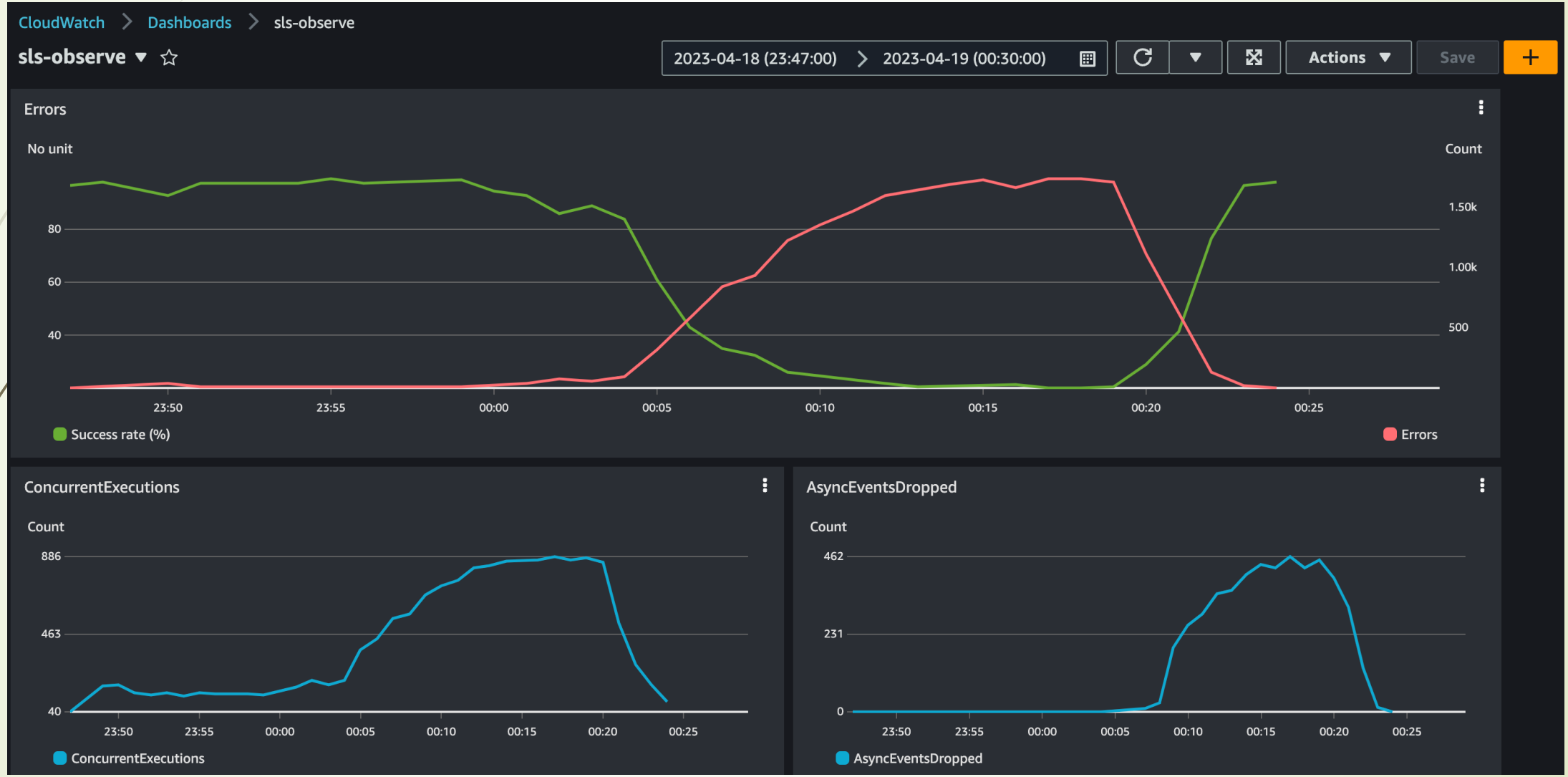
- Used a Logging framework (AWS Lambda Powertools for Python)
  - Structured Logging
  - Extracts Correlation ID from payload

```
1
2 @logger.inject_lambda_context(
3     log_event=True, correlation_id_path="detail.meta_data.correlat
4 )
5 @event_source(data_class=EventBridgeEvent)
6 def handler(event: EventBridgeEvent, context: LambdaContext):
7     logger.info({"status": "START", "message": "Processing Deliver
8
9     api_body: Dict[str, Any] = event.detail
10
11     correlation_id: str = event.detail.get("meta_data", {}).get(
12         "correlation_id", "undefined"
13     )
14
15     # remove meta_data from the API body
16     del api_body["meta_data"]
17
18     # read endpoint and API key
19     try:
20         # read endpoint and API Key
21         endpoint: str = parameters.get_parameter("/sls-observe/del
22         api_key: str = ssm_provider.get("/sls-observe/delivery/api
23
24         # robustly try and retry API Delivery (uses tenacity retry
25         response = try_api_delivery(endpoint, api_key, correlation
```

