



Serverless NFRs

Robert Bulmer
Software Architect @ City Electric

Electrical Wholesaler ()

What are NFRs?



Functional Requirements

Functional requirements are the customer / business requirements.

i.e. System must provide a CSV file of all active customers within the user specified date range

What are NFRs?



Non-Functional Requirements

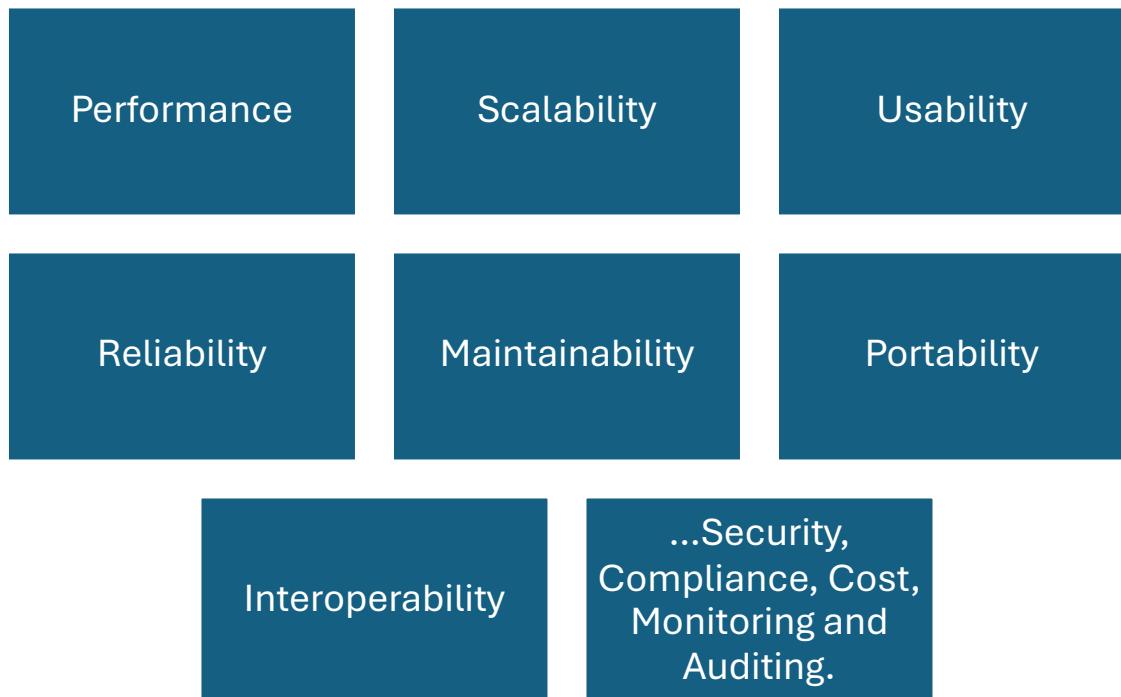
Non-Functional requirements are the technical system requirements that need to be met to fulfill those functional requirements.

i.e. System must provide a CSV file of all active customers within the user specified date range -> ?

The system must be able to allow 10 concurrent users to generate the file.

The system must be able to produce a file for our growing customer base of 1,500,000 customers.

NFR Categories



A.K.A – The ‘ilities’



My Key NFRs

- ✓ Maintainability
- ✓ Portability
- ✓ Cost
- ✓ Performance
- ✓ Scalability





Maintainability

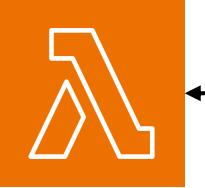
Often, we want to make sure a system can be maintained and used again.

Maintainability consists of requirements to do with ***reusability*** and ***analysability***.

Reusability, can part of the system be used by another system.

With **Analysability**, can we easily identify faults and improvements we need to make.

Maintainability



Example (Reusability):

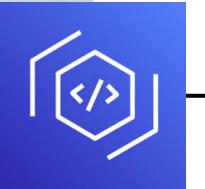


Functional Requirement: The integration must convert the Country ISO Code to the ISO Name – i.e. GBR to United Kingdom of Great Britain and Northern Ireland



Non-Functional Requirement: All standard data transformations are reusable across services.

Implementation: Publish a shared utility library to the company registry (AWS Code Artifact) to transform standard ISO values



Maintainability



Example (Reusability):



Functional Requirement: The integration must transform Customer data into the format required by the downstream integration.



Non-Functional Requirement: All architecture patterns that follow the company standards must be reusable across services.

