

AWS
re:Invent

D O P 2 0 8 - R

Amazon's approach to failing successfully

Becky Weiss

Senior Principal Engineer
Amazon Web Services

This is a talk about failing ... successfully



Agenda

Never waste a failure: The AWS approach to post-mortems

Seeing your failures before your customers do

How AWS can help you fail successfully

Never waste a failure: Post-mortems at AWS



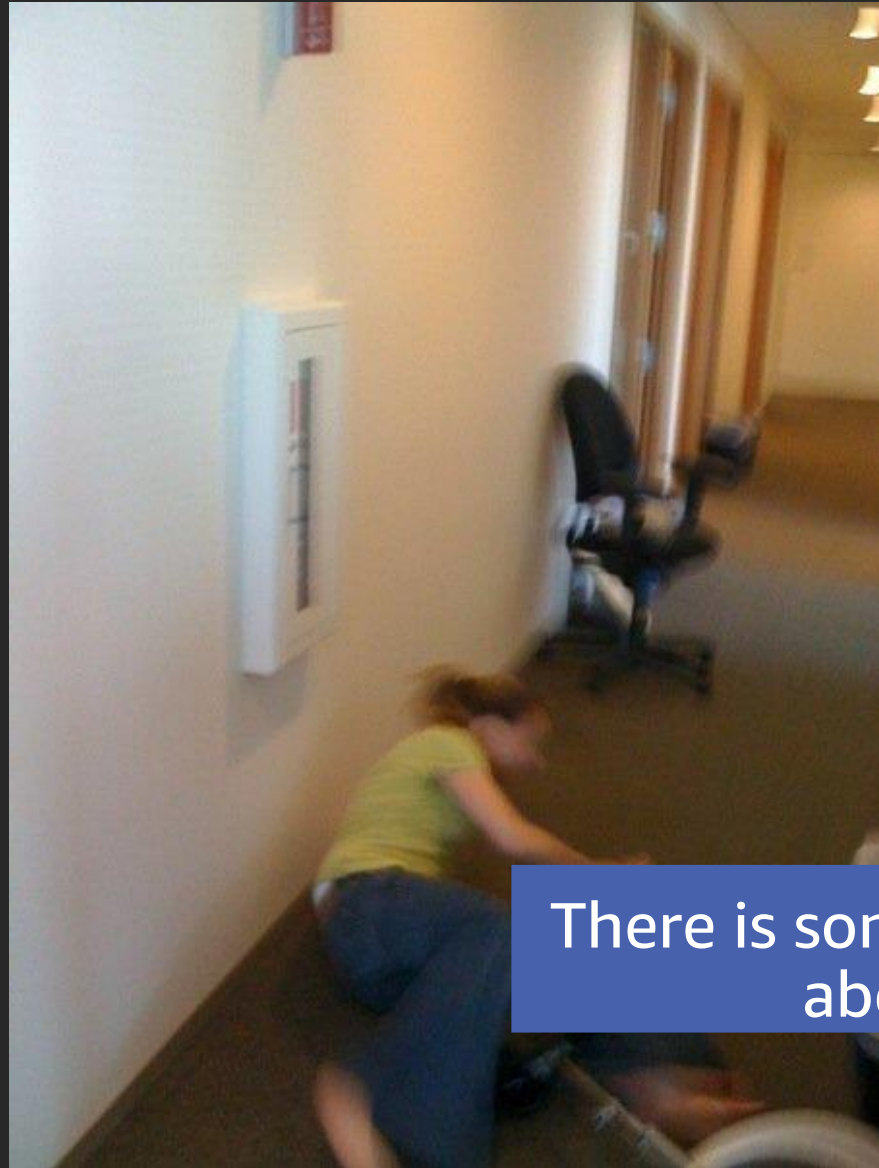
Werner Vogels 

@Werner

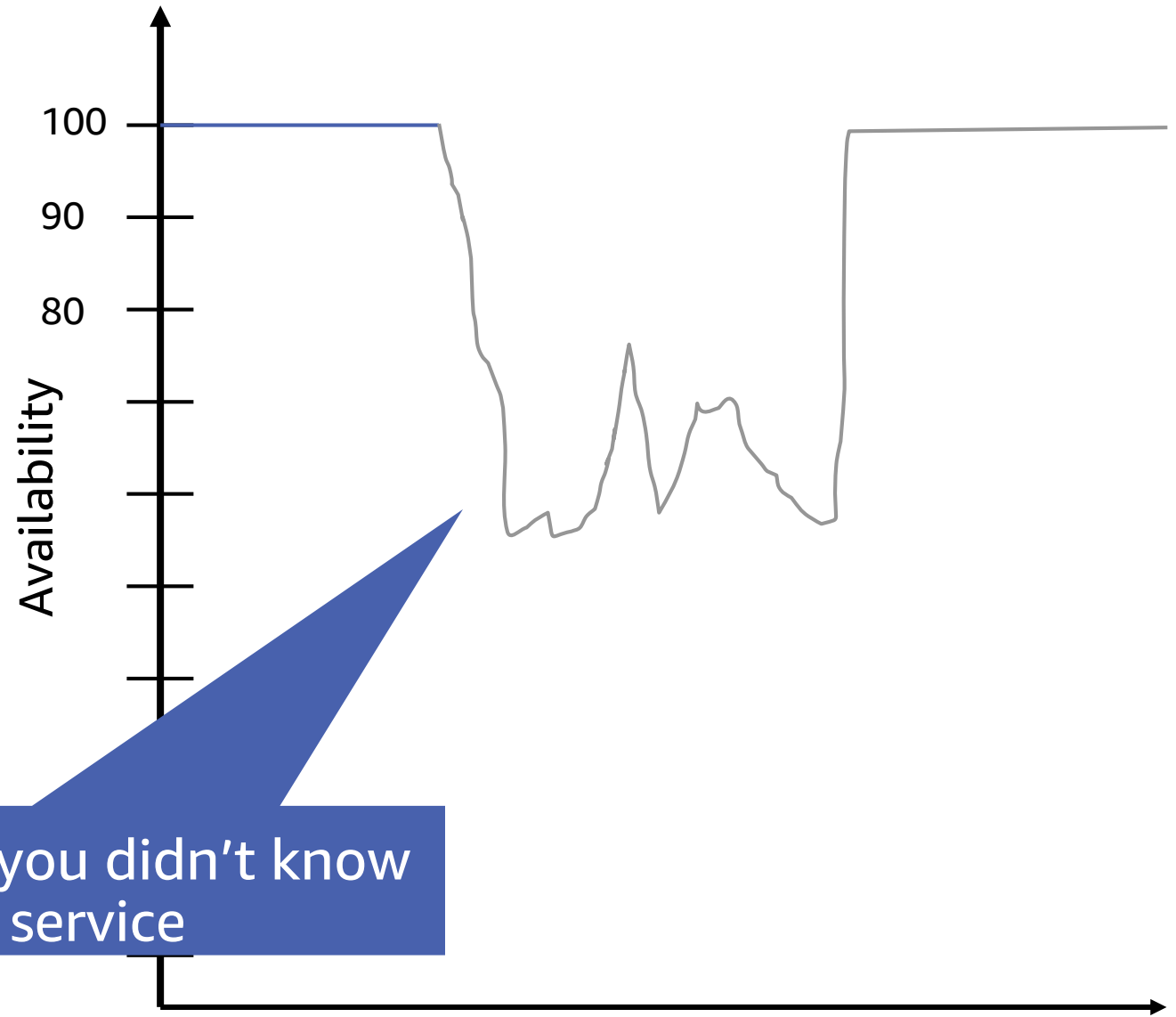
Follow



There is no compression algorithm for
experience.



There is something you didn't know about your service



COE: Correction of error

Structured analysis of customer-impacting events

Reflection of Amazon's peculiar culture

Goes well beyond "How do we prevent this from happening again"



We take these very seriously

[89458] Decaf coffee brewed in a non-decaf pot

Summary

At 8:45 am Pacific on 4/24/2018 on the 12th floor of Alexandria, an operator inadvertently brewed decaf coffee into the "medium roast, non-decaf" coffee pot. The operator was attempting to brew decaf coffee, but chose the wrong pot to brew it into.

Metrics / Graphs

- Number of pots of coffee brewed incorrectly during incident: 1
- Number of customers who unknowingly took a cup of decaf coffee: 2 (estimated)

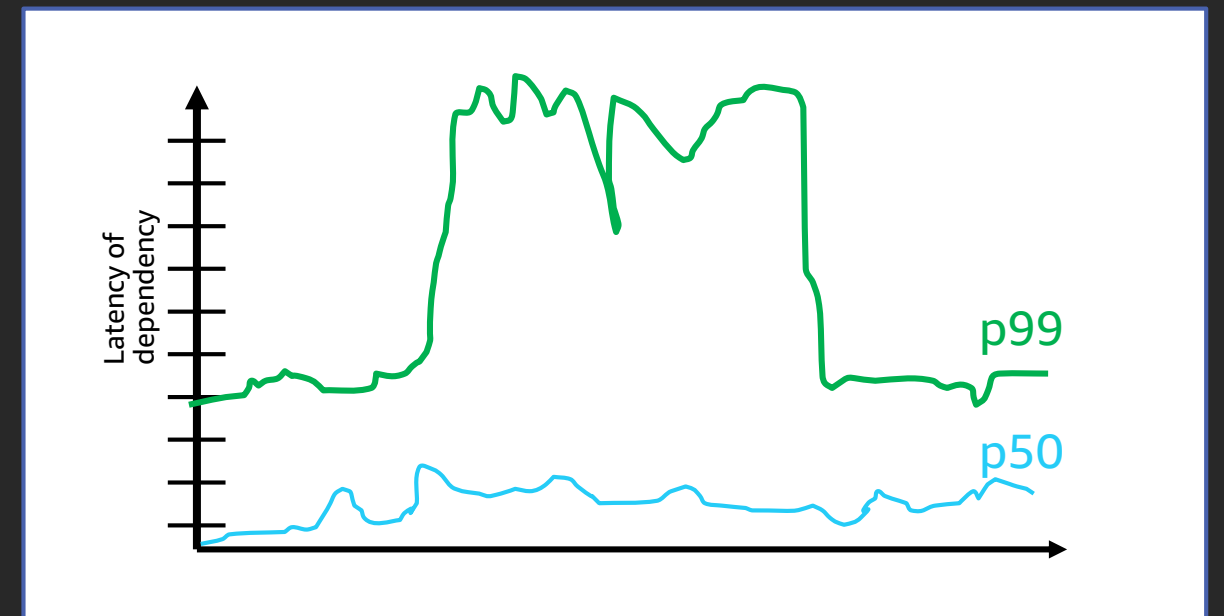
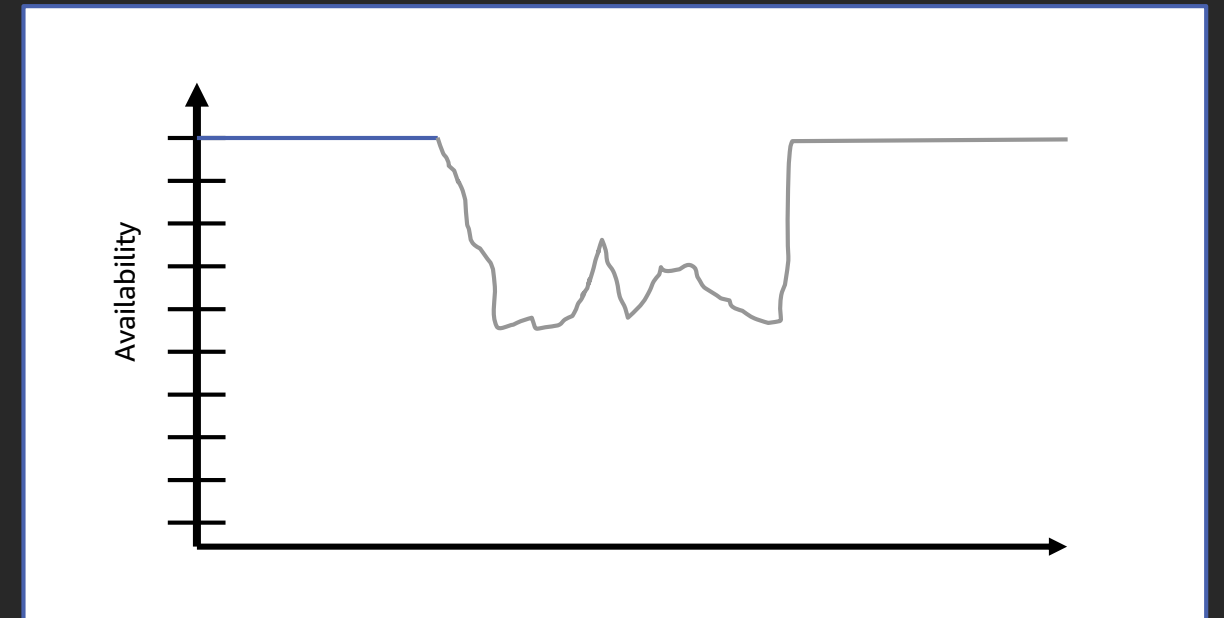
Customer Impact

While there are no metrics available to prove this, we know that there were at least two people who took coffee from the pot. One engineer had a half cup of coffee at his desk. That engineer did not move the pot from the brewing station into its holding spot, so that suggests that at least one other person had coffee. The one confirmed engineer chose to drink the cup of coffee anyway.



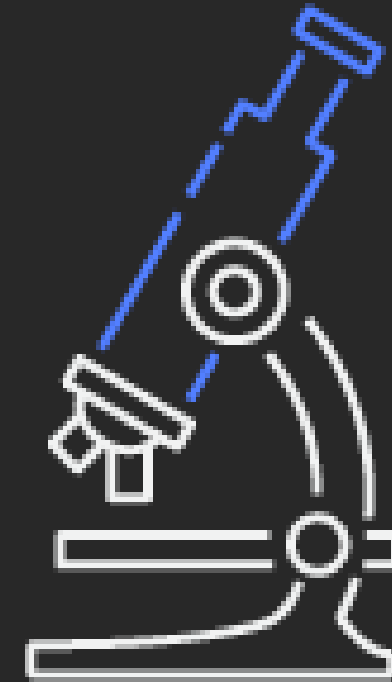
COEs start with the customer and work backward

- **Summary**
 - Narrative description of what happened
- **Metrics and graphs**
 - Primary impact and supporting graphs
 - If they don't exist, that's something to fix
- **Customer impact**
 - How many customers affected
 - What was the impaired experience



Areas of focus

- Root cause: Why? (x5)
- Blast radius: How widespread was the impact?
- Duration: For how long?
- What can others learn?



Toyota's Five-Whys approach to root cause

The vehicle will not start. ← the problem

Why? - The battery is dead. (First why)

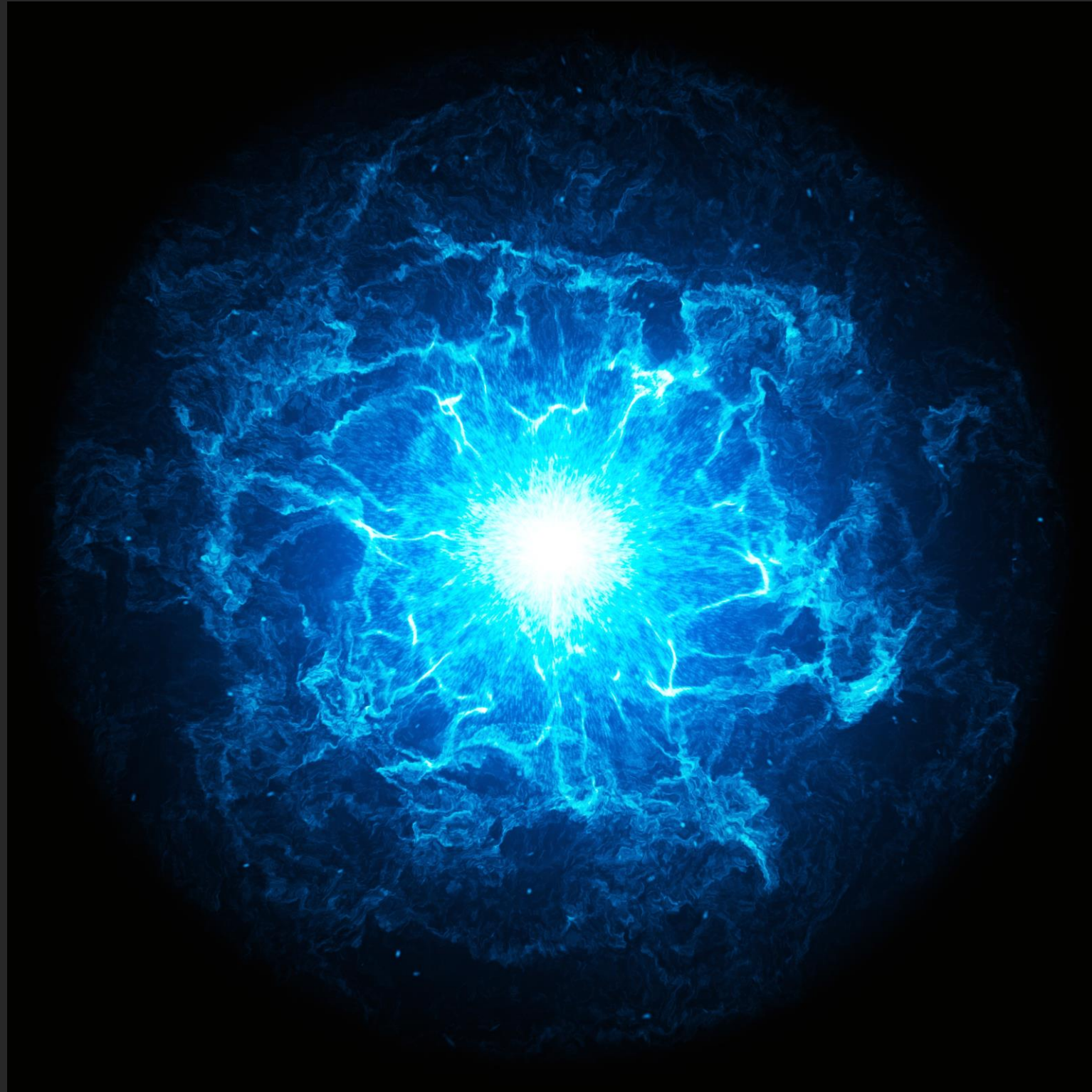
Why? - The alternator is not functioning. (Second why)

Why? - The alternator belt has broken. (Third why)

Why? - The alternator belt was well beyond its useful service life and not replaced. (Fourth why)

Why? - The vehicle was not maintained according to the recommended service schedule. (Fifth why, a root cause)

Blast radius



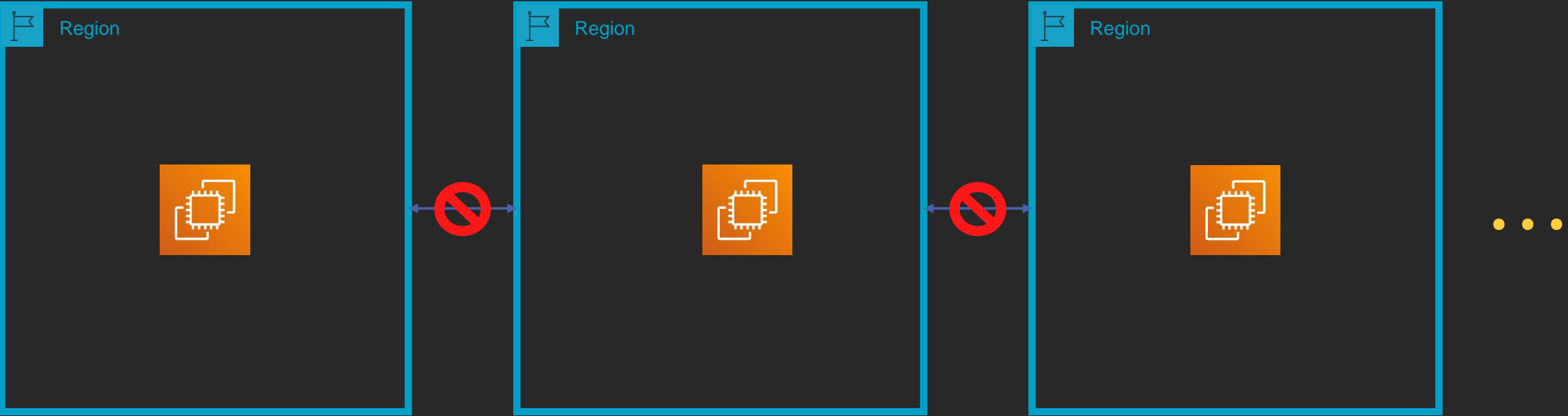
Blast radius



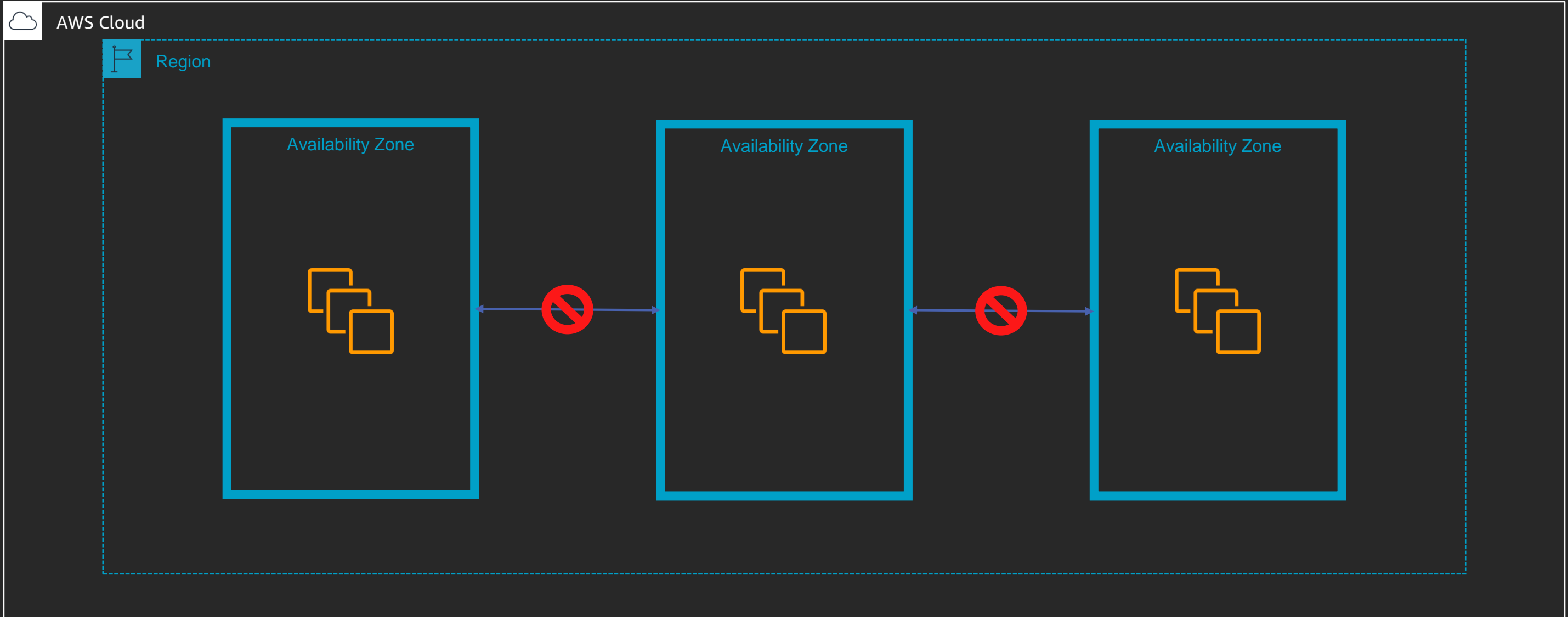
“Consider how blast radius could be reduced. As a thought exercise, how could you cut the blast radius for a similar event in half?”

