

Praktische Abschlussarbeit

Praktische Abschlussarbeit

- Projektbeschrieb
- Vorgehen
- AWS Secrets Manager
- MongoDB
- S3 Bucket

Lambda

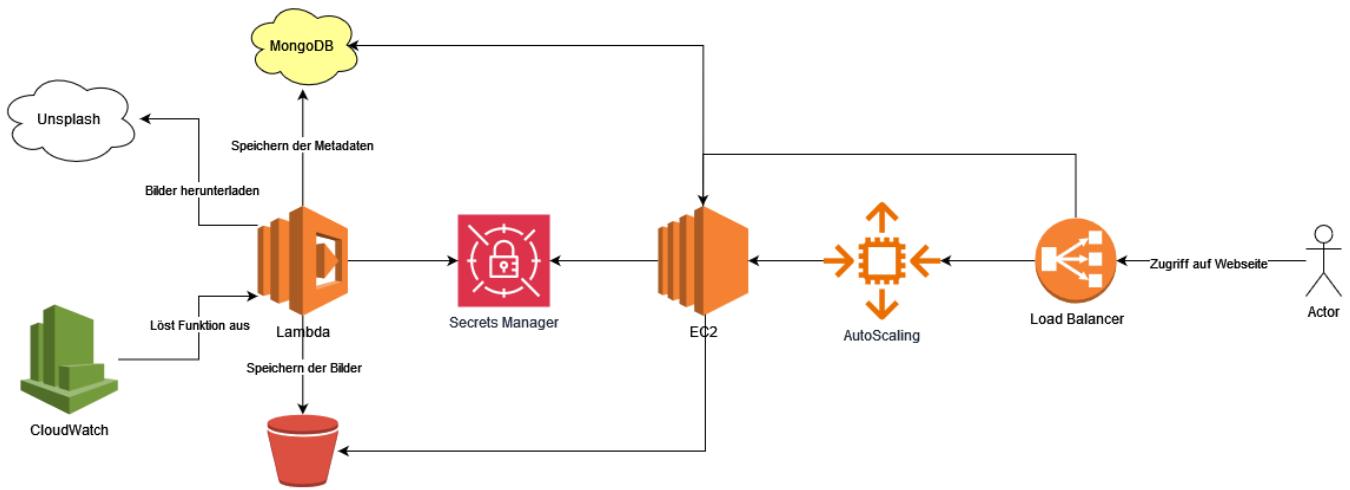
- Lambda Layer
- Lambda Function

- EC2 Webserver
- Automatisieren mit Event Bridge
- Load Balancer und AutoScaling
- Hosted Zone
- Reflexion und Abschluss

Projektbeschrieb

Die Abschlussarbeit, welche im Rahmen des Moduls 364 bearbeitet wurde, hatte das Ziel, eine Webseite zu kreieren, die Bilder anzeigt und deren Metadaten darstellt.

Die Logik dahinter ist allerdings einiges komplizierter. So wurden unter anderem externe Services, aber hauptsächlich AWS-spezifische Funktionalitäten, zu Gebrauch gemacht.



Komponentendiagramm

Prinzipiell lässt sich der ganze Vorgang in zwei Teile unterteilen: die Lambda-Funktion und der Webserver. Diese umfassen die Logik für die Bildverarbeitung und deren Darstellung.

Während die spezifischen Services und Funktionen zu den entsprechenden Komponenten in den folgenden Abschnitten genauer erläutert werden, folgt nun eine Übersicht, wie die Applikation funktioniert:

In einem Zwei-Minuten-Takt, welcher durch einen Cron-Job ausgelöst wird, wird die Lambda-Funktion von CloudWatch gestartet. Die Lambda-Funktion beginnt dann, zuerst die neusten Anmeldeinformationen (Credentials) aus dem Secrets Manager zu holen. Mit der von Unsplash zur Verfügung gestellten API wird ein zufällig ausgewähltes Bild im Format "regular", welches einer Breite von 1080 Pixeln entspricht, heruntergeladen. Bevor das Bild jedoch im S3 Bucket persistent gespeichert wird, ruft die Lambda-Funktion den "Rekognition"-Dienst auf, welcher das Bild analysiert. Als Resultat erhalten wir eine von künstlicher Intelligenz erstellte Bildbeschreibung. Die fünf Merkmale mit der grössten Konfidenz sowie die Metadaten (Unsplash-ID des Bildes, Beschreibung des Bildes und Benutzername des Fotografen), welche wir ebenfalls von Unsplash erhalten, werden anschliessend mithilfe der MongoAPI in eine MongoDB geschrieben. Diese NoSQL-Datenbank erlaubt eine freiere Wahl der Entitäten. Das heisst, die Daten folgen in der Regel keiner strengen Richtlinie, man kann also beliebige Daten speichern. Das erlaubt uns, in Zukunft mehr oder andere Metadaten zu speichern. Ist das Speichern der Metadaten erfolgreich, wird das Bild im S3 Bucket gespeichert und die Lambda-Funktion wird beendet.

Der Webserver, welcher auf einer EC2-Instanz läuft, bezieht ebenfalls die Anmeldeinformationen aus dem Secrets Manager und greift mit diesen per MongoAPI auf die MongoDB zu. Anschliessend wird im S3 Bucket nachgeschaut, ob die Bild-ID, welche von den Metadaten bezogen wurde, auch im S3 Bucket vorliegt. Wenn ja, wird das Bild mit den

entsprechenden Metadaten angezeigt.

Schliesslich sieht der Benutzer alle zwei Minuten ein neues Bild mit den dazugehörigen Metadaten auf einer Homepage.

Vorgehen

Für die Durchführung dieser Aufgabe wird folgendermassen fortgefahrene:

1. Secrets Manager (wird während der Bearbeitung der Aufgabe ständig aktualisiert)
2. MongoDB erstellen (IP Access beachten)
3. S3 Bucket erstellen (Public Access konfigurieren, sodass die Webseite darauf zugreifen kann)
4. Lambda Layer erstellen (beinhaltet Axios und MongoDB, also Dependencies für Lambda)
5. Lambda-Funktion erstellen (Node 20.x)
6. EC2-Webserver erstellen und dynamisch Inhalte von S3 und MongoDB laden
7. EventBridge-Trigger alle 2 Minuten
8. Load Balancer
9. Auto Scaling Service (für Redundanz)
10. Hosted Zone (Subdomain)

AWS Secrets Manager

Damit die Credentials für die jeweiligen Services nicht hardcodiert werden müssen, wird der AWS Secrets Manager verwendet. Dieser erlaubt es, zentralisiert Key-Value-Pairs zu speichern und diese beispielsweise in EC2 oder Lambda zu verwenden.