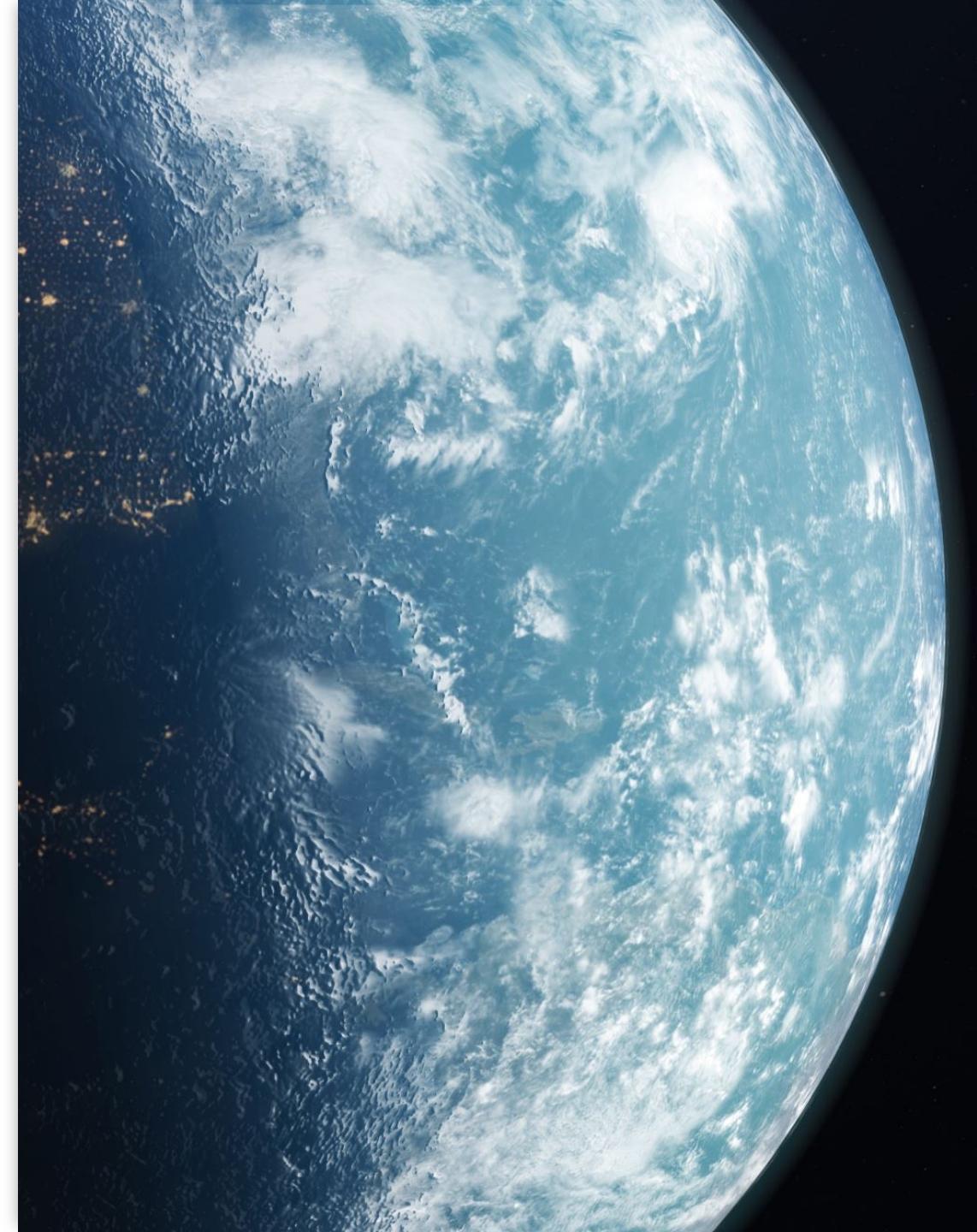
Implementation: *The integration should create an AWS CDK Construct to allow reuse of the IAC.*



Portability

With portability we often ask ourselves;

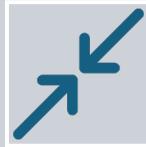
- ***Adaptability*** can we adapt this for another environment?
- ***Installability*** can we deploy and install the system on another platform or in another region or account.



Portability

Serverless Internationalisation

Example of building our serverless solutions once and with extensibility so they can be consumed in different locales safely, written in the AWS CDK and Typescript.



Example (Adaptability):

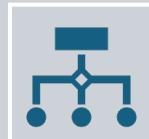


Serverless Advocate · [Follow](#)

11 min read · Jun 30, 2022



Functional Requirement: The integration must support UKI and NA customers



Non-Functional Requirement: Use *Internationalisation in the code base to adapt the deployment to the correct region*



Portability



Example (*Installability*):



Functional Requirement: The integration must support UKI and NA customers



Non-Functional Requirement: *The AWS CDK app must be configurable to deploy separately to multiple AWS regions*



Cost

No business has infinite budget. During normal operation it is not uncommon to be asked how much your application is going to cost the company.

They often want a monthly, yearly and future growth figure.



Cost



Example (Cost):



Functional Requirement: The integration must support our 100,000 Customer base



Non-Functional Requirement: *The integration must cost less than \$1,000/month (est. using the AWS pricing calculator).*

Implementation: Must have AWS budget alerts enabled and the application must be tagged for suitable identification





Performance

With performance we often look at **response time** and **throughput**.

Response time being how quickly the application reacts to input or an event.

Throughput is typically how many transactions can we process within a given time frame.



Performance

When should we use Serverless?

When you have periodic unpredictable, spiky, workloads. *There are other use cases where you might want to leverage the agility and developer experience that Serverless provides.*

If you're comparing serverless instances to regular provisioned instances, then you're comparing apples and oranges... They are for two different use cases. **Think "Serverless-first" not "serverless-only"**

Typically, when people say performance, they want to know exact milliseconds of the database queries. While that contributes to the performance NFR... we often need to look at a higher level first.

Performance – Response Time

What response time do we need from our system?

Typically, web application *users expect a Flow/Continuity response within 1 second*;
This usually involves refreshing data, reloading web pages etc.





Artillery

cloud-scale load testing

<https://www.npmjs.com/package/artillery>

```
artillery > ! performance-test.yml
  1 config:
  2   target: '{{env.API_URL}}'
  3   phases:
  4     - duration: 10 # Length of test
  5       arrivalRate: 1 # Number of users
  6       rampTo: 1 # Increase Users over time
  7       name: Test Phase
  8   plugins:
  9     ensure: {}
 10    apdex: {}
 11    metrics-by-endpoint: {}
 12  apdex:
 13    threshold: 500
 14  ensure:
 15    thresholds:
 16      - http.response_time.p99: 500
 17      - http.response_time.p95: 300
 18  scenarios:
 19    - flow:
 20      - get:
 21        url: '/customer/{{env.ID_UNDER_TEST}}'
 22
```



Artillery cloud-scale load testing

<https://www.npmjs.com/package/artillery>

Summary report @ 10:19:45(+0100)

apdex.frustrated:	4
apdex.satisfied:	6
apdex.tolerated:	0
http.codes.200:	10
http.downloaded_bytes:	1130
http.request_rate:	1/sec
http.requests:	10
http.response_time:	
min:	61
max:	3334
mean:	1310.2
median:	89.1
p95:	3197.8
p99:	3197.8
http.responses:	10
plugins.metrics-by-endpoint./prod//customer/403fdfbd-d1c2-405a-a6e5-9540775b...: .	10
plugins.metrics-by-endpoint.response_time./prod//customer/403fdfbd-d1c2-405a-a6e5-9540775baba3: .	
min:	61
max:	3334
mean:	1310.2
median:	89.1
p95:	3197.8
p99:	3197.8
vusers.completed:	10
vusers.created:	10
vusers.created_by_name.0:	10
vusers.failed:	0
vusers.session_length:	
min:	139.4
max:	3489.4
mean:	1388.2
median:	159.2
p95:	3262.4
p99:	3262.4

