Projektdokumentation Netzwerke

Projekt „Chatastrophy“

Nils Taglieber, Niklas Rotgeri, Wilko Kramer, Lenn Schneidewind, Edgar Scaruppe, Julian Peters

Inhalt

1. Einleitung

Das Projekt umfasst die Entwicklung eines Online-Chats mit dem Namen „Chatastrophy“. Dieser Online-Chat basiert auf der serverlosen Computerplattform Lambda von AWS. Das Chat-System ermöglicht Benutzern die Echtzeitkommunikation über WebSockets. AWS-Lambda wird eingesetzt, um den WebSocket-Server zu implementieren und die erforderliche Logik für den Chat bereitzustellen. DynamoDB wird verwendet, um Chatverläufe und andere relevante Daten persistent zu speichern. Darüber hinaus verwaltet das API-Gateway die WebSocket-Verbindungen und fungiert als Schnittstelle zwischen den Clients und der Lambda-Funktion.

Zusätzlich zu dem API-Gateway und DynamoDB werden auch andere AWS-Services in diesem Projekt verwendet, auf welche in diesem Projekt-Bericht eingegangen wird. Zudem wird der Technologie-Stack vorgestellt und die Architektur visualisiert und näher erläutert.

2. Projektübersicht

Projektziele und Anforderungen

Das Ziel des Projekts besteht darin, ein benutzerfreundliches Online-Chat-System zu entwickeln, das eine reibungslose Kommunikation und Interaktion zwischen den Benutzern ermöglicht. Das System soll eine Echtzeitkommunikation unterstützen, sodass Benutzer in der Lage sind, Nachrichten in Echtzeit auszutauschen.

Ein weiteres wichtiges Ziel ist die Persistenz der Chatnachrichten. Das System soll in der Lage sein, die Chatnachrichten dauerhaft zu speichern, um den Benutzern den Zugriff auf vergangene Unterhaltungen und den Chatverlauf zu ermöglichen. Hierbei soll die NoSQL-Datenbank DynamoDB verwendet werden, um eine effiziente und skalierbare Speicherung der Nachrichten zu gewährleisten.

Die Benutzeroberfläche des Chat-Systems sollte benutzerfreundlich und intuitiv gestaltet sein. Das Frontend sollte eine angenehme Benutzererfahrung bieten und verschiedene Funktionen wie das Senden von Nachrichten, das Anzeigen von Benutzerprofilen und das Scrollen durch den Chatverlauf ermöglichen.

Um diese Ziele zu erreichen, wird das Projekt verschiedene AWS-Services und Frameworks nutzen. AWS-Lambda wird verwendet, um die Backend-Logik zu implementieren und die Kommunikation zwischen den Benutzern zu steuern. DynamoDB dient hier als Datenbank zur Speicherung der Chatnachrichten und anderer relevanter Informationen. Der S3-Bucket wird genutzt, um statische Ressourcen wie den Frontend-Code und andere Dateien zu speichern und das Serverless-Framework soll die Bereitstellung und Verwaltung der AWS-Ressourcen erleichtern, indem es eine abstrakte Schicht über den Serverless-Compute-Dienst von AWS bietet.

