

GOTO AMSTERDAM 2024

#GOTOams

Developing highly scalable
image storage solution with
AWS Serverless at ip.labs

by Firdaws Aboulaye and
Vadym Kazulkin



GOTO
Guide

LET US HELP YOU

Ask questions
through **the app**



also remember to rate session



THANK YOU!

#GOTOams

Contact



Vadym Kazulkin

ip.labs GmbH Bonn, Germany



Co-Organizer of the Java User Group Bonn



v.kazulkin@gmail.com



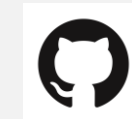
<https://www.linkedin.com/in/vadymkazulkin>



[@VKazulkin](https://twitter.com/VKazulkin)



<https://dev.to/vkazulkin>



<https://github.com/Vadym79/>



<https://de.slideshare.net/VadymKazulkin/>

Contact



Firdaws Aboulaye

ip.labs GmbH Bonn, Germany

Software Engineer



aboufred@gmail.com



<https://www.linkedin.com/in/faboulaye>



<https://github.com/faboulaye>

About ip.labs



Agenda

Application architecture back in 2017 in the datacenters

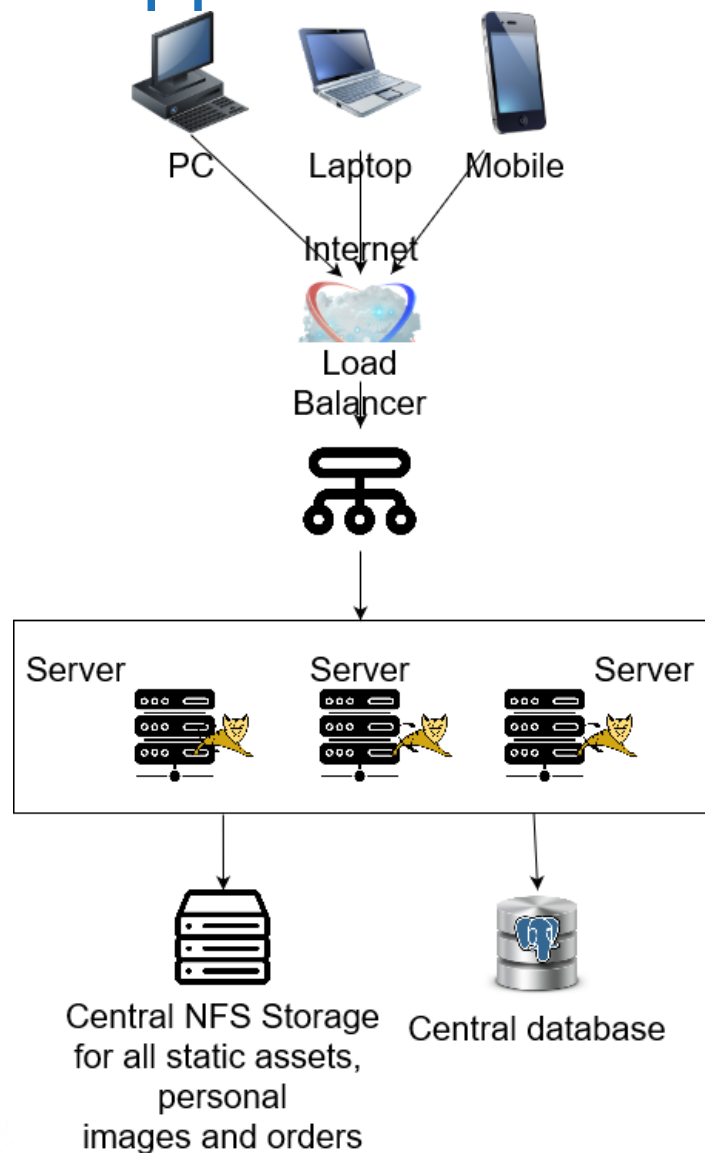
Application architecture back in 2018-2019 after initial migration to AWS

Motivation behind the reimplementation of our image storage solution to one based on AWS Serverless services

Current architecture of our image storage solution

Challenges and lessons learned

Application architecture back in 2017 in the datacenters

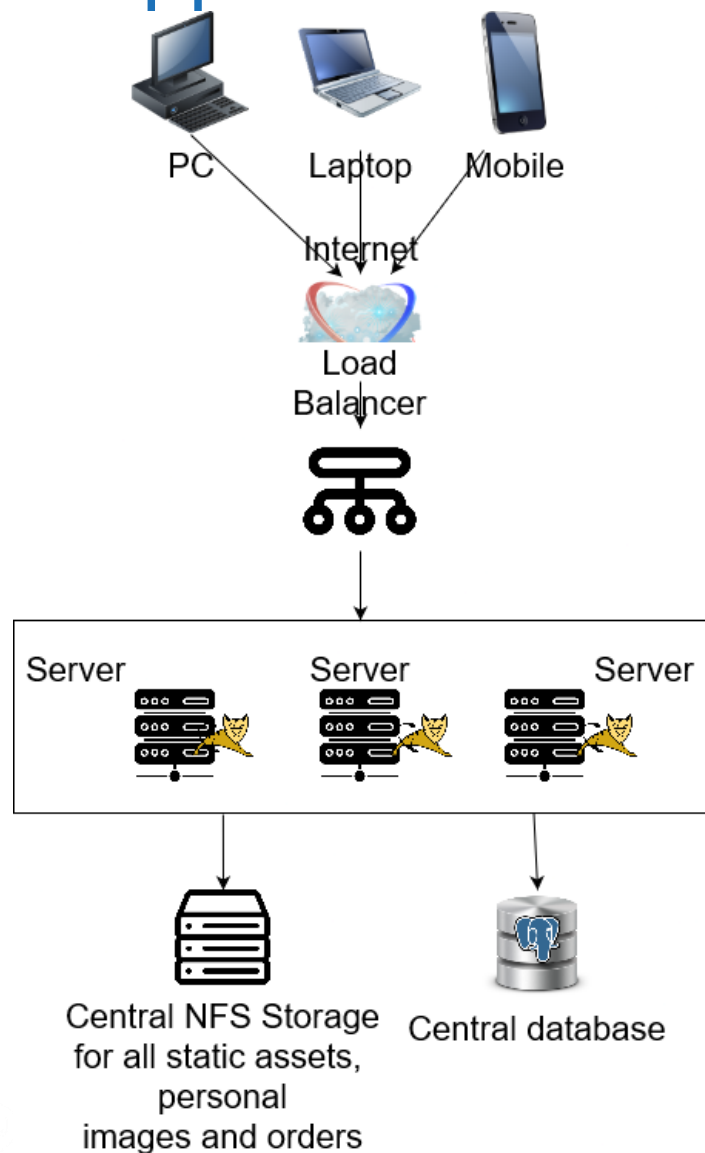


Challenges with the operations and application:

- The same application was deployed in different datacenters across the globe
- Mainly in Europe and Asia
- Different vendors for hardware (load balancer, virtualization, NFS storage) used



Application architecture back in 2017 in the datacenters

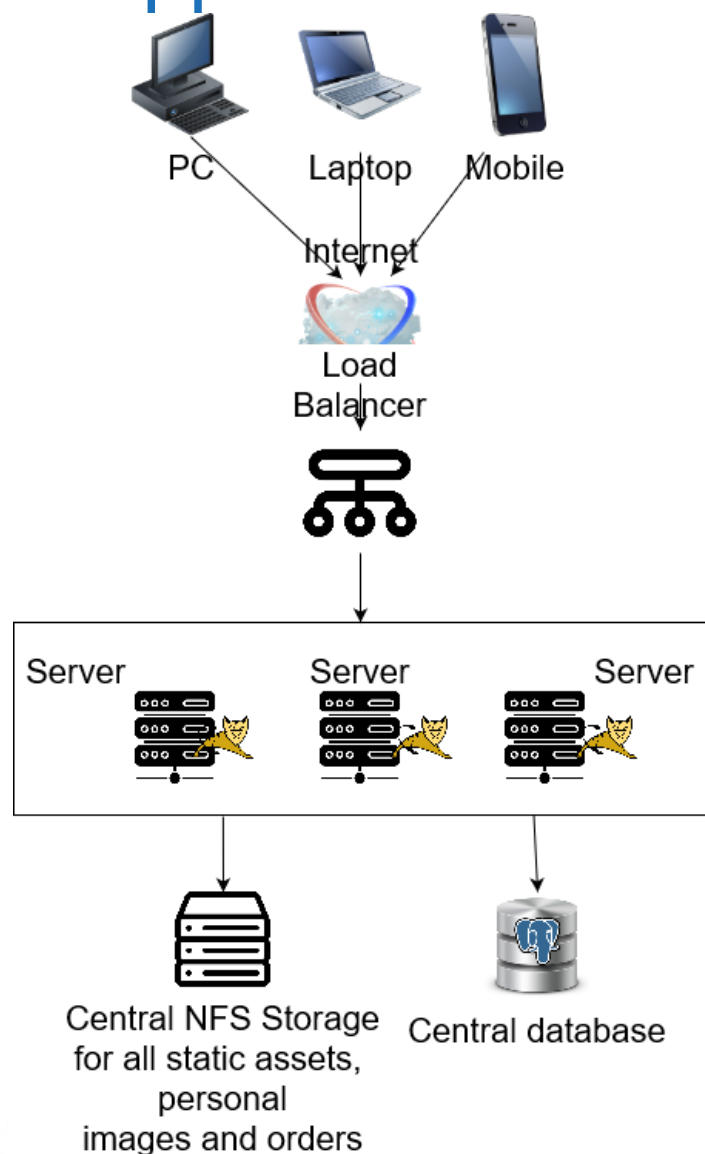


Challenges with the operations and application:

- No complete internal access to all datacenters
- No setup and deployment automation in place