Blast radius containment as a core design tenet

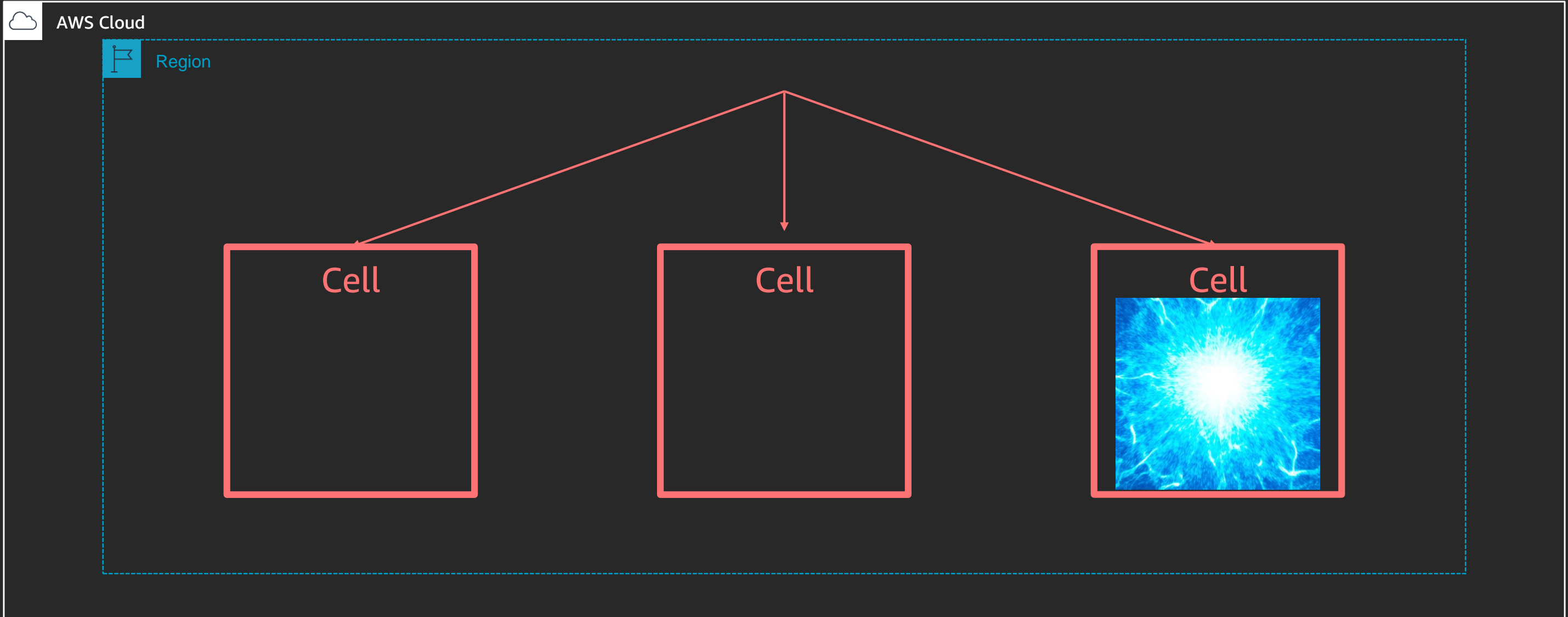
AWS Cloud



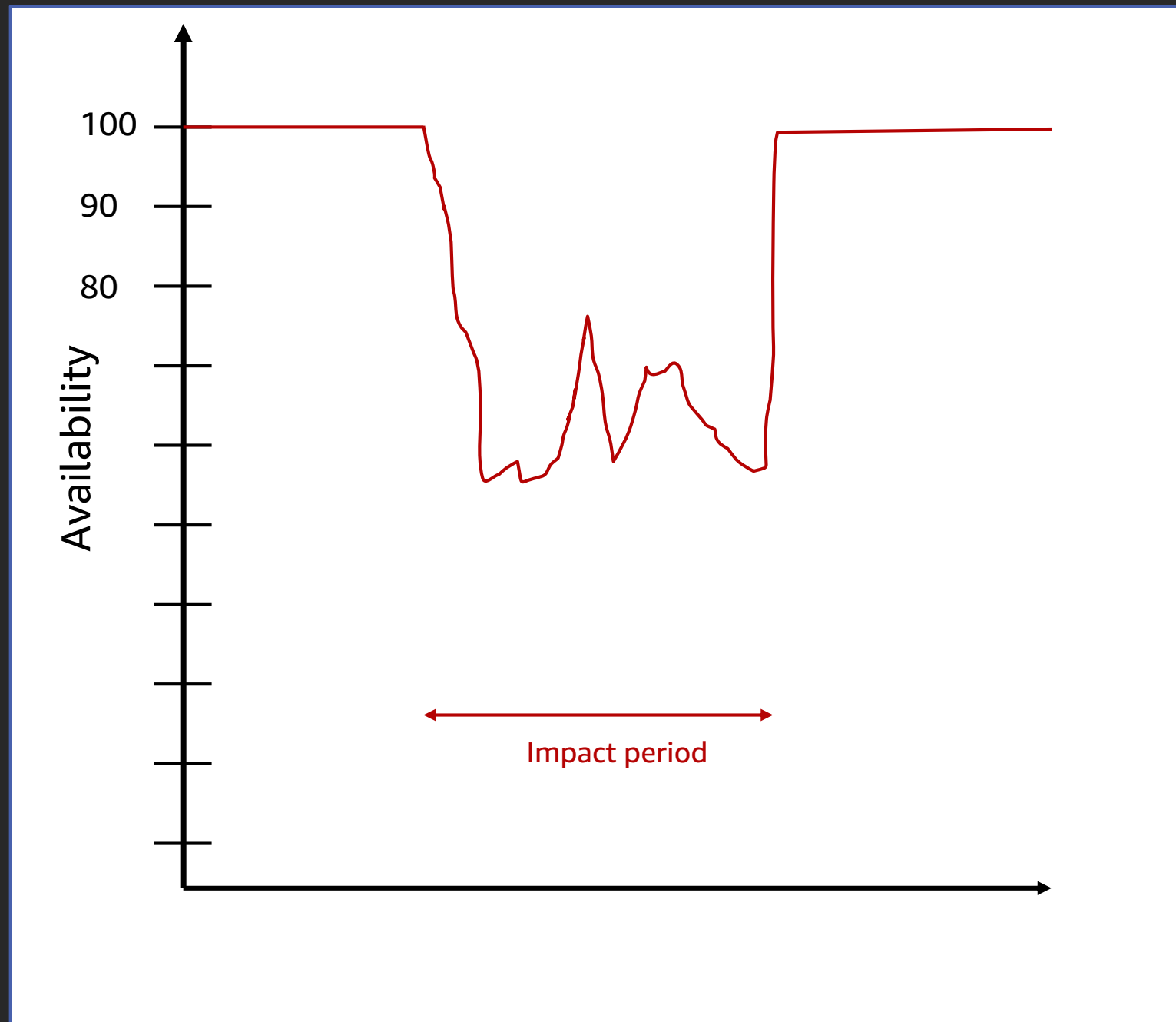
Blast radius containment as a core design tenet



Blast radius containment as a core design tenet

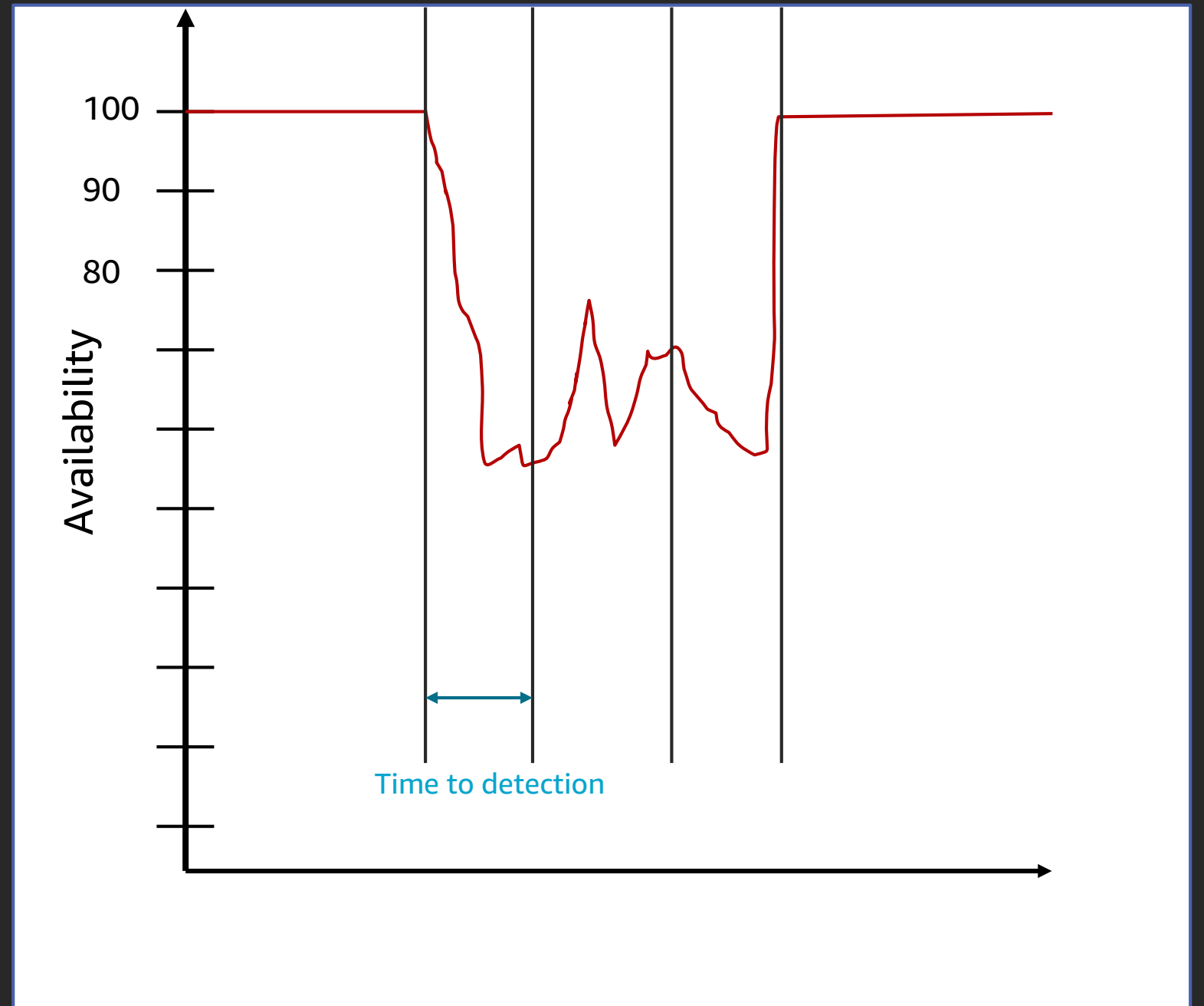


Controlling event duration



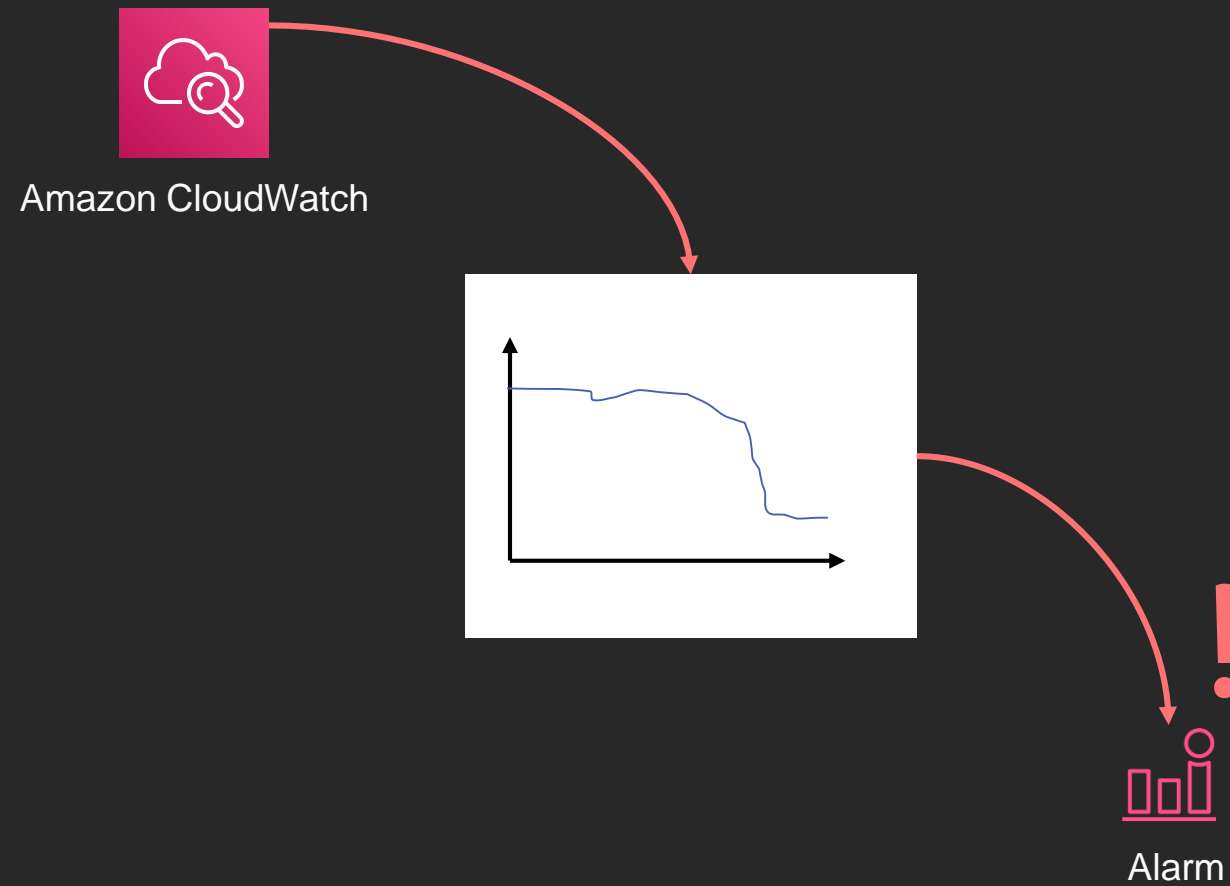
Controlling duration: Improving incident response

- “How was the event detected?”
- “How could time to detection be improved? As a thought experiment, how would you have cut the time in half?”



Controlling duration: Improving incident response

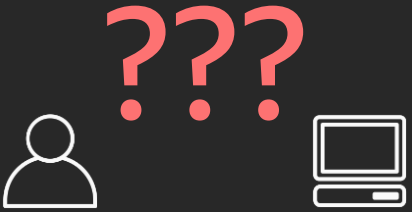
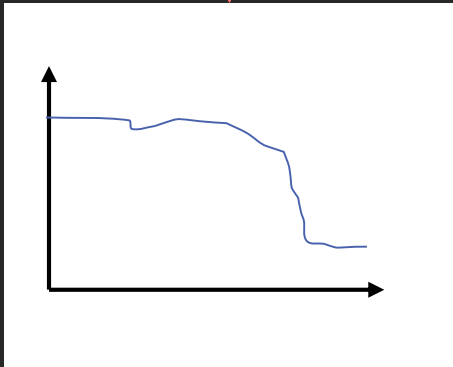
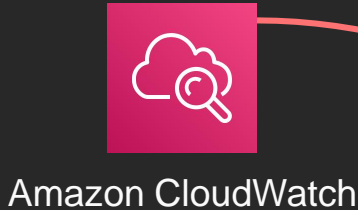
Good



Controlling duration: Improving incident response

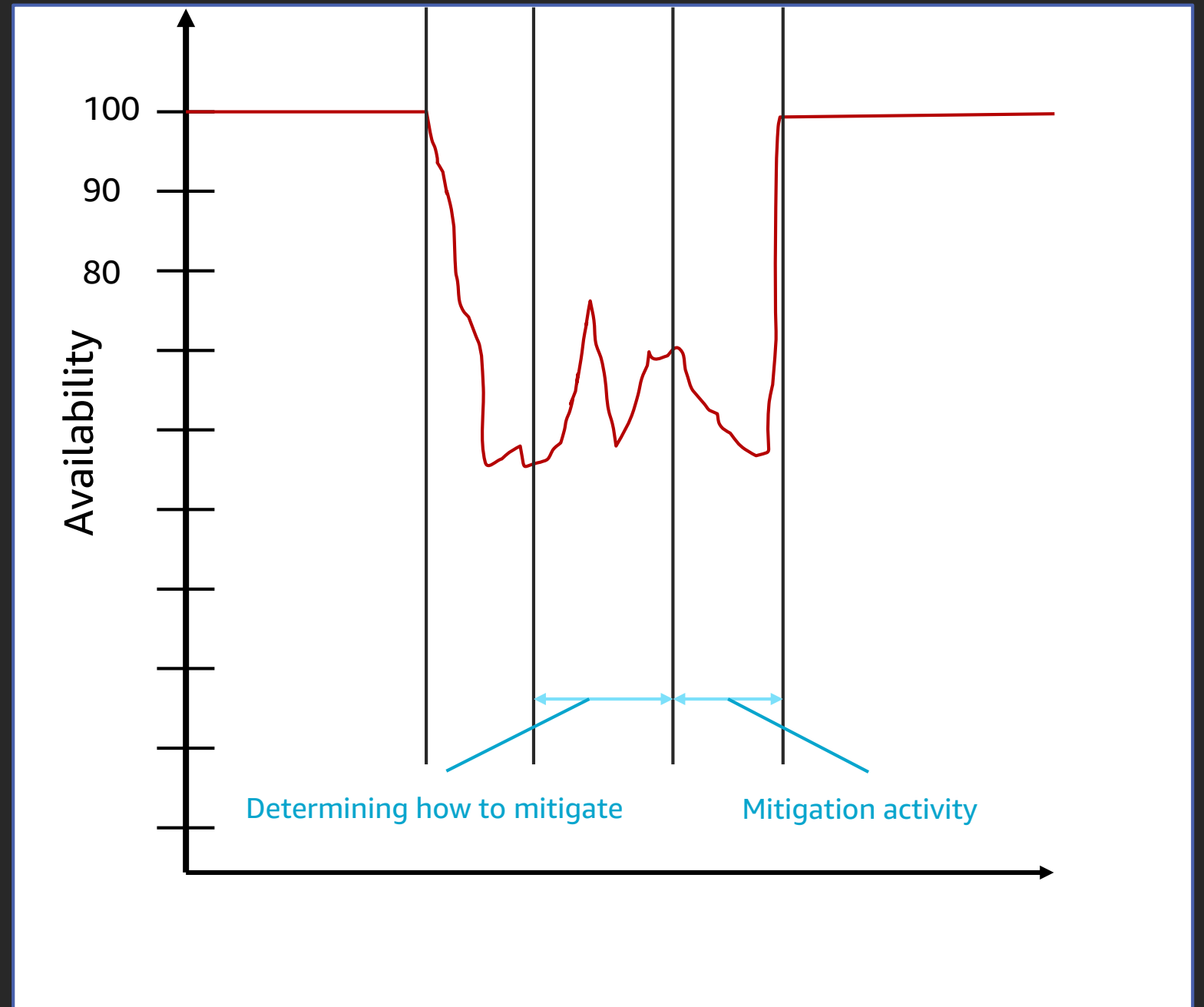
Good

Bad



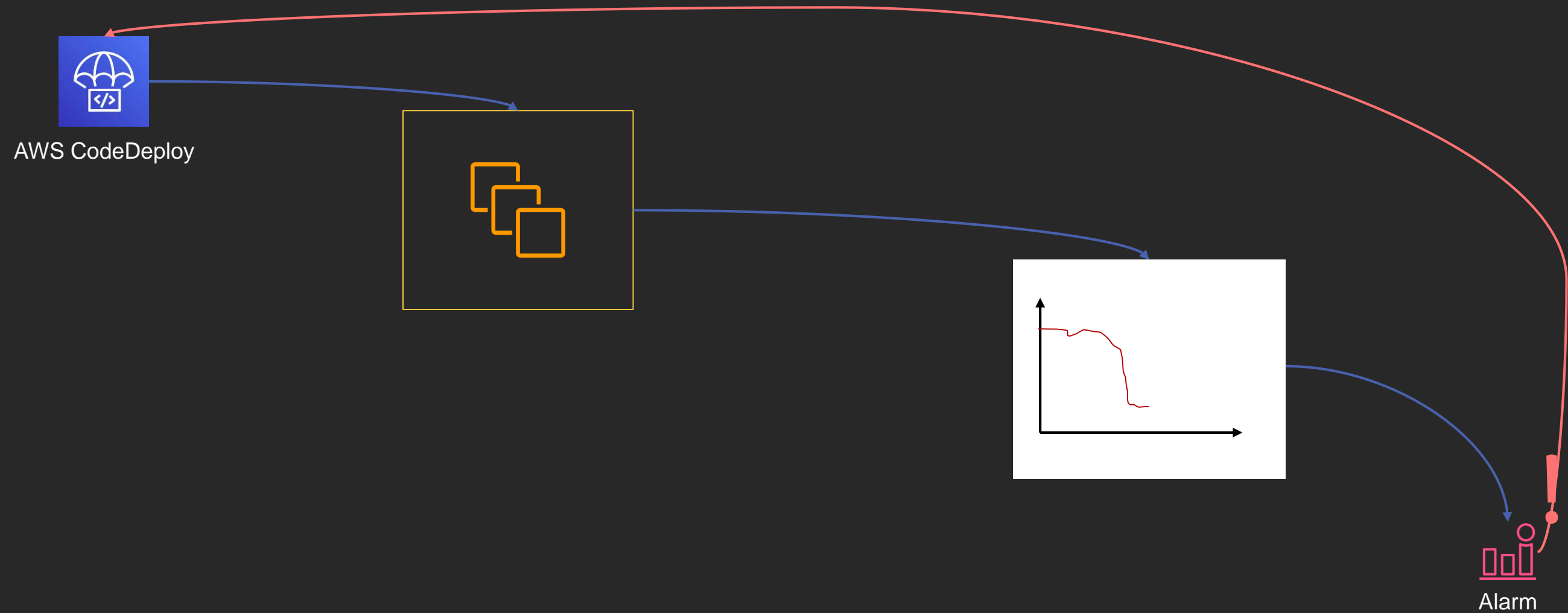
Controlling duration: Improving time to mitigation

- “How did you reach the point where you knew how to mitigate the impact?”
- “How could time to mitigation be improved? As a thought experiment, how would you have cut the time in half?”