AWS Secrets Manager => Store a new secret => Other type of secret

Damit die Secrets auch von EC2 oder Lambda ausgelesen werden können, müssen entsprechende Berechtigungen der Lab Role zugewiesen werden. Hierbei existieren verschiedene Möglichkeiten, um Berechtigungen zu erteilen. Eine Möglichkeit ist es, über das Identity and Access Management (IAM) der Rolle den Zugriff für den Secrets Manager zu erteilen. Leider geht das aufgrund der Benutzerberechtigung der AWS-Umgebung im Learner Lab nicht. Das bedeutet, dass ich mit meinem Account keine Berechtigungen im IAM verteilen kann (so hat es zumindest während der Bearbeitung der Aufgabe ausgesehen). Des Weiteren

gibt es die Möglichkeit, via AWS Secrets Manager bei jedem Schlüssel eigene Berechtigungen festzulegen. Dies kann einfach mit folgendem JSON unter Resource permissions beim jeweiligen Key angegeben werden:

```
{  
    "Version" : "2012-10-17",  
    "Statement" : [ {  
        "Effect" : "Allow",  
        "Principal" : {  
            "AWS" : "arn:aws:iam::561824754533:role/LabRole"  
        },  
        "Action" : "secretsmanager:GetSecretValue",  
        "Resource" : "*"  
    }, {  
        "Effect" : "Allow",  
        "Principal" : {  
            "AWS" : "arn:aws:iam::561824754533:role/EMR_EC2_DefaultRole"  
        },  
        "Action" : "secretsmanager:GetSecretValue",  
        "Resource" : "*"  
    } ]  
}
```

Dabei muss der ARN von der Rolle angegeben werden, welche die Berechtigung GetSecretValue erhalten soll. In meinem Fall ist das die LabRole.

Das Angeben der Berechtigung wird für jedes Secret wiederholt.

Schliesslich habe ich die folgenden Schlüsselpaare:

```
MONGODB_CLUSTER  
MONGODB_PASSWORD  
MONGODB_USERNAME  
MONGODB_COLLECTION_NAME  
MONGODB_DATABASE_NAME  
S3_BUCKET_NAME  
UNSPLASH_ACCESS_KEY
```

Choose secret type

Secret type [Info](#)

Credentials for Amazon RDS database

Credentials for Amazon DocumentDB database

Credentials for Amazon Redshift data warehouse

Credentials for other database

Other type of secret API key, OAuth token, other.

Key/value pairs [Info](#)

[Key/value](#) [Plaintext](#)

S3_BUCKET_NAME	praktischepruefungbucket
+ Add row	

Encryption key [Info](#)

You can encrypt using the KMS key that Secrets Manager creates or a customer managed KMS key that you create.

aws/secretsmanager [▼](#) [C](#)

[Add new key](#)

Konfiguration des Secrets für den S3 Bucket

Abschliessend erlaubt es der Secrets Manager einfach und konsequent, Anmeldeinformationen zu speichern. Der Vorteil dieses Services ist, dass er vollumfänglich von den anderen Services (Lambda, EC2) unterstützt wird und ein (mehr oder weniger) einfaches Auslesen ermöglicht. Des Weiteren können für den produktiven Betrieb Rollen und Berechtigungen spezifisch gesetzt werden, um die Sicherheit zu erhöhen.

MongoDB

Bei der MongoDB gibt es nicht viel zu beachten. Nebst einer leeren Instanz musste nur noch der Network access konfiguriert werden. Dabei wurde eingestellt, dass alle IP-Adressen Zugriff auf die Datenbank haben. Natürlich werden immer noch Anmeldeinformationen verlangt, doch wird die Firewall somit etwas entschärft. Der Grund dafür ist, dass die Services, die die MongoDB benötigen, einer öffentlichen IP zugewiesen sind. Diese IP kann sich immer wieder ändern (wenn nicht explizit definiert), was dazu führen würde, dass bei jeder Änderung der IP-Adresse der Network Access auf die neue IP angepasst werden müsste. Dies wird mit dem Setzen des IP-Ranges 0.0.0.0/0 umgangen, birgt aber ein höheres Sicherheitsrisiko. Da es

sich hierbei um keine produktive Umgebung handelt, ist dies jedoch kein Problem.

The screenshot shows the MongoDB Compass interface. At the top, it displays 'SEBASTIAN'S ORG - 2024-06-06 > PROJECT 0 > DATABASES'. Below this, the title 'Cluster346' is shown with a user icon. A navigation bar includes 'Overview', 'Real Time', 'Metrics', 'Collections' (which is underlined in green), 'Atlas Search', 'Performance Advisor', and 'Online'. Under 'Collections', there are buttons for '+ Create Database' and 'Search Namespaces'. A sidebar lists databases: 'praktischePruefungDB' (expanded) and 'praktischePruefungC...'. The main panel shows the 'praktischePruefungDB.praktischePruefungCollection' collection with details: 'STORAGE SIZE: 4KB', 'LOGICAL DATA SIZE: 0B', 'TOTAL DOCUMENTS: 0', and 'INDEXES TOTAL SIZE: 4KB'. It features tabs for 'Find', 'Indexes', 'Schema Anti-Patterns' (with a count of 1), 'Aggregation', and 'Search'. A link 'Generate queries from natural language in Compass' is also present.

MongoDB für die praktische Prüfung

Als Alternative zur gewählten MongoDB wären alle anderen NoSQL-Datenbanken in Frage gekommen, da die Anforderung vorsieht, dass Metadaten einfach erweitert werden können. Dies lässt sich mit NoSQL-Datenbanken umsetzen, da diese keinen strengen Bedingungen folgen. Die möglicherweise optimale Technologie wäre jedoch AWS DynamoDB. Dies ist eine hauseigene NoSQL-Datenbank von AWS und integriert sich entsprechend gut mit den anderen von AWS gebrauchten Services. Trotzdem ist der Zugriff auf MongoDB einfach und benötigt im Prinzip nur die API. Da wir in Aufgabe 6 bereits mit MongoDB gearbeitet haben, ist es trivial, diese bereits implementierte Technologie wiederzuverwenden.