Checks:

fail: http.response_time.p95 < 300
fail: http.response_time.p99 < 500



Artillery

cloud-scale load testing

<https://www.npmjs.com/package/artillery>

```
artillery > ! performance-test.yml
  1 config:
  2   target: '{{env.API_URL}}'
  3   phases:
  4     - duration: 1 # Length of test
  5       arrivalRate: 1 # Number of users
  6       rampTo: 1 # Increase Users over time
  7       name: WarmUp Phase
  8     - pause: 5
  9       name: WarmUp Wait Phase
 10    - duration: 10 # Length of test
 11      arrivalRate: 1 # Number of users
 12      rampTo: 1 # Increase Users over time
 13      name: Test Phase
 14   plugins:
 15     ensure: {}
 16     apdex: {}
 17     metrics-by-endpoint: {}
 18   apdex:
 19     threshold: 500
 20   ensure:
 21     thresholds:
 22       - http.response_time.p99: 500
 23       - http.response_time.p95: 300
 24   scenarios:
 25     - flow:
 26       - get:
 27         url: '/customer/{{env.ID_UNDER_TEST}}'
 28
```

1 failed... = Lambda Cold Starts



Artillery cloud-scale load testing

<https://www.npmjs.com/package/artillery>

Summary report @ 11:07:07(+0100)

```
apdex.frustrated: ..... 1
apdex.satisfied: ..... 10
apdex.tolerated: ..... 0
http.codes.200: ..... 11
http.downloaded_bytes: ..... 1243
http.request_rate: ..... 1/sec
http.requests: ..... 11
http.response_time:
    min: ..... 52
    max: ..... 3178
    mean: ..... 369
    median: ..... 85.6
    p95: ..... 162.4
    p99: ..... 162.4
http.responses: ..... 11
plugins.metrics-by-endpoint./prod//customer/403fdbd-d1c2-405a-a6e5-9540775b... 11
plugins.metrics-by-endpoint.response_time./prod//customer/403fdbd-d1c2-405a-a6e5-9540775baba3:
    min: ..... 52
    max: ..... 3178
    mean: ..... 369
    median: ..... 85.6
    p95: ..... 162.4
    p99: ..... 162.4
vusers.completed: ..... 11
vusers.created: ..... 11
vusers.created_by_name.0: ..... 11
vusers.failed: ..... 0
vusers.session_length:
    min: ..... 116.5
    max: ..... 3306.9
    mean: ..... 448.5
    median: ..... 156
    p95: ..... 232.8
    p99: ..... 232.8
```

Checks:

```
ok: http.response_time.p95 < 300
ok: http.response_time.p99 < 500
```

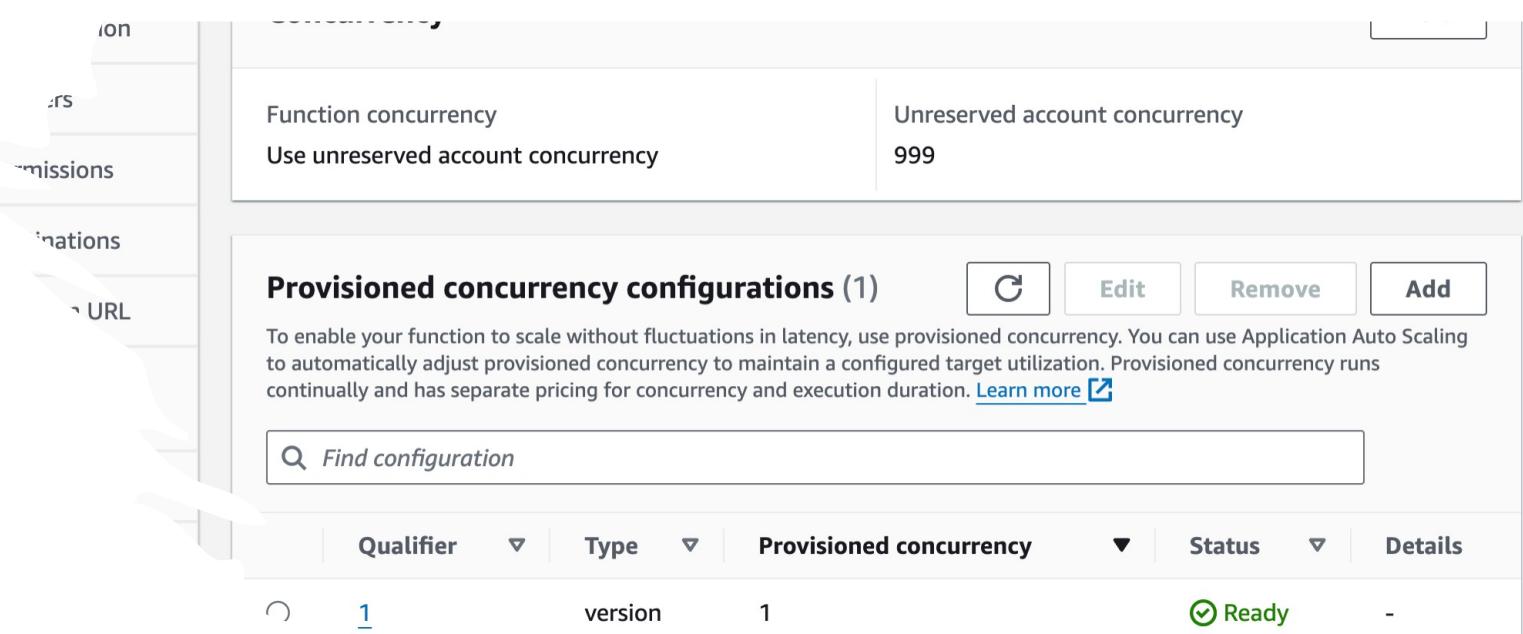
Apdex score: 0.9090909090909091 (good)

Cold Starts

There are numerous ways to resolve cold starts.

If you need spiky regular throughput with a fast response time, then **Provisioned Concurrency** is your solution.

You can now also use Auto-Scaling to automatically adjust the provisioned concurrency using CloudWatch Metrics!



The screenshot shows the AWS Lambda console interface for managing provisioned concurrency. On the left, there's a sidebar with navigation links: 'Function', 'Logs', 'Metrics', 'Traces', 'Deployments', and 'URL'. The main area has tabs for 'Function concurrency' and 'Unreserved account concurrency'. Under 'Function concurrency', it says 'Use unreserved account concurrency' and shows a value of '999'. Below this, a section titled 'Provisioned concurrency configurations (1)' is shown, with a table. The table has columns for 'Qualifier', 'Type', 'Provisioned concurrency', 'Status', and 'Details'. One row is visible with a 'version' value of '1' and a 'Ready' status. There are buttons for 'Edit', 'Remove', and 'Add' at the top of this section. A search bar at the bottom of the table says 'Find configuration'.