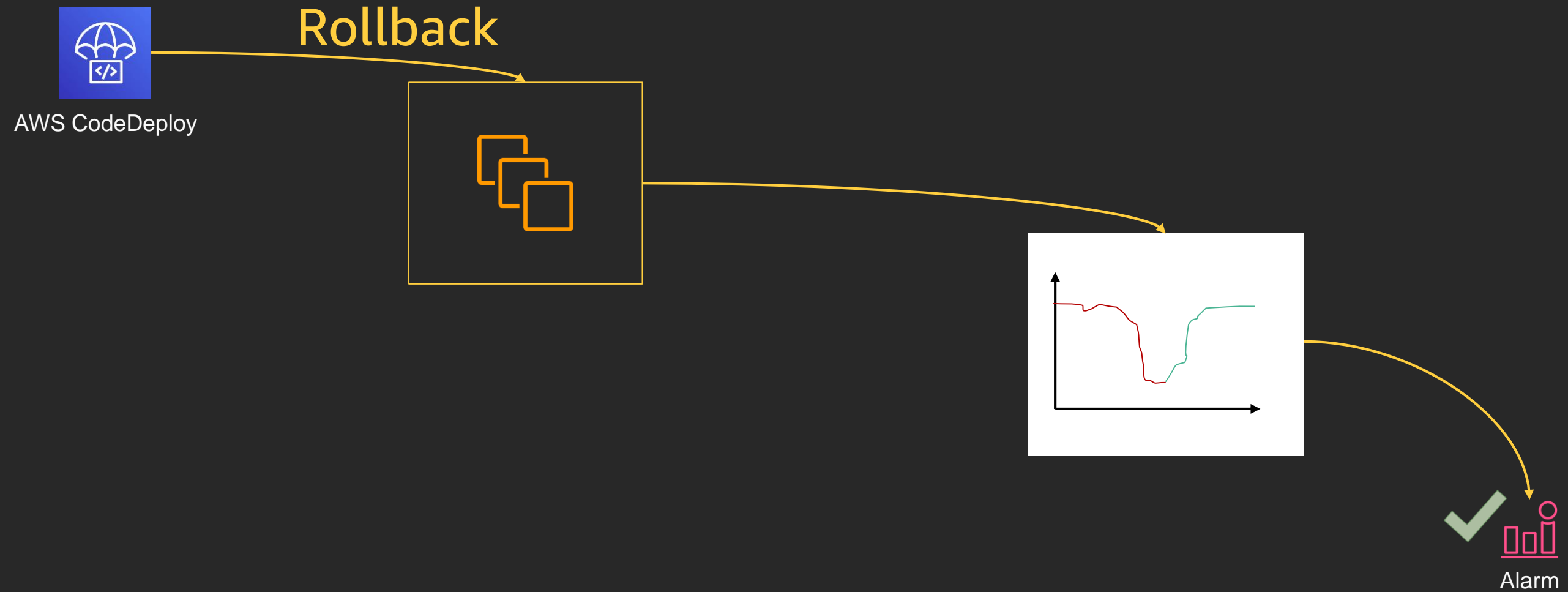
Controlling duration: Improving time to mitigation

Example: Alarm-based automatic rollback

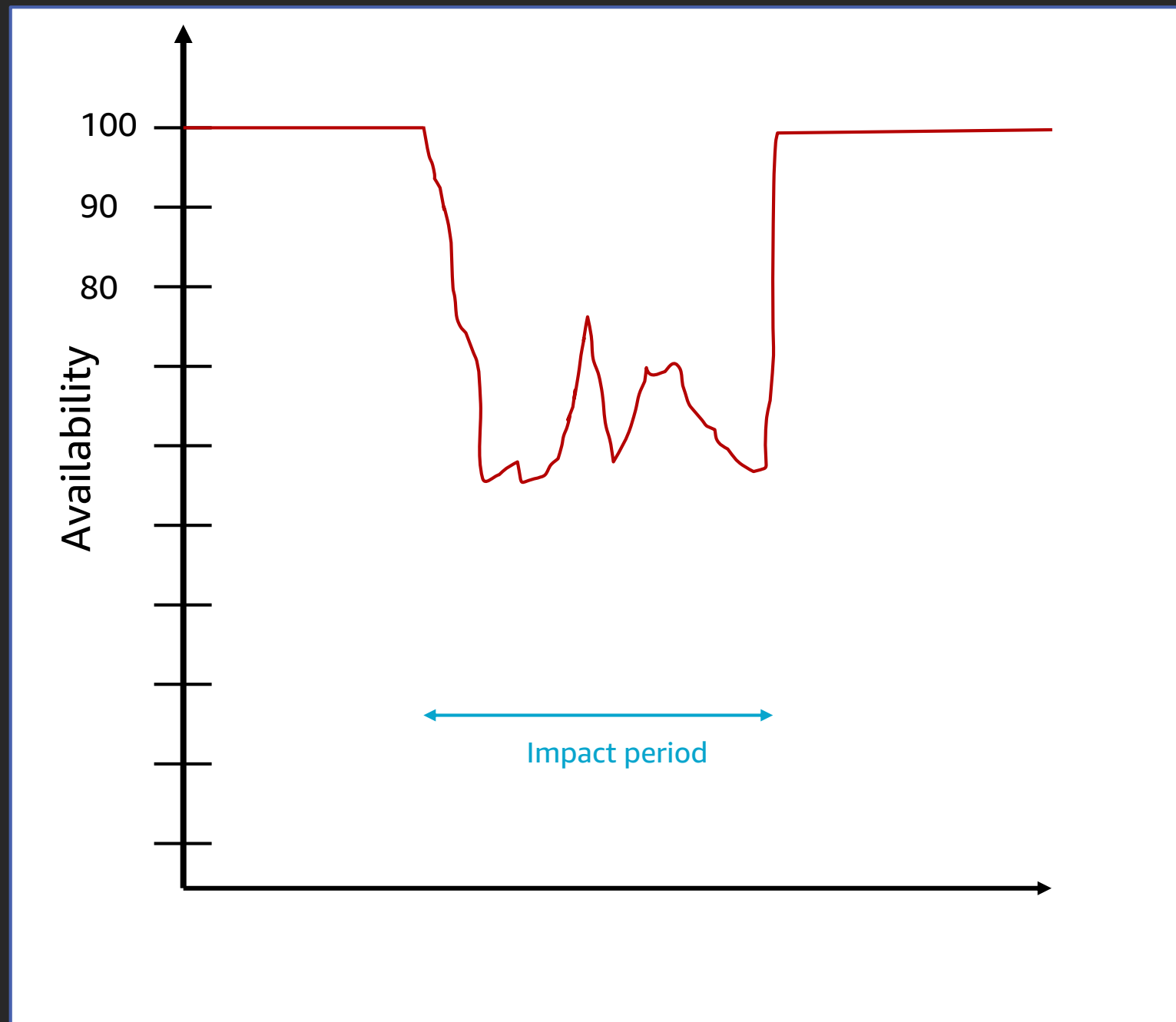


Controlling duration: Improving time to mitigation

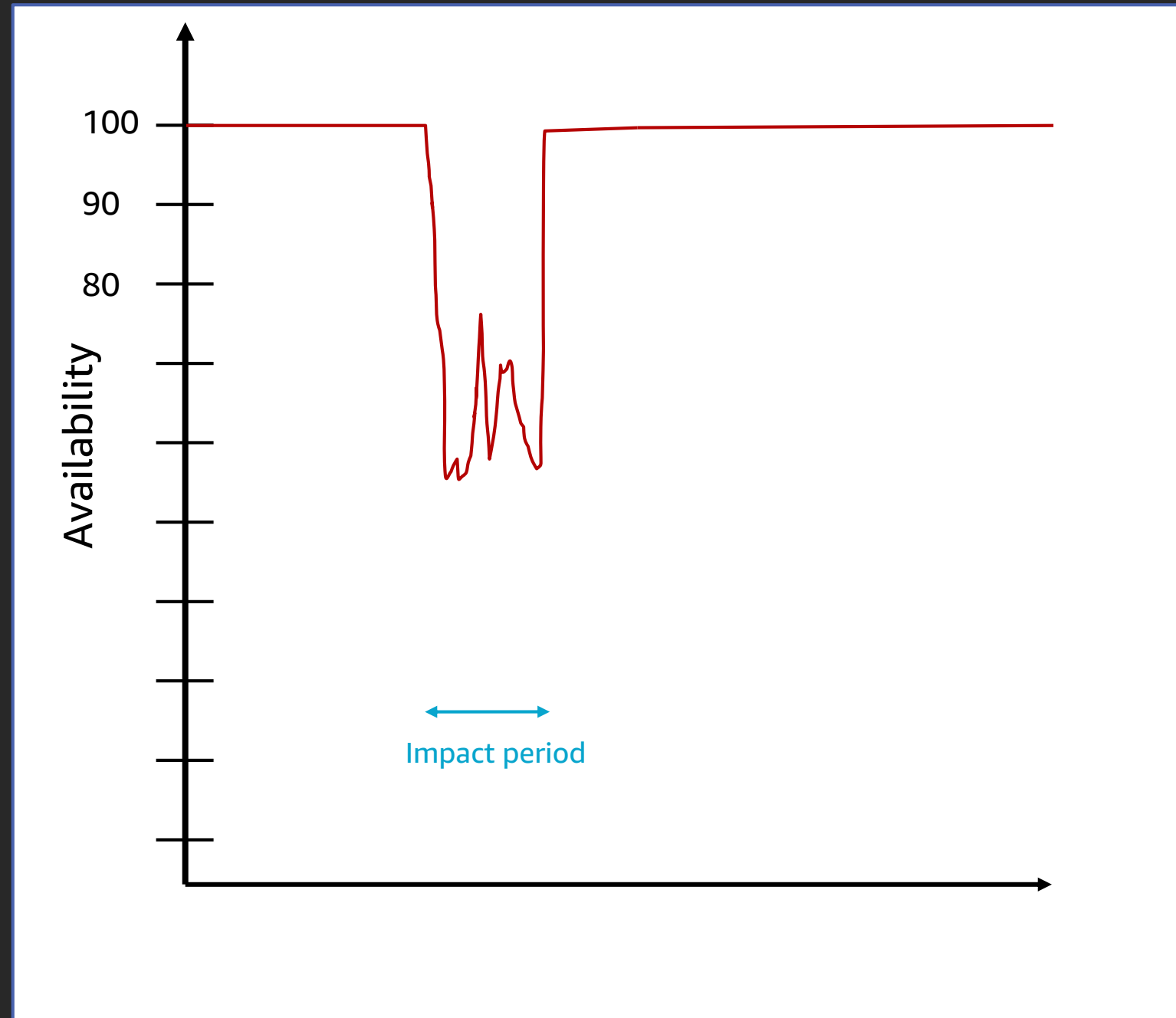
Example: Alarm-based automatic rollback



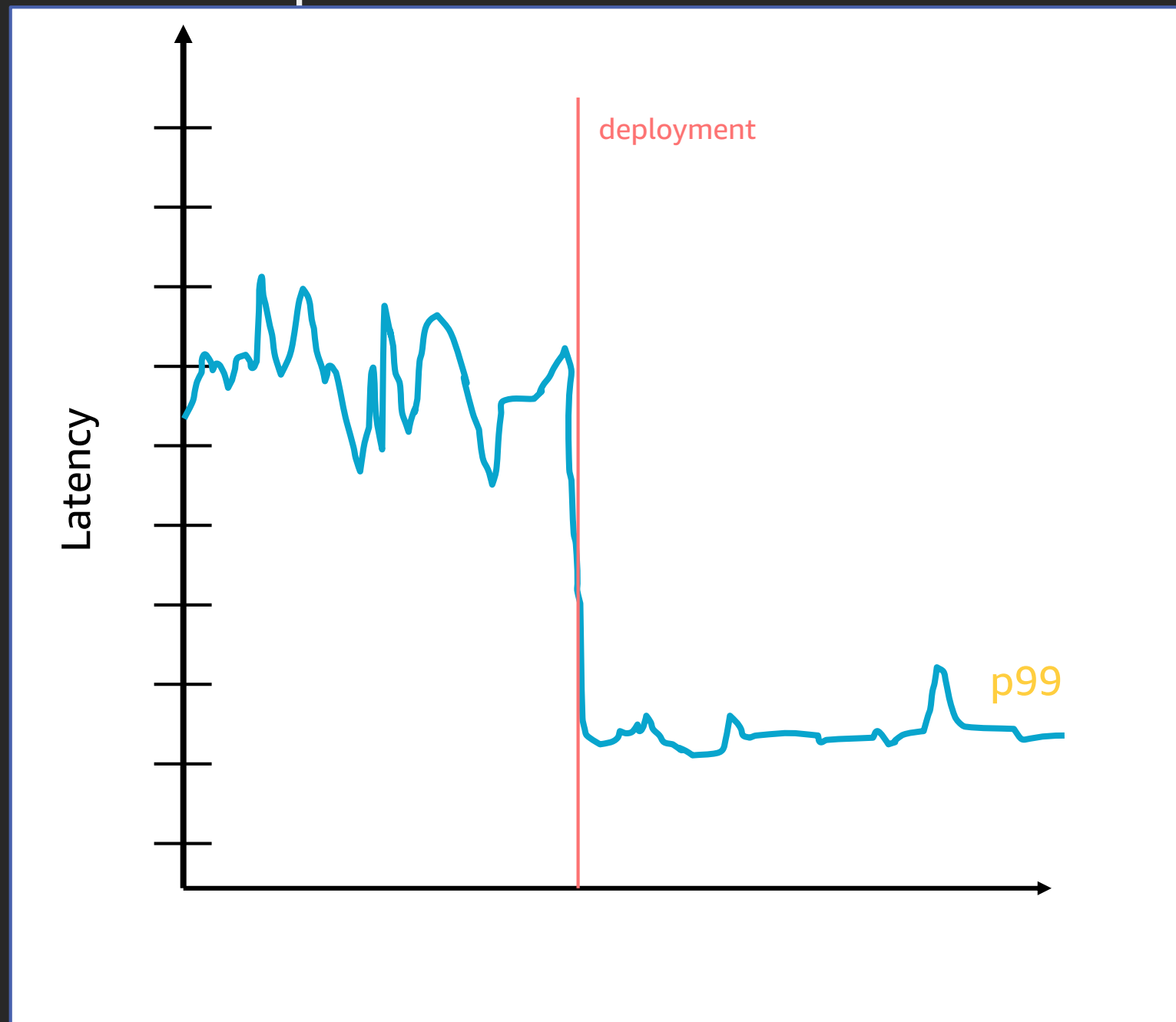
Controlling event duration



Controlling event duration



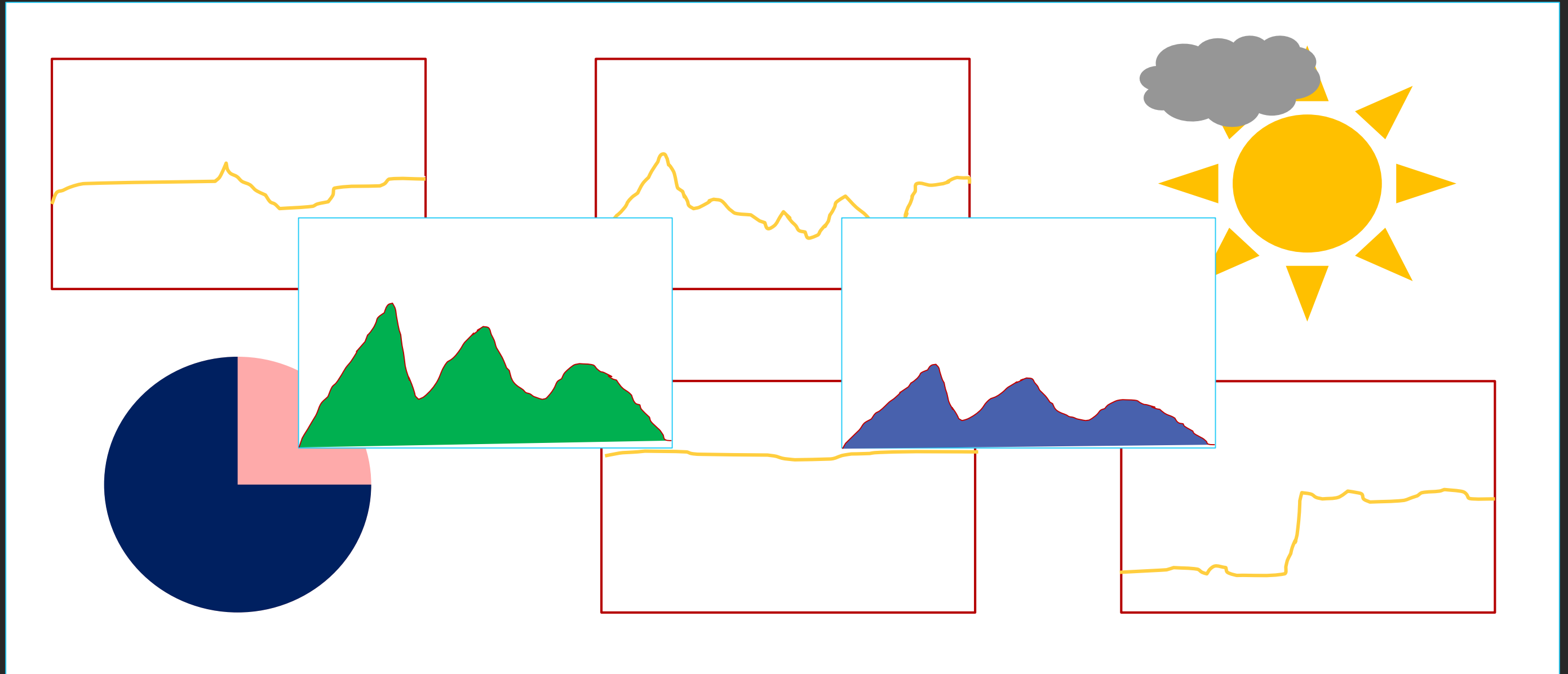
Wins are just as important as failures



Watching yourself fail (before your customers do)



Metrics are very interesting, and that can be a problem



Health metrics and diagnostic metrics

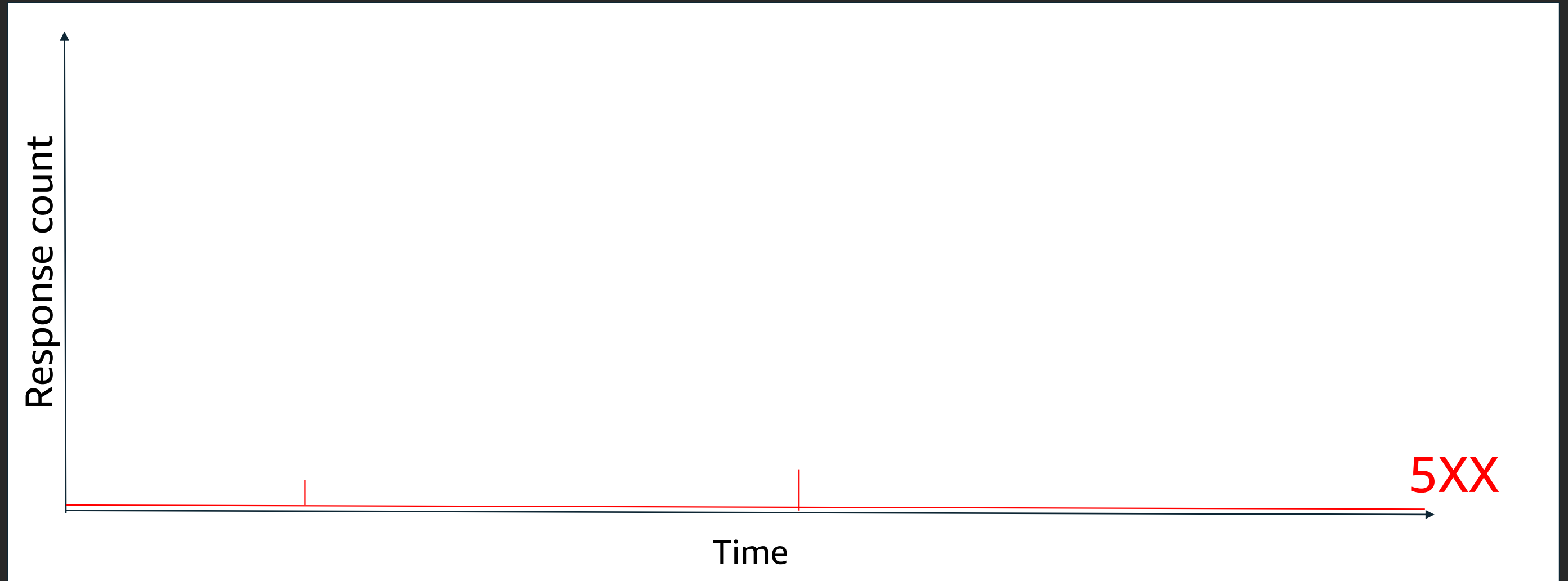
Health metrics

- *Answers* the question: Am I failing?
- *Does not answer* the question: Why am I failing?
- *Always* set alarms on these
- Be conservative in defining