S3 Bucket

Der Bucket muss so konfiguriert werden, dass Bilder im Internet öffentlich zugänglich gemacht werden können. Dabei kann entweder global festgelegt werden, dass jedes Bild veröffentlicht wird, oder man entscheidet sich jeweils genau, welches öffentlich gemacht werden soll. Ich habe mich für letzteres entschieden, um mehr Kontrolle über die Inhalte zu haben. Zum Beispiel möchte ich nicht, dass ein persönliches Bild automatisch online geht...

Damit dies bewerkstelligt werden konnte, mussten die ACLs aktiviert werden und Block all public access abgewählt werden. Diese Einstellungen erlauben es, dass Berechtigungen einzeln auf die jeweiligen Bilder gesetzt werden können. In unserem Fall bedeutet das, ein Bild öffentlich zu machen. Des Weiteren wird so ermöglicht, dass wir über die Lambda-Funktion die Berechtigungen für ein Bild setzen können.

Bucket name | [Info](#)
praktischepruefungbucket
Bucket name must be unique within the global namespace and follow the bucket naming rules. [See rules for bucket naming](#)

Copy settings from existing bucket - optional
Only the bucket settings in the following configuration are copied.
[Choose bucket](#)
Format: s3://bucket/prefix

Object Ownership [Info](#)

Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.

ACLs disabled (recommended)
All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.

ACLs enabled
Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.

⚠ We recommend disabling ACLs, unless you need to control access for each object individually or to have the object writer own the data they upload. Using a bucket policy instead of ACLs to share data with users outside of your account simplifies permissions management and auditing.

Object Ownership

Bucket owner preferred
If new objects written to this bucket specify the bucket-owner-full-control canned ACL, they are owned by the bucket owner. Otherwise, they are owned by the object writer.

Object writer
The object writer remains the object owner.

ⓘ If you want to enforce object ownership for new objects only, your bucket policy must specify that the bucket-owner-full-control canned ACL is required for object uploads. [Learn more](#)

Block Public Access settings for this bucket

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. [Learn more](#)

Block all public access
Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

Block public access to buckets and objects granted through new access control lists (ACLs)
S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.

Block public access to buckets and objects granted through any access control lists (ACLs)
S3 will ignore all ACLs that grant public access to buckets and objects.

Block public access to buckets and objects granted through new public bucket or access point policies
S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.

Block public and cross-account access to buckets and objects through any public bucket or access point policies
S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

Einstellungen des S3 Buckets

Für das Speichern des Bildes gibt es mehrere Alternativen, die jeweils Vor- und Nachteile

haben. Zum Beispiel wäre es möglich gewesen, MongoDB zu verwenden. Dies hätte die Speicherung und Abfrage vereinfacht, ist jedoch suboptimal, da Bilder nicht direkt im JPG- oder PNG-Format gespeichert werden können. Stattdessen müssten sie entweder als binäre Daten oder mithilfe von GridFS in MongoDB-Dokumente aufgeteilt werden, um gespeichert zu werden. Ein Nachteil dabei ist eine erhöhte Latenz und somit Einbussen in der Performance, da die Bilder verarbeitet werden müssen.

Die womöglich sinnvollste Alternative wäre jedoch das Erstellen eines AWS Elastic File Systems. Dies ist ein Datenspeicher, der zwischen EC2-Instanzen geteilt wird und ebenfalls Daten persistent speichert. Ein Vorteil dabei ist, dass Daten einfach abrufbar sind und sogar mit Lambda-Funktionen kompatibel sind. Trotzdem habe ich mich für den S3 Bucket entschieden, da ich bereits einige Aufgaben mit S3 Buckets erledigt habe und dieser die Vorteile bietet, möglicherweise performanter zu sein, da er über HTTP/S anstatt über NFS eingebunden wird.

Schlussendlich wäre es auch möglich gewesen, Elastic Block Storage zu verwenden. Dies eignet sich jedoch aufgrund von Skalierungs- und Wartungsproblemen nicht so gut wie S3. Daher habe ich mich für die konventionelle Methode entschieden - mit S3 Bucket, Lambda-Funktion und EC2-Instanz.

Lambda

Nachdem die MongoDB erstellt, deren Credentials im Secrets Manager hinterlegt und der Bucket konfiguriert wurden, kann nun der zweite Teil angegangen werden: das Erstellen der Lambda-Funktion.

Die folgende Lambda-Funktion basiert auf Node.js 20.x. Um mit der Unsplash-API und der MongoDB-API arbeiten zu können, benötigt sie Dependencies. Diese sind nicht direkt in der Laufzeitumgebung integriert und müssen daher manuell hinzugefügt werden. Dies kann jedoch mithilfe einiger Online-Ressourcen relativ einfach implementiert werden, indem ein sogenanntes Lambda-Layer verwendet wird.

Während der Bearbeitung der Aufgabe habe ich darüber nachgedacht, ob es sinnvoller gewesen wäre, eine andere Entwicklungsumgebung wie etwa C# oder Python zu verwenden, da dort möglicherweise die benötigten Libraries/Dependencies bereits enthalten sind.

Allerdings wird in [diesem](#) Artikel erwähnt, dass der Benutzer selbst die benötigten Abhängigkeiten prüfen muss. Welche jedoch bereits von Lambda selbst integriert sind, habe ich leider nicht gefunden. Daher spielt es meiner Meinung nach keine Rolle, welche

Umgebung verwendet wird, da überall Dependencies/Libraries als Layer hinzugefügt werden müssen.

Lambda Layer

Das Lambda Layer stellt der zugehörigen Lambda-Funktion benötigte Dependencies bereit. Um die Dependencies auf AWS hochladen zu können, müssen diese zunächst auf dem lokalen Rechner heruntergeladen und in eine ZIP-Datei gepackt werden.

Hier sind die Schritte dazu:

1. Einen lokalen Ordner erstellen
2. Im Ordner die Befehle npm init -y und npm install axios mongodb ausführen
3. Den lokalen Ordner in eine ZIP-Datei packen

Auf diese Weise werden die Dependencies/Libraries im richtigen Format vorliegen, das von Lambda gelesen werden kann.

Die Einbindung in Lambda erfolgt wie folgt:

1. Zur Lambda-Konsole navigieren
2. Den Bereich Layers auswählen
3. Ein neues Layer erstellen ("Create layer")
4. Das Layer konfigurieren, indem die Laufzeit angegeben wird und die ZIP-Datei hochgeladen wird

Create layer

Layer configuration

Name
praktischePruefung_lambdaLayer

Description - *optional*
Description

Upload a .zip file
 Upload a file from Amazon S3

Upload

lambda.zip
2.14 MB

For files larger than 10 MB, consider uploading using Amazon S3.

