Stadt-Land-Fluss

Online-Multiplayer-Game

Studiengang: Medieninformatik

Team-Mitglieder:
Sophia Schwalb, Natalie Hussfeldt, Anh Thu Bui, Gamze Isik

Gliederung

1. Projektidee und Inspiration
2. Projektorganisation
3. Frameworks & Cloud Services & Architektur
4. Serverless & FaaS
5. CI/CD Pipeline, Test
6. Probleme
7. Lessons Learned
8. Ausblick
9. Demo

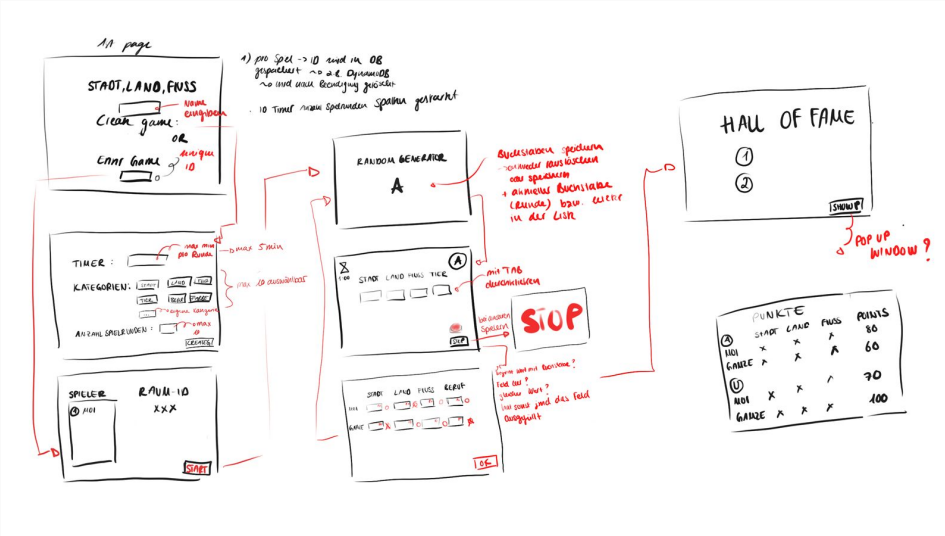
Projektidee und Inspiration

Projektidee und Inspiration

Erste Ideen : VVS Daten, Praktikumsdatenbank

Inspiration : Skribbl IO

Spielidee Stadt-Land-Fluss :
- Online Spiel
- Raumerstellung & Spieldaten bestimmen
- Beitreten des Raums
- Waiting Room
- Letter Generator & Spielrunde
- Anzeigen der Eingaben der Spieler
- Hall Of Fame