Application architecture back in 2017 in the datacenters

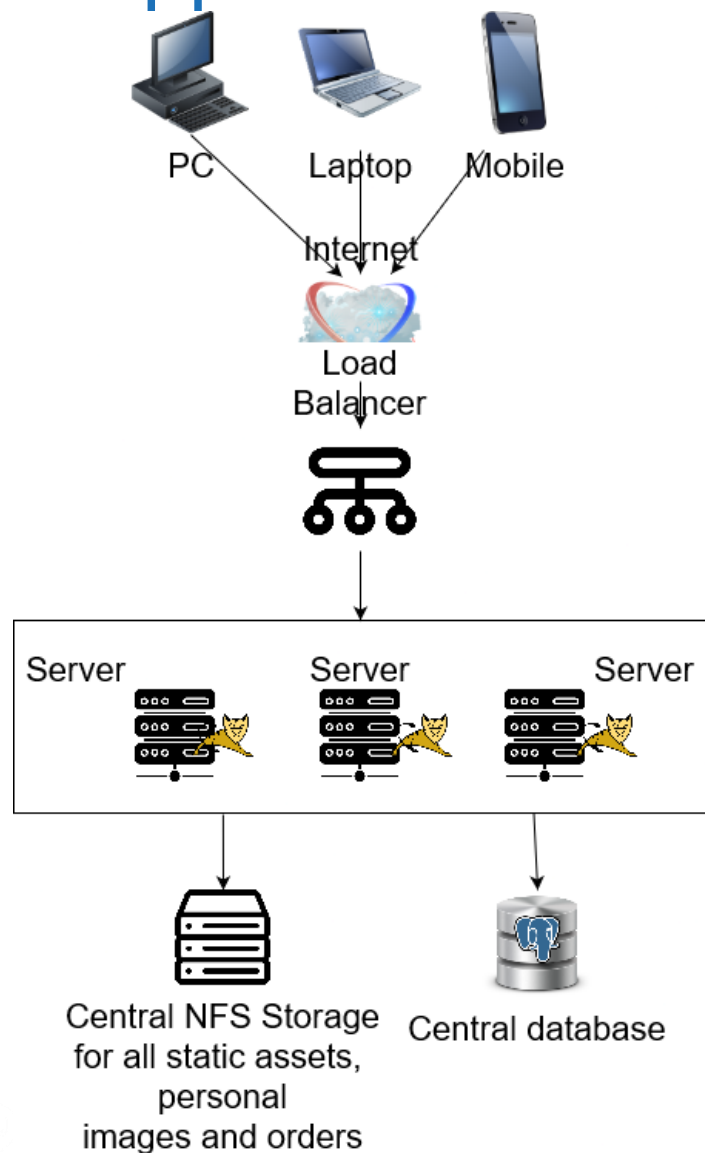


Challenges with the operations and application:

- Difficulties to rollout our product globally with the common stack
- No undependended staging and test setup possible
 - Central load balancer and NFS storage
- Developers hardly understood operations

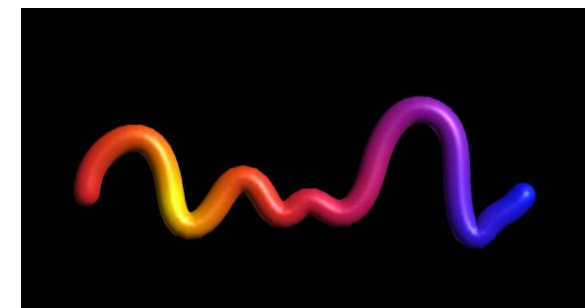


Application architecture back in 2017 in the datacenters



Challenges with the operations and application:

- Scalability issues due to the spikey business
- Related to number of uploaded and downloaded images
- Mainly because of the central NFS storage in use



AWS Migration

Main focus of migration was:

- Use the same AWS services in all regions
 - The ability to go globally and have the complete internal ownership of operations
- Automate everything
 - Use Infrastructure as a Code
- Provide undependend production-near staging and test environments



AWS Migration

Main focus of migration was:

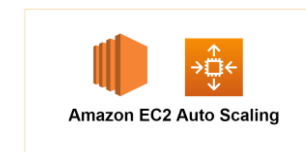
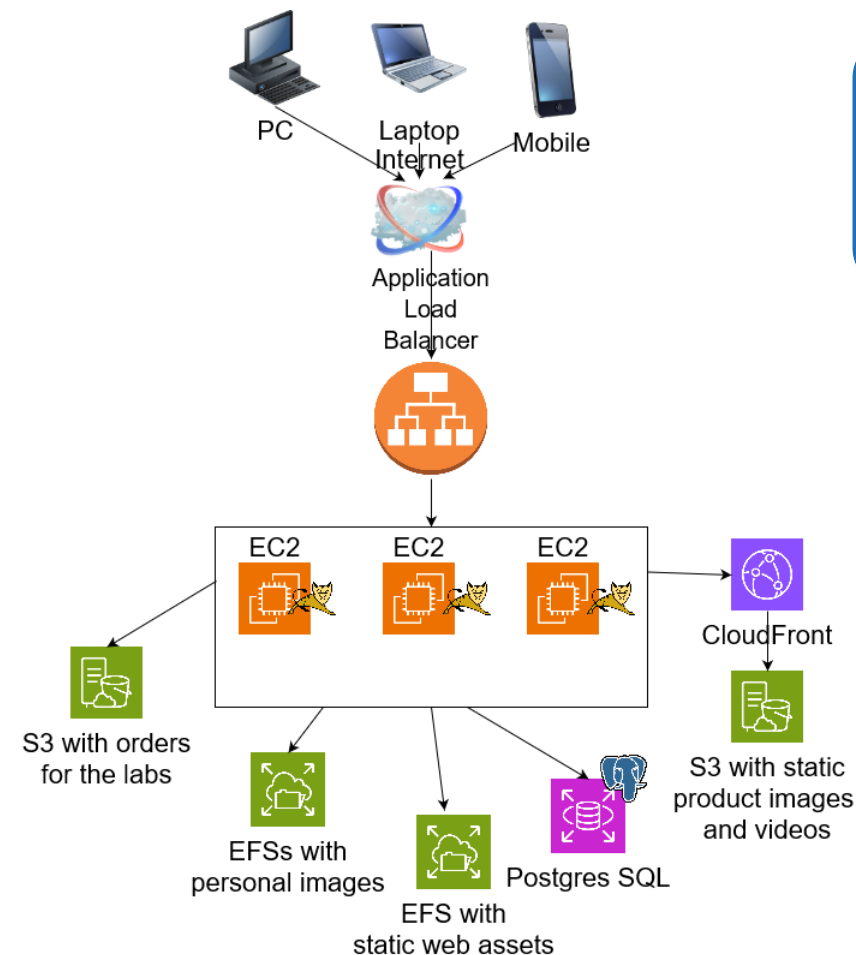
- Upskill developers with AWS operational skills
- Improve stability of the image storage related parts of the application
- Increase speed of delivery



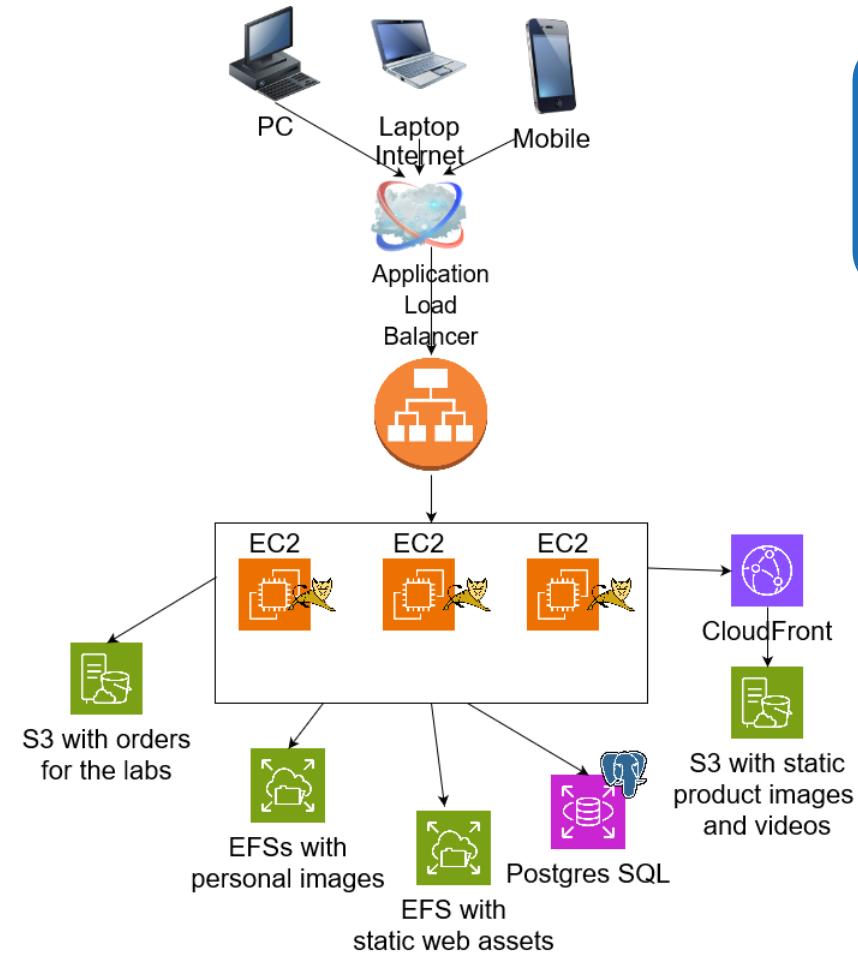
Application architecture back in 2018-2019 after initial migration to AWS

During the migration to AWS we:

- Automated everything
- On the infrastructural and deployment level
- No hot patches, only immutable infrastructure
- Introduced several independent core microservices
- Used different auto-scaling policies for them



Application architecture back in 2018-2019 after initial migration to AWS



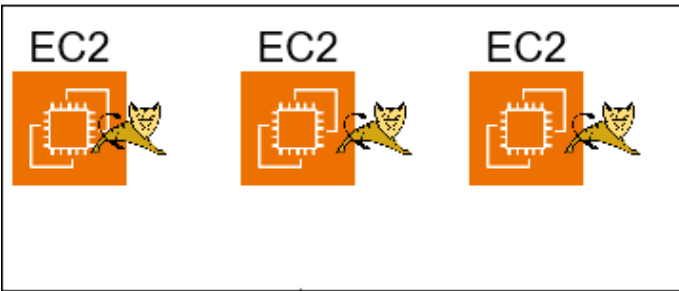
During the migration to AWS we:

- Separated the storage for static web assets, static product images and videos and personal images
- Separated image uploads and order downloads by the labs
- Improved and but not completely solved stability issues with image storage for personal images
- Set base to increase speed of delivery



Motivation behind the reimplementing of our image storage solution

Application
Load
Balancer



EFSs with
personal images

Technical challenges remained:

- Web Application deployed on each EC2 server had front and backend bundled together
- Frontend mainly contained photo product builders/editors and shop frontend



Motivation behind the reimplementing of our image storage solution

Application
Load
Balancer



EC2



EC2



EC2



EFSs with
personal images

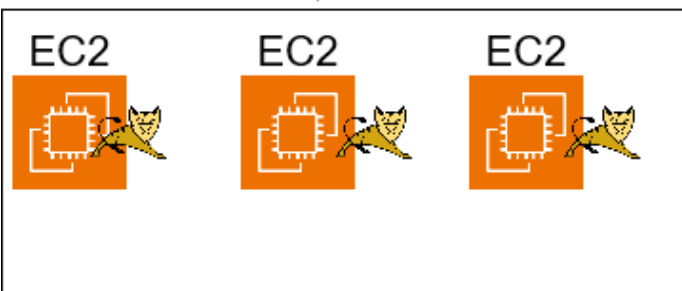
Technical challenges remained:

- Backend contained nearly all internal services bundled together as monolith
- Photo product creation, save and load of cloud products, user management, payment, ordering, external integrations