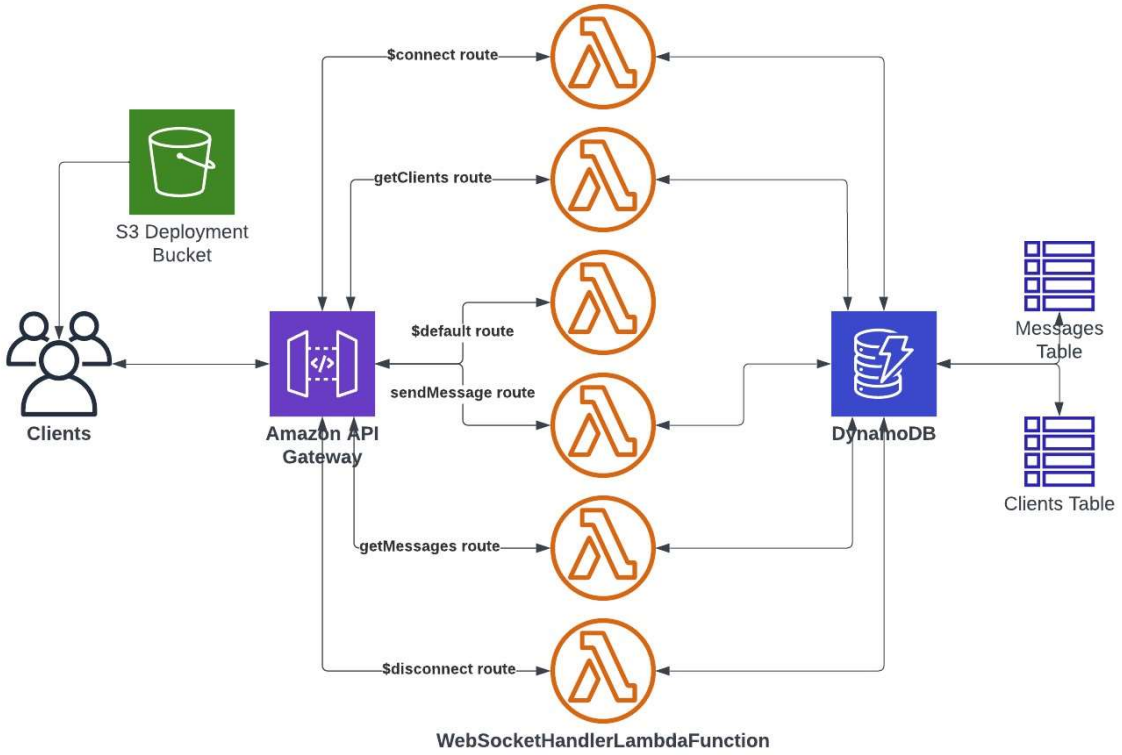
Technologie Stack

Als integrierte Entwicklungsumgebung (IDE), wird Visual Studie Code verwendet, um den Code für den Chat zu entwickeln. Node.js wird verwendet, um den WebSocket-Server zu implementieren und die Logik für den Chat zu schreiben und AWS-Lambda, um bestimmte Funktionen über das API-Gateway im Zusammenhang mit dem Chat zu implementieren, also bspw. das Aufrufen und Senden von Nachrichten oder Clients. Der S3-Bucket ist ein skalierbarer Objektspeicherdienst und Teil der AWS-Services Welt. Er kommt zum Einsatz, um Dateien wie Multimedia-Inhalte des Front-Ends zu speichern, die im Zusammenhang mit dem Chat stehen. DynamoDB, als NoSQL-Datenbankdienst von AWS, dient dazu, Chat-Verlaufsdaten, Benutzerprofile oder andere Informationen im Zusammenhang mit dem Chat zu speichern. Außerdem sind für die Entwicklung des Chats zwei Frameworks zum Einsatz gekommen, um den Umgang mit AWS-Lambda und die Front End- Entwicklung zu vereinfachen. Das Serverless Framework wurde hier verwendet, um die WebSocket-Serveranwendung und AWS-Lambda-Funktionen zu verwalten und bereitzustellen, Tailwind CSS, um das Frontend-Design für den Chat zu gestalten und anpassbare UI-Komponenten zu erstellen. Weiter kommen anderer AWS-Services zum Einsatz die mittels des Serverless-Frameworks konfiguriert wurden.

(Quelle: Eigene Darstellung)

3. Architektur

High-Level-Architektur des Chat-Systems



(Quelle: Eigene Darstellung, vgl. AWS-Lambda, s. Anhang)

Die High-Level-Architektur des Chat-Systems basiert auf einer serverlosen Architektur und umfasst verschiedene AWS-Services. Das Chat-System ermöglicht Benutzern die Echtzeitkommunikation über WebSockets. Dabei interagieren diese mit dem Chat-System über einen Webbrowser, indem eine Verbindung zum Chat-Server hergestellt wird, wodurch Nachrichten gesendet und empfangen werden können.