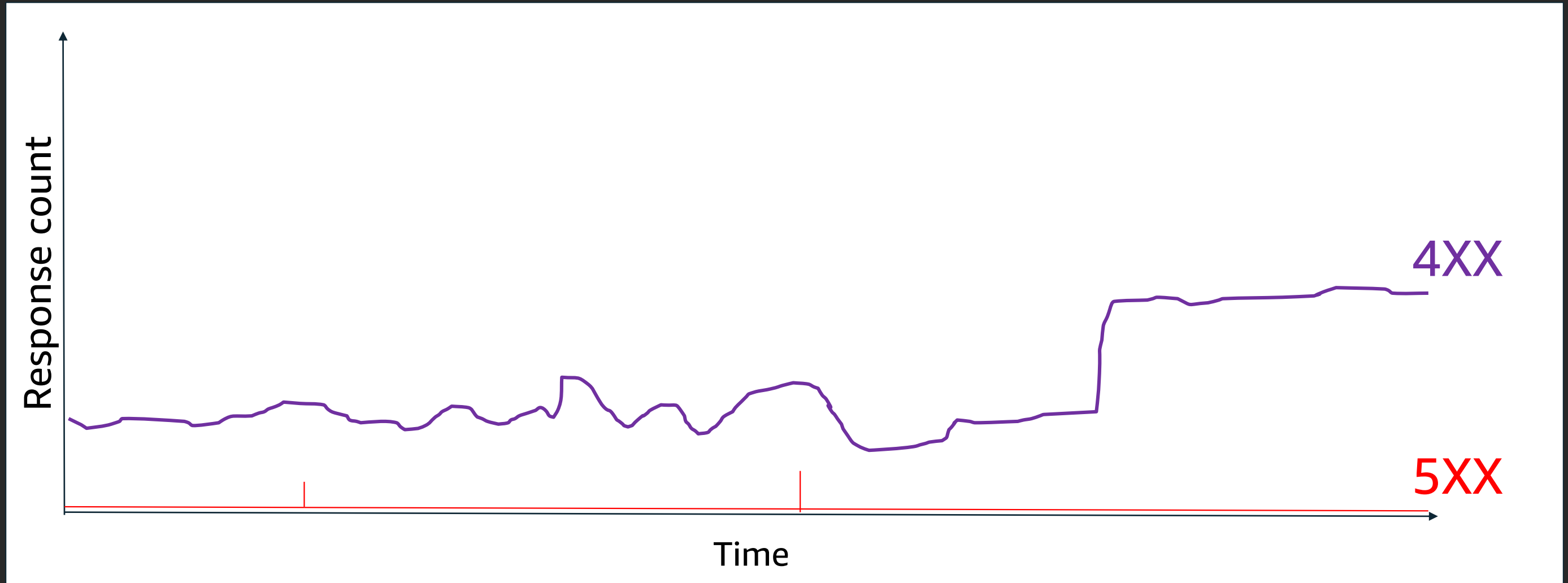
Diagnostic metrics

- *Answers* the question: What is the value of this thing I measured?
- *Might answer* the question: Why isn't my system working?
- *Sometimes* set alarms on these
- Be liberal in defining

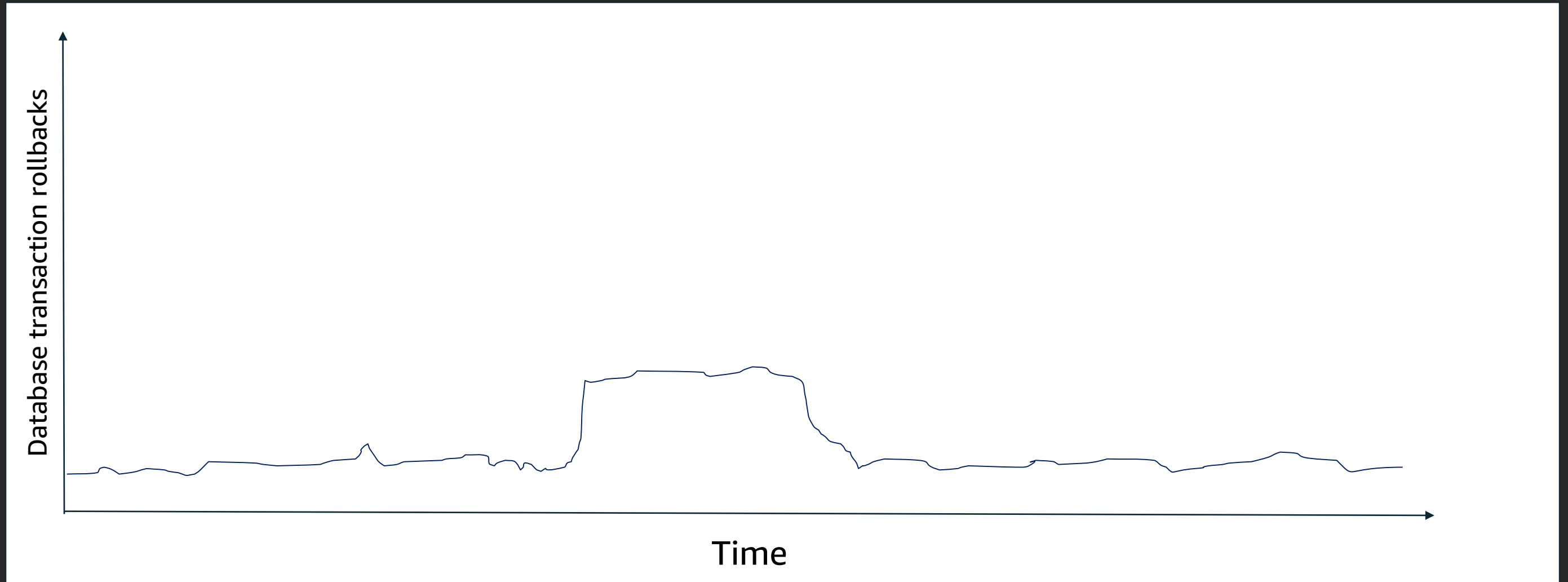
Health or diagnostic?



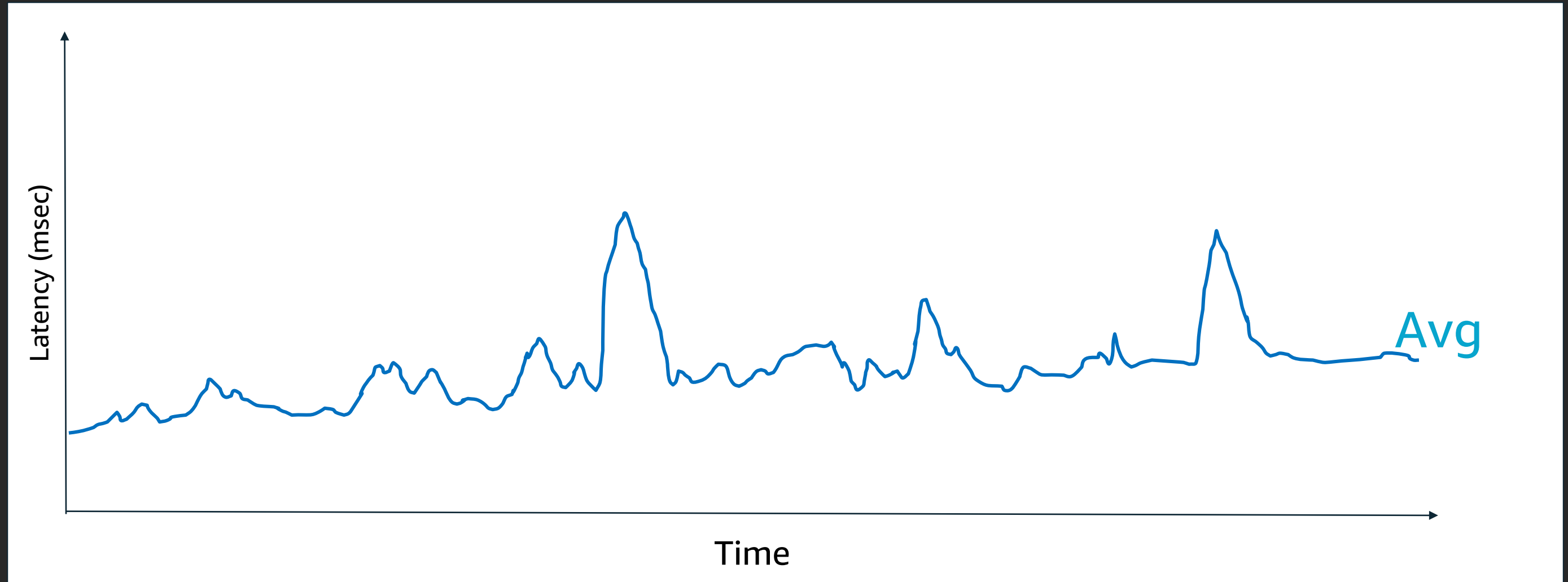
Health or diagnostic?



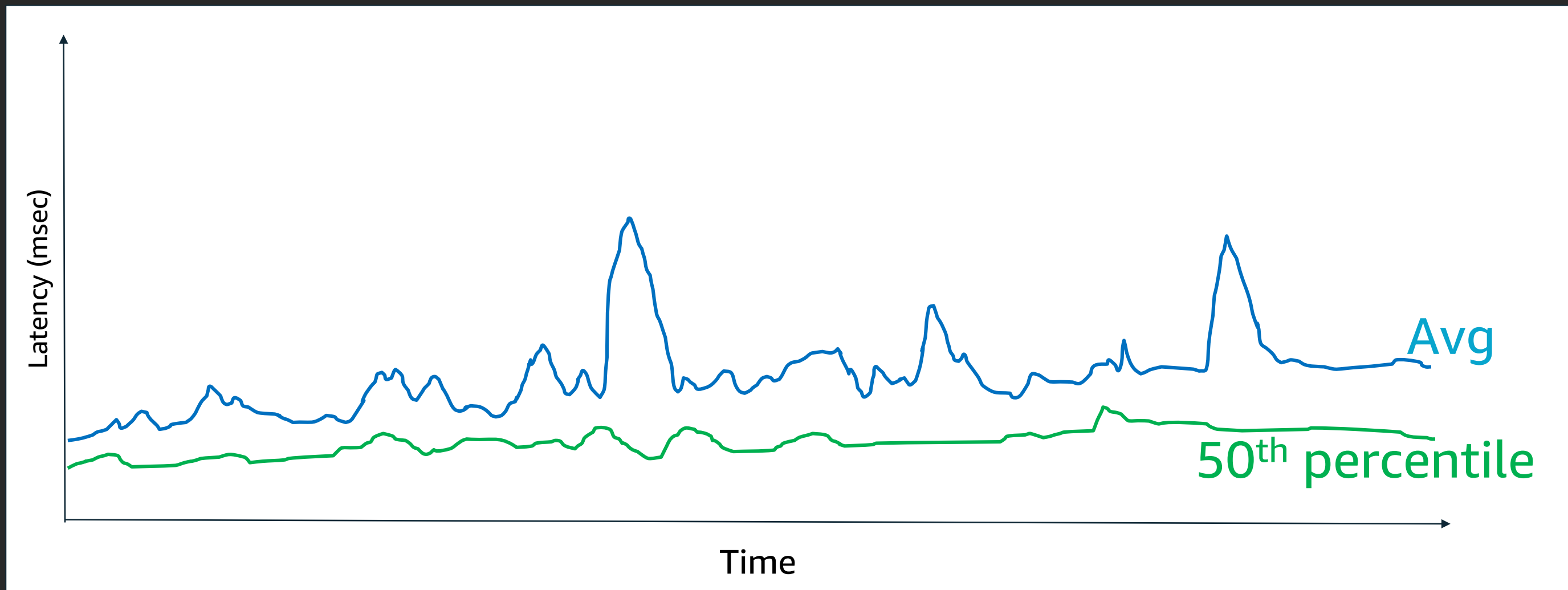
Health or diagnostic?



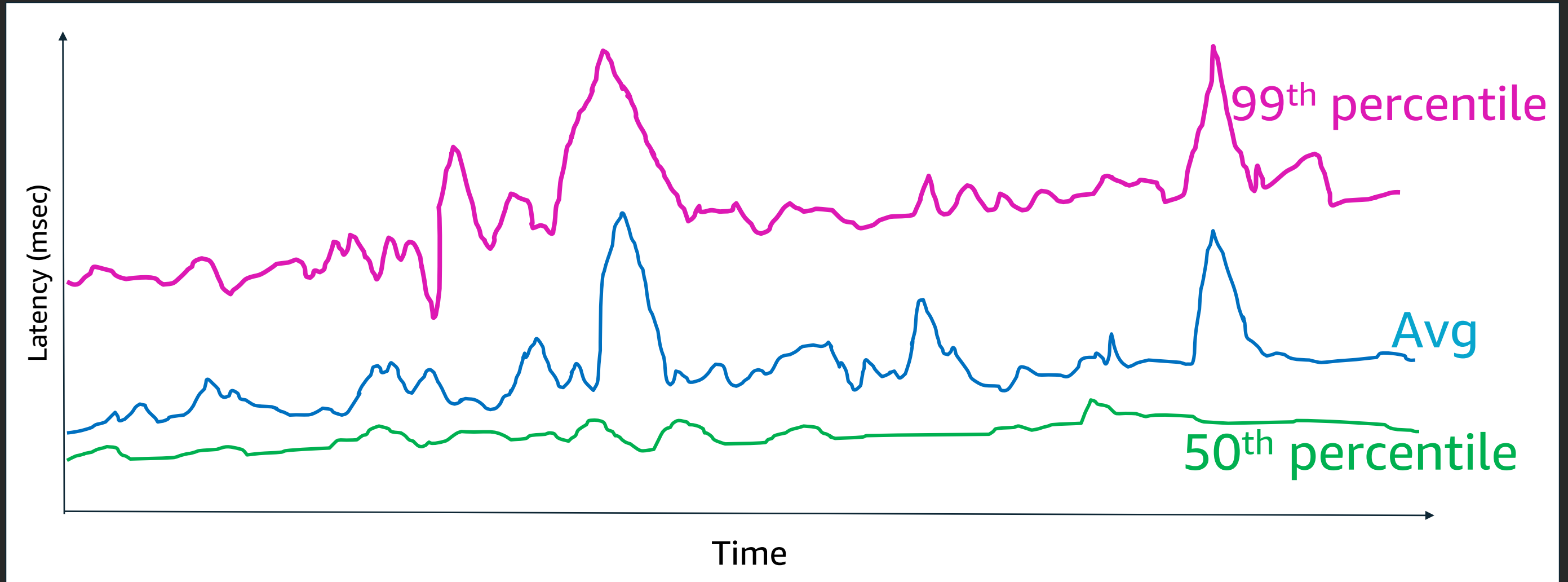
Health or diagnostic?



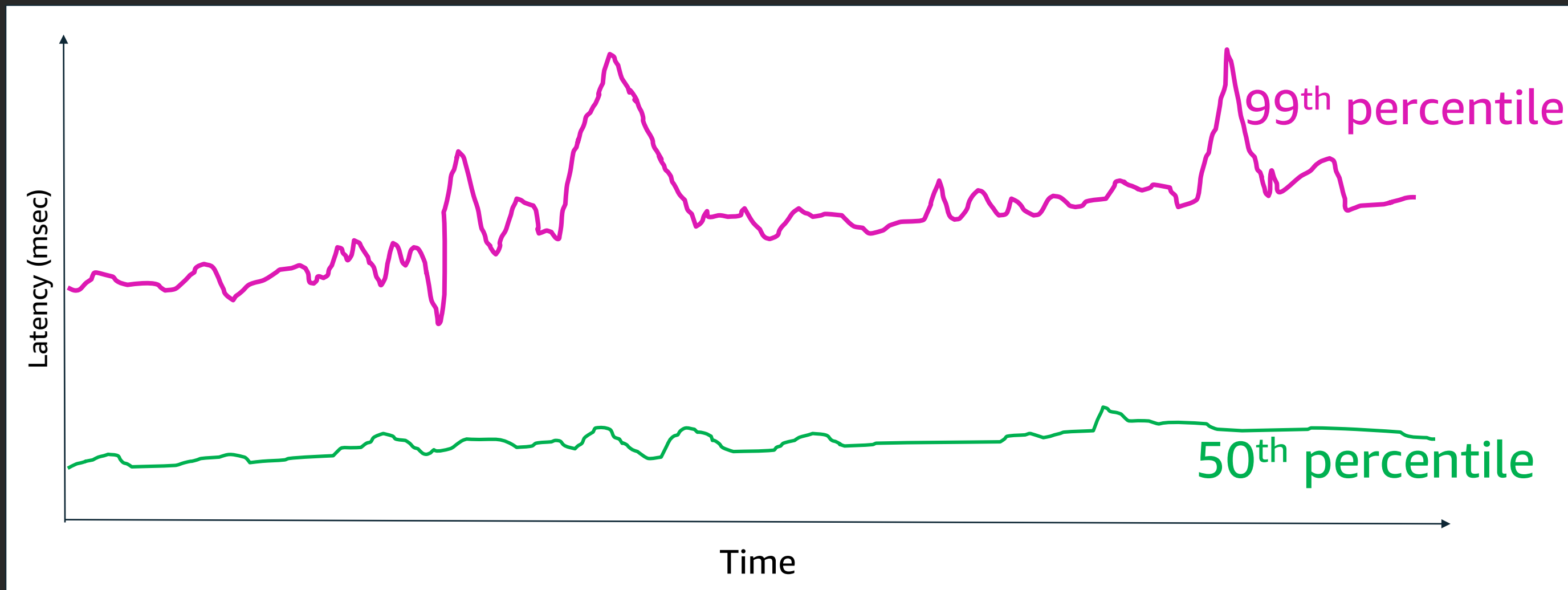
Health or diagnostic?



Health or diagnostic?