Qualifier	Type	Provisioned concurrency	Status	Details
1	version	1	Ready	-

AWS Lambda Power Tuning

The screenshot shows the AWS Serverless Application Repository interface. In the top navigation bar, there are links for 'aws', 'Services', a search bar with placeholder 'Search', and an 'Option+' button. On the left, a sidebar has 'Serverless Application Repository' at the top, followed by 'Available applications' (which is orange) and 'Published applications'. The main area is titled 'Available applications' and shows two tabs: 'Public applications (858)' (which is orange) and 'Private applications'. Below these tabs is a search bar containing 'aws-lambda-power-tuning'. Underneath the search bar is a checked checkbox labeled 'Show apps that create custom IAM roles or resource policies'. A detailed card for the 'aws-lambda-power-tuning' application is displayed. The card title is 'aws-lambda-power-tuning'. It includes a warning icon and the text: 'Creates custom IAM roles or resource policies'. The description states: 'AWS Lambda Power Tuning is an open-source tool that can help you visualize and fine-tune the memory/power configuration of Lambda functions. It runs in your AWS account - powered by AWS Step Functions - and it supports multiple optimization strategies.' Below the description are several blue tags: 'lambda', 'state-machine', 'optimization', 'step-functions', and 'power'. At the bottom of the card, it says 'Alex Casalboni' with a GitHub icon, '20.3K deployments', and 'AWS verified author'.

The screenshot shows a GitHub repository page for 'github.com/alexcasalboni/aws-lambda-power-tuning/tree/master'. The top navigation bar includes back and forward arrows, a refresh icon, and a search bar with the URL. Below the navigation is a header with 'README' and 'Apache-2.0 license'. The main content area features a large heading 'AWS Lambda Power Tuning'. Below the heading are several status indicators: a 'Build Status' badge, a 'coverage 100%' badge, a 'Maintained? yes' badge, and an 'issues 6 open' badge. There is also an 'Open Source' button with a heart icon. The description text reads: 'AWS Lambda Power Tuning is a state machine powered by AWS Step Functions that helps you optimize your Lambda functions for cost and/or performance in a data-driven way.' Another section of text states: 'The state machine is designed to be easy to deploy and fast to execute. Also, it's language agnostic so you can optimize any Lambda functions in your account.' The final section of text explains: 'Basically, you can provide a Lambda function ARN as input and the state machine will invoke that function with multiple power configurations (from 128MB to 10GB, you decide which values). Then it will analyze all the execution logs and suggest you the best power configuration to minimize cost and/or maximize performance.'

Power Tuning

Step Functions X Execution started successfully X

State machines Activities

Developer resources

- Online learning workshop
- Local Development
- Data flow simulator
- Feature spotlight
- Documentation