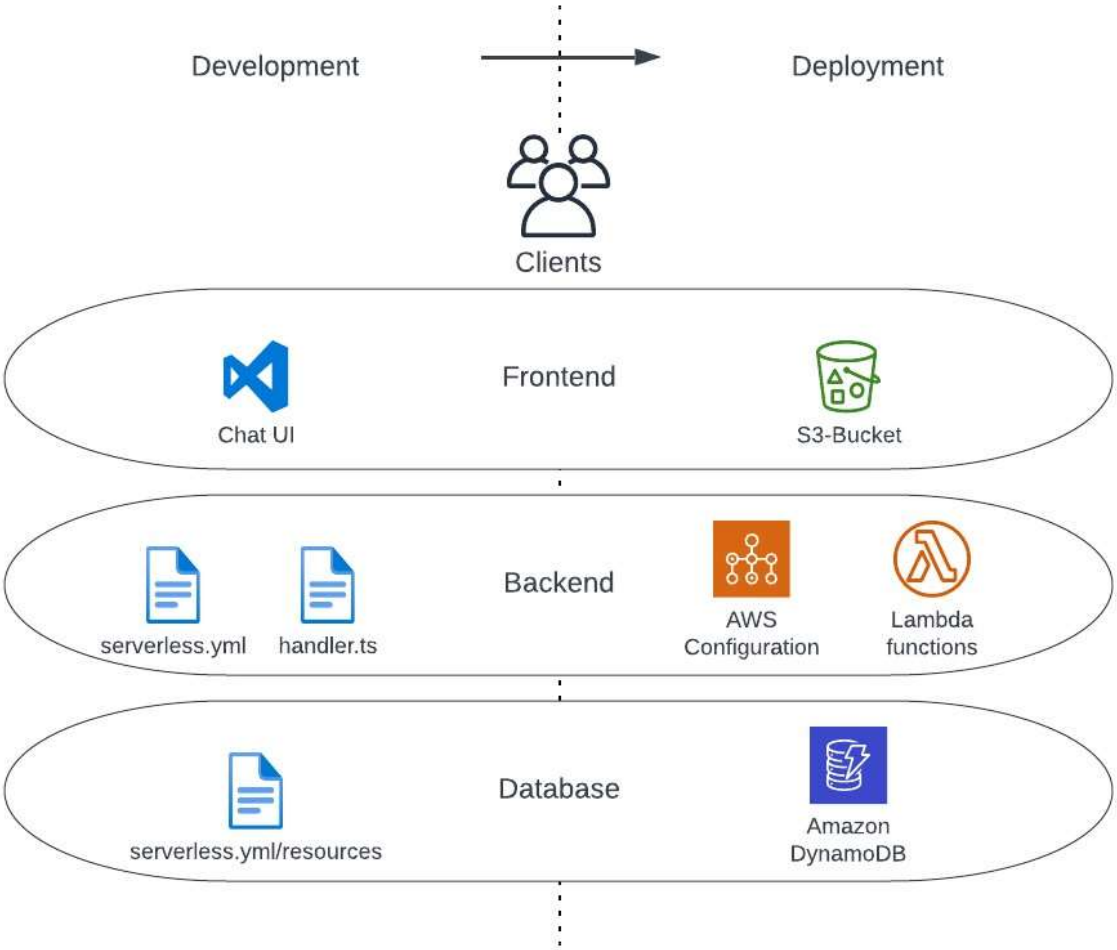
Das API-Gateway ist ein AWS-Service innerhalb der AWS-Welt und fungiert als zentrale Verbindungsstelle zwischen den Benutzer-Clients und dem WebSocket-Server. Es nimmt eingehende WebSocket-Anfragen entgegen und leitet sie an die entsprechenden AWS-Lambda-Funktionen weiter. Die WebSocket-Verbindung und die Chat-Logik werden durch AWS-Lambda-Funktionen implementiert. Wenn eine WebSocket-Verbindung hergestellt wird, eine Nachricht empfangen wird oder die Verbindung geschlossen wird, werden die entsprechenden Lambda-Funktionen aufgerufen.[1] Sie verarbeiten die eingehenden Anfragen, führen die erforderliche Logik aus und senden die entsprechenden Antworten an die Clients.