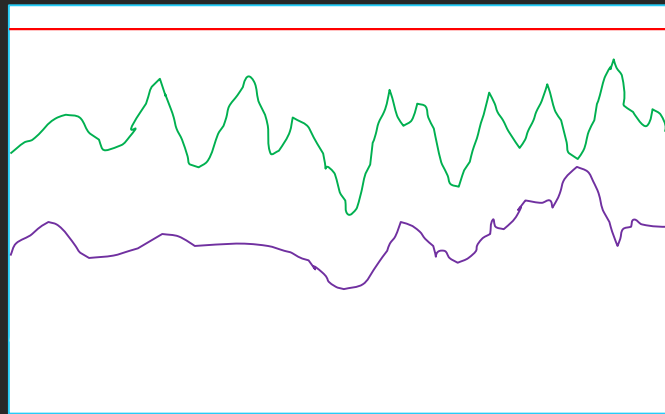
Percentiles >> Avg



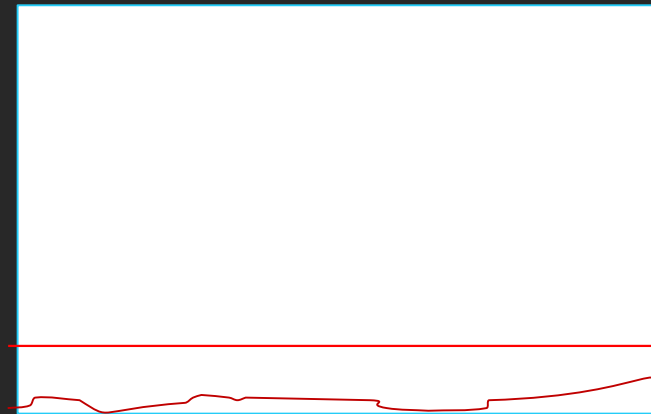
Layout of a great dashboard

Health metrics at the top

Latency percentiles



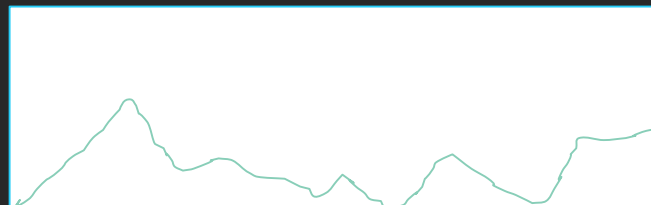
Faults



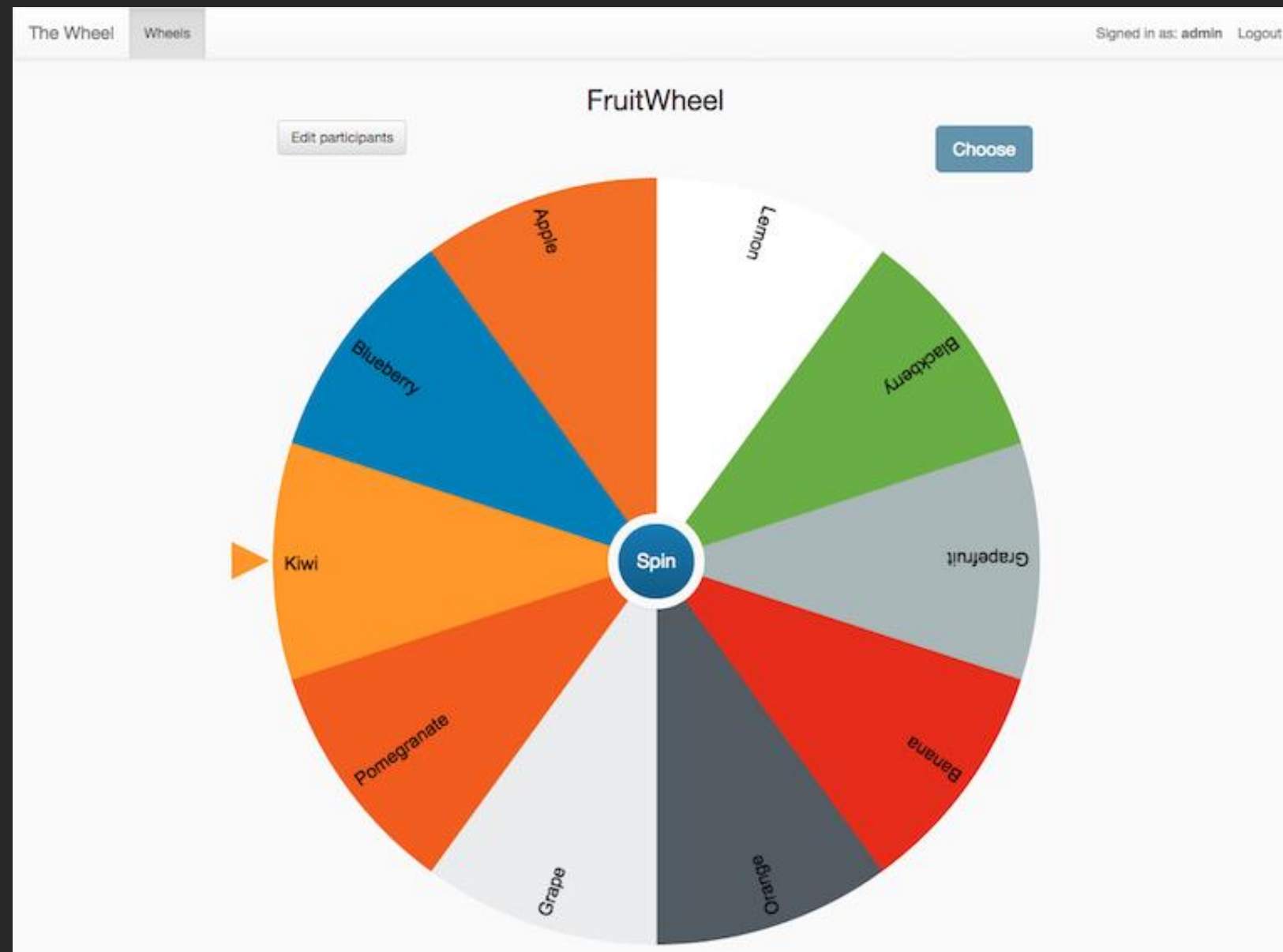
Volume



Key diagnostic metrics below the fold

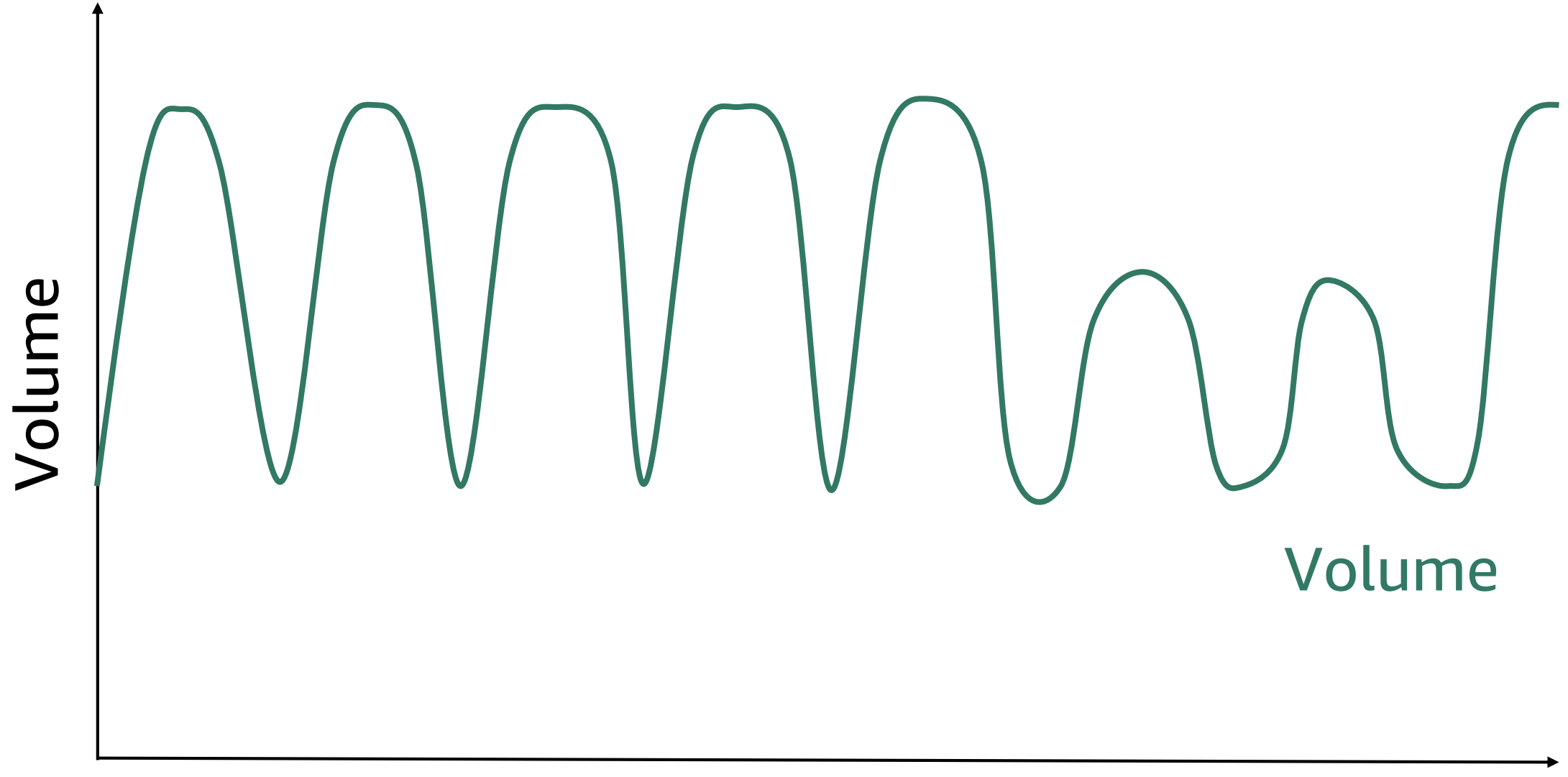


The AWS Ops Wheel



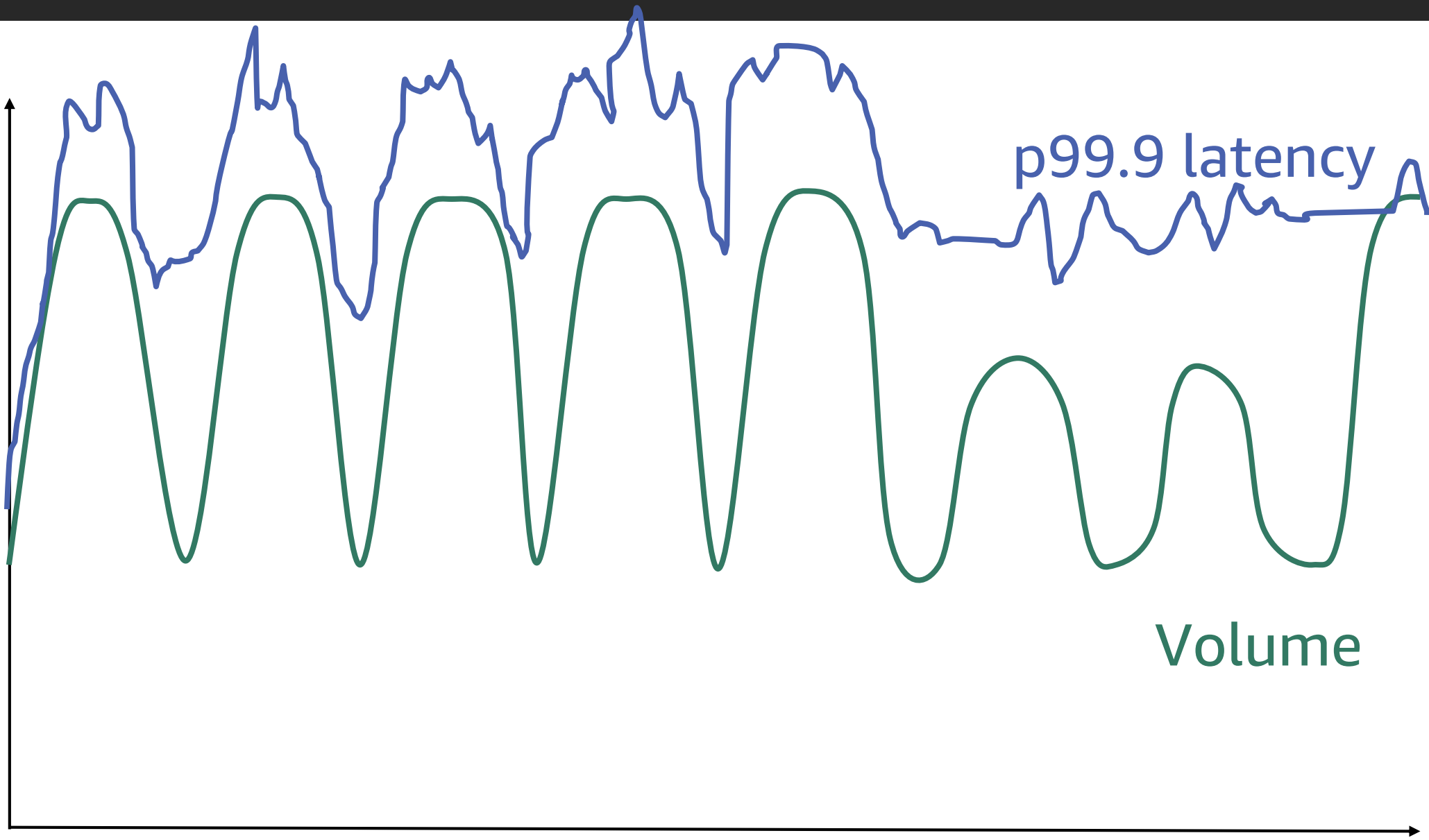
Looking for failure in your metrics





Time scale: ~one week

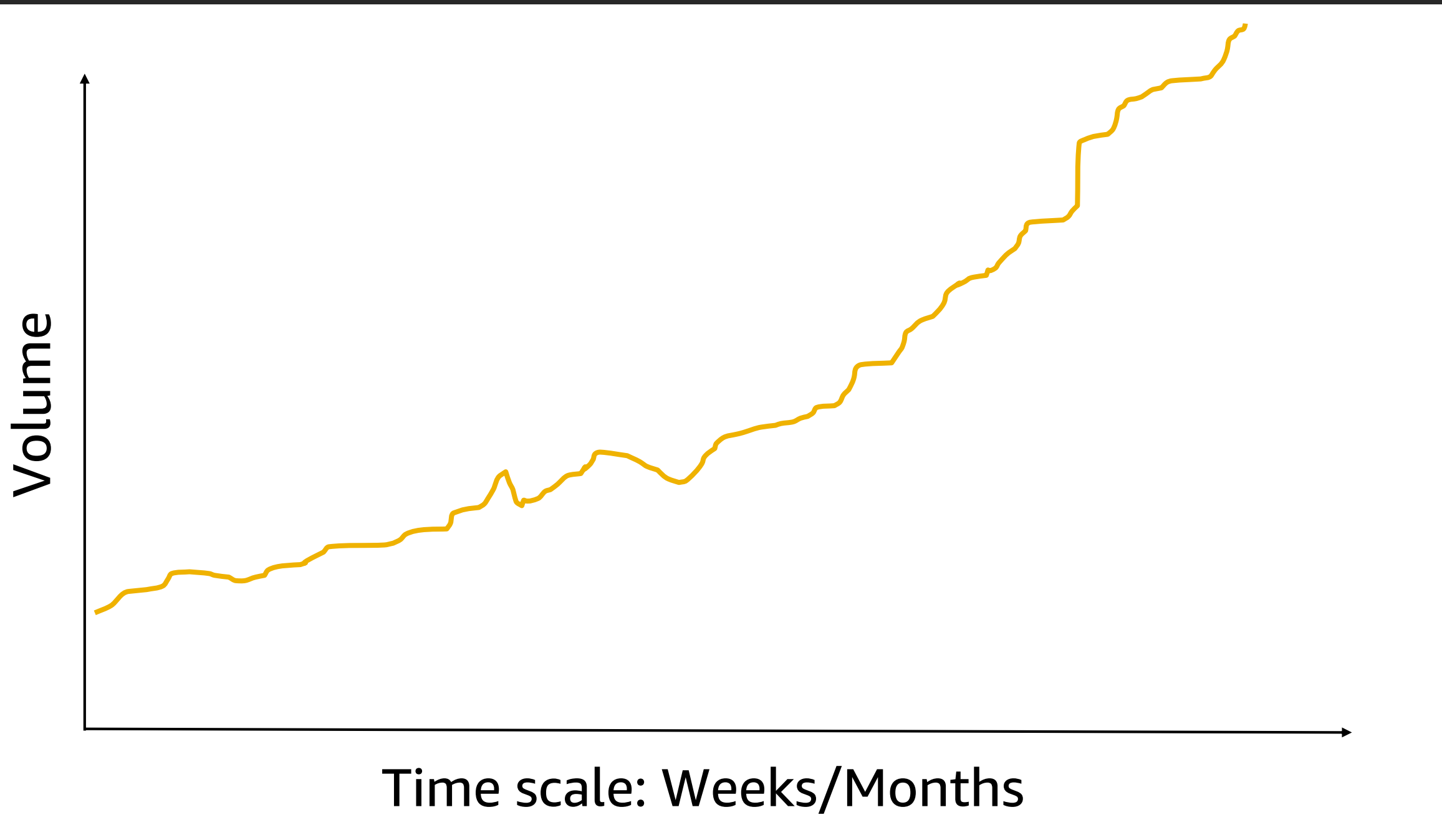
Volume

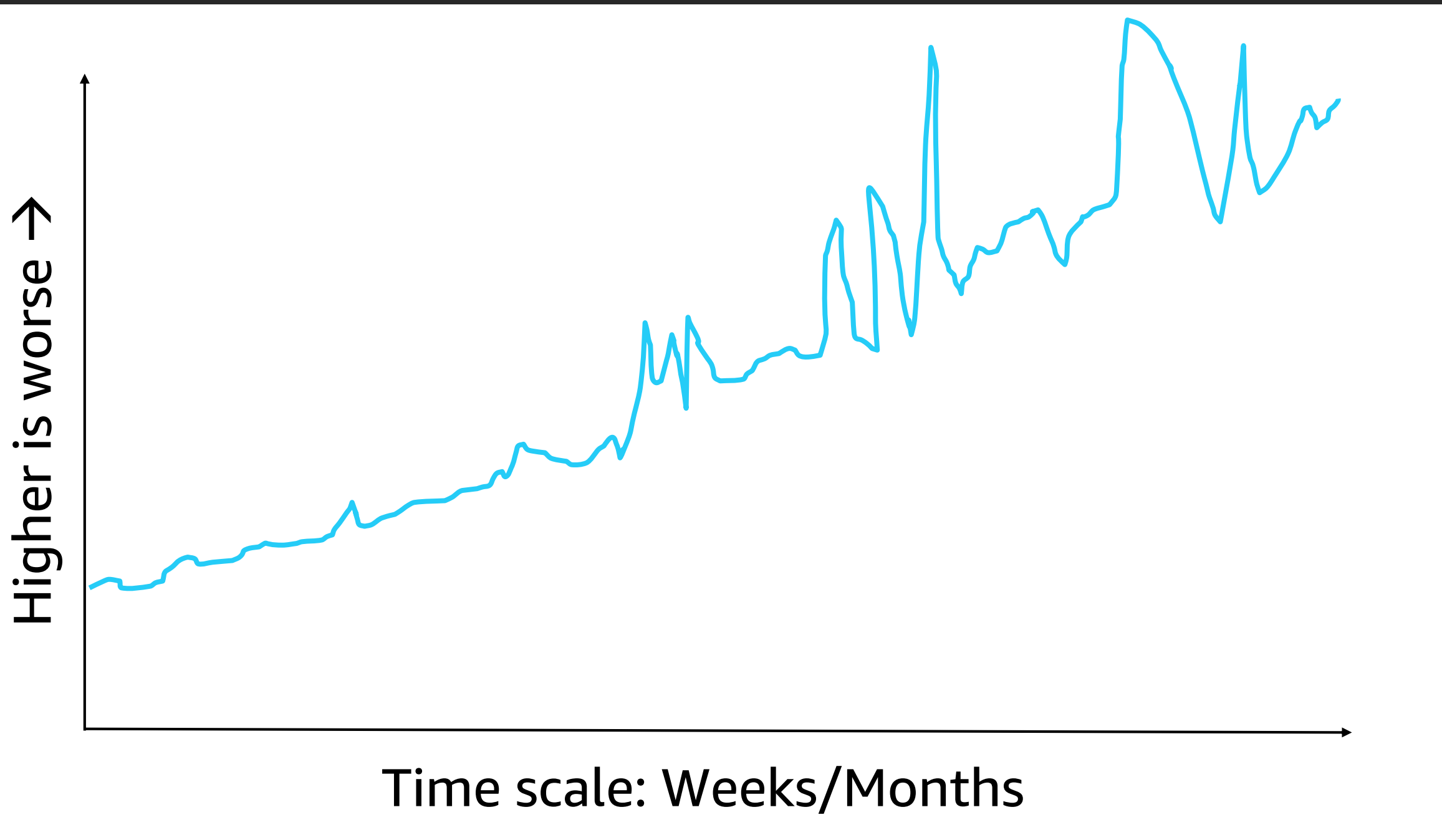


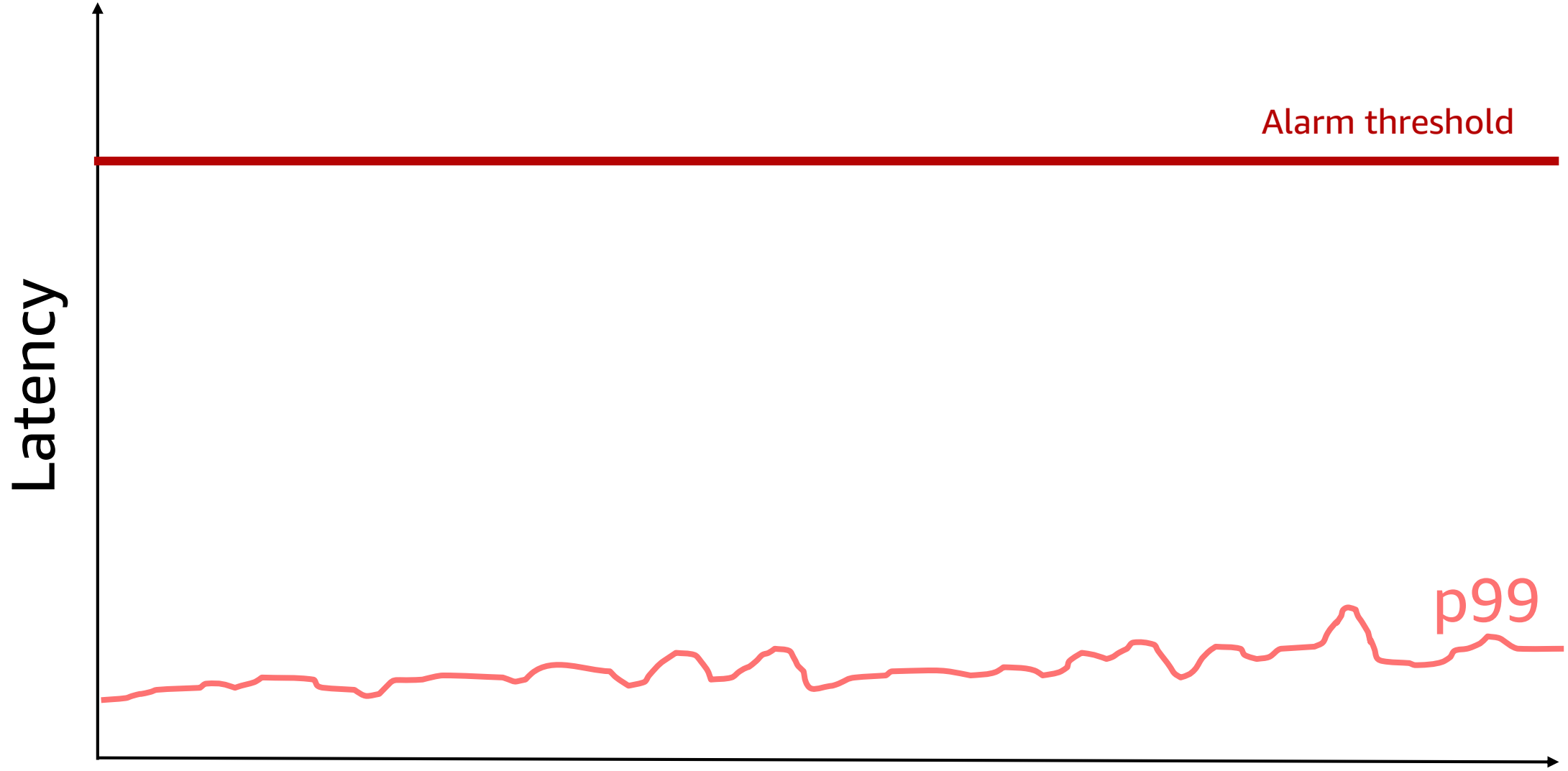
Time scale: ~one week

Volume