Join our feedback panel

Graph view Table view

Graph view Actions ▾

Optimizer Test state

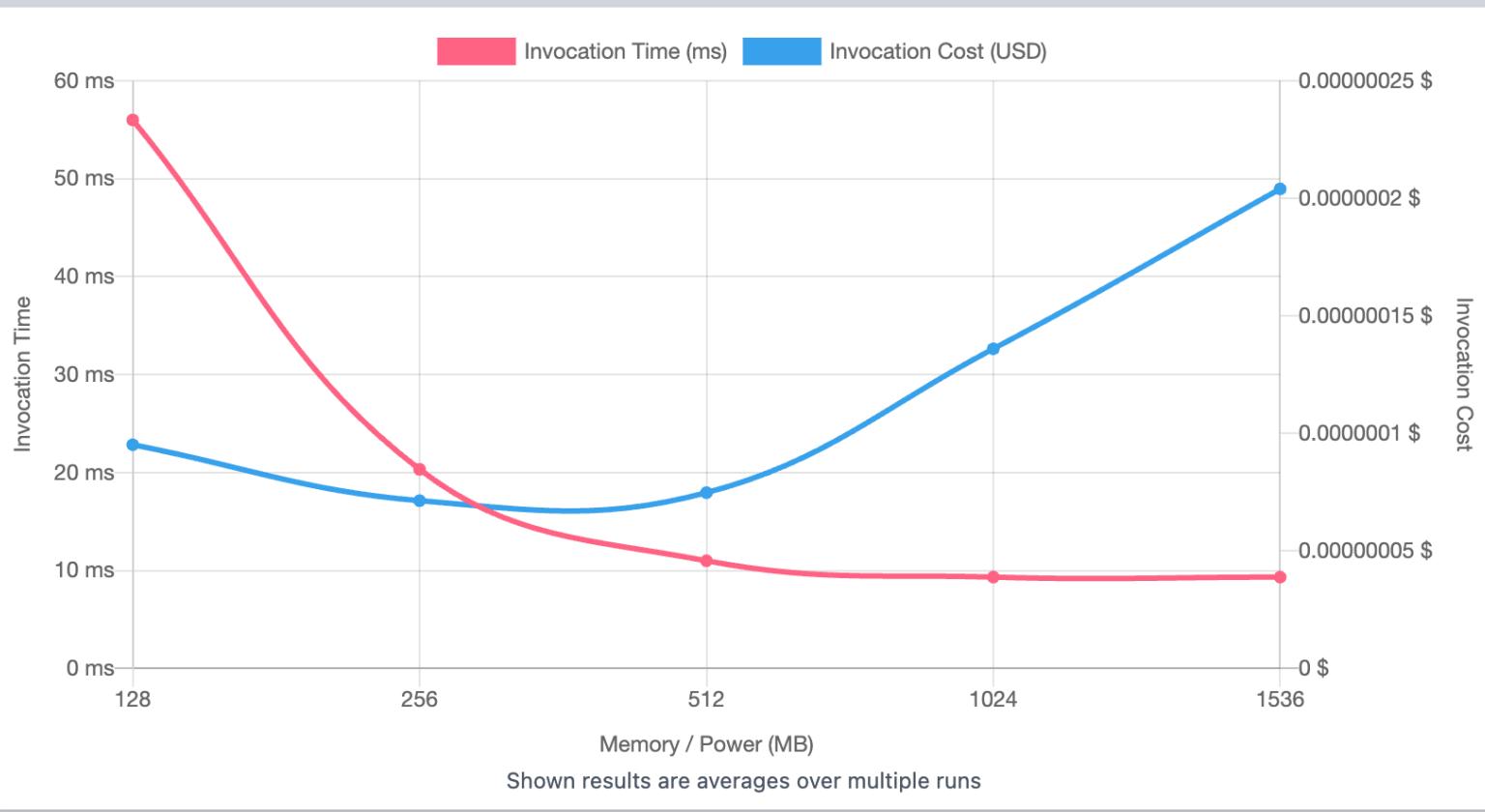
Logs | Lambda | Log group

Input Output Details Definition Events

```
1 {  
2   "power": 256,  
3   "cost": 7.13999999999999e-8,  
4   "duration": 20.333333333333332,  
5   "stateMachine": {  
6     "executionCost": 0.00028,  
7     "lambdaCost": 0.000265489,  
8     "visualization": "https://lambda-power-  
tuning.show/#gAAAAQACAAQABg==;AABgQquqokEAADBVVUVQVVVFUE=;wXDMM5FUmT08  
oaAzZQcSNBgLWzQ="  
9   }  
10 }
```

Power Tuning

AWS Lambda Power Tuning Results



Best Cost
256MB

Best Time
1024MB

Worst Cost
1536MB

Worst Time
128MB

Compare

Code Optimisation

```
src > adapters > secondary > customer > TS get-customer.adapter.ts > [🔗] getCustomer
1  import { Customer } from '@models/customer';
2  import { connect } from '@shared/databases-services/customer-service/connection';
3  import { customerService } from '@shared/databases-services/customer-service/customer/services';
4
5  export const getCustomer = async (id: string): Promise<Customer> => {
6    |   console.log('Secondary Adapter: Connecting to database');
7    |   💡 await connect();
8
9    |   console.log('Secondary Adapter: Retrieving customer Id');
10   |   const { _id, ...customer } = (
11     |       await customerService.getCustomer(id)
12     |   ).toObject();
13
14   |   return customer as Customer;
15 };
16
```

Code Optimisation

```
src > shared > databases-services > customer-service > connection > TS connect.ts > ...
1  import { connect as mongooseConnect, ConnectOptions, Mongoose } from 'mongoose';
2  import { fetchConnectionSecret } from './fetch-connection-secret';
3  import { config } from '@config/config';
4
5  let connection: Mongoose;
6
7  const secretName = config.get('counterDbConnectionSecret');
8
9  const defaultConnOptions: ConnectOptions = {
10    // Options such as pool size, ssl etc
11  };
12
13 export async function connect(options: ConnectOptions = {}) {
14  if (connection) {
15    return connection;
16  }
17
18  console.log('DB Service: Fetching Connection String');
19  console.log(`DB Service: **** ${secretName}`);
20  const dbConnectionString = await fetchConnectionSecret(secretName);
21
22  console.log('DB Service: Connecting to database');
23  connection = await mongooseConnect(dbConnectionString, {
24    ...defaultConnOptions,
25    ...options,
26  });
27  console.log('DB Service: Connected');
28
29  return connection;
30 }
```

Code Optimisation

```
src > shared > databases-services > customer-service > connection > TS connect.ts > ...
1   import { connect as mongooseConnect, ConnectOptions, Mongoose } from 'mongoose';
2   import { fetchConnectionSecret } from './fetch-connection-secret';
3   import { config } from '@config/config';
4
5   let connection: Mongoose;
6
7   const secretName = config.get('counterDbConnectionSecret');
8
9   const defaultConnOptions: ConnectOptions = {
10    // Options such as pool size, ssl etc
11  };
12
13  export async function connect(options: ConnectOptions = {}) {
14    if (connection) {
15      return connection;
16    }
17
18    console.log('DB Service: Fetching Connection String');
19    console.log(`DB Service: **** ${secretName}`);
20    const dbConnectionString = await fetchConnectionSecret(secretName);
21
22    console.log('DB Service: Connecting to database');
23    connection = await mongooseConnect(dbConnectionString, {
24      ...defaultConnOptions,
25      ...options,
26    });
27    console.log('DB Service: Connected');
28
29    return connection;
30  }
```

Code Optimisation

```
src > shared > databases-services > customer-service > connection > TS fetch-connection-secret.ts
  1 import {
  2   GetSecretValueCommand,
  3   SecretsManagerClient,
  4 } from '@aws-sdk/client-secrets-manager';
  5
  6 const client = new SecretsManagerClient();
  7
  8 interface ConnectionSecret {
  9   dbConnectionString: string;
 10 }
 11
 12 export const fetchConnectionSecret = async (
 13   secretName: string
 14 ): Promise<string> => {
 15   const response = await client.send(
 16     new GetSecretValueCommand({
 17       SecretId: secretName,
 18     })
 19   );
 20   const connectionSecret: ConnectionSecret = JSON.parse(
 21     response.SecretString || ''
 22   );
 23
 24   if (!connectionSecret || !connectionSecret.dbConnectionString) {
 25     throw new Error('Unable to parse connection details');
 26   }
 27
 28   return connectionSecret.dbConnectionString;
 29 }
 30
```

Code Optimisation

Also, Avoid sizable packages where possible.

Such as Sequelize ORM

<https://www.npmjs.com/package/sequelize>



Sequelize

Sequelize is a modern TypeScript and Node.js ORM for Oracle, Postgres, MySQL, MariaDB, SQLite and SQL Server, and more. Featuring solid transaction support, relations, eager and lazy loading, read replication and more.