Compatible architectures - *optional* [Info](#)
Choose the compatible instruction set architectures for your layer.
 x86_64
 arm64

Compatible runtimes - *optional* [Info](#)
Choose up to 15 runtimes.
Runtimes ▾ [C](#)

Node.js 20.x X

License - *optional* [Info](#)

Konfiguration des Lambda Layers

Lambda Function

Um die Lambda-Funktion zu erstellen, folge diesen Schritten:

1. Navigiere zur Lambda-Konsole
2. Wähle den Reiter Functions
3. Klicke auf Create function
4. Wähle Node.js 20.x als Laufzeitumgebung aus
5. Bei der Option Change default execution role wähle die LabRole aus

Create function Info

Choose one of the following options to create your function.

Author from scratch

Start with a simple Hello World example.

Use a blueprint

Build a Lambda application from sample code and configuration presets for common use cases.

Container image

Select a container image to deploy for your function.

Basic information

Function name

Enter a name that describes the purpose of your function.

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime Info

Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.



Architecture Info

Choose the instruction set architecture you want for your function code.

x86_64

arm64

Permissions Info

By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

Change default execution role

Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

- Create a new role with basic Lambda permissions
- Use an existing role
- Create a new role from AWS policy templates

Existing role

Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.



[View the LabRole role](#) on the IAM console.

Einstellungen der Lambdafunktion

Das Auswählen der Execution Role hat zwei Gründe: Erstens kann die Funktion nur erstellt werden, wenn eine Rolle angegeben wird, die Berechtigungen hat, auf CloudWatch (Logging Service) zu schreiben. Zweitens greifen wir in der Funktion auf den API-Dienst von MongoDB zu. Dafür werden Credentials benötigt, die im Secrets Manager definiert sind. Damit diese ausgelesen werden können, haben wir bei den jeweiligen Secrets im Secrets Manager angegeben, welche Rollen Zugriff auf die Ressourcen haben. Da dies die LabRole ist, wird diese ausgewählt.

Da die benötigten Dependencies nun vorhanden sind, wird die Lambda-Funktion erstellt, die das Bild von Unsplash herunterlädt und in den S3 Bucket speichert sowie die Metadaten in MongoDB ablegt.

Unter Layers wird der zuvor erstellte Layer hinzugefügt. Dazu wird der ARN (eindeutige ID) des Layers verwendet, der sich in der Detailansicht des Lambda Layers befindet.

The screenshot shows the 'Add layer' configuration page. At the top, 'Function runtime settings' are listed with 'Runtime' set to 'Node.js 20.x' and 'Architecture' set to 'x86_64'. Below this, the 'Choose a layer' section is shown. It includes a note about choosing layers with compatible runtime and instruction set architecture or specifying an ARN. Three options are available: 'AWS layers', 'Custom layers', and 'Specify an ARN'. The 'Specify an ARN' option is selected, with the ARN 'arn:aws:lambda:us-east-1:561824754533:layer:praktischePruefung_lambdaLayer:1' entered into the input field. A 'Verify' button is next to the input field. Below the ARN input, there are sections for 'Description' (empty), 'Compatible runtimes' (Node.js 20.x), and 'Compatible architectures' (x86_64).

Konfiguration, um Layer an der Funktion anzubinden

Der **Code** wird anschliessend in die Datei index.mjs geschrieben und muss mit dem Button Deploy gespeichert werden.

Es ist wichtig zu beachten, dass die Timeout-Dauer erhöht werden muss. Aufgrund des Fetchens von Unsplash und anderen Operationen könnte die Lambda-Funktion länger als drei Sekunden dauern und daher einen Timeout verursachen. Dies kann unter Configuration => General configuration angepasst werden. Ich verwende eine Timeout-Dauer von 30 Sekunden. Bevor mit der Einrichtung der EC2-Instanz fortgefahren wird, sollte ein Test durchgeführt werden, der mit dem entsprechenden Button ausgeführt werden kann. Der Test-Event kann mit der Standardkonfiguration durchgeführt werden und benötigt lediglich einen Namen. Ein erfolgreicher Test gibt einen HTTP-Code 200 zurück. Zudem sollte ein öffentlich einsehbares Bild im Bucket vorhanden sein und in MongoDB der Metadateneintrag gemäss den Anforderungen gespeichert sein.

```

Test Event Name
praktischePruefungTestEv

Response
{
  "statusCode": 200,
  "body": "{\"message\":\"Success\",\"metadata\":{\"id\":\"Kwi4AXxGuz0\",\"description\":\"a monkey hanging on to a tree branch\",\"author\":\"malcoo\"}}"
}

```

Antwort des Tests i.O.

praktischePruefungDB.praktischePruefungCollection

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 353B TOTAL DOCUMENTS: 1 INDEXES TOTAL SIZE: 36KB

[Find](#)

[Indexes](#)

[Schema Anti-Patterns](#) 0

[Aggregation](#)

[Search Indexes](#)

Generate queries from natural language in Compass

[Filter](#)