Die DynamoDB ist die Inhouse-NoSQL-Datenbank von AWS und wird in diesem Projekt als Datenbank verwendet, um die Chatverläufe und andere relevante Informationen persistent zu speichern. So kann gewährleistet werden, dass ein Verlauf der Unterhaltungen bereitgestellt werden kann, indem vergangene Nachrichten abgerufen werden. Weiter wird ein S3-Bucket

[1] Handler-File, siehe Anhang

verwendet, um Dateien wie HTML-, CSS- und Bild-Dateien der Web-Applikation zu speichern.[2] Das AWS-Identity and Access Management (IAM) ermöglicht darüber hinaus die Verwaltung von Benutzerberechtigungen und -zugriff auf die verschiedenen AWS-Ressourcen.[3] AWS CloudWatch kann zur Überwachung und Protokollierung der Anwendung eingesetzt werden.[4]

Komponenten und ihre Funktionen (Frontend, Backend, Datenbank)



Quelle: Eigene Darstellung

Backend

Der Handler ist eine Funktion oder ein Skript, das in der AWS-Lambda-Umgebung ausgeführt wird, wenn eine Anfrage an die Lambda-Funktion gesendet wird. Er fungiert als Einstiegspunkt für die Lambda-Funktion und steuert den Ablauf der Verarbeitung. Sollten Anfragen über den WebSocket eingehen werden diese durch den Handler über das API-Gateway entgegengenommen. Der Handler ist für die Verarbeitung der eingehenden Anfragen zuständig und ruft die entsprechenden Funktionen auf, um die erforderliche Logik auszuführen. Das können Aktionen wie das Speichern einer Chatnachricht in der Datenbank,

[2] S3-Buckets, siehe Anhang
[3] IAM-Rollenvergabe, siehe Anhang
[4] CloudWatch, siehe Anhang

das Abrufen von Nachrichten oder das Verarbeiten von Benutzeraktionen sein. Der Handler orchestriert die Verarbeitungsschritte und gibt die entsprechenden Antworten an das API-Gateway zurück, welche dann zu den Clients gelangen.

Die Serverless.yml-Datei ist eine Konfigurationsdatei im YAML-Format, in der die Einstellungen und Definitionen für das Serverless-Projekt festgelegt werden. Sie enthält Informationen wie den Namen des Dienstes, die Region, in der der Dienst bereitgestellt werden soll, die AWS-Ressourcen, die verwendet werden sollen, und die einzelnen Funktionen, die erstellt werden sollen. Sie ermöglicht die einfache Konfiguration und Verwaltung der Infrastruktur und Ressourcen über das Serverless-Framework. Außerdem erlaubt es die Lambda-Funktionen, deren Auslöser, Umgebungsvariablen, Berechtigungen, Abhängigkeiten und andere wichtige Einstellungen zu definieren und andere AWS-Services wie die DynamoDB-Tabellen, S3-Buckets, API-Gateway und andere Ressourcen zu konfigurieren und mit den Lambda-Funktionen zu verbinden, wodurch eine nahtlose Integration und Kommunikation zwischen den verschiedenen Komponenten des Chat-Systems gewährleistet wird.

Frontend

Der Amazon Simple Storage Service (S3) dient als zentraler Speicherort für verschiedene Ressourcen, die im Zusammenhang mit der Chat-Anwendung stehen. Dies umfasst die statischen Dateien der Benutzeroberfläche, wie HTML-, CSS- und JavaScript-Dateien. Während der Entwicklungsphase der Anwendung wird die Build-Version der Benutzeroberfläche erstellt, die alle erforderlichen Ressourcen für den Betrieb der Anwendung enthält. Die Build-Version wird in den S3-Bucket hochgeladen werden, um sie dort zu speichern und als statische Ressource bereitzustellen. Darüber hinaus können auch andere Dateien im S3-Bucket gespeichert werden, wie beispielsweise Profilbilder der Benutzer oder Logos, die im Chat-System verwendet werden.

Database

Die Datenbank dient als zentraler Speicherort für die Chatnachrichten und Benutzerprofile. Dabei werden die erforderlichen Ressourcen innerhalb der Serverless.yml-File konfiguriert und bereitgestellt, einschließlich der Erstellung der DynamoDB-Tabelle und der Definition von Indizes und Schemata. Sie speichert die gesendeten Chatnachrichten und deren Inhalte, wie Absender, Zeitstempel und Empfänger. Jede Nachricht wird als ein Eintrag in der Datenbank gespeichert, um eine dauerhafte Aufzeichnung und den Zugriff auf vergangene Unterhaltungen zu ermöglichen. Die Datenbank erfasst zudem aktuell verbundene Clients, um eine Zuordnung der connectionId zu gewährleisten.