Motivation behind the reimplementation of our image storage solution

Application
Load
Balancer



EFSs with
personal images

Technical challenges remained:

- Central EFS for image upload/download still used from all EC2 servers
- Scalability issues on the shared EFS propagates to all EC2 servers and make the main application unavailable



Motivation behind the reimplementing of our image storage solution

Application
Load
Balancer



EC2



EC2



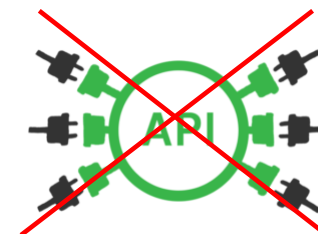
EC2



EFSs with
personal images

Technical challenges remained:

- No clearly defined APIs for image storage solution
- Low testability of the image storage solution
- No clearly ownership for image storage solution



Motivation behind the reimplementation of our image storage solution

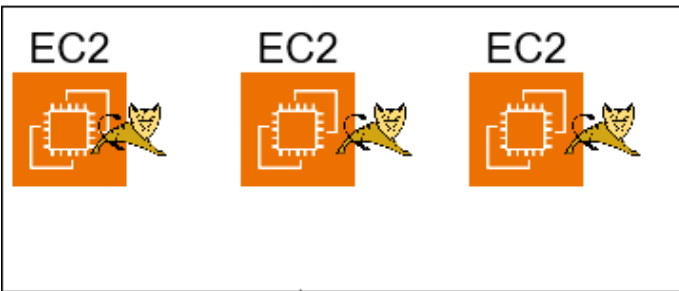
Product/Strategical challenges remained:

- Our main application also contained completely self-written eCommerce system
- Shop frontend, shop backend, user management, payment and ordering workflow, statistics/BI
- Our application contained external APIs to enable User SSO and External Cart integrations
- Integration and maintenance costs were high for our customers

CUSTOM
E-COMMERCE
DEVELOPMENT



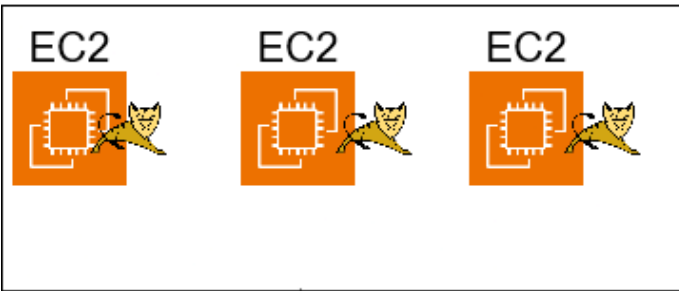
Application
Load
Balancer



EFSs with
personal images

Motivation behind the reimplementing of our image storage solution

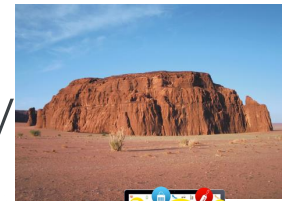
Application
Load
Balancer



EFSs with
personal images

Product/Strategical challenges remained:

- Business expressed the need to support direct integrations with the popular eCommerce solutions
- Magento/Adobe Cloud, Shopify
- Image handling/workflow was a part of this monolithical front and backend
- Although it provided generic functionality it was tightly coupled to the self-written eCommerce solution



CUSTOM
E-COMMERCE
DEVELOPMENT



EFSs with
personal images

Why we chose AWS Serverless for image the storage solution?

- We already experimented with AWS Serverless services for the new development on a smaller scale since our early days in AWS in 2018
- We saw a lot of benefits to fully utilize the power of the AWS cloud
 - further increasing the speed of delivery
 - focus on our core capabilities by relying on the AWS managed services

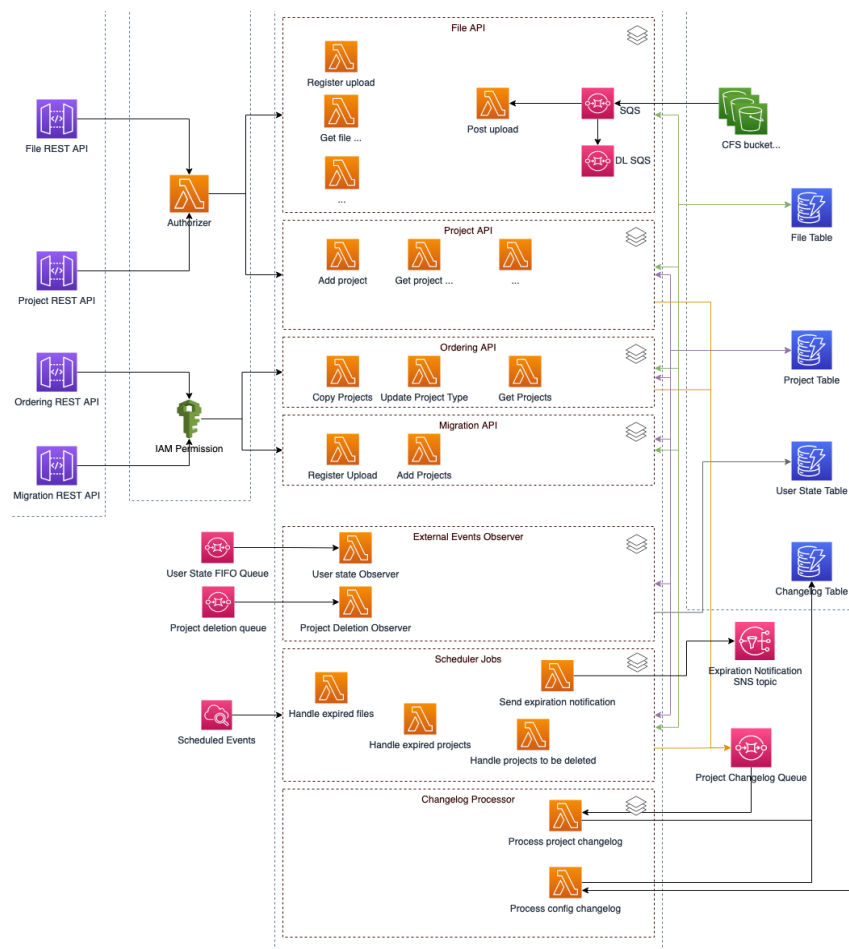


Why we chose AWS Serverless for image the storage solution?

- It became also a cultural thing
 - Developers started to learn, build and share knowledge in AWS Serverless



Current architecture our image storage solution



File storage modules

- File API
- Project API
- Ordering API
- Scheduler Jobs
- Migration API
- ...

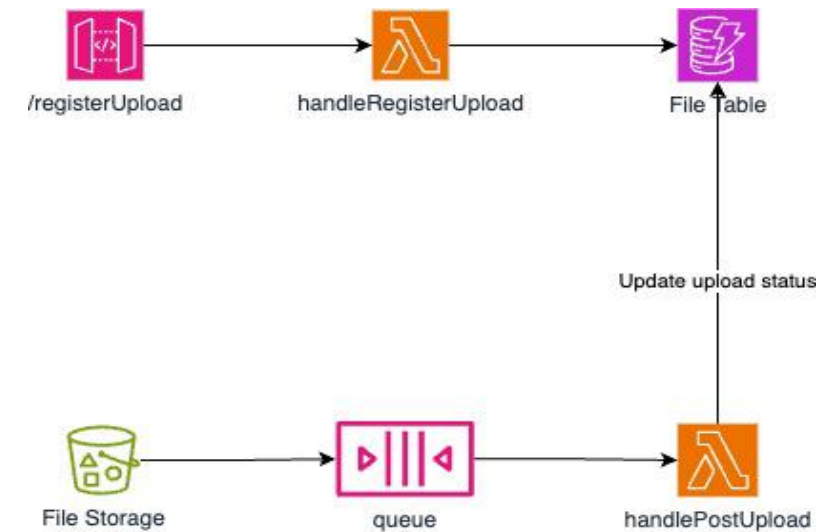
File Storage - modules

File API

Handles customer file operations, secured by Lambda authorizer.

Key actions include:

- registering uploads
- retrieving file information
- generating S3 pre-signed URLs.



File Storage - modules

Project API

- Manages customer projects and cart items, secured by Lambda authorizer.
- Supports adding, updating, renaming and retrieving projects.

Main concerns

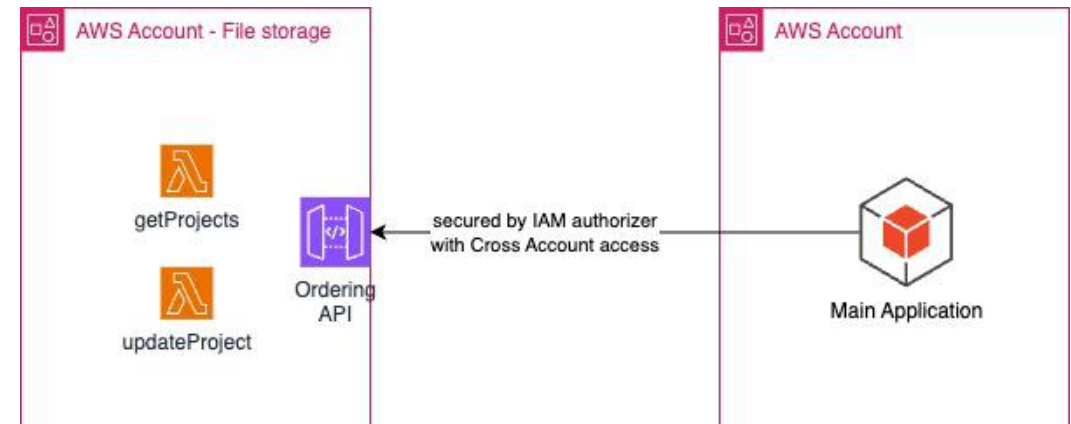
- Project content too big to be saved in DynamoDB
- Eventual consistency
- ✓ Save the content in S3 and store the object key in DynamoDB

File Storage - modules

Ordering API

Prepare data for ordering process

- creating project copies
- retrieving projects by ID
- updating project types.