Type a query: { field: 'value' }

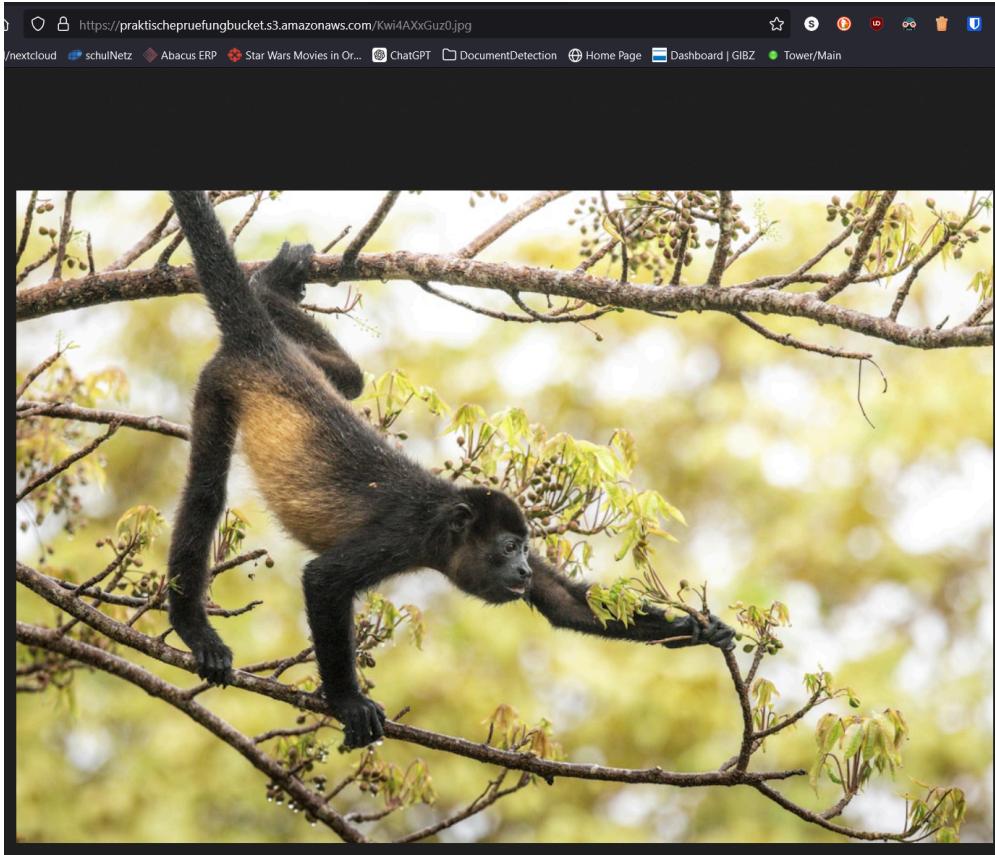
QUERY RESULTS: 1-1 OF 1

```

_id: ObjectId('6676bd9ad86f883b2475a609')
id : "Kwi4AXxGuz0"
description : "a monkey hanging on to a tree branch"
author : "malcoo"
▼ labels : Array (5)
  ▼ 0: Object
    Name : "Animal"
    Confidence : 99.99984741210938
  ▼ 1: Object
    Name : "Mammal"
    Confidence : 99.99984741210938
  ▼ 2: Object
    Name : "Monkey"
    Confidence : 99.99984741210938
  ▼ 3: Object
    Name : "Wildlife"
    Confidence : 99.99984741210938
  ▼ 4: Object
    Name : "Tree"
    Confidence : 56.06834411621094

```

Erster Eintrag von Metadaten in der MongoDB



Öffentliches Bild im S3 Bucket

EC2 Webserver

Für das Erstellen der Instanz wird eine [Cloud-Init](#) Datei verwendet.

Konfiguriert wurde Folgendes:

1. Ubuntu 22.04 als Betriebssystem
2. Ein neues Key Pair wurde erstellt, um die praktischen Prüfungen von anderen Aufgaben zu trennen
3. Eine neue Security Group wurde erstellt, die als Outbound-Rule den Zugriff von allen IP-Adressen (0.0.0.0/0) erlaubt und als Inbound-Rule die Ports SSH (22), HTTP (80) und HTTPS (443) zulässt. Auch wenn HTTPS nicht verwendet wird, vermeidet diese Konfiguration unerwartete Timeout-Probleme beim Zugriff auf die Webseite
4. Es wurde die IAM-Rolle ausgewählt, die Zugriff auf den Secrets Manager hat, wie bereits für die Lambda-Funktion definiert. Diese Rolle ermöglicht es der EC2-Instanz, die erforderlichen Schlüsselpaare aus dem Secrets Manager

auszulesen

Der Webserver läuft mithilfe von Apache und zeigt kontinuierlich die neuen Inhalte aus der MongoDB zusammen mit den jeweiligen Bildern an, die über ein PHP-Skript bereitgestellt werden.

Bei der Konfiguration der EC2-Instanz gab es einige Herausforderungen, insbesondere beim Auslesen der Secrets aus dem Secrets Manager. Verschiedene Ansätze wurden getestet:

1. Direktes Auslesen der Credentials aus dem Secrets Manager im MongoAPI Call.
Dies garantiert stets aktuelle Secrets und wird alle zwei Minuten durch einen Cron Job aktualisiert.
2. Abrufen der Secrets per Cron Job und Speichern als Umgebungsvariablen (ENV).
Der MongoAPI Call greift dann auf diese ENV zu und ruft alle zwei Minuten die API ab. Hierbei muss sichergestellt werden, dass die Credentials stets aktuell sind.
3. Ein Skript, das die Secrets aus dem Secrets Manager liest und direkt im gleichen Skript den API Call durchführt. Dieses Skript wird als Cron Job ausgeführt.

Nur die letzte Methode hat funktioniert, wenn auch nicht optimal. Das Skript `credentials.sh` wird beim Erstellen der EC2-Instanz aufgerufen, um die Secrets aus dem Secrets Manager abzurufen und als ENV auf dem System zu speichern. Gleichzeitig wird der API Call initialisiert, der alle Minute (statt alle zwei Minuten, um immer auf dem neuesten Stand von S3 Bucket/MongoDB zu sein) die Metadaten aus der MongoDB abruft und in einer Datei speichert.

Ein Problem hierbei ist, dass Änderungen an den Secrets nicht automatisch übernommen werden. Die aktualisierten Daten werden durch ein PHP-Skript angezeigt, das das Bild aus dem S3 Bucket lädt und die entsprechenden Metadaten aus der Datei holt. Der Verbindungsstring für den S3 Bucket muss hierbei hartcodiert werden, da der Zugriff auf den Secrets Manager oder die ENV-Variablen nicht implementiert wurde.

Trotz der Funktionalität ist dieses Cloud-Init Skript nicht für den produktiven Einsatz geeignet. Die Verwaltung der Credentials bleibt eine grosse Herausforderung, die stets aktuell und sicher aufbewahrt werden müssen. Dennoch war es möglich, die Credentials aus dem Secrets Manager zu lesen und zu verwenden.

Automatisieren mit Event Bridge

Da der Webserver nun funktioniert, möchten wir, dass alle zwei Minuten ein Bild von Unsplash heruntergeladen und verarbeitet wird. Dies kann sehr einfach mit Amazon EventBridge umgesetzt werden.

Hier sind die Schritte:

1. Navigiere zur Amazon EventBridge Konsole.
2. Wähle Schedules.
3. Klicke auf Create schedule.
4. Unter Schedule pattern gib die Cron Expression `*/2 * * * *` ein, um festzulegen, dass die Regel alle zwei Minuten ausgeführt wird.
5. Wähle als Target AWS Lambda aus.
6. Wähle die entsprechende Lambda-Funktion aus.
7. Wähle bei Action after schedule completion die Option NONE aus, da der Vorgang kontinuierlich laufen soll.
8. Bei den Permissions wähle die LabRole aus, um der Regel die benötigten Berechtigungen zu geben.