4. Implementierung

Für die Implementierung wird das Serverless-Framework verwendet. Um die Verbindung zu AWS-Lambda zu ermöglichen, wird hier der AWS CLI sowie Node.js und npm benötigt und demnach entsprechend installiert. Das Serverless-Framework dient dazu die Entwicklung, Bereitstellung und Skalierung der Anwendung zu vereinfachen indem beispielsweise AWS-Lambda-Funktionen und die zugehörige Infrastruktur verwaltet werden. Außerdem wird die einfache Konfiguration von AWS-Ressourcen über das Command-line Interface ermöglicht.

5. Fazit

In dieser Projektdokumentation haben wir ein Chat-System auf Basis von WebSockets und AWS detailliert beschrieben und dokumentiert. Das Projekt hatte das Ziel, eine Plattform für den Austausch von Nachrichten in Echtzeit bereitzustellen.

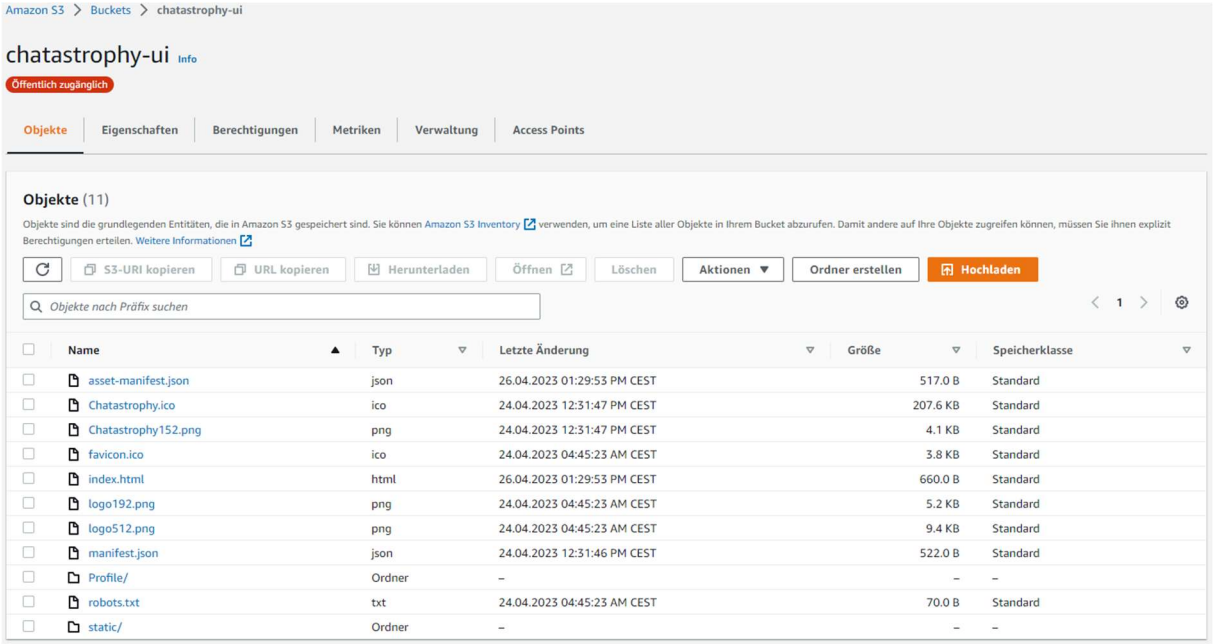
Im Verlauf des Projekts wurden verschiedene Technologien und AWS-Services eingesetzt, um die gesteckten Ziele zu erreichen. Die Integration von WebSockets ermöglicht dabei eine effiziente bidirektionale Kommunikation zwischen den Clients und dem Backend, was zu einer schnellen und reibungslosen Chat-Erfahrung führt.