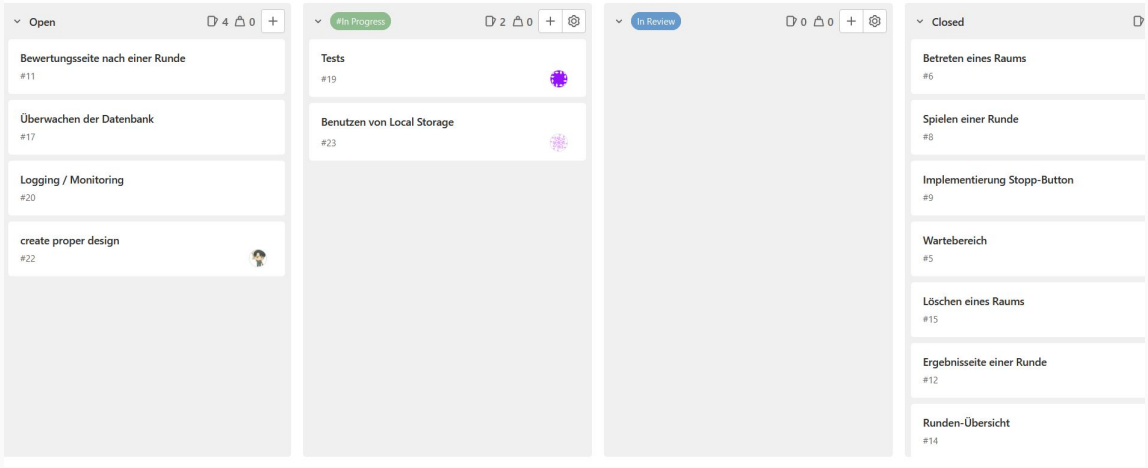
Projektorganisation

Projektorganisation

- Als Kommunikationsmittel wurden Discord und Whatsapp genutzt
- Regelmäßige Rücksprachen & Meetings im Projektteam
- Bei Problemen Austausch zu Lösungsansätzen

GitLab :

- Erstellen von Issues
- Arbeitsaufteilung über Zuweisung von Issues



Frameworks & Cloud Services & Architektur

Framework

Frontend

- Angular als Typescript basiertes Front-End Webapplikationsframework
- Viele Beispiele zur Anwendung von Websockets und AWS im Zusammenhang mit Angular

Backend

- Python für die Serverseite

Serverless Framework

- AWS Datenbanken, API-Routen, Lambdas lokal über eine serverless.yaml verwalten und deployen

Framework

serverless.yml

```yaml
service: aws-serverless-websockets
provider:
  name: aws
  runtime: python3.9
  region: eu-central-1
  environment:
    CONNECTION_DB_TABLE: ${self:resources.Resources.My
  iamRoleStatements:
    - Effect: Allow
      Action:
        - "dynamodb:PutItem"
        - "dynamodb:GetItem"
        - "dynamodb:DeleteItem"
        - "dynamodb:Scan"
        - "dynamodb:UpdateItem"
        - "dynamodb:Query"
        - "dynamodb:BatchWriteItem"
        - "dynamodb:UpdateItem"
      Resource:
        - Fn::GetAtt: [MyAppTable, Arn]
    - Effect: Allow
      Action:
        - "execute-api:ManageConnections"

      Resource:
        - "arn:aws:execute-api:*:*:**/@connections/*"
```

```yaml
functions:
  defaultHandler:
    handler: handler.defaultHandler
    events:
      - websocket:
          route: $default

  broadcast_to_room:
    handler: broadcast_message_handler.broadcas
    events:
      - websocket:
          route: broadcast_to_room
  stop_round:
    handler: broadcast_message_handler.stop_rou
    events:
      - websocket:
          route: stop_round
```

```yaml
resources:
  Resources:
    MyAppTable:
      Type: "AWS::DynamoDB::Table"
      Properties:
        AttributeDefinitions:
          - AttributeName: "roomId"
            AttributeType: "S"
        KeySchema:
          - AttributeName: "roomId"
            KeyType: "HASH"
        BillingMode: PAY_PER_REQUEST
        TableName: game_data
```

Cloud Services/Komponenten

AWS

- DynamoDB
- Lambdas
- CloudWatch
- API Gateway
- Kostenloses Kontingent für AWS
- Serverless (Functions, Resources, IAM Roles)
- S3

DynamoDB


```
{
 "roomId": "9b2efd07-dba3-4cc5-96a4-f5a49691b27e",
 "categories": [
  "Fluss",
  "Name",
  "Essen",
  "Film"
 ],
 "game_players": [
  {
   "Lena": {
    "connectionId": "Vx6OCcL2FiACHKg=",
    "next_round": false,
    "points": [
     5,
     10
    ],
    "status": "active",
    "values": [
     [
      "",
      "",
      "Fisch",
      ""
     ],
     [
      "",
      "",
      "Pasta",
      ""
     ]
    ]
   }
  },
  {
   "Mia": {
    "connectionId": "Vx6TjcuMFiACFGg=",
    "next_round": false,
    "points": [
     65,
     70
    ],
    "status": "inactive",
    "values": [
     [
      "Fluss",
      "Frank",
      "Fisch",
      "Fluch der Karibik"
     ],
     [
      "Po",
      "Paul",
      "Pizza",
      "Pirates of the Caribbean"
     ]
    ]
   }
  }
 ],
 "number_of_players": 2,
 "rounds": 2,
 "timer": "120",
 "used_letters": [
  "F",
  "P"
 ]
}
```