The screenshot shows the 'Schedule' configuration page in the AWS EventBridge console. The 'Cron expression' field contains `*/2 * * * ?`, indicating a rate of every 2 minutes. Below the field is a detailed breakdown of the cron expression: Minutes: `*/2`, Hours: `*`, Day of month: `*`, Month: `*`, Day of week: `?`, Year: `*`. A 'Copy cron expression' button is visible. Under 'Next 10 trigger dates', it lists the following times: Tue, 25 Jun 2024 21:00:00 (UTC+02:00), Tue, 25 Jun 2024 21:02:00 (UTC+02:00), Tue, 25 Jun 2024 21:04:00 (UTC+02:00), Tue, 25 Jun 2024 21:06:00 (UTC+02:00), Tue, 25 Jun 2024 21:08:00 (UTC+02:00), Tue, 25 Jun 2024 21:10:00 (UTC+02:00), Tue, 25 Jun 2024 21:12:00 (UTC+02:00), Tue, 25 Jun 2024 21:14:00 (UTC+02:00), Tue, 25 Jun 2024 21:16:00 (UTC+02:00), and Tue, 25 Jun 2024 21:18:00 (UTC+02:00).

Die nächsten Ausführzeiten des Events

Load Balancer und AutoScaling

Mit der bisher umgesetzten Applikation werden alle zwei Minuten Bilder von Unsplash heruntergeladen und mit den jeweiligen Metadaten auf der Homepage angezeigt. In diesem Abschnitt wird ein Load Balancer und ein Auto Scaling Service erstellt. Diese Kombination ermöglicht es, stets eine gewünschte Anzahl von EC2-Instanzen laufen zu haben und die Last gleichmäßig auf diese zu verteilen. Zudem werden die Instanzen in verschiedenen Availability Zones ausgeführt, was die Redundanz erhöht.

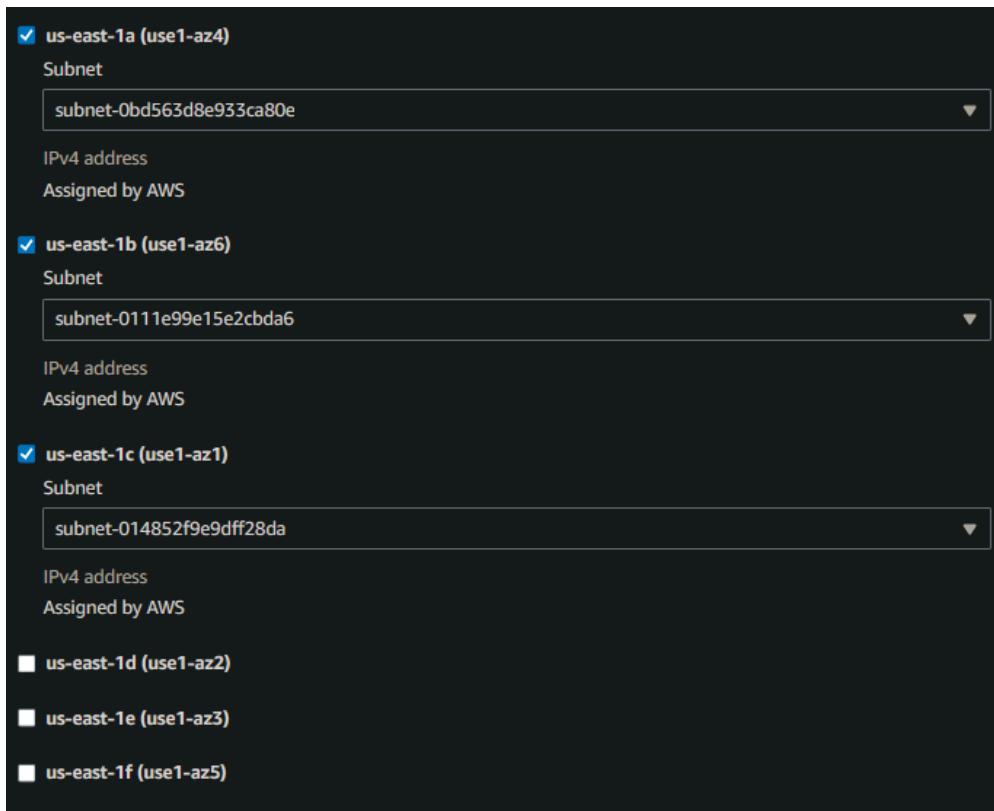
Der folgende Ablauf wird nicht im Detail dokumentiert, da dies bereits in der Aufgabe 06: Scaling ausführlicher beschrieben ist. Dennoch ist der Ablauf wie folgt:

1. Erstellen der Target Group:
 - i. Gehe zu EC2
 - ii. Wähle Target Groups
 - iii. Klicke auf Create target group
 - iv. Wähle als Target Instances

The screenshot shows the AWS CloudFormation console interface for creating a new target group. At the top, there is a table with columns: Instance ID, Name, State, and Security groups. One instance is selected: i-02af94cd9a76058ca, named praktischePruefungEC2, with a state of Running and security group prak. Below the table, it says "1 selected". Under "Ports for the selected instances", the port number 80 is entered. A note below says "1-65535 (separate multiple ports with commas)". There is a button "Include as pending below". At the bottom, a message says "1 selection is now pending below. Include more or register targets when ready." The next section is titled "Review targets". It shows a table with the same header: Targets (1). It lists one target: i-02af94cd9a76058ca, named praktischePruefungEC2, with a state of Running and security group praktischePruefungSG. There are buttons for "Remove all pending" and "Show only pending". The page footer says "Einstellungen der Target Group".