Während des Projekts haben wir wertvolle Erfahrungen gesammelt und Herausforderungen gemeistert. Für die Zukunft gibt es verschiedene Möglichkeiten zur Weiterentwicklung und Verbesserung des Systems. Die Integration zusätzlicher Services wie CloudWatch zur Qualitätssicherung, Authentifizierung zur Datenkonformität und -Integrität oder Funktionen wie Dateiübertragung, oder Emojis könnte die Benutzererfahrung weiter verbessern. Zudem können Performance-Optimierungen und Fehlerbehebungen vorgenommen werden, basierend auf dem Feedback der Benutzer und den gesammelten Betriebsmetriken.

Insgesamt war das Projekt erfolgreich und hat die gesteckten Ziele erreicht. Die Entwicklung eines Chat-Systems auf Basis von WebSockets und AWS hat uns wertvolle Einblicke in die Bereiche Netzwerke, AWS und Cloud-Infrastruktur gegeben. Das erworbene Wissen und die gewonnenen Erfahrungen können wir in zukünftigen Projekten nutzen, um ähnliche Anforderungen zu erfüllen und innovative Lösungen zu entwickeln.

Anhang

S3-Bucket



API-Gateway

Handler-File (Upload in Lambda Funktion)

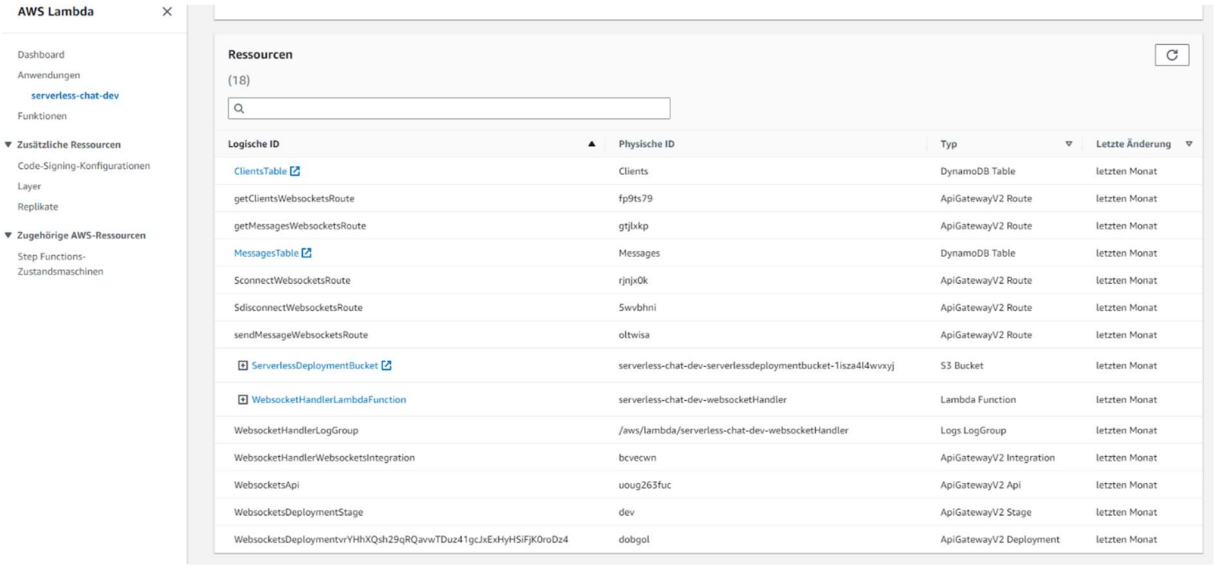
```ts
export const handle = async (
  event: APIGatewayProxyEvent,
): Promise<APIGatewayProxyResult> => {
  console.log("event:", event);
  const connectionId = event.requestContext.connectionId as string;
  const routeKey = event.requestContext.routeKey as Action;

  try {
    switch (routeKey) {
      case "$connect":
        return handleConnect(connectionId, event.queryStringParameters);
      case "$disconnect":
        return handleDisconnect(connectionId);
      case "getClients":
        return handleGetClients(connectionId);
      case "sendMessage":
        return handleSendMessage(
          await getClient(connectionId),
          parseSendMessageBody(event.body),
        );
      case "getMessages":
        return handleGetMessages(
          await getClient(connectionId),
          parseGetMessageBody(event.body),
        );
      default:
        return responseForbidden;
    }
  } catch (e) {
    if (e instanceof HandlerError) {
      await postToConnection(
        connectionId,
        JSON.stringify({ type: "error", message: e.message }),
      );
      return responseOK;
    }

    throw e;
  }
};
```