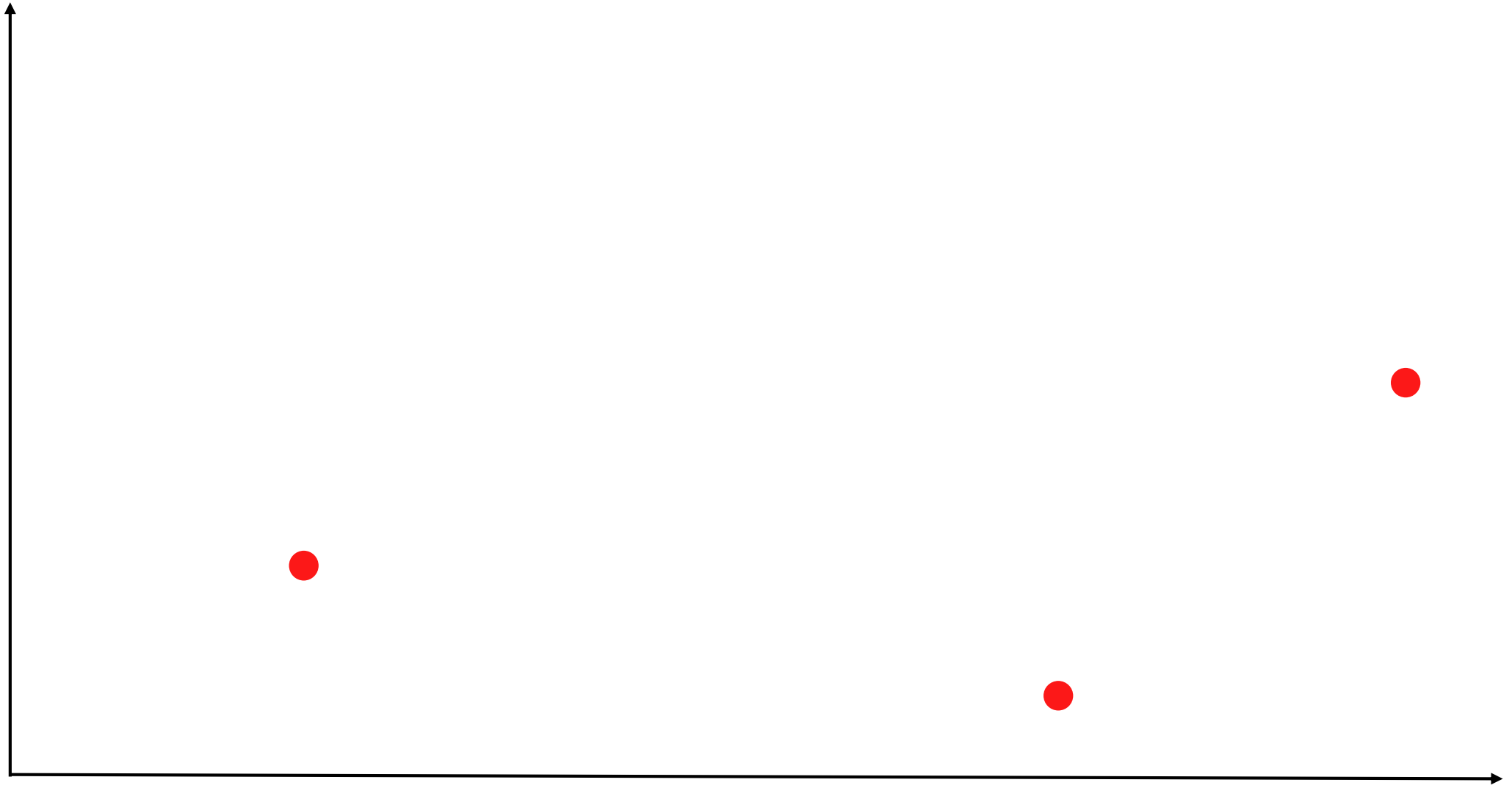
p99.9 latency



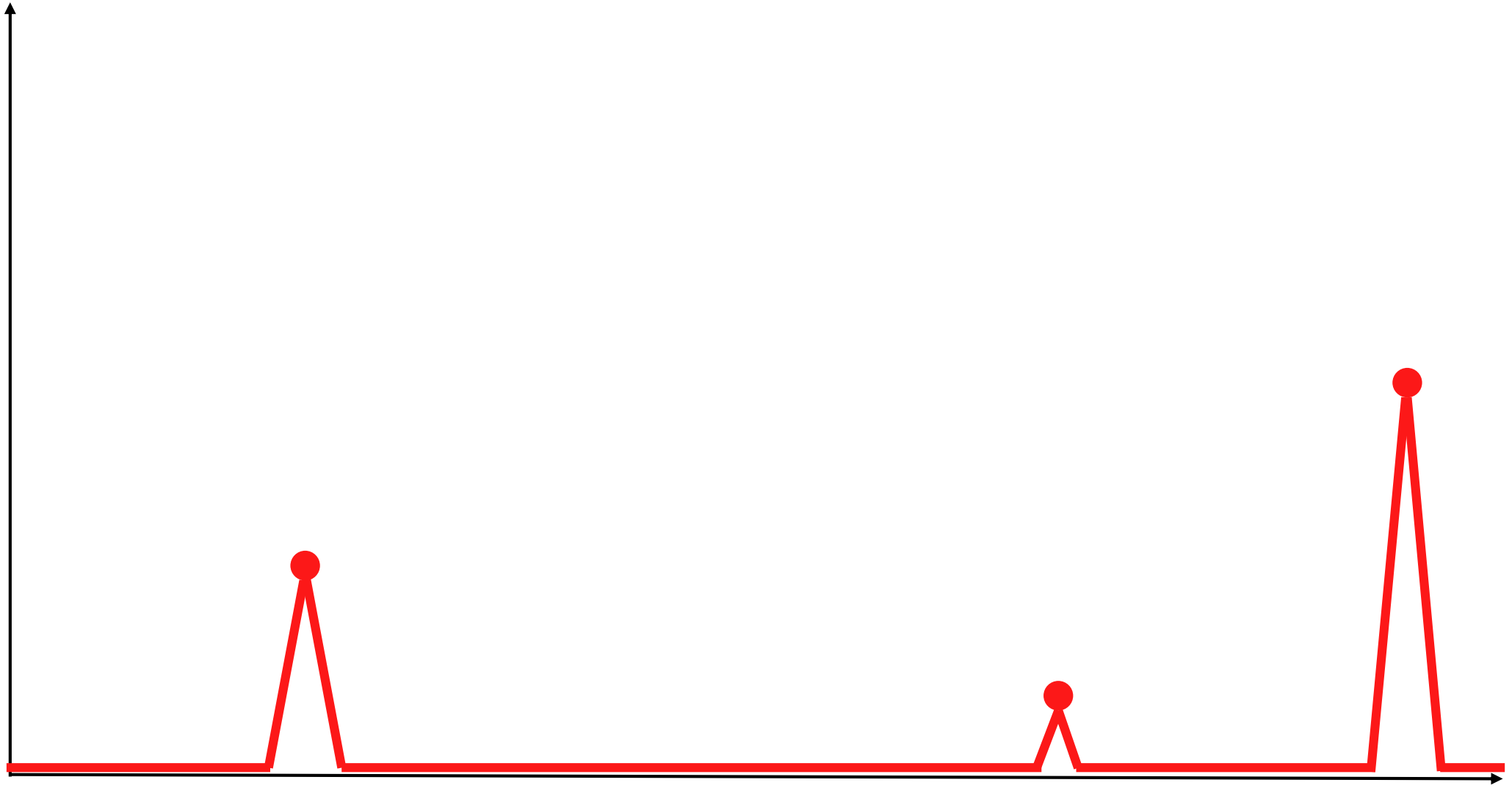


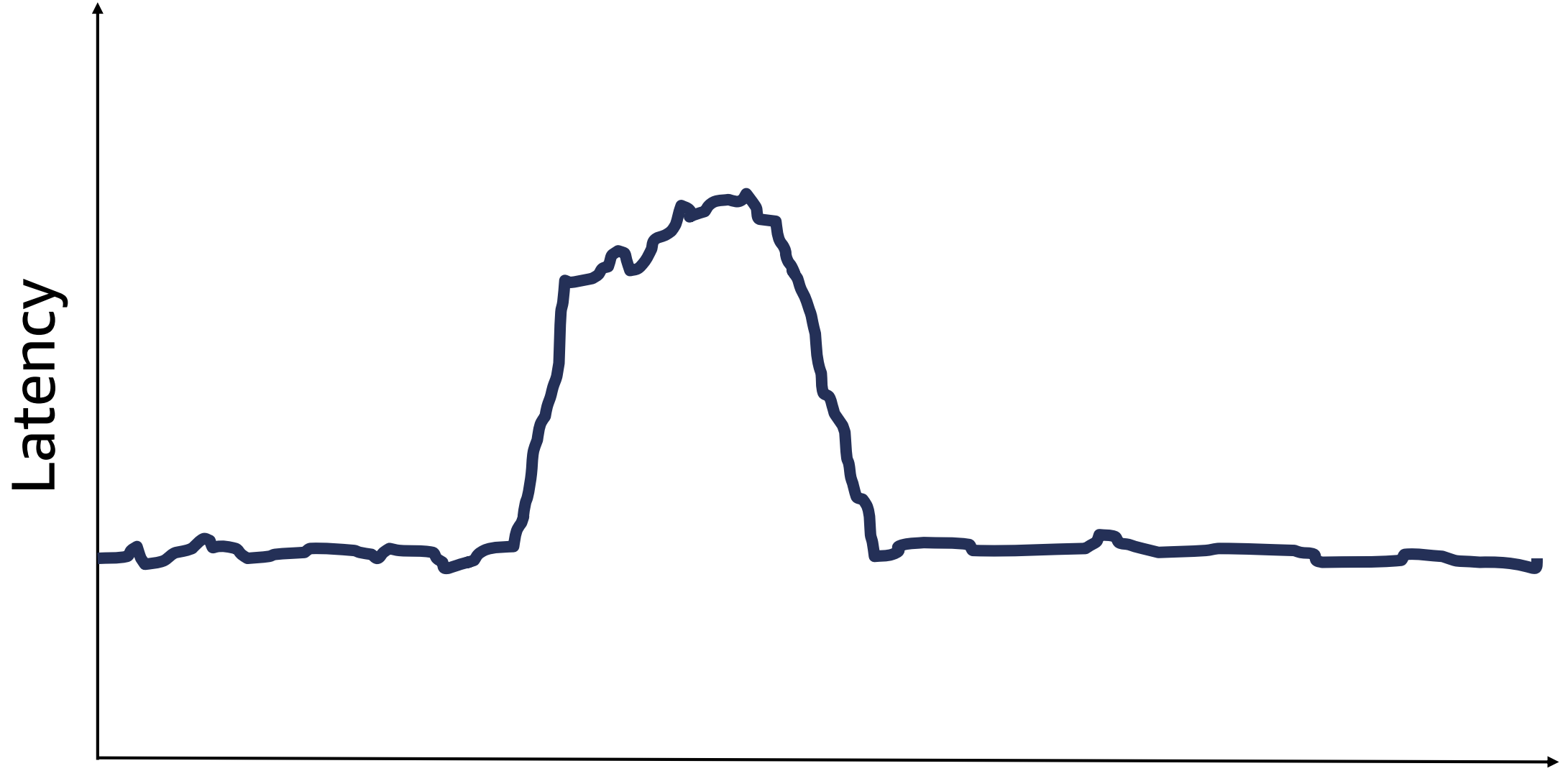


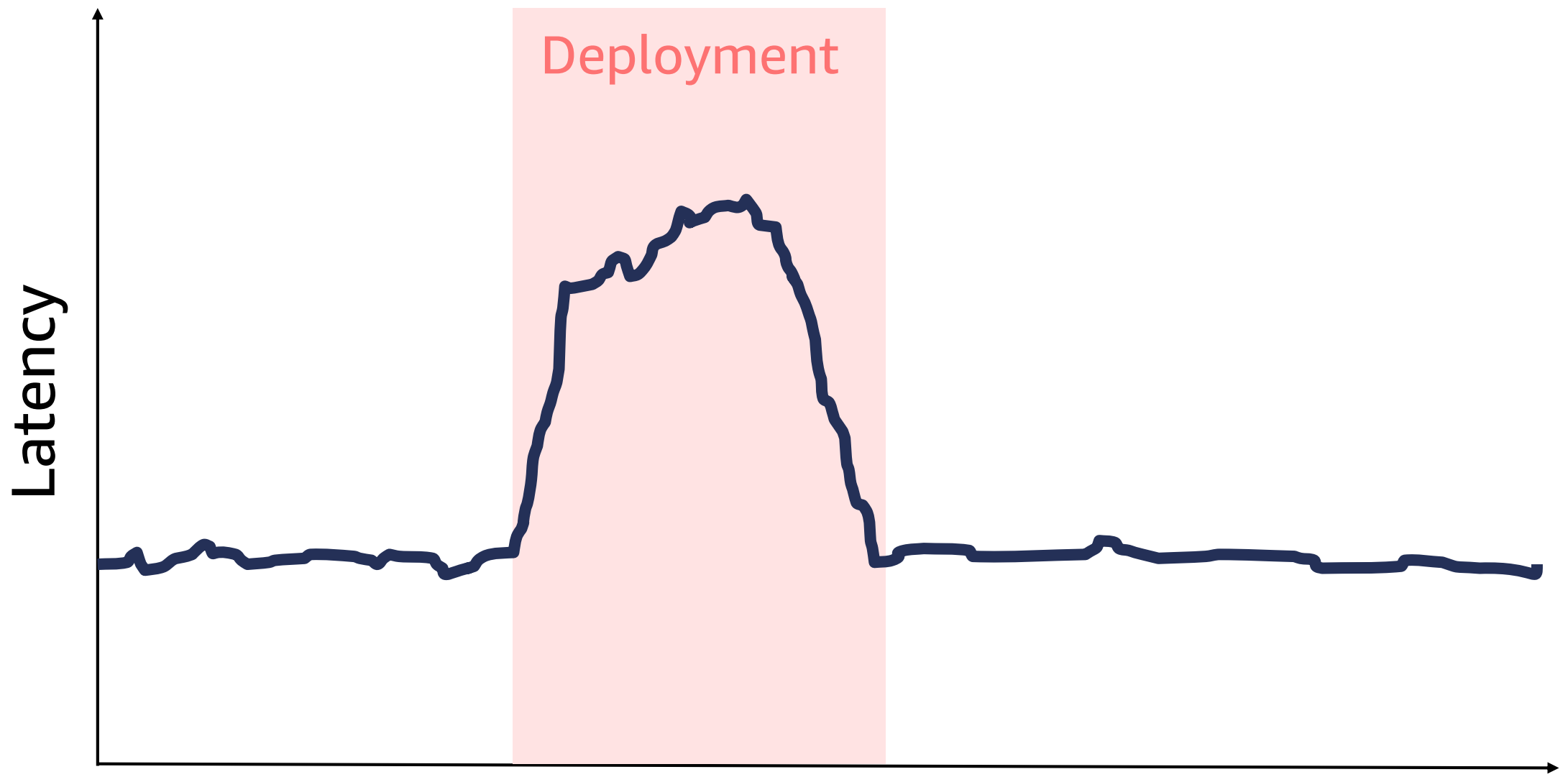
Failures

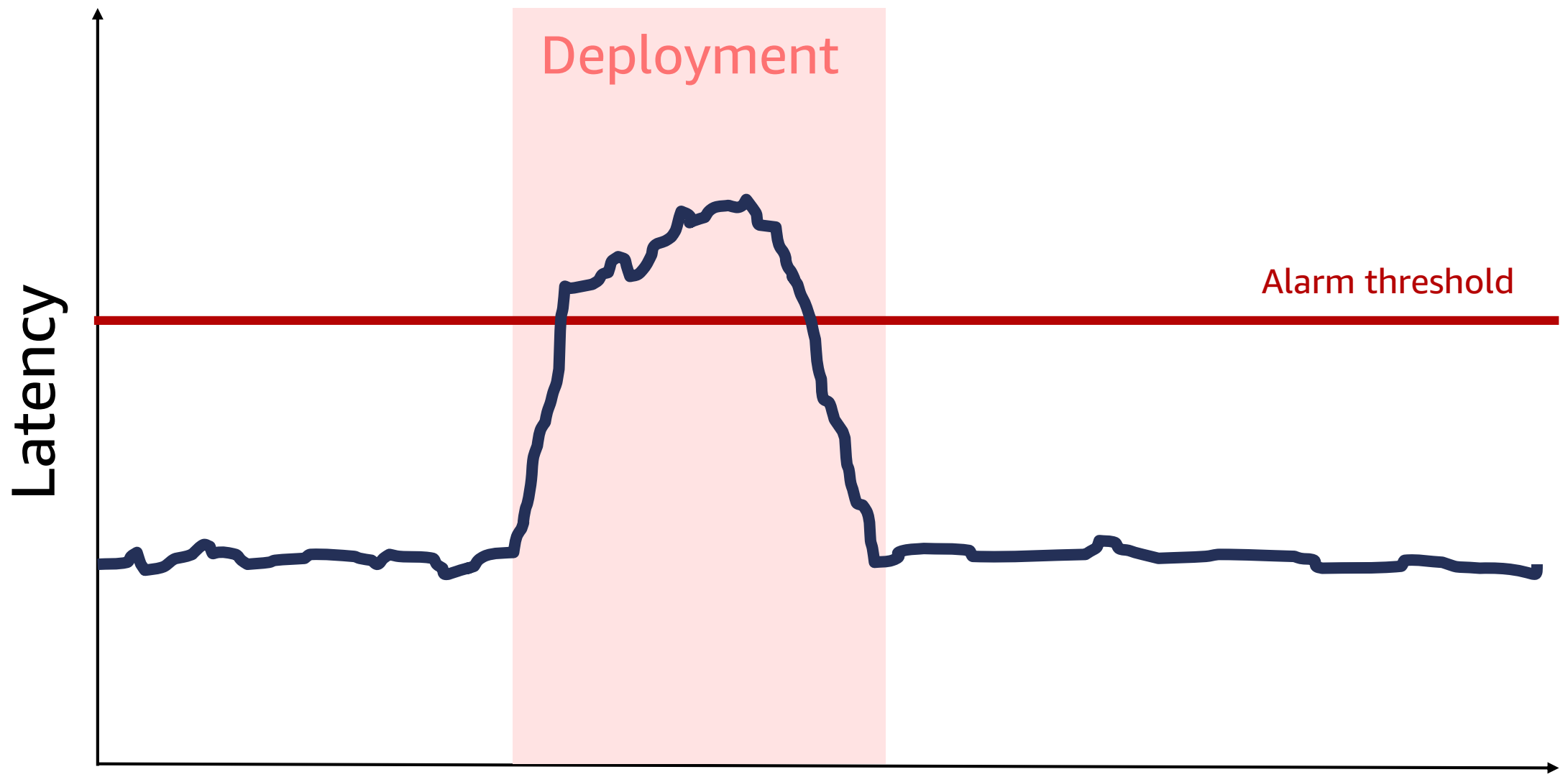


Failures









How AWS can help you

Key service: Amazon CloudWatch

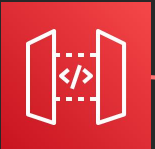
Automatically published metrics



AWS Lambda



Amazon Simple Notification Service (Amazon SNS)



Amazon API Gateway



Amazon Simple Storage Service (Amazon S3)



Amazon DynamoDB



Amazon Elastic Compute Cloud (Amazon EC2)



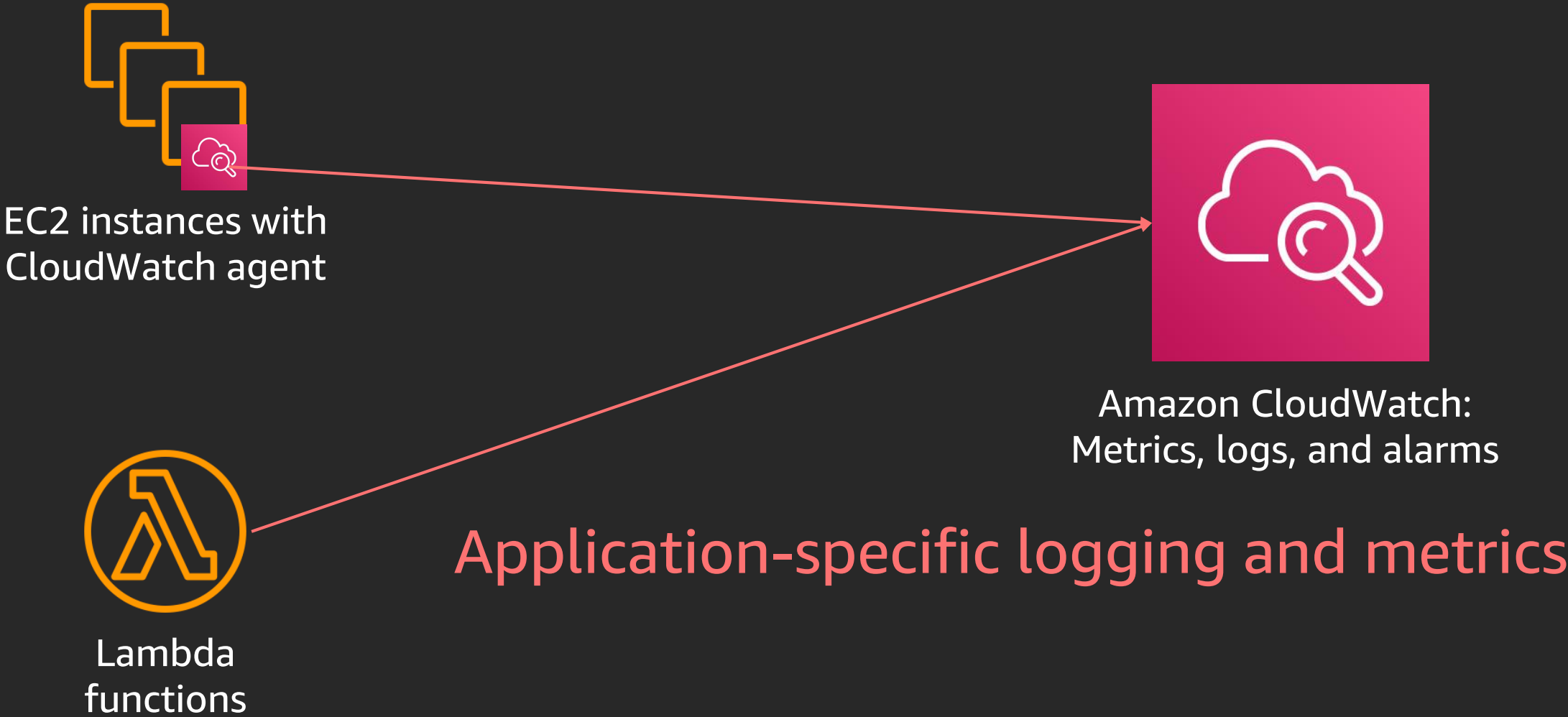
Billing

(and many others)