- NoSQL - Einträge in der Datenbank variieren und folgen keinem starren Muster
- Nicht-relationaler Ansatz, unabhängig von einem festen Tabellenschema
- Schnelle NOSQL-Schlüssel-Wert-Datenbank für beliebig große Datenmengen
- Python Boto3 Bibliothek

```
region = 'eu-central-1'
api = boto3.client('apigatewaymanagementapi',
                   endpoint_url='https://tjmy88rryf.execute-api.eu-central-1.amazonaws.com/dev/',
                   region_name=region)
dynamo = boto3.client('dynamodb', region_name=region)
```

- get_item()
- put_item()
- update_item()
- delete_item()

Lambdas

Functions (17)

Last fetched 1 minute ago | Actions

Filter by tags and attributes or search by keyword

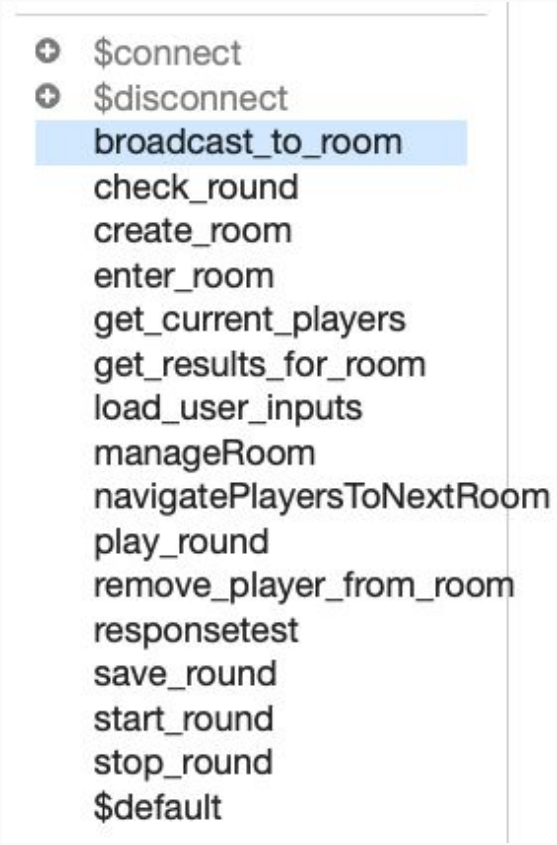
Function name	Description	Package type	Runtime
manageRoom	-	Zip	Python 3.9
aws-serverless-websockets-dev-stop_round	-	Zip	Python 3.9
aws-serverless-websockets-dev-check_round	-	Zip	Python 3.9
server_response_test	-	Zip	Python 3.9
aws-serverless-websockets-dev-play_round	-	Zip	Python 3.9
aws-serverless-websockets-dev-get_current_players	-	Zip	Python 3.9
aws-serverless-websockets-dev-remove_player_from_room	-	Zip	Python 3.9
aws-serverless-websockets-dev-enter_room	-	Zip	Python 3.9
aws-serverless-websockets-dev-load_user_inputs	-	Zip	Python 3.9
aws-serverless-websockets-dev-start_round	-	Zip	Python 3.9
aws-serverless-websockets-dev-navigatePlayersToNextRoom	-	Zip	Python 3.9
aws-serverless-websockets-dev-save_round	-	Zip	Python 3.9
aws-serverless-websockets-dev-defaultHandler	-	Zip	Python 3.9
aws-serverless-websockets-dev-create_room	-	Zip	Python 3.9
aws-serverless-websockets-dev-broadcast_to_room	-	Zip	Python 3.9
get_user_name	-	Zip	Python 3.9
aws-serverless-websockets-dev-get_results_for_room	-	Zip	Python 3.9