Rollenvergabe innerhalb der serverless.yml File (Konfiguration des IAM)

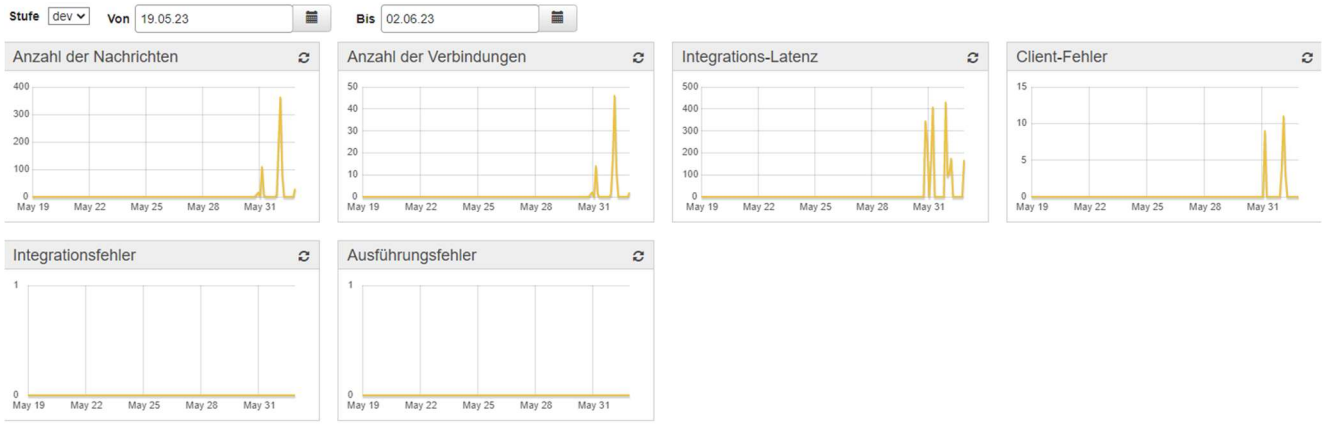
```yaml
iam:
  role:
    statements:
      - Effect: Allow
        Action:
          - "dynamodb:PutItem"
          - "dynamodb:GetItem"
          - "dynamodb:DeleteItem"
          - "dynamodb:Scan"
        Resource:
          - { "Fn::GetAtt": ["ClientsTable", "Arn"] }
      - Effect: Allow
        Action:
          - "dynamodb:Query"
        Resource:
          Fn::Join:
            - "/"
            - - { "Fn::GetAtt": ["ClientsTable", "Arn"] }
              - "index"
              - "*"
      - Effect: Allow
        Action:
          - "dynamodb:PutItem"
          - "dynamodb:GetItem"
          - "dynamodb:DeleteItem"
          - "dynamodb:Scan"
        Resource:
          - { "Fn::GetAtt": ["MessagesTable", "Arn"] }
      - Effect: Allow
        Action:
          - "dynamodb:Query"
        Resource:
          Fn::Join:
            - "/"
            - - { "Fn::GetAtt": ["MessagesTable", "Arn"] }
              - "index"
              - "*"
```

IAM in AWS

AWS Lambda Visualisierung der Schnittstellen



CloudWatch (hier Amazon API Gateway)



Benutzerhinweise

Aufruf der Website:
http://chatastrophy-ui.s3-website.us-east-1.amazonaws.com/

GitHub-Repository:
https://github.com/NiklasRtg

Hinweise:

1. Sollte eine Session bereits über dieselbe connectionId (z.B.: gleicher Browser) geöffnet sein, werden auf der neuen keine Clients angezeigt, da hier die Connection unterbunden wird (um das gleichzeitige Senden von Nachrichten zu vermeiden)
2. Falls Sie mit sich selbst schreiben wollen, empfehlen wir, den Inkognito-Modus zu verwenden, da hier eine neue connectionId vom Browser vergeben wird
3. Es werden nur aktive Clients angezeigt, Chatverläufe sind also nur aufrufbar, wenn diese auch online sind
4. Ein Responsive Design ist nur bedingt implementiert, bitte öffnen Sie die Website über einen großen Bildschirm (Laptop, Computer)