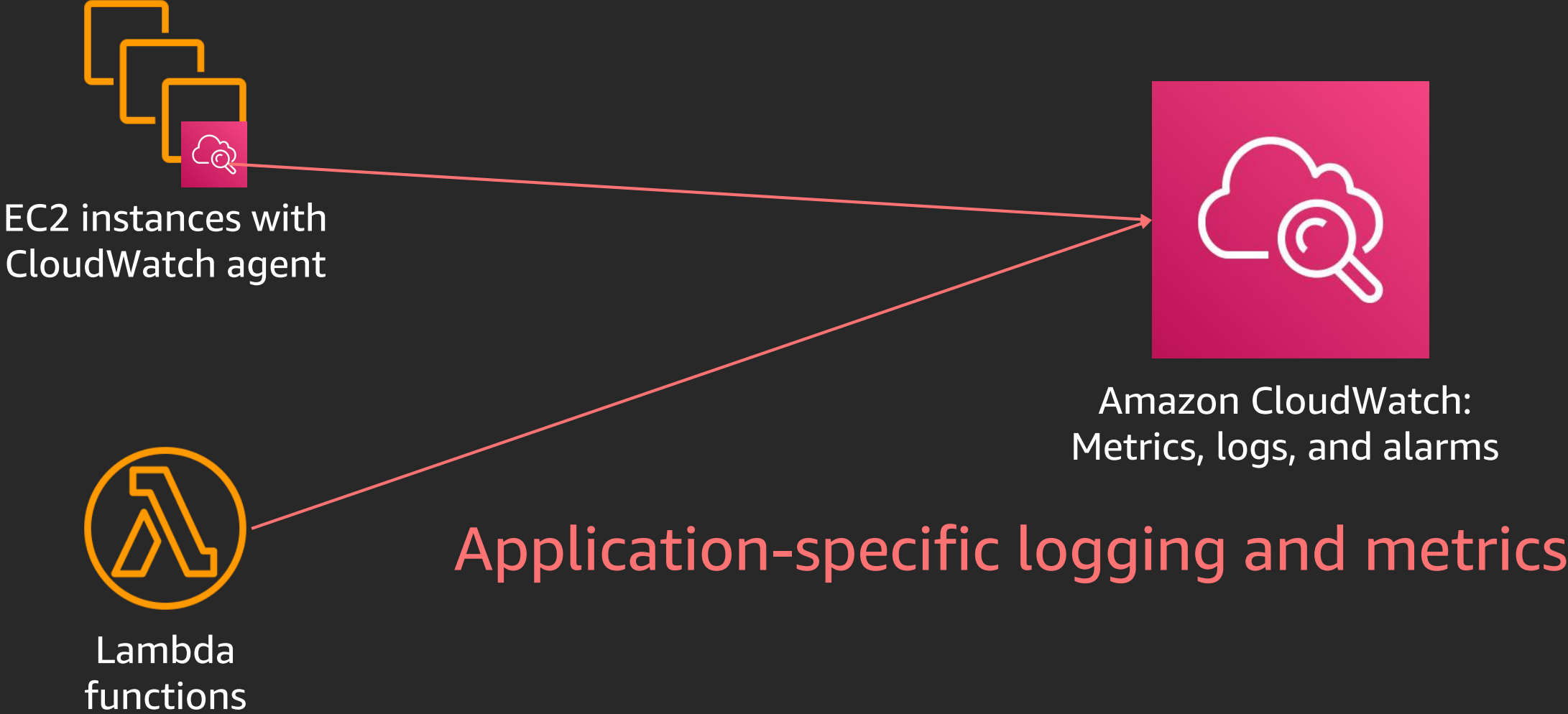


Amazon CloudWatch:
Metrics, logs, and alarms

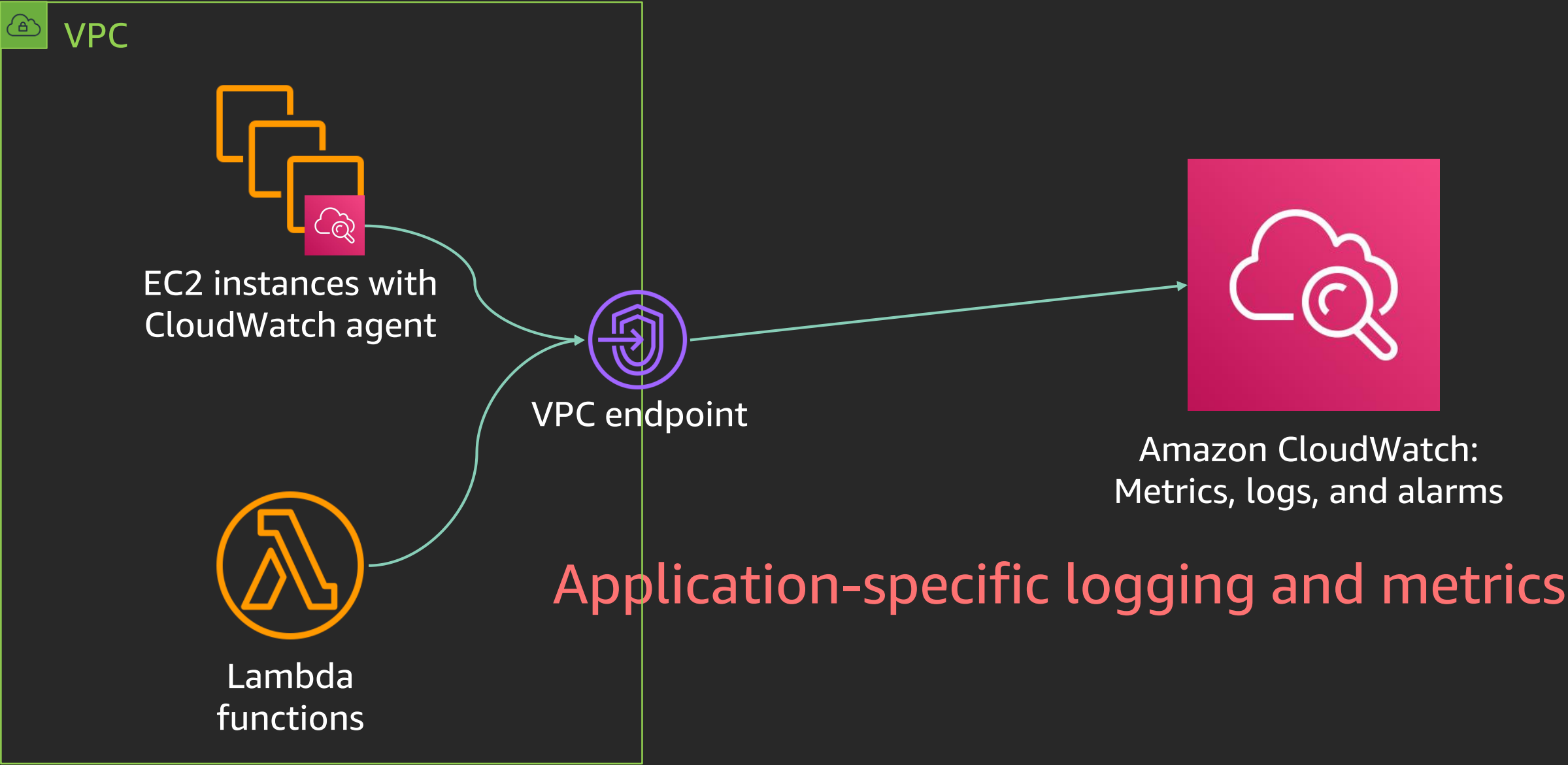
Key service: Amazon CloudWatch



Key service: Amazon CloudWatch

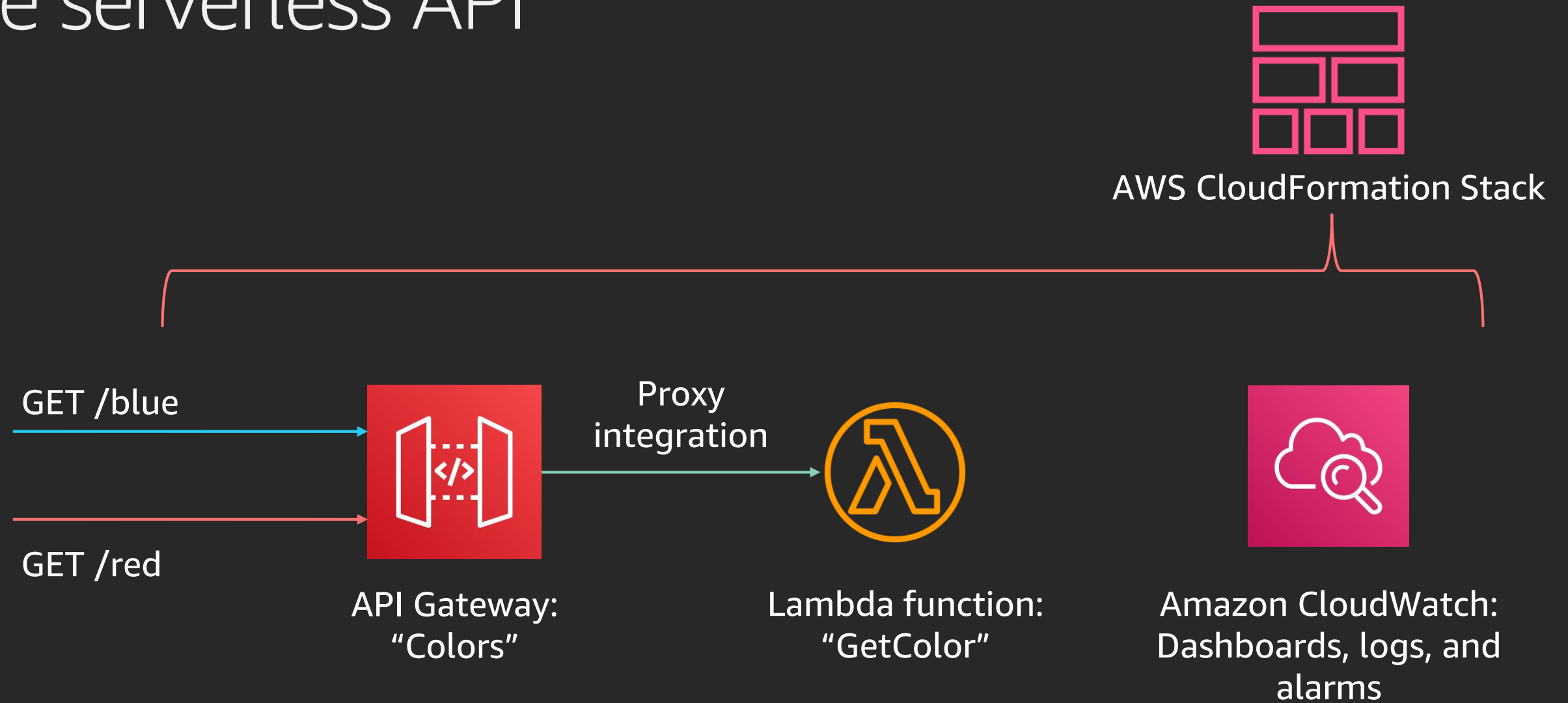


Key service: Amazon CloudWatch



Example: Monitoring a serverless application

A simple serverless API



CloudWatch default view

CloudWatch: Overview Time range 1h 3h 12h 1d 3d 1w custom Actions Refresh

All resources

Alarms by AWS service

Services	Status	Alarm	Insufficient	OK
API Gateway	🚨	1	-	2
DynamoDB	✅	-	-	8
Billing	?	-	-	-
CloudWatch Events	?	-	-	-
CloudWatch Logs	?	-	-	-
EC2	?	-	-	-
Elastic Block Store	?	-	-	-
Lambda	?	-	-	-
Network ELB	?	-	-	-
RDS	?	-	-	-

Recent alarms

color-api-BlueFaultRateAlarm-8LGCM... 🚨

No unit

Fault rate > 0.02 for 1 datapoints within 1 minute

Legend: Fault rate

color-api-LatencyP99Alarm-1BW0KB... ✅

Milliseconds

Latency > 300 for 2 datapoints within 2 minutes

Legend: Latency

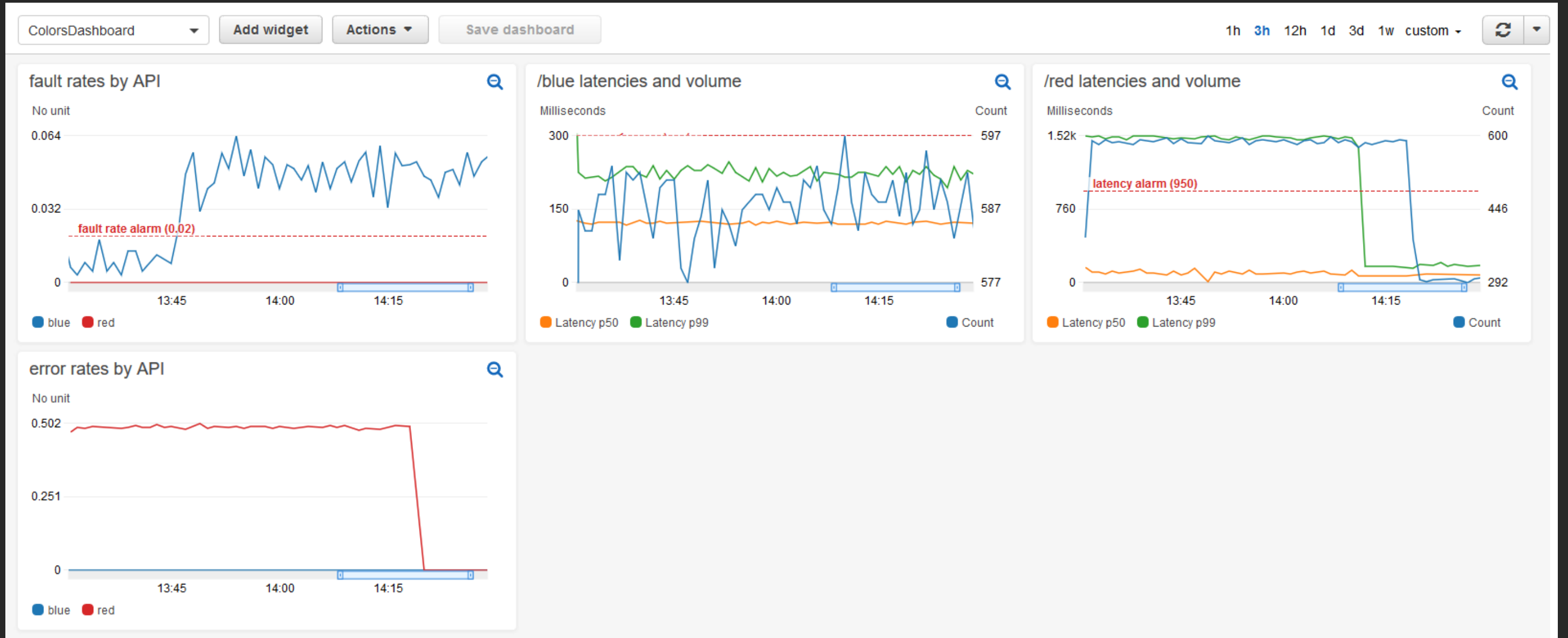
color-api-RedFaultRateAlarm-3OYV2... ✅

No unit

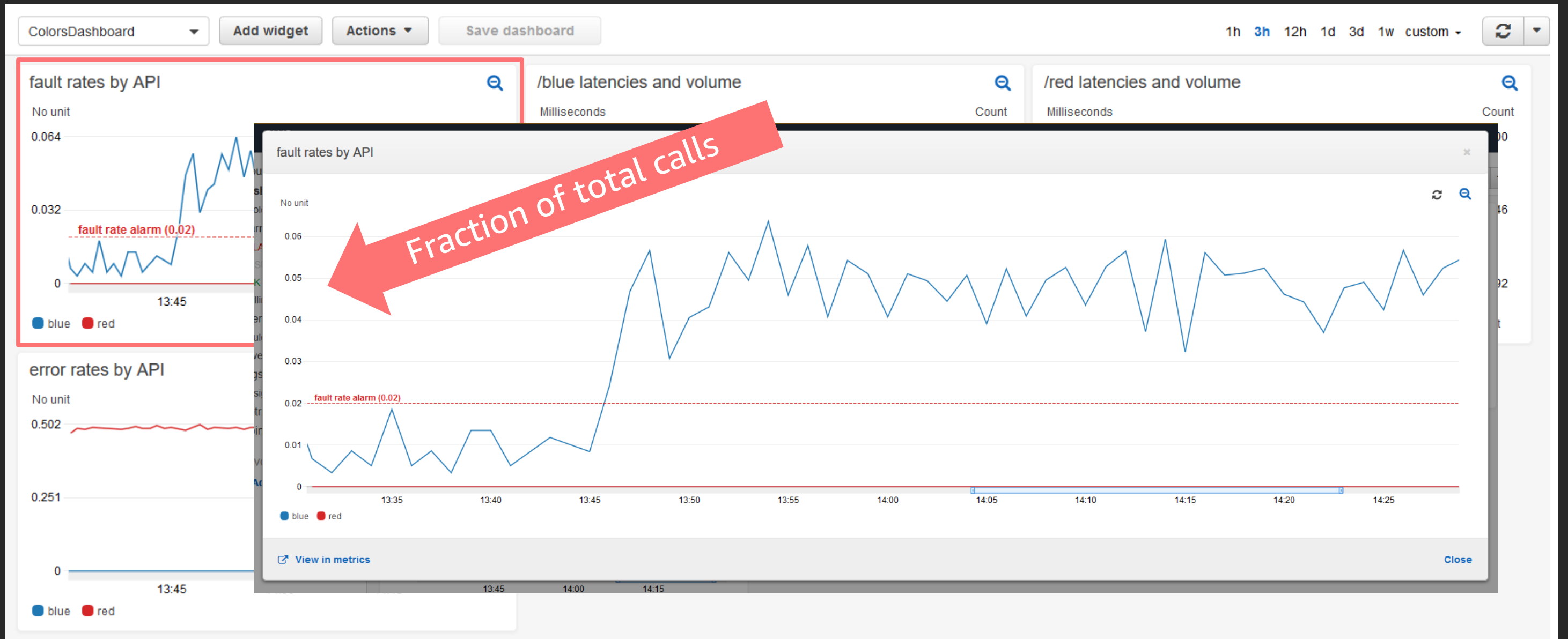
Fault rate > 0.02 for 1 datapoints within 1 minute

Legend: Fault rate

Custom dashboard



Custom dashboard: Metric math



Provisioning dashboards with AWS CloudFormation

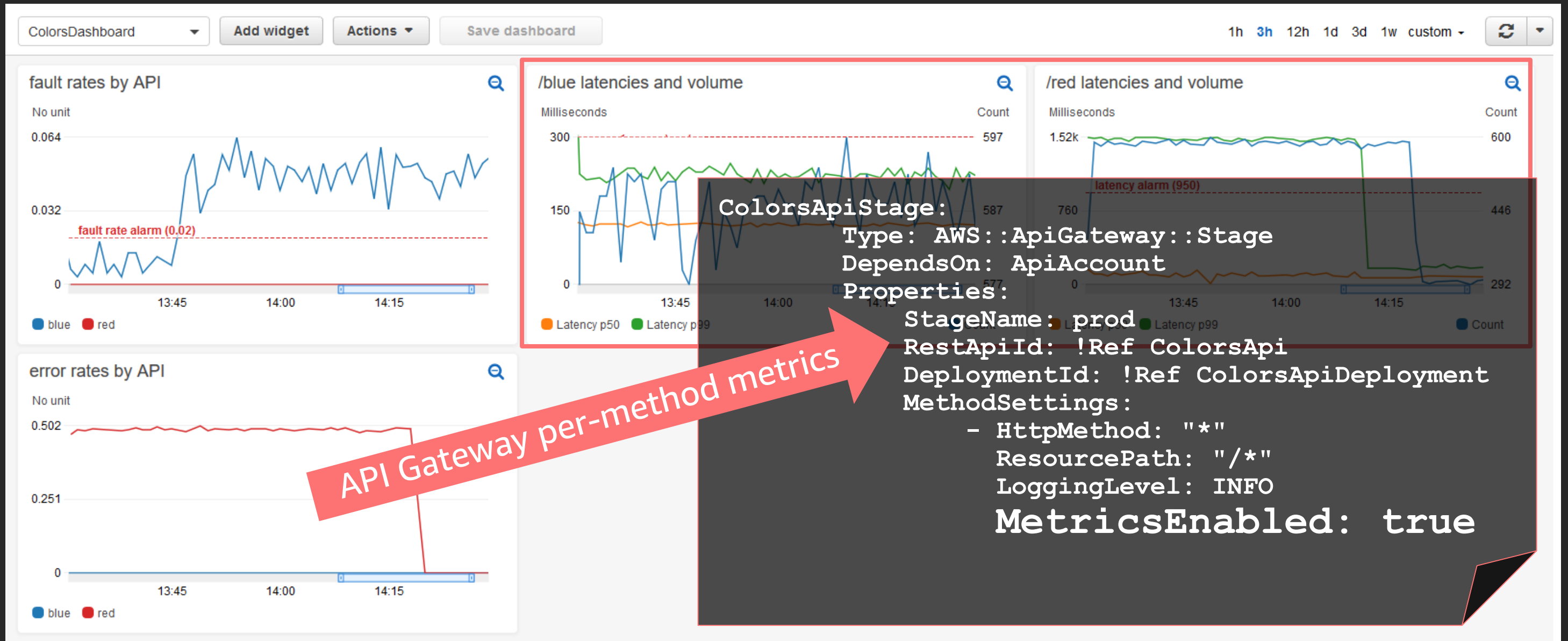
ColorsDashboard: Type: AWS::CloudWatch::Dashboard Properties: DashboardName: ColorsDashboard DashboardBody: !Sub |

```
    "widgets": [
      {
        "type": "metric",
        "x": 0,
        "y": 0,
        "width": 8,
        "height": 6,
        "properties": {
          "metrics": [
            "AWS/ApiGateway",
            "5XXError",
            "ApiName",
            "ColorsApi",
            "Resource",
            "/blue",
            "Stage",
            "${ColorsApiStage}",
            "Method",
            "GET",
            { "id": "m1", "visible": false } ],
            [
            ".",
            "Count",
            ".",
            ".",
            ".",
            ".",
            ".",
            ".",
            ".",
            ".",
            { "id": "m2", "visible": false } ],
            [ { "expression": "m1 / m2", "label": "blue", "id": "e1" } ],
            [ "AWS/ApiGateway", "5XXError", "ApiName", "ColorsApi", "Resource", "/red", "Stage", "${ColorsApiStage}", "Method", "GET", { "id": "m3", "visible": false } ],
            [
            ".",
            "Count",
            ".",
            ".",
            ".",
            ".",
            ".",
            ".",
            ".",
            ".",
            { "id": "m4", "visible": false } ],
            [ { "expression": "m3 / m4", "label": "red", "id": "e2" } ]
          ]
        }
      }
    ]
```