Lambdas

```
createRoom(): void {
  const roomId = uuidv4();
  websocket.send(JSON.stringify({
    action: 'create_room',
    roomId: roomId,
    timer: this.selectedTimerOption,
    rounds: this.rounds,
    usedLetters: [],
    categories: this.selectedCategories,
    userName: this.username,
    numberOfPlayers: this.numberPlayers
  }));
```

```
import json
import boto3
import uuid
import random

region = 'eu-central-1'
api = boto3.client('apigatewaymanagementapi',
                   endpoint_url='https://tjmy88rryf.execute-api.eu-central-1.amazonaws.com/dev/',
                   region_name=region)
dynamo = boto3.client('dynamodb', region_name=region)


# creates a new room with the given values or default values, if no values are given
def create_room(event, context):
  print(event)

  connection_id = event['requestContext']['connectionId']
  request_body = json.loads(event['body'])

  room_id = str(request_body['roomId']) if 'roomId' in request_body else str(uuid.uuid4())
  timer = request_body['timer'] if 'timer' in request_body else "180"
  rounds = request_body['rounds'] if 'rounds' in request_body else 3
  number_of_players = request_body['numberOfPlayers'] if 'numberOfPlayers' in request_body else 2
  categories = request_body['categories'] if 'categories' in request_body else ['Stadt', 'Land', 'Fluss']
  used_letters = request_body['usedLetters'] if 'usedLetters' in request_body else []
  user_name = request_body['userName'] if 'userName' in request_body else None
  users = [
    {user_name: {'status': 'active', 'points': [], 'connectionId': connection_id, 'next_round': False, 'values': []}}]

  if user_name is None:
    print('Error while creating new room: no user name')
    api.post_to_connection(ConnectionId=connection_id, Data=json.dumps(
      {'statusCode': 400, 'method': 'create_room',
       'body': 'Failed to create new room: user name is required'}).encode('utf-8'))
    return {'statusCode': 400, 'body': 'Failed to create new room: user name is required'}

  try:
    print(f'Saving new room with room id {room_id}')
    json_obj_room = {'roomId': room_id, 'timer': str(timer), 'rounds': rounds, 'number_of_players': number_of_players,
                     'used_letters': used_letters, 'categories': categories, 'game_players': users}
    game_data_table = boto3.resource('dynamodb', region_name=region).Table('game_data')
    game_data_table.put_item(Item=json_obj_room)

    api.post_to_connection(ConnectionId=connection_id, Data=json.dumps(
      {'statusCode': 200, 'method': 'create_room', 'body': 'success'}).encode('utf-8'))
    return {'statusCode': 200, 'body': 'Successfully created new room'}

  except Exception as e:
    print(f'Error while saving new room in DB: {e}')
    api.post_to_connection(ConnectionId=connection_id, Data=json.dumps(
      {'statusCode': 400, 'method': 'create_room',
       'body': 'Some error occurred while saving new room in DB: ' + json.dumps(e)}).encode('utf-8'))
    return {'statusCode': 400, 'body': 'Error while saving new room in DB'}
```

API Gateway



- Erstellung eines eigenen Websocket-API Gateways
- Lambdas werden durch eine Websocket-API getriggert
- Über ein API-Gateway können Serverless-Funktionen bereitgestellt und verwaltet werden
- übersetzt Anfragen an einen einzelnen Endpunkt und leitet sie an eine andere Functions as a Service-Funktion weiter