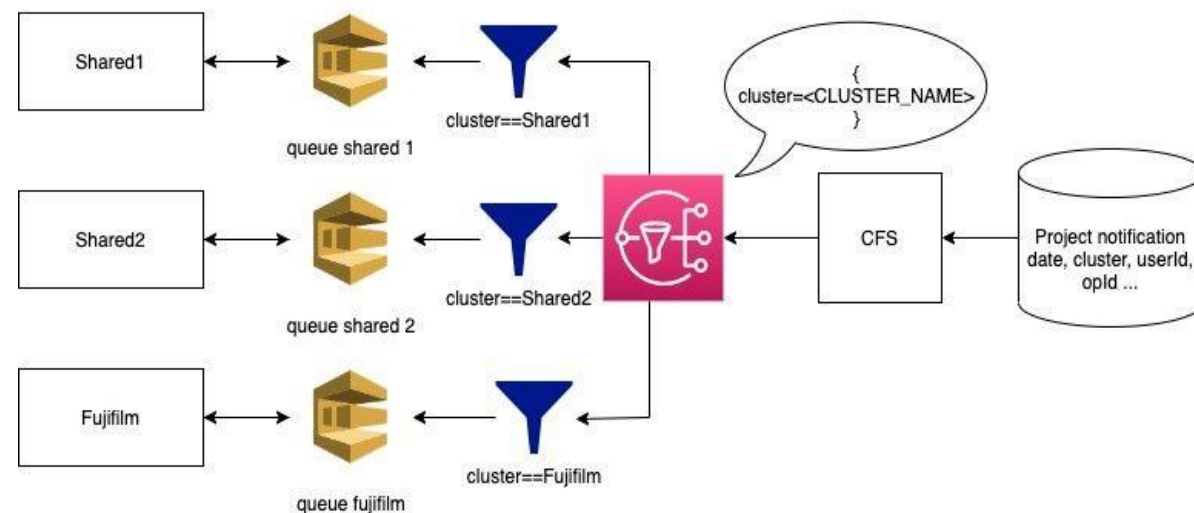


File Storage – modules

Scheduler Jobs

- Projects and Files Expiration management
- Send notification before expiration
- Cleaning job

Publish/Subscribe Fan-Out Pattern in Serverless Architectures



File Storage – modules

Migration API

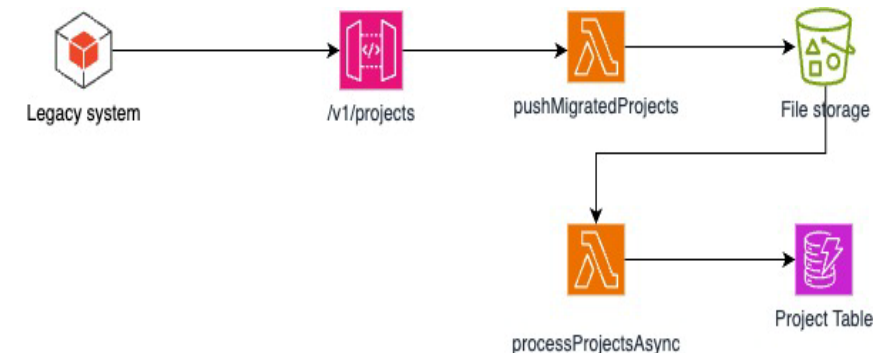
Migrate existing projects from the legacy system to the new file storage.

Type of migration:

- Batch mode
- On demand

Challenges

- Timeout during migration
- Retry on failed migrated project



Security



User data and authentication are not handled by the file storage



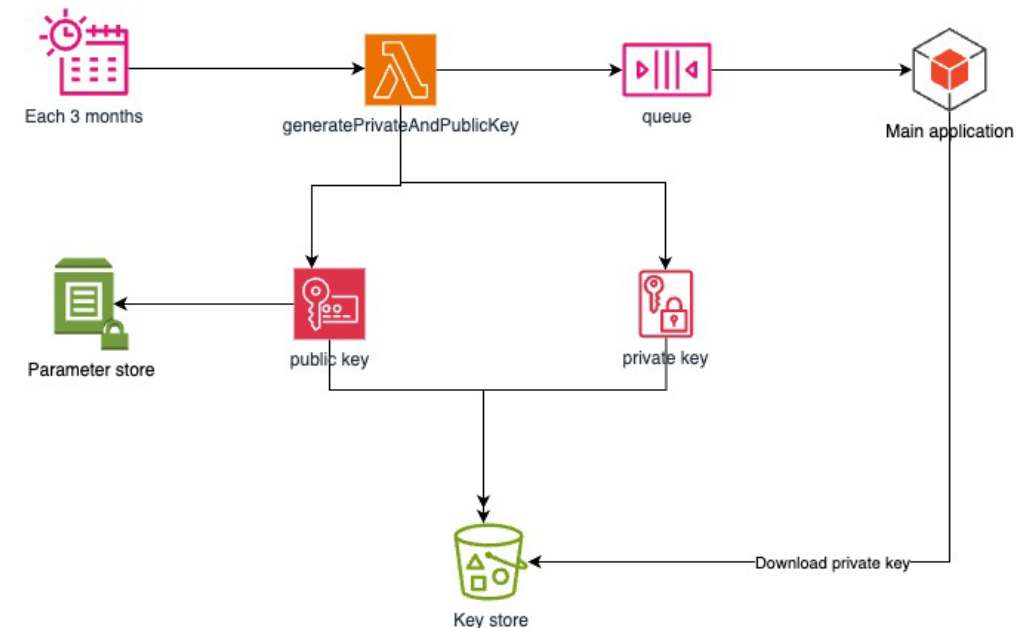
Secure request via token validation (Lambda authorizer) or IAM authorization



Private and public keys are used to generate and validate token. These keys are renewed each 3 month

Security – Renew private / public keys

- Rotate private and public keys every three months
- Notify legacy software to retrieve the private key from S3



Additional benefits of this architecture

- Enables seamless integration of file storage into eCommerce platforms (Magento, Shopify)
- Improved reporting of project with more metrics
- Simple to extend functionality with the current file storage APIs
 - Picture uploads through QR code scanning
 - Introduce project sharing capabilities

Challenges and lessons learned



Architect with AWS Serverless quotas and technical concepts in mind



General architectural decisions

Aurora Serverless vs DynamoDB

DynamoDB Provisioned Throughput vs On-Demand Capacity



Challenging to trace all requests

Usage of X-Ray add another complexity on the code

Service Quotas

Service Quotas



Dashboard

AWS services

Quota request history

Organization

Quota request template

Dashboard

[Modify dashboard cards](#)

Amazon Athena

Total quotas: 6



Amazon DynamoDB

Total quotas: 9



Amazon Elastic Block Store (Amazon EBS)

Total quotas: 40



Amazon Elastic Compute Cloud (Amazon EC2)

Total quotas: 137



Amazon Relational Database Service (Amazon RDS)

Total quotas: 28



Amazon Virtual Private Cloud (Amazon VPC)

Total quotas: 27



AWS CloudFormation

Total quotas: 24



AWS Key Management Service (AWS KMS)

Total quotas: 53



AWS Lambda

Total quotas: 22

Pending service quota requests (0)

Recently resolved service quota requests (0)

Service Quotas

Service Quotas ×

[Dashboard](#)
[AWS services](#)
[Quota request history](#)
▼ **Organization**
[Quota request template](#)

[Service Quotas](#) > [AWS services](#)

AWS services

Service

[Access Analyzer](#)

[Amazon API Gateway](#)

[Amazon AppFlow](#)

Service Quotas

Service Quotas ×

[Dashboard](#)
[AWS services](#)
[Quota request history](#)
▼ **Organization**
[Quota request template](#)

[Service Quotas](#) > [AWS services](#) > [Amazon API Gateway](#)

Amazon API Gateway

Service quotas

Request quota increase

×

< 1 > ⚙

	Quota name ▲	Applied quota value	AWS default quota value	Adjustable ▼
☐	API Stage throttles in a usage plan	20	20	Yes
☒	Throttle burst rate	5.000	5.000	No
☐	Throttle rate	10.000	10.000	Yes

account and current region limits

Service Quotas

Concurrent executions

Request increase at account-l

Details

Description

The maximum number of events that functions can process simultaneously in the current Region.

Utilization

0

Applied quota value

1.000

AWS default quota value

1.000

Adjustability

Account-level

Monitoring

Request history

Alarms

Tags

Monitoring

This displays a customizable CloudWatch embedded graph. [Learn more](#)

1h

3h

12h

1d

3d

1w

Add to dashboard

Concurrent executions - (Utilization %)



Service Quotas Request History

[Service Quotas](#) > Quota request history

Recent quota increase requests

View open quota increase requests or requests that were closed in the last 90 days.

Q

Filter by service or quota

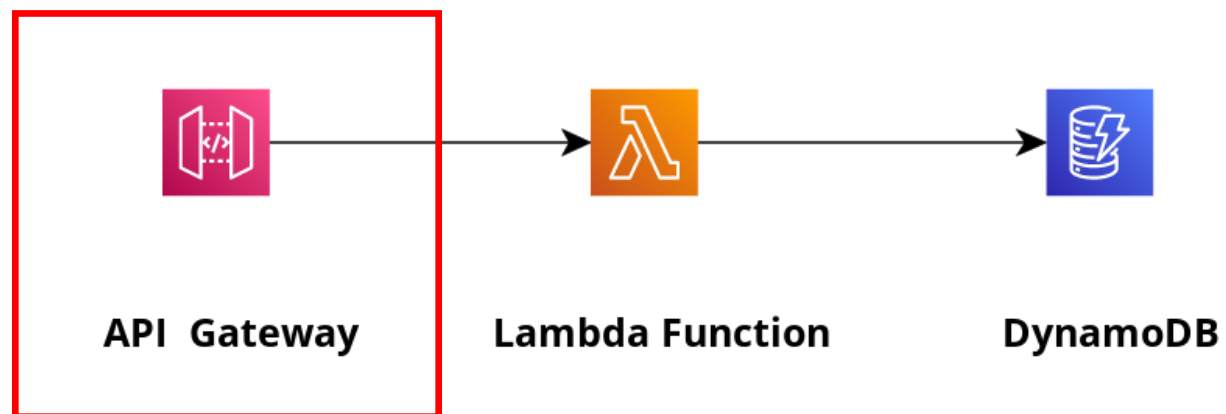
<

1

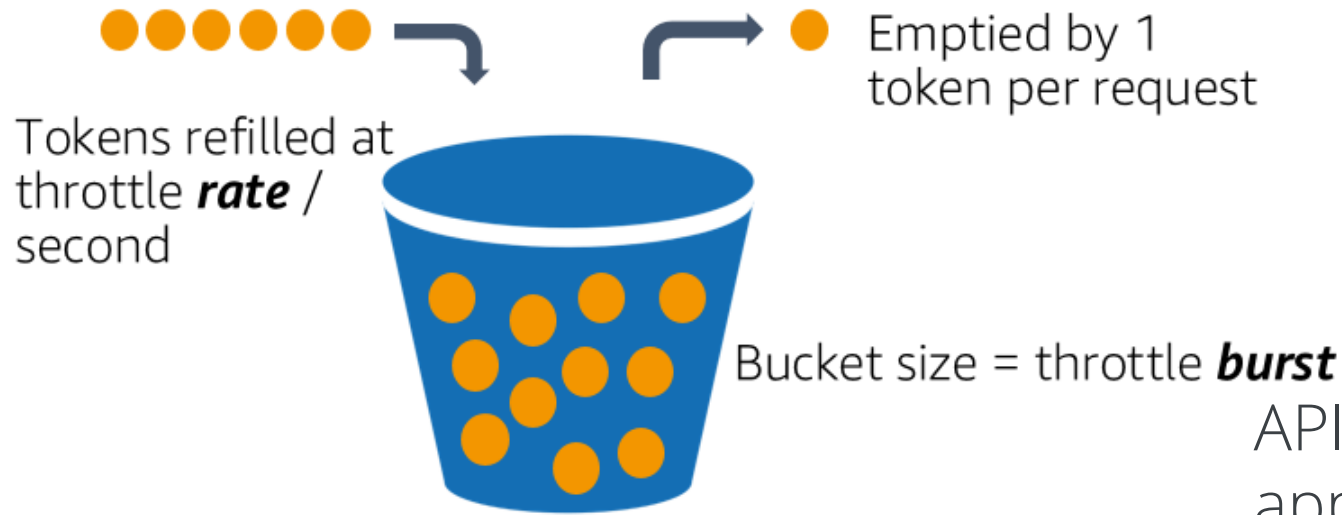
>

Service	Quota name	Status	Requested quota value	Request date	Last updated date
Amazon API Gateway	Throttle rate	Approved	30.000	5. Sept. 2023	5. Sept. 2023
AWS Lambda	Concurrent executions	Quota requested	3.000	5. Sept. 2023	5. Sept. 2023