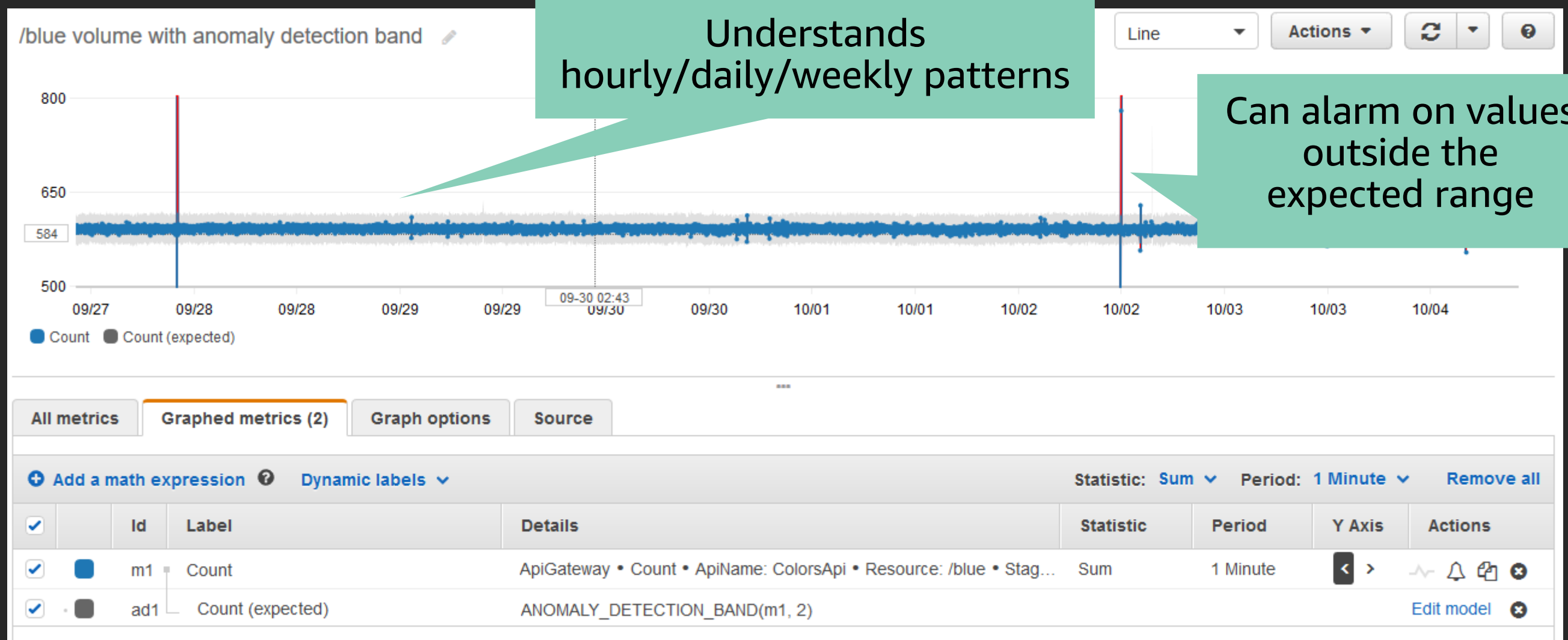
API Gateway per-method metrics

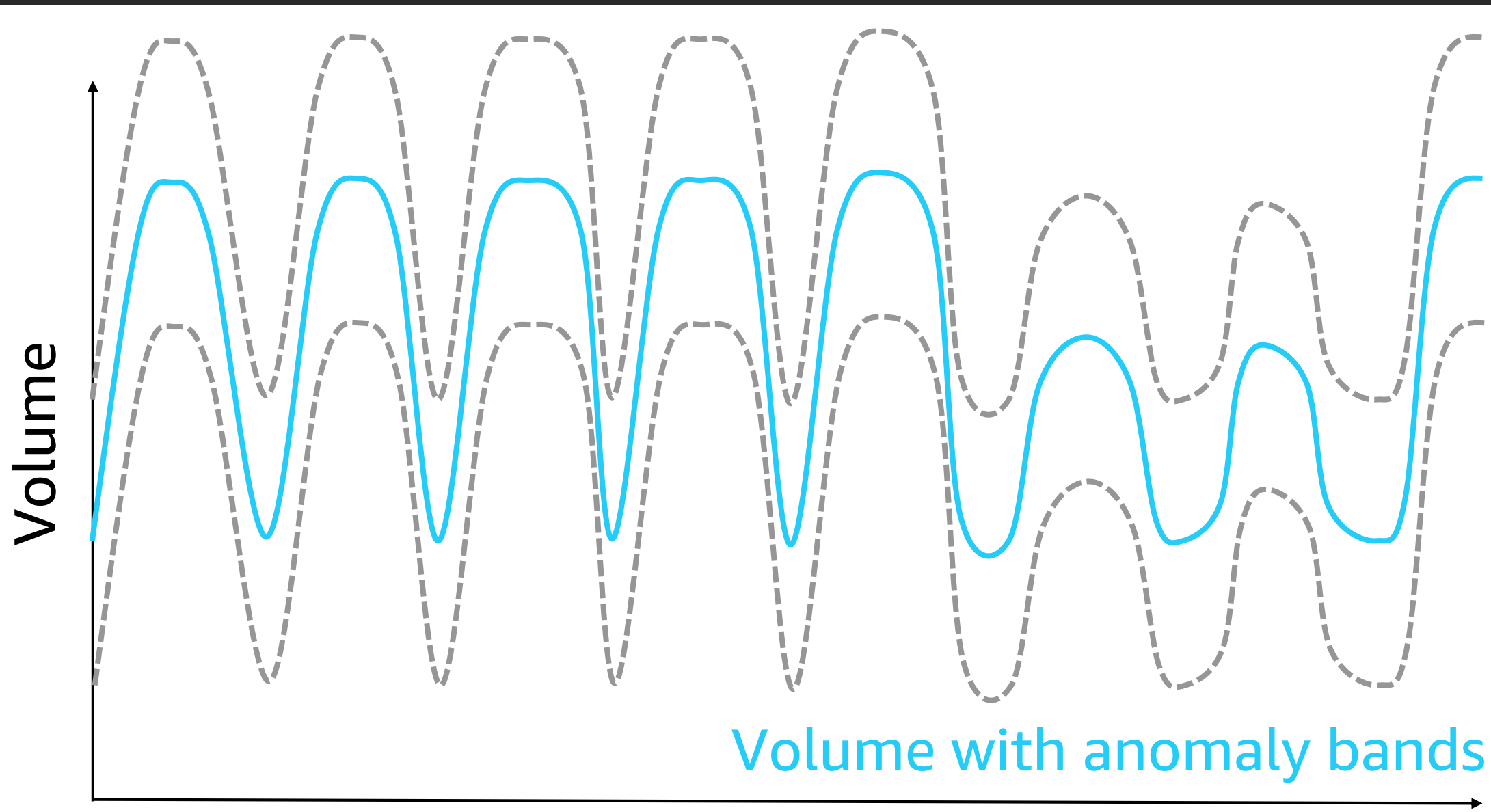
AWS CloudFormation stack

Custom dashboard: Breakdown by method



New: CloudWatch Anomaly Detection

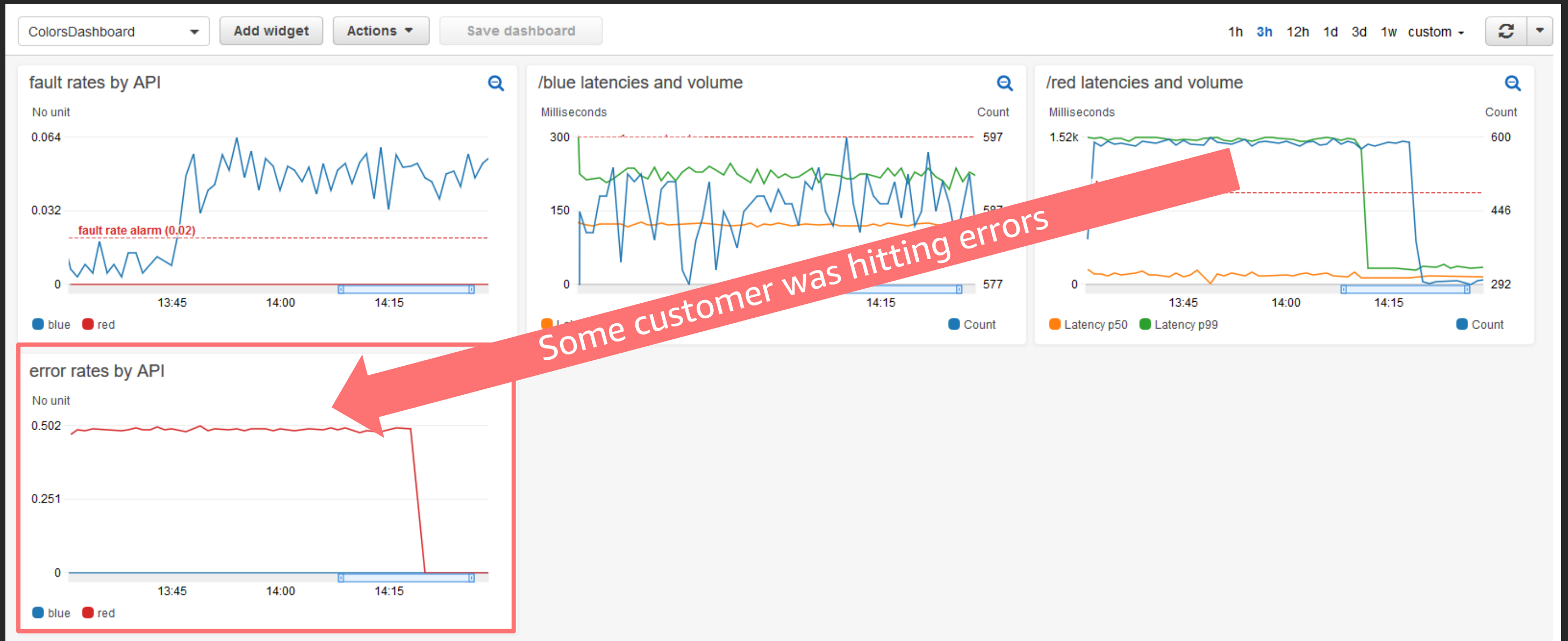




Volume with anomaly bands

Time scale: ~one week

Diagnosing unexpected behavior



Fast diagnostics with CloudWatch Logs Insights

CloudWatch Dashboards

Alarms

- ALARM 1
- INSUFFICIENT 0
- OK 10

Billing

Events

- Rules
- Event Buses

Logs

Insights

Metrics

Favorites

[+ Add a dashboard](#)

[Add to dashboard](#) [Actions](#)

[Learn how to write log queries](#)

color-api-ApiAccessLogGroup-12QZHY8H43LY5 15m 30m 1h **6h** 12h 1d custom

```
fields @timestamp, requestId, resourcePath, status
| filter status >= 400 and status < 500
| sort @timestamp desc
| limit 20
```

[Run query](#) [Sample queries](#) [Have feedback? Email us.](#)

Logs Visualization

Distribution of log events over time

14,380 records matched | 102,649 records (25.4 MB) scanned in 2.9s @ 35,432 records/s (8.8 MB/s)

#	@timestamp	requestId	resourcePath	status
▶ 1	2019-03-12 14:40:25.654	c5f69d72-44d4-11e9-9791-3dd462249c26	/red	429
▶ 2	2019-03-12 14:38:42.040	8834600b-44d4-11e9-8f52-fd038b18c837	/red	429
▶ 3	2019-03-12 14:38:41.874	881b0bc5-44d4-11e9-897b-a373d3f4f232	/red	429
▶ 4	2019-03-12 14:38:41.232	87b915d0-44d4-11e9-8406-df892c565959	/red	429
▶ 5	2019-03-12 14:38:40.848	877e7d62-44d4-11e9-bd94-05e2616cf493	/red	429
▶ 6	2019-03-12 14:38:40.429	873e8dd3-44d4-11e9-aec3-3b2d7332e244	/red	429

Commands

- fields
- filter
- stats
- sort
- limit
- parse

Discovered fields

- @logStream
- @message
- @timestamp
- caller
- httpMethod
- integrationLatency
- requestId
- requestTime
- resourcePath
- responseLength
- status

Fast diagnostics with CloudWatch Logs Insights

color-api-ApiAccessLogGroup-12QZHY8H43LY5

```
fields @timestamp, requestId, resourcePath, status
| filter status >= 400 and status < 500
| sort @timestamp desc
| limit 20
```

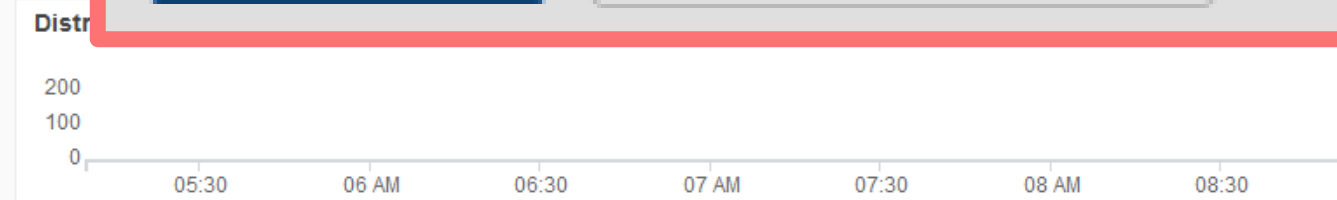
Run query

Sample queries

Discovered fields

Search for a field

@logStream	100%
@message	100%
@timestamp	100%
caller	100%
httpMethod	100%
integrationLatency	100%
requestId	100%
requestTime	100%
resourcePath	100%
responseLength	100%
status	100%



14,380 records matched | 102,649 records (25.4 MB) scanned in 2.9s @ 35,432 records/s (8.8 MB/s)

#	@timestamp	requestId	resourcePath	status
1	2019-03-12 14:40:25.654	c5f69d72-44d4-11e9-9791-3dd462249c26	/red	429
2	2019-03-12 14:38:42.040	8834600b-44d4-11e9-8f52-fd038b18c837	/red	429
3	2019-03-12 14:38:41.874	881b0bc5-44d4-11e9-897b-a373d3f4f232	/red	429
4	2019-03-12 14:38:41.232	87b915d0-44d4-11e9-8406-df892c565959	/red	429
5	2019-03-12 14:38:40.848	877e7d62-44d4-11e9-bd94-05e2616cf493	/red	429
6	2019-03-12 14:38:40.429	873e8dd3-44d4-11e9-aec3-3b2d7332e244	/red	429

Fast diagnostics with CloudWatch Logs Insights

The screenshot displays the AWS CloudWatch Logs Insights interface. On the left, a navigation sidebar includes sections for CloudWatch Dashboards, Alarms (with 1 ALARM, 0 INSUFFICIENT, and 10 OK indicators), Billing, Events, Rules, Event Buses, Logs, Insights (selected), Metrics, and Favorites. The main area shows a query for the log group 'color-api-ApiAccessLogGroup-12QZHY8H43LY5' with the following fields and filters:

```
fields @timestamp, requestId, resourcePath, status
| filter status >= 400 and status < 500
| sort @timestamp desc
| limit 20
```

Buttons for 'Run query' and 'Sample queries' are visible. Below the query, there are tabs for 'Logs' and 'Visualization'. The 'Visualization' tab is active, showing a bar chart titled 'Distribution of log events over time' from 05:30 to 11 AM. A red box highlights the visualization area, which shows a significant spike in log events between 09:30 and 10:30 AM. Below the chart, a summary states: '14,380 records matched | 102,649 records (25.4 MB) scanned in 2.9s @ 35,432 records/s (8.8 MB/s)'. At the bottom, a table of log records is shown:

#	@timestamp	requestId	resourcePath	status
1	2019-03-12 14:40:25.654	c5f69d72-44d4-11e9-9791-3dd462249c26	/red	429
2	2019-03-12 14:38:42.040	8834600b-44d4-11e9-8f52-fd038b18c837	/red	429
3	2019-03-12 14:38:41.874	881b0bc5-44d4-11e9-897b-a373d3f4f232	/red	429
4	2019-03-12 14:38:41.232	87b915d0-44d4-11e9-8406-df892c565959	/red	429
5	2019-03-12 14:38:40.848	877e7d62-44d4-11e9-bd94-05e2616cf493	/red	429
6	2019-03-12 14:38:40.429	873e8dd3-44d4-11e9-aec3-3b2d7332e244	/red	429

On the right side of the interface, a list of fields is visible: httpMethod, integrationLatency, requestId, requestTime, resourcePath, responseLength, and status.

Fast diagnostics with CloudWatch Logs Insights

The screenshot displays the AWS CloudWatch Logs Insights interface. At the top, a log group is selected: `color-api-ApiAccessLogGroup-12QZHY8H43LY5`. A query is entered in the text area:

```
fields @timestamp, requestId, resourcePath, status
| filter status >= 400 and status < 500
| sort @timestamp desc
| limit 20
```

Below the query, a summary indicates: `14,380 records matched | 102,649 records (25.4 MB) scanned in 2.9s @ 35,432 records/s (8.8 MB/s)`. A table of results is shown, with the first seven rows highlighted by a red box. The table has the following columns: #, @timestamp, requestId, resourcePath, and status.

#	@timestamp	requestId	resourcePath	status
▶ 1	2019-03-12 14:40:25.654	c5f69d72-44d4-11e9-9791-3dd462249c26	/red	429
▶ 2	2019-03-12 14:38:42.040	8834600b-44d4-11e9-8f52-fd038b18c837	/red	429
▶ 3	2019-03-12 14:38:41.874	881b0bc5-44d4-11e9-897b-a373d3f4f232	/red	429
▶ 4	2019-03-12 14:38:41.232	87b915d0-44d4-11e9-8406-df892c565959	/red	429
▶ 5	2019-03-12 14:38:40.848	877e7d62-44d4-11e9-bd94-05e2616cf493	/red	429
▶ 6	2019-03-12 14:38:40.429	873e8dd3-44d4-11e9-aec3-3b2d7332e244	/red	429
▶ 7	2019-03-12 14:38:40.015	86ff6292-44d4-11e9-a2ad-3d300d85d51a	/red	429

On the right side of the console, there is a 'Commands' panel with options like `fields`, `filter`, `stats`, `sort`, `limit`, and `parse`. Below that is a 'Discovered fields' panel with a search bar and a list of fields including `@logStream`, `@message`, `@timestamp`, `caller`, `httpMethod`, `integrationLatency`, `requestId`, `requestTime`, `resourcePath`, `responseLength`, and `status`.

Fast diagnostics with CloudWatch Logs Insights

CloudWatch
Dashboards
Alarms
ALARM 1
INSUFFICIENT 0
OK 10
Billing
Events
Rules
Event Buses
Logs
Insights
Metrics
Favorites
+ Add a dashboard

ColorApiStage:
Type: AWS::ApiGateway::Stage
DependsOn: ApiAccount
Properties:
...
AccessLogSetting:
DestinationArn: !GetAtt ApiAccessLogGroup.Arn
Format: !Sub |
{
 "requestId": "\$context.requestId",
 "httpMethod": "\$context.httpMethod",
 "resourcePath": "\$context.resourcePath",
 "status": "\$context.status",
 "responseLength": "\$context.responseLength"
}

Run query Sample queries

Distribution of log events over time

14,380 records matched | 102,649 records (25.4 MB) scanned in 2.9s @ 35,432 records/s (6.8 MB/s)

#	@timestamp	requestId	resourcePath	status
▶ 1	2019-03-12 14:40:25.654	c5f69d72-44d4-11e9-9791-3dd462249c26	/red	429
▶ 2	2019-03-12 14:38:42.040	8834600b-44d4-11e9-8f52-fd038b18c837	/red	429
▶ 3	2019-03-12 14:38:41.874	881b0bc5-44d4-11e9-897b-a373d3f4f232	/red	429
▶ 4	2019-03-12 14:38:41.232	87b915d0-44d4-11e9-8406-df892c565959	/red	429
▶ 5	2019-03-12 14:38:40.848	877e7d62-44d4-11e9-bd94-05e2616cf493	/red	429
▶ 6	2019-03-12 14:38:40.429	873e8dd3-44d4-11e9-aec3-3b2d7332e244	/red	429

Commands
fields
filter
stats
sort
limit
parse

Discovered fields
Search for a field
@logStream
@message
@timestamp
caller
httpMethod
integrationLatency
requestId
requestTime
resourcePath
responseLength
status

Wrap-up

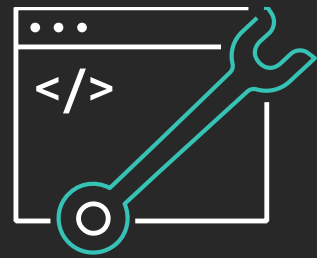
Takeaways

- Never waste a failure:
Effective post-mortems
- Catch failures before your customers do:
Effective dashboarding and metrics-reading
- Use AWS tools to gain visibility and insight into your application



Learn DevOps with AWS Training and Certification

Resources created by the experts at AWS to propel your organization and career forward



Take free digital training to learn best practices for developing, deploying, and maintaining applications



Classroom offerings, like DevOps Engineering on AWS, feature AWS expert instructors and hands-on activities



Validate expertise with the **AWS Certified DevOps Engineer - Professional** or **AWS Certified Developer - Associate** exams

Visit aws.amazon.com/training/path-developing/

Thank you!

Becky Weiss

becky@amazon.com



Please complete the session survey in the mobile app.