2. Konfigurieren des Load Balancers:

- i. Gehe zu EC2
- ii. Wähle Load Balancer
- iii. Klicke auf Create Load Balancer
- iv. Wähle Application Load Balancer
- v. Konfiguriere die Availability Zones, in denen der Load Balancer laufen soll
- vi. Passe die Security Groups entsprechend an
- vii. Wähle die zuvor erstellte Target Group aus, um die Last auf die EC2-Instanzen zu verteilen



Die selektierten Availability Zones für mehr Redundanz

Summary			
Review and confirm your configurations. Estimate cost			
Basic configuration Edit	Security groups Edit	Network mapping Edit	Listeners and routing Edit
praktischePruefungLB <ul style="list-style-type: none"> Internet-facing IPv4 	<ul style="list-style-type: none"> praktischePruefungSG sg-08a682ac9974a87af 	VPC vpc-0e70f2a6594cd526c <ul style="list-style-type: none"> us-east-1a subnet-0bd563d8e933ca80e us-east-1b subnet-0111e99e15e2cbda6 us-east-1c subnet-014852f9e9dff28da 	<ul style="list-style-type: none"> HTTP:80 defaults to praktischePruefungTG
Service integrations Edit			Tags Edit
AWS WAF: None			None
AWS Global Accelerator: None			

Zusammenfassung des Load Balancers

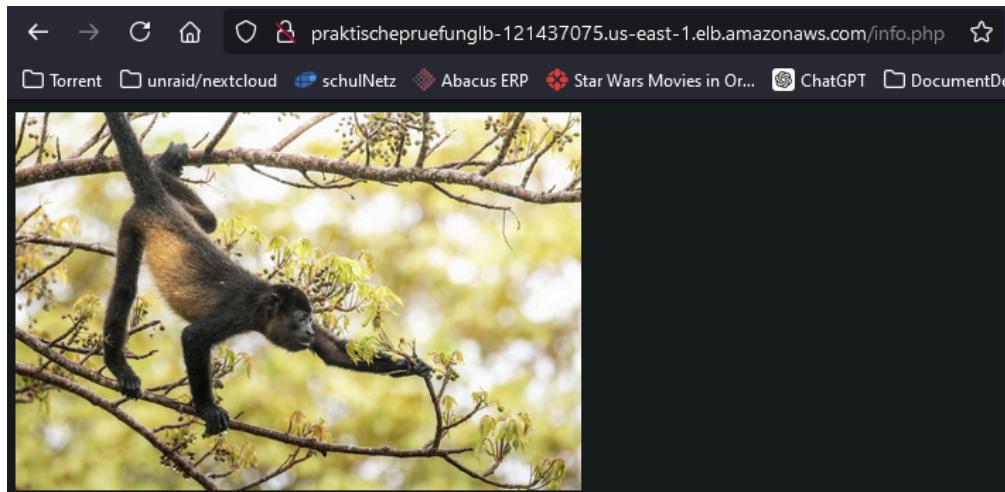


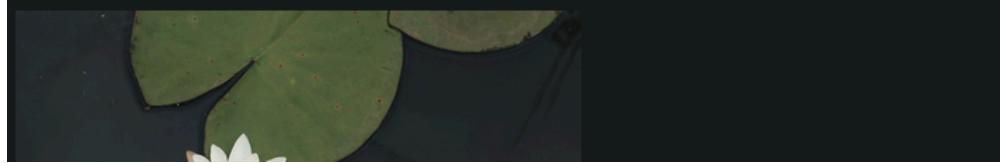
Photo ID: Kwi4AXxGuz0

Description: a monkey hanging on to a tree branch

Author: malcoo

Labels:

Animal (Confidence: 99.999847412109),
Mammal (Confidence: 99.999847412109),
Monkey (Confidence: 99.999847412109),
Wildlife (Confidence: 99.999847412109),
Tree (Confidence: 56.068344116211).



Erreichbar: Der Webserver via Load Balancer

Bis zu diesem Zeitpunkt haben wir einen LoadBalancer mit einer öffentlich erreichbaren URI, doch zeigt diese auf immer die gleiche Instanz. Dies wollen wir nun ändern.

3. Erstellen eines Launch Templates:

2. Gehe zu EC2
3. Wähle Launch Templates
4. Konfiguriere das Launch Template ähnlich wie die zuvor erstellte EC2-Instanz
4. Definieren des Auto Scaling Services:
 - i. Gehe zu EC2
 - ii. Wähle Auto Scaling Groups
 - iii. Wähle das zuvor erstellte Launch Template aus
 - iv. Wähle die gleichen Availability Zones wie beim Load Balancer aus
 - v. Wähle Attach to an existing load balancer
 - vi. Wähle die entsprechende Target Group aus
 - vii. Aktiviere die Elastic Load Balancing Gesundheitsüberprüfungen
 - viii. Setze die Mindestanzahl von Instanzen auf 2

Sobald die Instanzen den Status "Healthy" erreichen, bleibt die Webseite über die URI des Load Balancers weiterhin erreichbar. Jetzt besteht eine gewisse Redundanz mit drei Instanzen (2 vom Auto Scaler, 1 manuell erstellt). Man kann sich immer auf die beiden Instanzen des Auto Scalers verlassen, es sei denn, es gibt einen weitreichenden AWS-Systemausfall, was jedoch sehr unwahrscheinlich ist.

Details

arn:aws:elasticloadbalancing:us-east-1:561824754533:targetgroup/praktischePruefungTG/d2c8dd428ef10355

Target type Instance	Protocol : Port HTTP: 80	Protocol version HTTP1	VPC vpc-0e70f2a6594cd526c	
IP address type IPv4	Load balancer praktischePruefungLB			
3 Total targets	<input checked="" type="radio"/> 3 Healthy	<input checked="" type="radio"/> 0 Unhealthy	<input checked="" type="radio"/> 0 Unused	<input checked="" type="radio"/> 0 Initial
	0 Anomalous			<input checked="" type="radio"/> 0 Draining

Distribution of targets by Availability Zone (AZ)
Select values in this table to see corresponding filters applied to the Registered targets table below.