[Getting Started](#)

[API Reference](#)

[Upgrade to v6](#)

[Support us](#)

Performance – Throughput

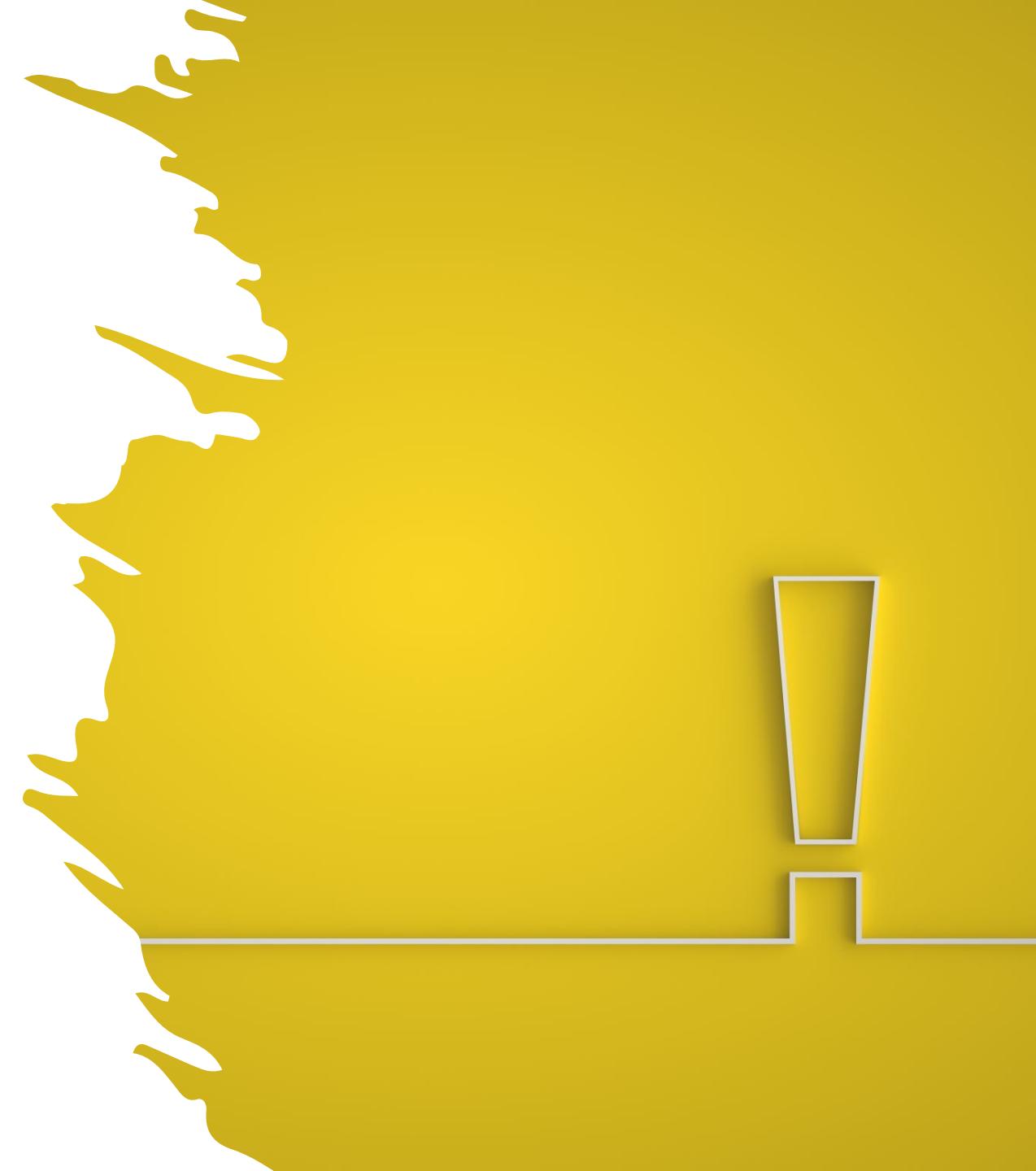
How many concurrent users are likely to use this application. We could have 100 users, 500 users or 10,000 users.

For Throughput -

Number of active users VS response time.

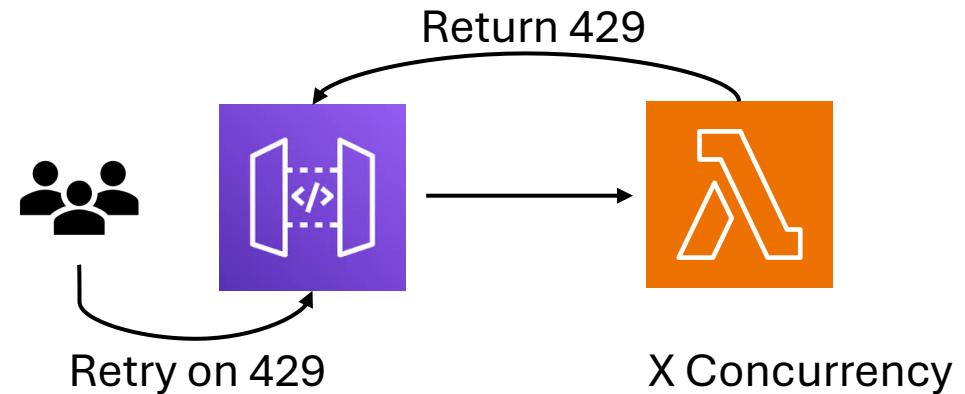
We can then use both to generate a baseline.

For example, the system needs to manage: *100 requests per second.*



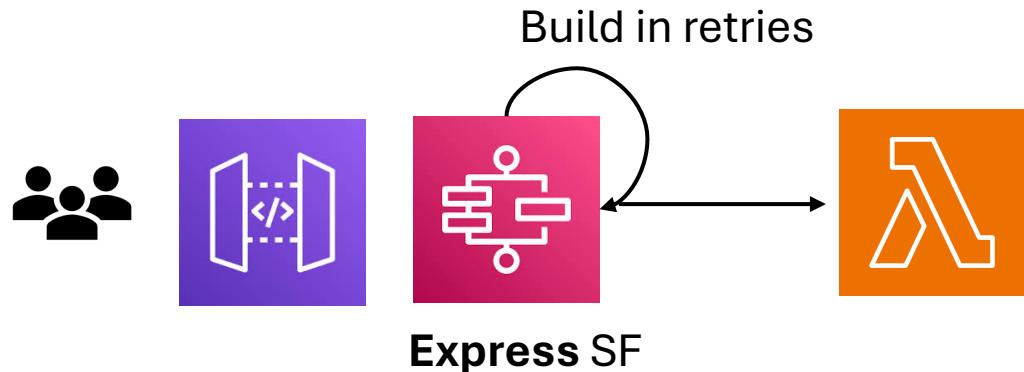
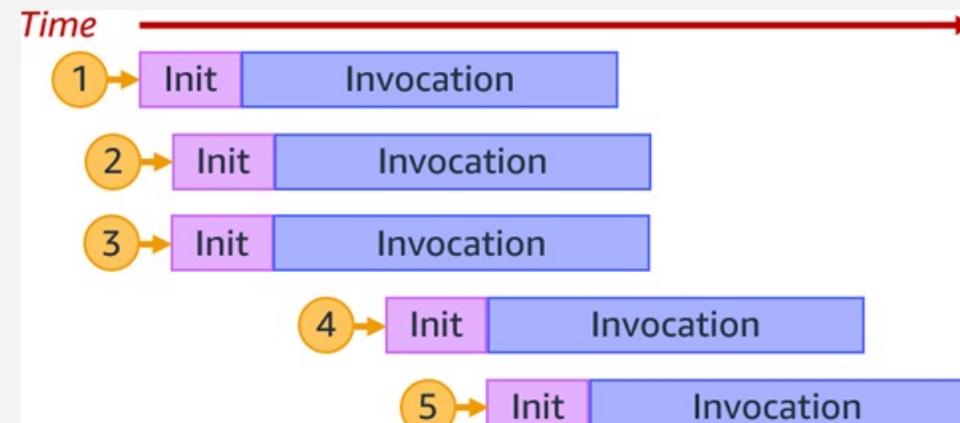
Performance –