Serverless Application



API Gateway Token Bucket Algorithm



Token bucket algorithm

- The **throttle rate** then determines how many requests are allowed per second
- The **throttle burst** determines how many **additional** requests are allowed per second

API Gateway throttling-related settings are applied in the following order:

- Per-client or per-method throttling limits that you set for an API stage in a usage plan
- Per-method throttling limits that you set for an API stage
- Account-level throttling per Region
- AWS Regional throttling

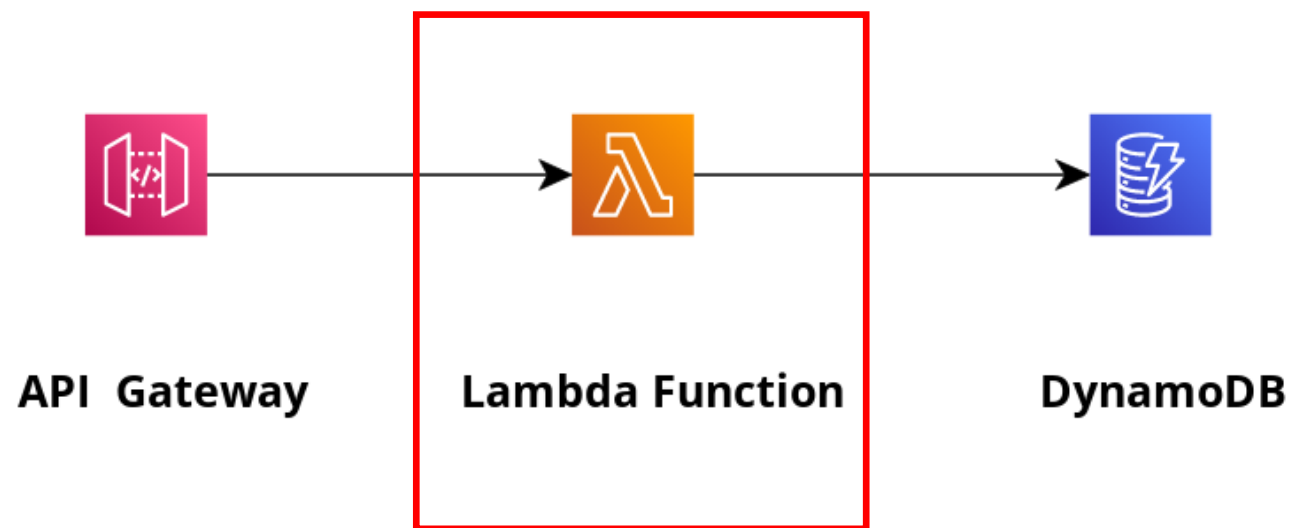
API Gateway Important Service Quotas

Quota	Description	Value	Adjustable
Default throughput / Throttle rate	The maximum number of requests per second that your APIs can receive	10.000	
Throttle burst rate	The maximum number of additional requests per second that you can send in one burst	5.000	

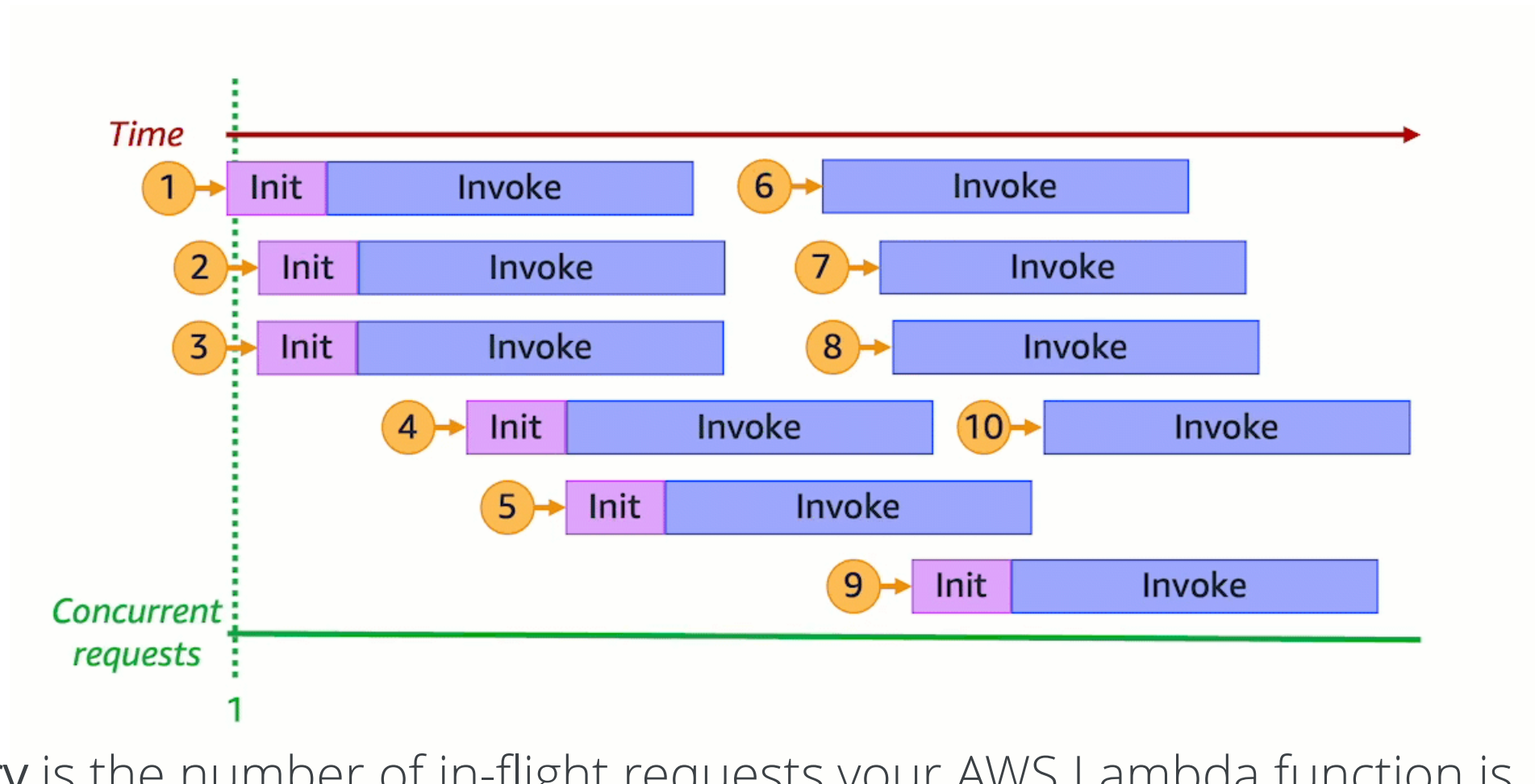
API Gateway Important Service Quotas

Quota	Description	Value	Adjustable	Mitigation
Max timeout	The maximum integration timeout in milliseconds	29 sec		1) Increase the limit 2) Lambda Function URL with response streaming
API Payload size	Maximum payload size for non WebSocket API	10 MB		1)The client makes an HTTP GET request to API Gateway, and the Lambda function generates and returns a presigned S3 URL 2)The client uploads the image to S3 directly, using the resigned S3 URL

Serverless Application



Lambda Concurrency

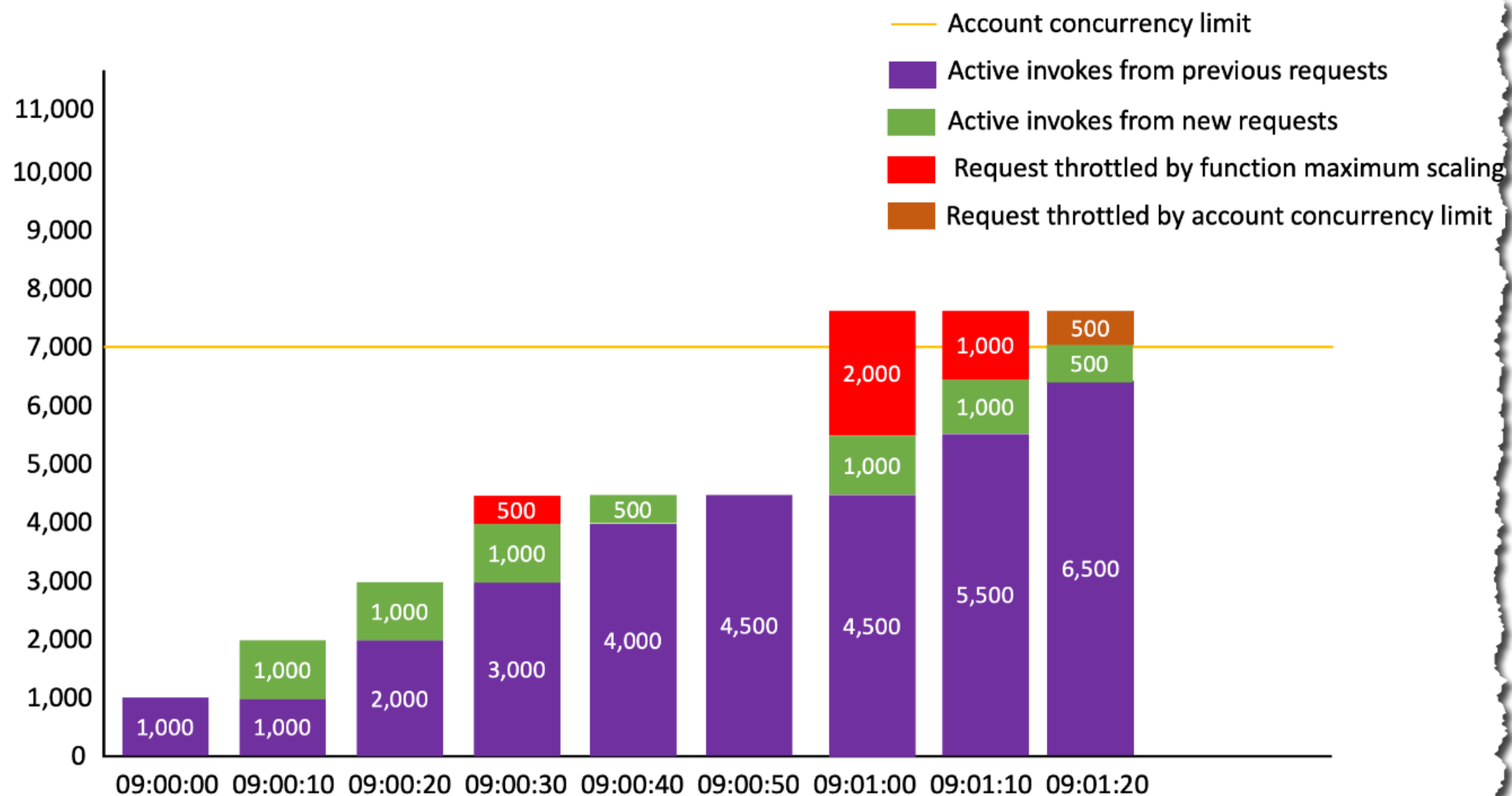


Concurrency is the number of in-flight requests your AWS Lambda function is handling at the same time