Targets	Monitoring	Health checks	Attributes	Tags
Registered targets (3) Info	Anomaly mitigation: Not applicable	C	Deregister	Register targets
Target groups route requests to individual registered targets using the protocol and port number specified. Health checks are performed on all registered targets according to the target group's health check settings. Anomaly detection is automatically applied to HTTP/HTTPS target groups with at least 3 healthy targets.				
<input type="text"/> Filter targets < 1 > @				
Instance ID	Name	Port	Zone	Health status
i-0662374ca1ec0cf1a		80	us-east-1a	Healthy
i-0bca20587d6e0ae64		80	us-east-1c	Healthy
i-02af94cdaf76058ca	praktischePrue...	80	us-east-1b	Healthy

Drei gesunde Instanzen

Hosted Zone

Um den Webserver unter der Subdomain [sebastian.m346.ch](#) erreichbar zu machen, müssen wir eine Hosted Zone in AWS Route 53 erstellen und konfigurieren:

1. Gehe zu AWS Route 53
2. Wähle Create hosted zone
3. Gib die Subdomain [sebastian.m346.ch](#) an
4. Der Administrator der Domain muss die Nameserver (NS), die in der Route 53 Oberfläche angezeigt werden, informiert bekommen. Der Administrator delegiert dann alle Verbindungen zur Subdomain an unsere Hosted Zone, damit wir diese verwenden können
5. Nachdem die Hosted Zone erstellt und konfiguriert ist, erstelle einen DNS-Eintrag (record):
 - i. Wähle den Typ Alias.
 - ii. Wähle den AWS Service, der automatisch konfiguriert werden soll.
 - iii. Wähle den Load Balancer, der mit deinem Webserver verbunden ist ([praktischePruefungLB-121437075.us-east-1.elb.amazonaws.com](#)).

Damit wird der Load Balancer unter der Subdomain [sebastian.m346.ch](#) erreichbar sein.

The screenshot shows the AWS Route 53 console. A new resource record set is being created for the subdomain 'sebastian.m346.ch'. The 'Datensatzname' field contains 'subdomain' and the 'Datensatztyp' field is set to 'A - Leitet den Datenverkehr an eine IPv4-Adresse und einige AWS-Ress...'. The 'Alias' checkbox is selected. In the 'Datenverkehr weiterleiten an' section, 'Alias zu Application und Classic Load Balancer' is chosen, and 'USA Ost (Nord-Virginia)' is selected. The target value is 'dualstack.praktischePruefungLB-121437075.us-east-1.elb.amazonaws.com'. The 'Einfaches Routing' checkbox is also checked. A note at the bottom says 'Verwenden Sie: „dualstack.praktischePruefungLB-121437075.us-east-1.elb.amazonaws.com“'.

Konfiguration der Subdomain in der Hosted Zone

Hat alles funktioniert, ist unser Webserver über die Seite [sebastian.m346.ch](#) aufrufbar.

Reflexion und Abschluss

Die Aufgabe erwies sich als komplexer als anfangs gedacht. Das Planen und

Zusammenstellen ging ziemlich gut, da die vorherigen Aufgaben in diesem Modul alle gewisse Teile dieser Aufgabe beinhalteten. Es war also prinzipiell ein Zusammenstellen verschiedener Dienste und möglicher Verbesserungen oder zusätzlicher Funktionalitäten. Secrets Manager, Events Bridge oder Lambda Layer waren zusätzliche Konfigurationen, die Neuland für mich waren und entsprechend Zeit gekostet haben.

Wie bereits gesagt, war das mit Abstand Schwierigste das Handling der Credentials. Zum einen Berechtigungen zu erteilen, sodass Lambda oder EC2 diese auslesen können, zum anderen die Speicherung, Aktualisierung und schliesslich deren Verwendung. Die Implementierung in der Lambdafunktion finde ich gelungen; stets wenn diese ausgeführt wird, holt sie sich die neuen Anmeldeinformationen. Die Werte sind nicht hardcodiert und erlauben ein einfaches Austauschen der Variablen im Secrets Manager. Bei der EC2-Instanz ist dies leider nicht der Fall. Die Credentials werden zwar verwendet, doch habe ich es nicht geschafft, das Aktualisieren dieser in einen Cron-Job o.ä. zu integrieren. Hierbei bedarf es mehr technisches Know-how, um eine sicherere und besser wartbare Lösung zu kreieren.

Als weiteren Verbesserungspunkt sehe ich eine optimiertere Abfrage zwischen EC2 und dem S3 Bucket. Zurzeit werden alle Bilder auf einmal aus dem Bucket bezogen, was je nach Menge sehr viel Zeit beansprucht. Dies könnte am einfachsten optimiert werden, indem die Webseite "Seiten" bekommt. Dies würde garantieren, dass auf jeder Seite nur beispielsweise 10 Bilder geladen werden. Möchte man mehr sehen, muss man auf die nächste Seite wechseln. Eine weitere Möglichkeit wäre das lokale Speichern der Bilder. Dies wäre zwar schneller beim Anzeigen, würde jedoch die Architektur komplexer machen. Ähnliches gilt für das Holen der Metadaten. Der Cron-Job sorgt dafür, dass jede Minute alle Daten ausgelesen und gespeichert werden. Dies erzeugt, abhängig von den Daten, massive Performanceprobleme und müsste für den produktiven Betrieb ebenfalls überarbeitet werden. Am besten wäre es, genau wie bei den Bildern, nur die Metadaten abzurufen, die auch wirklich angezeigt werden.

Den restlichen Ablauf schätze ich als erfüllt ein. Durch die Verwendung verschiedener Services entsteht eine dezentralisierte und modulare Architektur, die beliebig skaliert oder ausgetauscht werden kann, ohne grosse Unterbrüche zu erleiden. Wenn beispielsweise die EC2-Instanzen gewartet werden müssen oder ausfallen, läuft das Herunterladen und Bearbeiten der Bildinformationen trotzdem weiter, da eine Lambdafunktion benutzt wird, welche unabhängig davon arbeiten kann. Weiter ist die Lambdafunktion kostengünstiger, da sie nicht ständig läuft im Vergleich zu einer EC2-Instanz. Da der Benutzer ständig auf die

Website gelangen will, kommt man nicht um einen Webserver herum, der auf einer EC2-Instanz läuft. Um Redundanz und eine mögliche Downtime zu vermeiden, wurde auch ein Auto Scaling Service und Load Balancer integriert. Somit laufen stets mindestens zwei EC2-Instanzen in verschiedenen Availability Zones. Wird eine Instanz zu stark beansprucht, leitet der Load Balancer den Verkehr zur anderen Instanz um, was zu Leistungsverbesserungen führt.

Zusammenfassend bin ich also relativ zufrieden mit der Lösung. Sie weist redundante Aspekte für einen optimalen Betrieb auf und verfolgt einen modularen Ansatz. Es ist (teilweise) möglich, einen zentralisierten Credentials Manager zu verwenden, um die Anmeldeinformationen zuverlässig zu ändern, und es spart Kosten durch die Verwendung einer Lambdafunktion und einem S3 Bucket. Verbessert werden müssten jedoch die Verwaltung der Anmeldeinformationen und ein performanteres Holen und Abspeichern der Bilddateien, Metadaten sowie deren Anzeige auf der Webseite.