CloudWatch

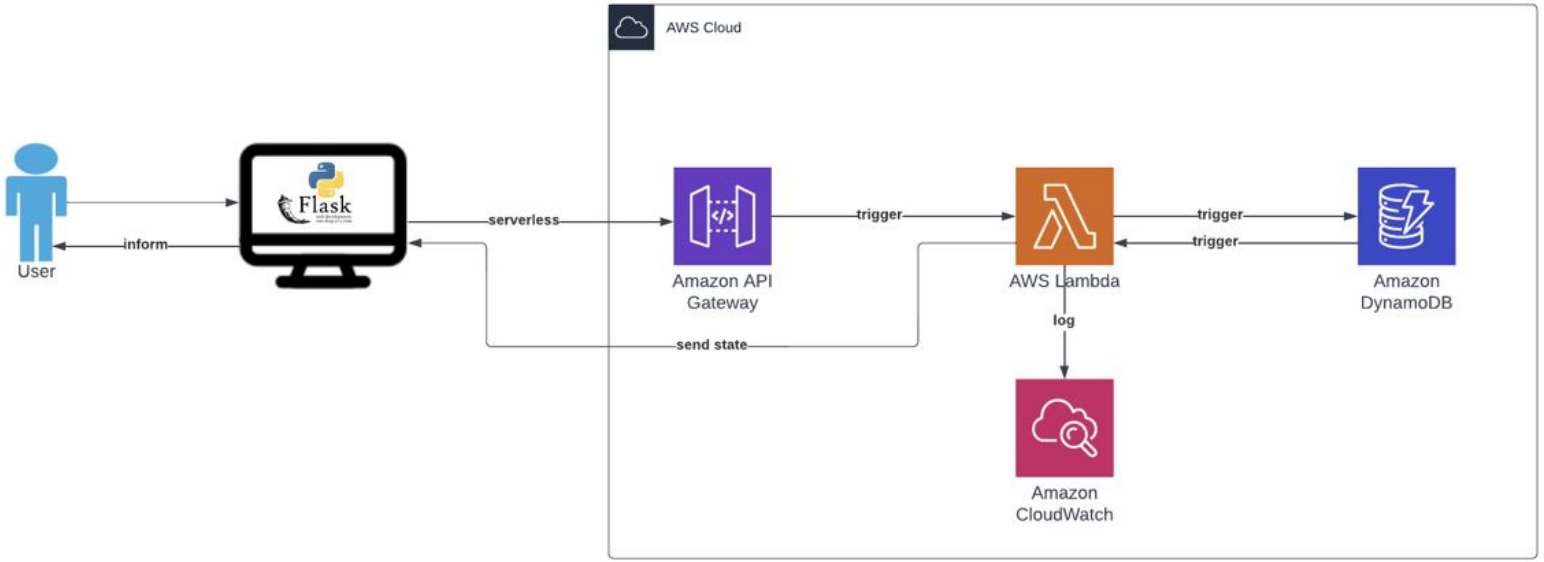
- Wird verwendet, um unsere API zu protokollieren
- Logging
- Fehler schneller finden

2022-07-23T15:33:33.052+02:00	created dict with username as key and inputs as values {'username': 'lena'}
2022-07-23T15:33:33.052+02:00	values list for user lena: [{'L': [{'S': 'rügen'}, {'S': 'russland'}, {'S': 'rhein'}]}]
2022-07-23T15:33:33.052+02:00	added user inputs of player lena: {'username': 'lena', 'Stadt': 'rügen', 'Land': 'russland', 'Fluss': 'rhein'}
2022-07-23T15:33:33.052+02:00	created dict with username as key and inputs as values {'username': 'mia'}
2022-07-23T15:33:33.052+02:00	values list for user mia: [{'L': [{'S': 'ravensburg'}, {'S': 'russland'}, {'S': ''}]}]
2022-07-23T15:33:33.052+02:00	added user inputs of player mia: {'username': 'mia', 'Stadt': 'ravensburg', 'Land': 'russland', 'Fluss': ''}
2022-07-23T15:33:33.052+02:00	dict: [{'username': 'lena', 'Stadt': 'rügen', 'Land': 'russland', 'Fluss': 'rhein'}, {'username': 'mia', 'Stadt': 'raven…
2022-07-23T15:33:33.052+02:00	saving points of last round for room with id b42b9bc2-ece3-481c-b166-3925727dd37d and user inputs: [['rügen', 'russland'…
2022-07-23T15:33:33.063+02:00	reformatting ['rügen', 'russland', 'rhein']
2022-07-23T15:33:33.063+02:00	checking input value rügen, letter R
2022-07-23T15:33:33.063+02:00	checking input value russland, letter R
2022-07-23T15:33:33.063+02:00	checking input value rhein, letter R
2022-07-23T15:33:33.063+02:00	reformatted user inputs: [['rügen', 'russland', 'rhein'], ['ravensburg', 'russland', '']]
2022-07-23T15:33:33.063+02:00	reformatting ['ravensburg', 'russland', '']
2022-07-23T15:33:33.063+02:00	checking input value ravensburg, letter R