Lambda Important Service Quotas **New**

Quota	Description	Value	Adjustable	Mitigation
Concurrent executions/ Concurrency limit	The maximum number of events that functions can process simultaneously in the current region	1,000		Rearchitect
Burst Concurrency Limit	After the initial burst, concurrency scales by 1000 executions every 10 seconds up to your account concurrency limit. Each function within an account now scales independently from each other	- US West (Oregon), US East (N. Virginia), Europe (Ireland)=3,000 - Asia Pacific (Tokyo), Europe (Frankfurt), US East (Ohio)=1,000 - All other Regions=500		Use provisioned concurrency

Lambda Concurrency



Lambda Concurrency and TPS



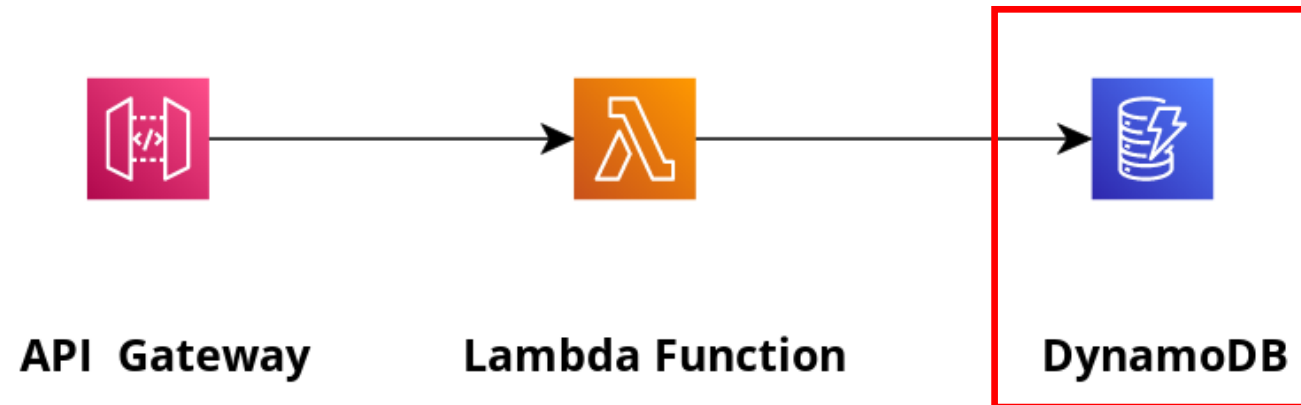
Lambda **concurrency** limit is a limit on the simultaneous in-flight invocations allowed at the same time

Transaction per second (TPS) = $\text{concurrency} / \text{function duration in seconds}$

Lambda Important Service Quotas

Quota	Description	Value	Adjustable	Mitigation
TPS (Transaction per Second)	The maximum number of TPS	$\text{TPS} = \min(10 \times \text{concurrency}, \text{concurrency} / \text{function duration in seconds})$	 	• If the function duration is exactly 100ms (or 1/10th of a second), both terms in the min function are equal • If the function duration is over 100ms, the second term is lower and TPS is limited as per concurrency/function duration • If the function duration is under 100ms, the first term is lower and TPS is limited as per 10 x concurrency

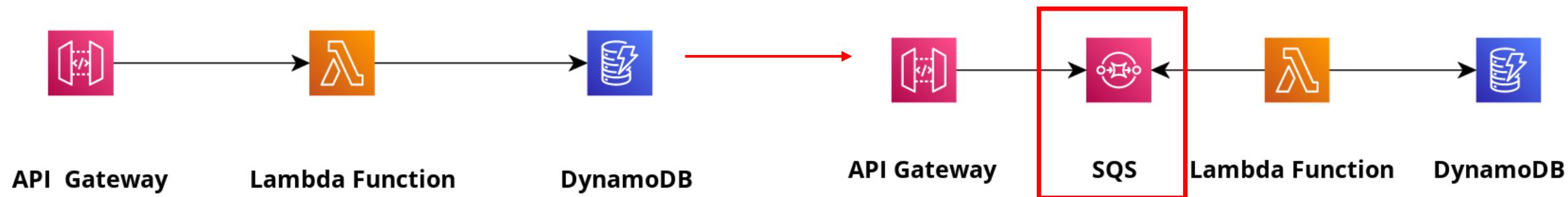
Serverless Application



DynamoDB Important Service Quotas

Quota	Description	Value	Adjustable	Mitigation
Table-level read/write throughput limit	The maximum number of read/write throughput allocated for a table or global secondary index	40.000 RCU/ 40.000 WCU		Ask for quote increase
Table-Level burst capacity for provisioned capacity mode	During an occasional burst of read or write activity, these extra capacity units can be consumed quickly	up to 300 seconds of unused RCUs and WCUs		
Partition-level read/write throughput	The maximum number of read/write throughput allocated for a partition	3000 RCU /1000 WCU		Use best practices to avoid hot partition

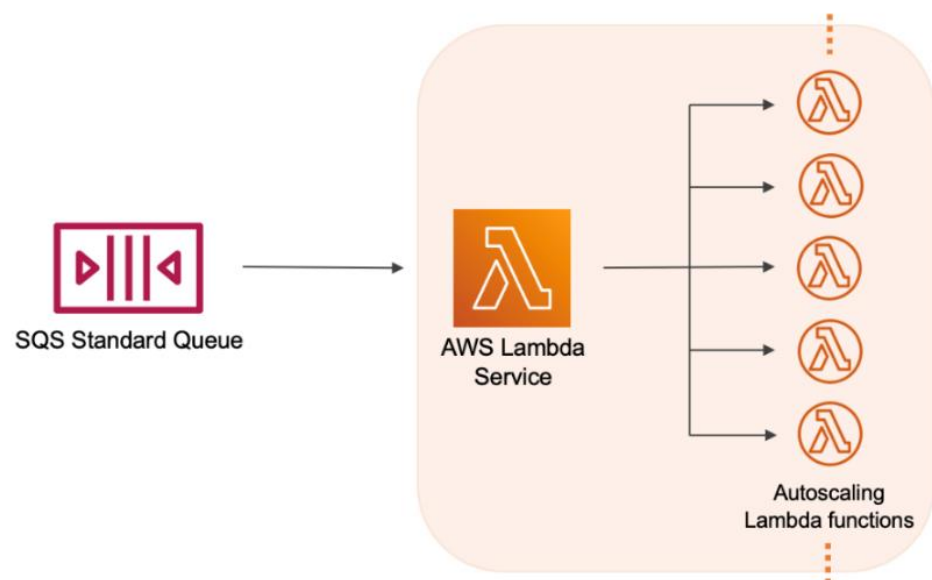
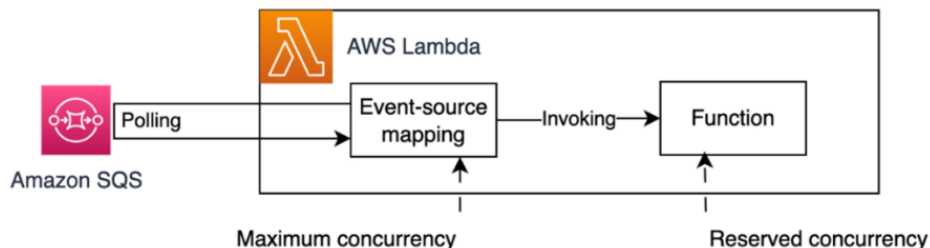
Serverless Application



SQS (Standard) Important Service Quotas

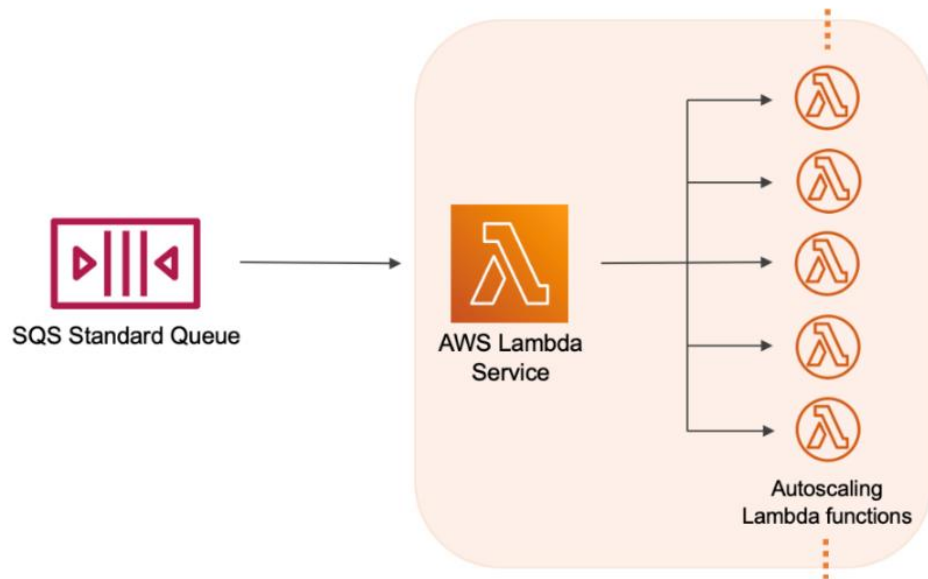
Quota	Description	Value	Adjustable
Throughput per Standard Queue	Standard queues support a nearly unlimited number of transactions per second (TPS) per API action.	Nearly unlimited	