- Managing throughput with lambda throttling



Synchronously

- Concurrent Executions - Default 1,000 per account – Soft Limit, up to tens of thousands



Express SF

Performance –

- Throttling our lambda input

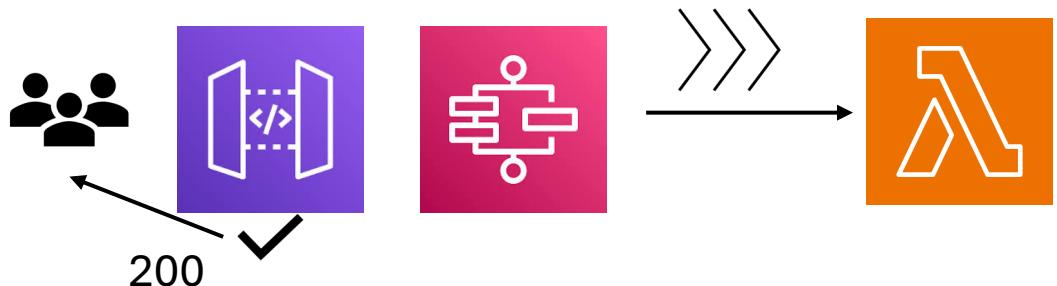
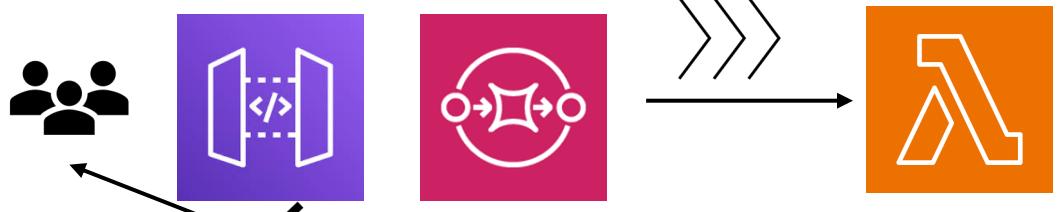
Asynchronously

- Concurrent Executions - Default 1,000 per account – Soft Limit, up to tens of thousands

Event Driven



Storage First





Scalability

Scalability is often measured with how the system reacts to an increase in load.

Typically, we load test our application so we are confident it will scale in production.

How many concurrent users or actions does the system need to support? What spikes will occur?

Special Events

- **Black Friday Sale**
- **Product Release**
- **Payday**
- **Advertising**



Typical Users



Unexpected
Users



Lots of HTTP 429 Errors

Reserved
Concurrency
100



Scalability

Scalability is often left behind and taken for granted with serverless applications. Engineers typically say, “lambda can scale indefinitely”, “DynamoDB will manage any throughput”.

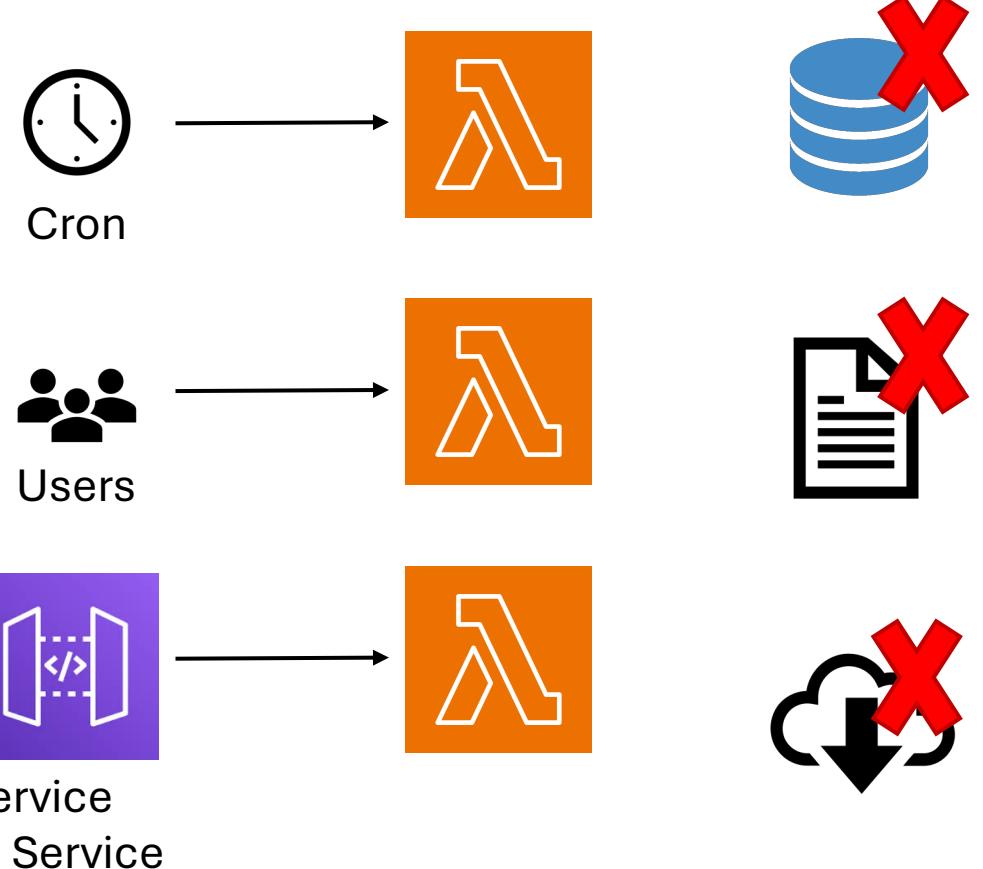
Well not without complications... scaling up for these services takes time, and your application will receive scalability errors (HTTP 429 -Too Many Requests) during the scaling out (horizontal scaling) period.