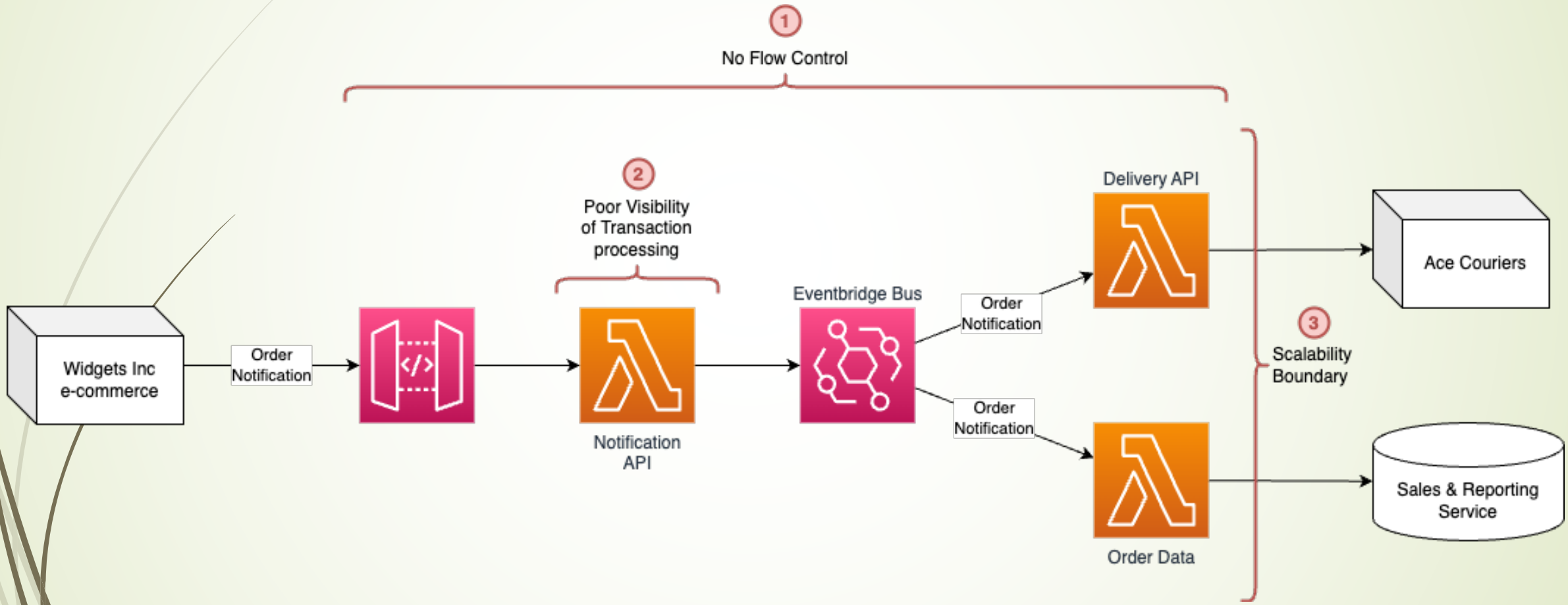
# Actual Demo Project



# What Happened at Scale ?



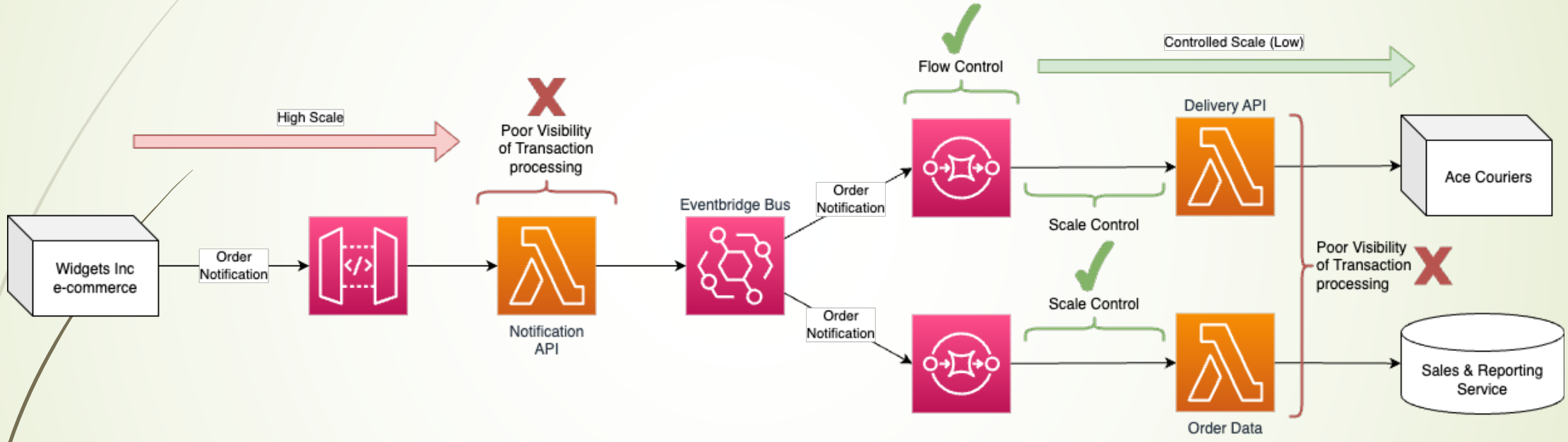
# What Went Wrong?



# Key Integration Considerations

- Understand your system Volumes
  - What is YOUR system's expected Transactions per second?
- Understand Downstream Constraints
  - Are you rate Limited?
  - What is their maximum capacity (TPS)
- Add-in Flow Control to your Architecture
- Introduce a **correlation_id** and **transaction_id** into your logging
- Do Structured logging (JSON format)
  - Enables queries
  - Can Ask questions
  - Can create new Metrics from log queries

# Adding in Flow Control



# What do Good Logs look like?

```
1 {
2     "level": "INFO",
3     "location": "handler:22",
4     "message":
5     {
6         "status": "START",
7         "message": "Processing Order Notification"
8     },
9     "timestamp": "2023-04-18 12:04:42,386+0000",
10    "service": "notify-handler",
11    "cold_start": true,
12    "function_name": "sls-observe-NotificationFunction-IORZrEPMkMaX",
13    "function_memory_size": "1024",
14    "function_arn": "arn:aws:lambda:ap-southeast-2:308836149415:function:sls-observe-NotificationFunction",
15    "function_request_id": "c8d0b6b9-b853-4a1a-b8bc-c8e5fa486fdb",
16    "correlation_id": "bb85f862-3e7f-4491-b3d2-8360a99ac8da",
17    "xray_trace_id": "1-643e8759-19a106fd752af65341bc2267"
18 }
```