Lambda scaling with SQS standard queues



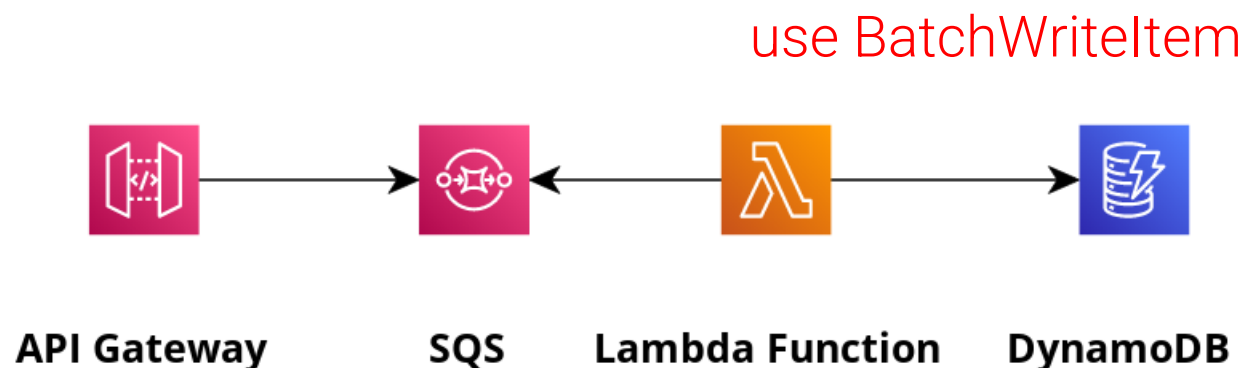
- When a Lambda function subscribes to an SQS queue, Lambda polls the queue as it waits for messages to arrive. It consumes messages in batches, starting with **5 functions** at a time
- If there are more messages in the queue, Lambda adds up to **300 functions/concurrent executions** per minute, up to **1,000 functions (or up to your account concurrency limit)**, to consume those messages from the SQS queue
- This scaling behavior is managed by AWS and cannot be modified
- To process more messages, you can optimize your Lambda configuration for higher throughput

Lambda scaling with SQS standard queues



- Increase the allocated memory for your Lambda function
- Optimize batching behavior:
 - by default, Lambda batches up to **10 messages** in a queue to process them during a single Lambda execution. You can increase this number up to **10,000 messages**, or up to **6 MB** of messages in a single batch for standard SQS queues
 - If each payload size is **256KB** (the maximum message size for SQS), Lambda can only take **23 messages** per batch, regardless of the batch size setting
 - Implement partial batch responses

Use BatchWriteItem request for storing to DynamoDB



- The **BatchWriteItem** operation puts or deletes multiple items in one or more tables.
- A single call to BatchWriteItem can transmit up to **16MB** of data over the network, consisting of up to **25 item** put or delete operations

General best practices for Service Quotas

- Know, understand and observe the service quotas
 - Architect with service quotas in mind
 - AWS adjusts them from time to time
 - In case I'd like to request the quota increase, provide a valid justification for the new desired value
- Service quotas are valid per AWS account (per region)
 - Use different AWS accounts for development and testing
 - Use different AWS accounts for independent (micro-)services
 - Separate AWS accounts on the team level
 - Use AWS Organizations

Recent AWS Serverless Services Quota Increases

AWS Lambda functions now scale 12 times faster when handling high-volume requests

by [Marcia Villalba](#) | on 26 NOV 2023 | in [Announcements](#), [AWS Lambda](#), [AWS re:Invent](#), [Compute](#) | [Permalink](#) | [Comments](#) | [Share](#)



Voiced by [Amazon Polly](#)

Now [AWS Lambda](#) scales up to 12 times faster. Each synchronously invoked Lambda function now scales by 1,000 concurrent executions every 10 seconds until the aggregate concurrency across all functions reaches the account's concurrency limit. In addition, each function within an account now scales independently from each other, no matter how the functions [are invoked](#). These improvements come at no additional cost, and you don't need to do any configuration in your existing functions.

Building scalable and high-performing applications can be challenging with traditional architectures, often requiring over-provisioning of compute resources or complex caching solutions for peak demands and unpredictable traffic. Many developers choose Lambda because it scales on-demand when applications face unpredictable traffic.

Before this update, Lambda functions could initially scale at the account level by 500–3,000 concurrent executions (depending on the Region) in the first minute, followed by 500 concurrent executions every minute until the account's concurrency limit is reached. Because this scaling limit was shared between all the functions in the same account and Region, if one function experienced an influx of traffic, it could affect the throughput of other functions in the same

Introducing the Data API for Amazon Aurora Serverless v2 and Amazon Aurora provisioned clusters

by [Steve Abraham](#) | on 21 DEC 2023 | in [Amazon Aurora](#), [Amazon RDS](#), [AWS Serverless Application Model](#), [Learning Levels](#), [MySQL compatible](#), [PostgreSQL compatible](#) | [Permalink](#) | [Comments](#) | [Share](#)

Traditionally, applications that communicate with relational databases use drivers that provide a persistent connection between the application and the database server. With the advent of serverless applications, including those created using [AWS Lambda](#), persistent connections from the application are no longer practical because serverless applications are, by nature, stateless. What's more, traditional database drivers often come with a set of complex tuning parameters that may be implemented incorrectly by developers who are less familiar with those drivers.

In November 2018, AWS [announced](#) the launch of the Data API for [Amazon Aurora Serverless](#). The Data API is an intuitive, secure HTTPS API for running SQL queries against a relational database that enables you to accelerate modern application development. The Data API appeals to customers of all sizes, from startups to enterprises, seeking to minimize the time-consuming network and application configuration tasks needed to securely connect to an [Amazon Aurora](#) database. The Data API eliminates the use of drivers and improves application scalability by automatically pooling and sharing database connections (connection pooling) rather than requiring you to manage connections. You can call the Data API via an [AWS SDK](#) or the [AWS Command Line Interface](#) (AWS CLI).

Recent AWS Serverless Services Quota Increases

Amazon DynamoDB introduces configurable maximum throughput for On-demand tables

Posted On: May 3, 2024

[Amazon DynamoDB](#) on-demand is a serverless, pay-per-request billing option that can serve thousands of requests per second without capacity planning. Previously, the on-demand request rate was only limited by the default throughput quota (40K read request units and 40K write request units), which uniformly applied to all tables within the account, and could not be customized or tailored for diverse workloads and differing requirements. Since on-demand mode scales instantly to accommodate varying traffic patterns, a piece of hastily written or unoptimized code could rapidly scale up and consume resources, making it difficult to keep costs and usage bounded.

Starting today, you can optionally configure maximum read or write (or both) throughput for individual on-demand tables and associated secondary indexes, making it easy to balance costs and performance. Throughput requests in excess of the maximum table throughput will automatically get throttled, but you can easily modify the table-specific maximum throughput at any time based on your application requirements. Customers can use this feature for predictable cost management, protection against accidental surge in consumed resources and excessive use, and safe guarding downstream services with fixed capacities from potential overloading and

AWS Lambda supports faster polling scale-up rate for Amazon SQS as an event source

Posted On: Nov 6, 2023

AWS Lambda now supports up to 5x faster polling scale-up rate (adding up to 300 concurrent executions per minute) for spiky Lambda workloads configured with [Amazon Simple Queue Service](#) (Amazon SQS) as an event source using [Lambda event source mapping](#) or [Amazon EventBridge Pipes](#). This enables customers building event-driven applications using Lambda and SQS queues (standard or first-in, first-out) to achieve more responsive scaling during a sudden burst of messages in their queues, and reduces the need to duplicate Lambda functions or SQS queues to achieve faster message processing.

AWS Lambda improves responsiveness for configuring stream and queue-based event sources

Posted On: Feb 29, 2024

AWS Lambda now improves the responsiveness for configuring Event Source Mappings (ESMs) and Amazon EventBridge Pipes with event sources such as self-managed Apache Kafka, Amazon Managed Streaming for Apache Kafka (MSK), Amazon DocumentDB, and Amazon MQ. This enhancement allows changes—such as updating, disabling, or deleting ESMs or Pipes—to take effect within 90 seconds, an improvement from the previous time frame of up to 15 minutes.

Customers building event-driven applications use self-managed Apache Kafka, Amazon MSK, Amazon DocumentDB, and Amazon MQ as event sources with Lambda, by configuring ESMs or EventBridge Pipes. Previously, configuration changes that updated, disabled, or deleted these ESMs or Pipes could take up to 15 minutes to go into effect. With today's announcement, these configuration changes will go into effect within 90 seconds, enabling customers to quickly update and test their applications.

This feature is generally available in all commercial [regions](#) that AWS Lambda and Amazon EventBridge Pipes are available in, except Asia Pacific (Hyderabad), Asia Pacific (Melbourne), Canada West (Calgary), Europe (Spain), and Europe (Zurich).