You may also depend on another service, or read from a file, call a downstream API.

All of these instances can cause your serverless applications to fail scalability checks.

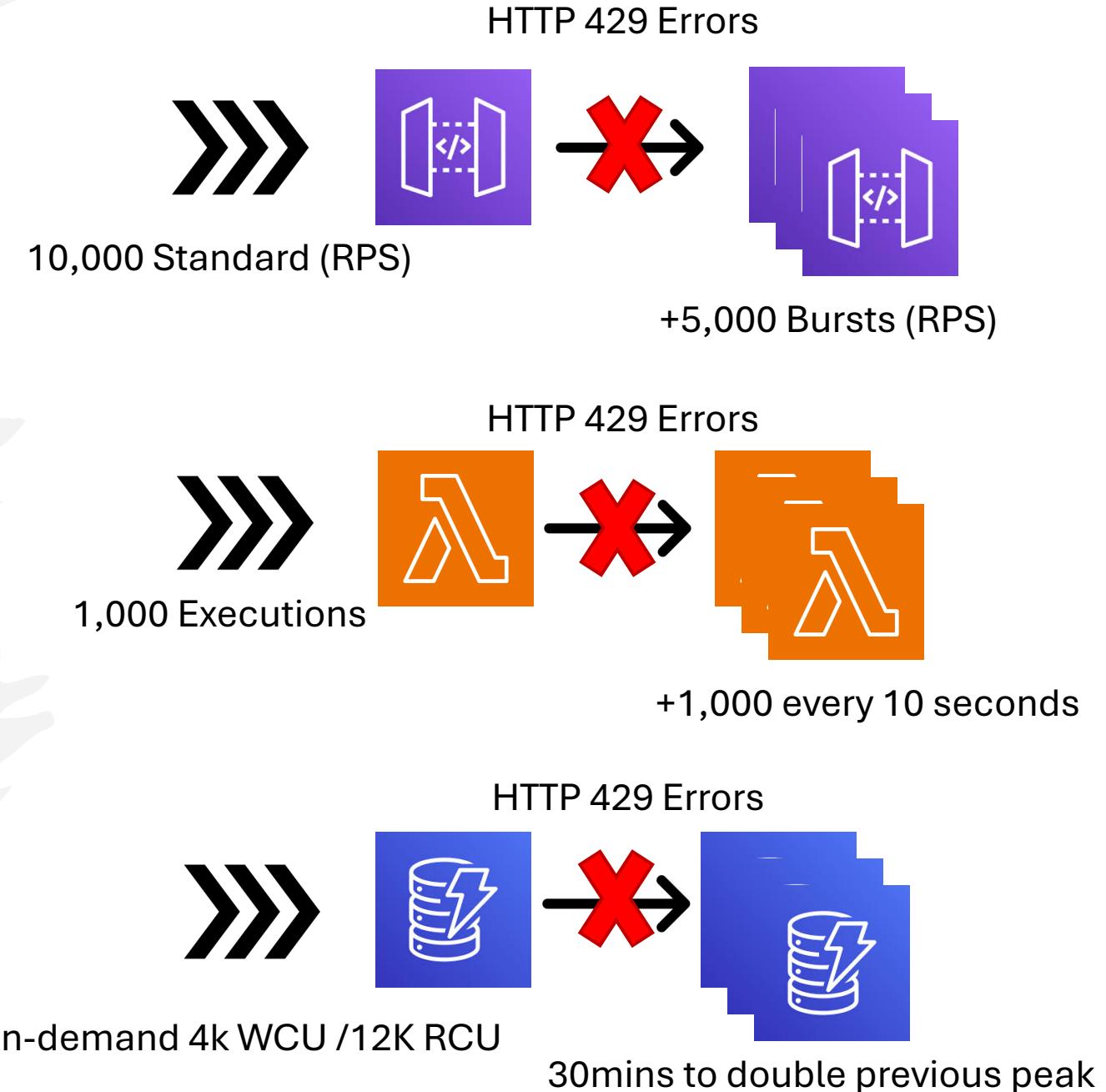
Scalability – Common Problems

- The dependency bottleneck
 - Database connections hit
 - File limits reached
 - Capability of External APIs



Scalability – Often taken for granted...

- Service Scaling
 - API Gateway
 - DynamoDB
 - Lambda



Scalability/ Throughput

Remember we mentioned Scalability with Throughput...

To scale safely, a well-designed system should be implementing at minimum some exception handling including basic back-off and retry.

```
22  async function getCustomerCall(id: string) {
23    try {
24      return await customerService.getCustomer(id);
25    } catch (error) {
26      // Advanced - Check error code. i.e 429
27      if (attempt < 3) {
28        attempt += 1;
29        const backoffDelay = 100;
30        console.warn(
31          `Request failed. Retry attempt ${attempt} after ${backoffDelay}ms`
32        );
33        await new Promise((resolve) => setTimeout(resolve, backoffDelay));
34        return getCustomerCall(id);
35      }
36    }
37    throw error;
38 }
```



The header features a black background with yellow geometric shapes resembling shards or arrows pointing upwards. In the top right corner is a circular portrait of a man with short brown hair, wearing a light-colored shirt and a lanyard, standing in front of a dark wooden paneled wall with a circular emblem. To his left, the text "Robert Bulmer" is written in bold yellow, with "ArchiTechInsights.com" in a smaller white sans-serif font below it. There are also small yellow arrow-like icons pointing towards the text.

In Summary

It's often easy to miss off Non-Functional Requirements, especially when teams are under pressure of delivery.

They are critical to the success of a software system because they define the system's operational qualities and constraints, ensuring it meets user needs and performs well under expected conditions.

Ignoring NFRs can lead to significant issues post-deployment, such as system crashes, security breaches, poor user satisfaction, and costly maintenance.

Properly defining and addressing NFRs from the beginning ensures a robust, secure, and efficient system that meets user expectations and performs well in real-world conditions.