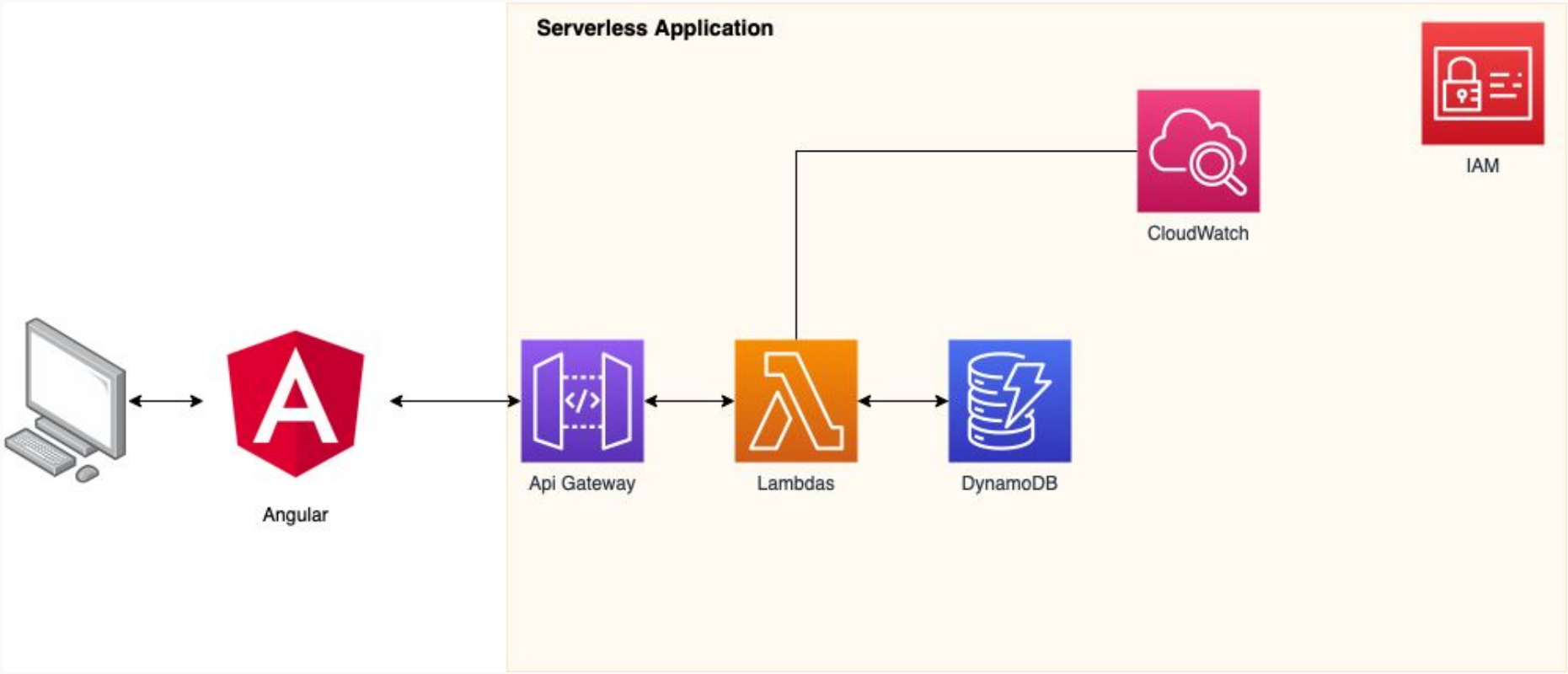
AWS S3 Bucket

- Laden des Build Ordners in S3 Bucket
- Bereitstellung der Webanwendung durch einen generierten Link

Alte Architektur



Neue Architektur



Serverless & FaaS

Serverless

- bedeutet nicht, dass kein Server existiert
- Cloud-Anbieter übernimmt Management des Servers
- Entwickler muss den Server somit nicht erstellen, nicht updaten und nicht skalieren, wenn die Anwendung wächst
- Entwickler kann sich ganz auf den Code seiner Anwendung konzentrieren

Warum haben wir Serverless gewählt ?