# The Power of Structured Logging

```
▼ 2      2023-04-18T22:04:42.391+10:00    notify-handler         START        Processing Order Notification
  Field                 Value
  @ingestionTime        1681819485805
  @log                  308836149415:/aws/lambda/sls-observe-NotificationFunction-IORZrEPMkMaX
  @logStream            2023/04/18/[$LATEST]842e5e9253644c93bcd8575e55ab665b
  @message              {"level":"INFO","location":"handler:22","message":{"status":"START","message":"Processing Order Notification"},"timestamp":"2023-04-18 12:04:42,38
  @timestamp            1681819482391
  cold_start            1
  correlation_id        bb85f862-3e7f-4491-b3d2-8360a99ac8da
  function_arn          arn:aws:lambda:ap-southeast-2:308836149415:function:sls-observe-NotificationFunction-IORZrEPMkMaX
  function_memory_size  1024
  function_name         sls-observe-NotificationFunction-IORZrEPMkMaX
  function_request_id   c8d0b6b9-b853-4a1a-b8bc-c8e5fa486fdb
  level                 INFO
  location              handler:22
  message.message       Processing Order Notification
  message.status        START
  service               notify-handler
  timestamp             2023-04-18 12:04:42,386+0000
  xray_trace_id         1-643e8759-19a106fd752af65341bc2267
```

# What does a “Story” look like?

| # | @timestamp | service | status | message |
|---|---|---|---|---|
| 1 | 2023-04-18T22:04:42.391+10:00 | notify-handler | | |
| 2 | 2023-04-18T22:04:42.391+10:00 | notify-handler | START | Processing Order Notification |
| 3 | 2023-04-18T22:04:42.391+10:00 | notify-handler | | |
| 4 | 2023-04-18T22:04:42.448+10:00 | notify-handler | COMPLETE | |
| 5 | 2023-04-18T22:04:43.366+10:00 | delivery-handler | | |
| 6 | 2023-04-18T22:04:43.366+10:00 | delivery-handler | START | Processing Delivery Notification |
| 7 | 2023-04-18T22:04:43.656+10:00 | delivery-handler | | |
| 8 | 2023-04-18T22:04:44.272+10:00 | slow-api | START | Slow processing... |
| 9 | 2023-04-18T22:04:44.272+10:00 | slow-api | | slow response, waiting 1 seconds |
| 10 | 2023-04-18T22:04:45.274+10:00 | slow-api | | |
| 11 | 2023-04-18T22:04:45.274+10:00 | slow-api | FAILED | random process failure |
| 12 | 2023-04-18T22:04:45.282+10:00 | delivery-handler | | 502 |
| 13 | 2023-04-18T22:04:48.332+10:00 | delivery-handler | | |
| 14 | 2023-04-18T22:04:48.375+10:00 | slow-api | | |
| 15 | 2023-04-18T22:04:48.375+10:00 | slow-api | START | Slow processing... |
| 16 | 2023-04-18T22:04:48.375+10:00 | slow-api | | slow response, waiting 10 seconds |
| 17 | 2023-04-18T22:04:58.385+10:00 | slow-api | COMPLETE | |
| 18 | 2023-04-18T22:04:58.394+10:00 | delivery-handler | | 200 |
| 19 | 2023-04-18T22:04:58.394+10:00 | delivery-handler | | processed |
| 20 | 2023-04-18T22:04:58.394+10:00 | delivery-handler | COMPLETE | |

# Unlocking Serverless Observability

- Think about your logging before you build!
- Write Log messages to tell a "Story"
- Log Transaction and Correlation ID
  - Make them part of your Message body (means data will pass through ALL cloud services)
- Track Key Metrics per Lambda (all available on Lambda Metric Console)
  - Concurrent Invocations (Should always alarm on account concurrency level!)
  - Duration – this is the fastest indicator of impending problems
  - Look at Cloudwatch Anomolay detection to assist with duration variance
- Log key business Metrics
  - The absence of metrics is equally important!

# Wrapup and References

- Open Source Tools used today:
  - AWS Lambda Powertools for Python
  - SAM Cli
  - Python
- Twitter: @walmsles
- Mastadon: @walmsles@hachyderm.io
- Currently Building Serverless DNA - serverlessdna.com
- Learning More about Serverless: serverlessland.com