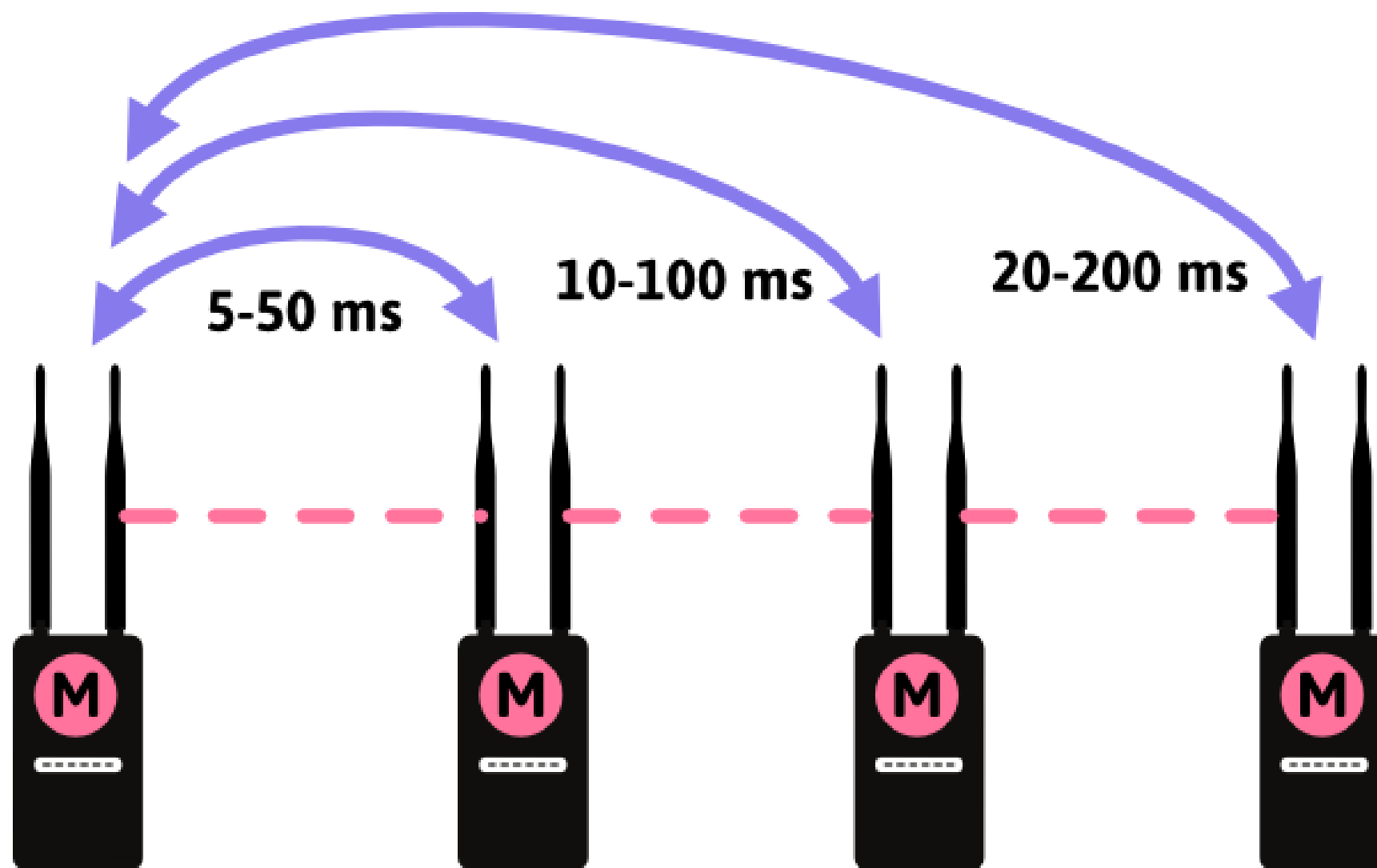
The improved responsiveness is automatically applied to all [Lambda ESMs](#) and [EventBridge Pipes](#) with self-managed Apache Kafka, Amazon MSK, Amazon DocumentDB, and Amazon MQ event sources, at no additional cost.

Amazon API Gateway integration timeout limit increase beyond 29 seconds

Posted on: Jun 4, 2024

Amazon API Gateway now enables customers to increase their integration timeout beyond the prior limit of 29 seconds. This setting represents the maximum amount of time API Gateway will wait for a response from the integration to complete. You can raise the integration timeout to greater than 29 seconds for Regional REST APIs and private REST APIs, but this might require a reduction in your account-level throttle quota limit. With this launch, customers with workloads requiring longer timeouts, such as Generative AI use cases with Large Language Models (LLMs), can leverage API Gateway.

It's also about latency



Serverless latency

SERVERLESS LATENCY

Best practices - runtime

Lambda	Type	Execution Context	Parallelism	Duration	Response
Fat	Sync	✗	✗	~371 ms	< ~400 ms
Fat	Sync	✓	✗	~331 ms	< ~400 ms
Fat	Sync	✓	✓	~273 ms	< ~300 ms
Single	Sync	✓	✓	~21 ms	< ~40 ms
EDA - EB	Async	✓	✓	~154 ms	< ~600 ms
EDA - SNS	Async	✓	✓	~162 ms	< ~400 ms
EDA - SQS	Async	✓	✓	~166 ms	< ~350 ms
EDA - SQS (Rust)	Async	✓	✓	~60 ms	< ~200 ms

aws © 2023, Amazon Web Services, Inc. or its affiliates.

ProSiebenSat.1 Media SE

Challenges and lessons learned



Architect with AWS Serverless quotas and technical concepts in mind



General architectural decisions

Aurora Serverless vs DynamoDB

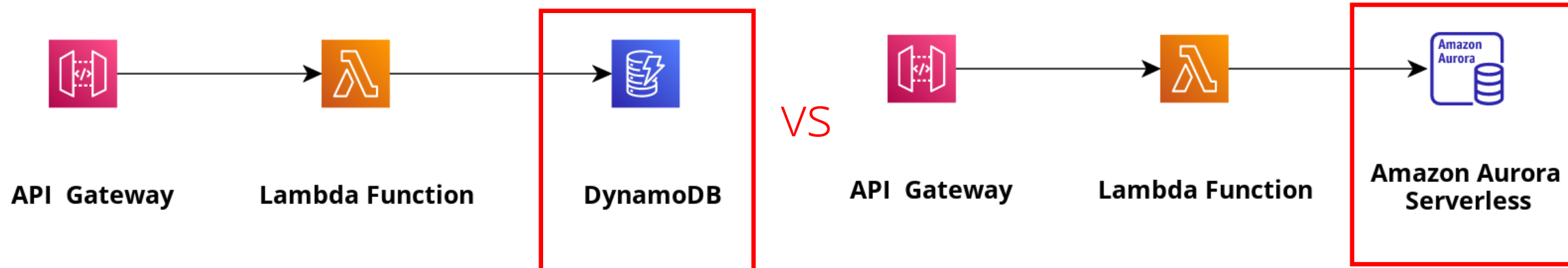
DynamoDB Provisioned Throughput vs On-Demand Capacity



Challenging to trace all requests

Usage of X-Ray add another complexity on the code

DynamoDB vs Aurora Serverless



Aurora Serverless v1 Important Service Quotas

Quota	Description	Value	Adjustable
Data API requests per second	The maximum number of requests to the Data API per second allowed in this account in the current AWS Region	1.000	

We are reaching out to let you know that as of **December 31, 2024**, Amazon Aurora will no longer support **Serverless version 1 (v1)**. As per the Aurora Version Policy [1], we are providing 12 months notice to give you time to upgrade your database cluster(s). Aurora supports two versions of Serverless. We are only announcing the end of support for Serverless v1. Aurora Serverless v2 continues to be supported. We recommend that you proactively upgrade your databases running Amazon Aurora Serverless v1 to Amazon Aurora Serverless v2 at your convenience before December 31, 2024.

Aurora Serverless v2 Important Service Quotas

Quota	Description	Value	Adjustable
Data API requests per second	The maximum number of requests to the Data API per second allowed	The max_connections value for Aurora Serverless v2DB instances is based on the memory size derived from the maximum ACUs. However, when you specify a minimum capacity of 0.5 ACUs on PostgreSQL-compatible DB instances, the maximum value of max_connections is capped at 2,000.	

Aurora Serverless v2 Important Service Quotas

The following table shows the default values for `max_connections` for Aurora Serverless v2 based on the maximum ACU value.

Maximum ACUs	Default maximum connections on Aurora MySQL	Default maximum connections on Aurora PostgreSQL
1	90	189
4	135	823
8	1,000	1,669
16	2,000	3,360
32	3,000	5,000
64	4,000	5,000
128	5,000	5,000

DynamoDB vs Aurora Serverless v2

	DynamoDB	DynamoDB + DAX
Investment in Knowledge	• Understanding of NoSQL databases • Understanding of single-table design principles	Same
Requires to put Lambda into VPC to access		

DynamoDB vs Aurora Serverless v2

	Aurora Serverless v2	Aurora Serverless v2 + Data API
Investment in Knowledge	Relational databases are familiar to many	Same
Engine Support	MySQL and PostgreSQL	Currently only PostgreSQL
Requires to put Lambda into VPC to access		
May require Amazon RDS Proxy for connection pooling		

DynamoDB Provisioned Throughput vs On-Demand Capacity Mode

On-Demand Capacity Mode is ideal for:

- Unknown workloads
- Frequently idle workloads
 - Staging and (individual) test environments
- Unpredictable application traffic
- Low management overhead (truly serverless mode) is preferable

DynamoDB Important Service Quotas

Quota	Description	Value	Adjustable
Initial throughput for on-demand capacity mode	Initial throughput for on-demand capacity mode	See futher details	

Initial Throughput for DynamoDB On-Demand Capacity Mode

Newly created table with on-demand capacity mode:

- enables newly created on-demand tables to serve up to **4,000 WCUs or 12,000 RCUs**
- If you exceed **double your previous traffic's peak within 30 minutes**, then you might experience throttling

Initial Throughput for DynamoDB On-Demand Capacity Mode

- Pre-warm the tables to the anticipated peak capacity of the spike
 - Performing the load test
- Creating table in provisioned mode with high enough WCUs/RCUs and then switch to on-demand mode



Challenges and lessons learned



Architect with AWS Serverless quotas and technical concepts in mind



General architectural decisions

Aurora Serverless vs DynamoDB

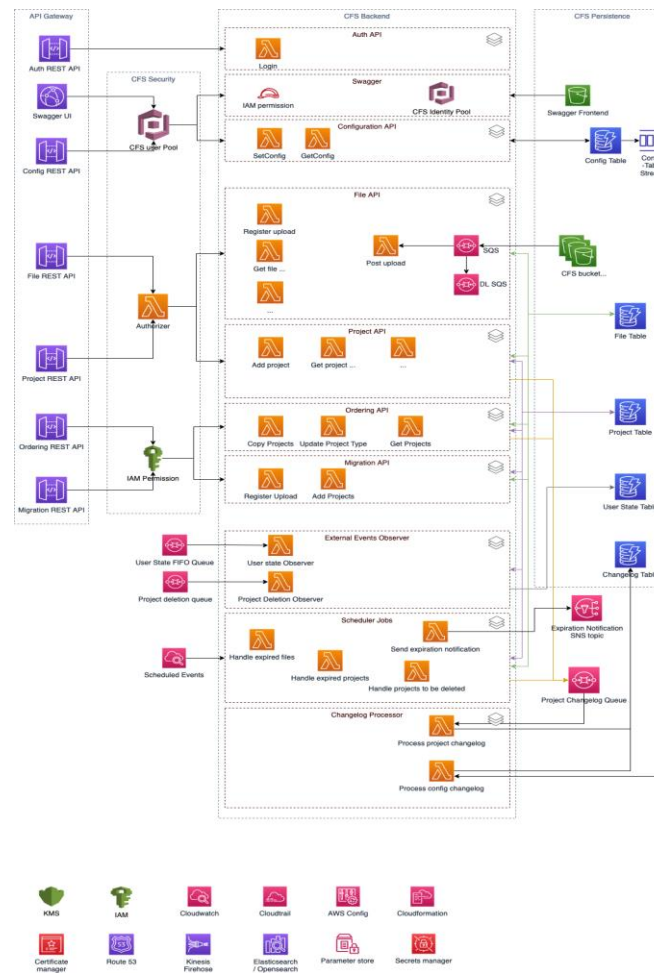
DynamoDB Provisioned Throughput vs On-Demand Capacity



Challenging to trace all requests

Usage of X-Ray add another complexity on the code

Current architecture our image storage solution



Serverless challenges



Gregor Hohpe

Serverless reduces the need for (readily available) ops skills but increases the demand for (less readily available) distributed system design skills.



GOTO
Guide



Remember to
rate this session

THANK YOU!



#GOTOams