- Idee schnell umsetzbar
- schnellere Einarbeitung als bei Aufsetzen von eigenem Server
- Interesse
- Bei viel Traffic keine Gedanken um Skalierung
- “Pay as you go” Lösungen der Cloudanbieter (zu Beginn zahlt man nichts und erst mit zunehmendem Nutzen)

Nachteile

- Serverless Dienste führen zu einer Abhängigkeit des Systems
- Wechsel des Cloud-Anbieters = viel Aufwand
- Cloud-Anbieter kann entscheiden, wann wichtiges Update durchgeführt wird

FaaS

- Fokus nur noch auf einzelne Funktionen, Funktionalität der Anwendung (“Geschäftslogik”)
- Infrastruktur, Betriebssystem, Laufzeitumgebung, Anwendung schon gegeben
- Logik implementiert man im Rahmen von Funktionen
- Funktionen sind zustandslos => man muss sich um Persistenz außerhalb kümmern

Vorteile	Nachteile
sehr einfach skalierbar, (Begrenzung durch Budget)	Coldstarts (Wenn bei Aufruf eine neue Instanz des Dienstes initialisiert werden muss, dauert das eine gewisse Zeit)
kostengünstig, da kein laufender Serverprozess	Abhängigkeit von Cloud-Anbieter
sehr einfach Funktionen zu warten und verändern (nur einzelne Funktionen müssen deployed werden)	
Funktionsaufruf sobald ein Triggerevent ausgelöst wurde (nur dann Rechenleistung belegt)	

Tests

CI/CD & Testing

PIE SOCKET

- Online Websocket Tester

GitLab CI/CD

- AWS Nutzer angelegt für die Pipeline
- 3 Stages : test, build, deploy
- Je nach Branch wird die Anwendung getestet oder deployed

Tests

- moto als Python-Bibliothek (AWS-Services einfach mocken)
- aufgegliedert in set/get/delete

Probleme

Probleme 

- Einarbeitung in verschiedene Frameworks (Flask, serverless, Angular etc.)
  - teilweise wenig Informationen im Internet zu finden, welche speziell zu unserem Projekt passen
- Timing-Probleme beim Abfragen und Anlegen von Daten
- DynamoDB in Kombination mit komplexen Datenstrukturen
- Deployment der Lambda-Funktionen
- Datenverlust bei Neuladen

Lessons Learned

Vor- und Nachteile Cloud Services

Serverless Framework

+	−
Verwaltung aller Elemente wie Datenbanken, API-Routen, Lambda Functions etc. über eine serverless.yml möglich	Einarbeitungszeit zu Beginn
AWS-Kontoschlüsseln/ Kontoanmeldeinformationen müssen nicht in Skripte/ Umgebungsvariabeln kopiert werden	

Vor- und Nachteile Cloud Services

DynamoDB

✚	⊖
Skalierbarkeit - unterstützt Tabellen jeder Größe	Nicht unbedingt für komplexe Datenstrukturen geeignet
Einfaches Abfragen, Anlegen und Manipulieren von Daten	
NoSQL - keine feste Datenstruktur	

Lessons Learned

- Grundverständnis von AWS Lambdas, API Gateway und serverless framework
- CloudWatch zu aktivieren, obwohl es was kostet, ist sehr hilfreich
- CI/CD Pipeline aufzubauen

- Aus Zeitgründen verfolgt man die funktionierende Lösung
- Mehr Zeit einplanen zum Einarbeiten in die Thematik
- Es wäre sehr sinnvoll, die Anwendung auch lokal aufzubauen
- Mehr Zeit einplanen für Security-Maßnahmen

Ausblick

- Design vervollständigen
- eigene Kategorien anlegen
- Spracheinstellung
- eigene Avatare
- Lambdas sollen nicht für produktiv und deployment Umgebung gleichzeitig deployed werden

Demo