Bier Pongo

Kevin Fischer
OTH Amberg-Weiden
k.fischer2@oth-aw.de

Markus Fleischmann
OTH Amberg-Weiden
m.fleischmann2@oth-aw.de

Darren Fürst
OTH Amberg-Weiden
d.fuerst@oth-aw.de

Simon Kleber
OTH Amberg-Weiden
s.kleber1@oth-aw.de

Fabian Klinger
OTH Amberg-Weiden
f.klinger1@oth-aw.de

Jakob Lindner
OTH Amberg-Weiden
j.lindner3@oth-aw.de

Christian Renner
OTH Amberg-Weiden
c.renner@oth-aw.de

***Abstrakt*— Cloudfähige, skalierbare, multiuser- und multidevice-fähige Bier Pong Social Media Platform.**

Schlüsselwörter—Beer Pong, Cloud, Data, Team

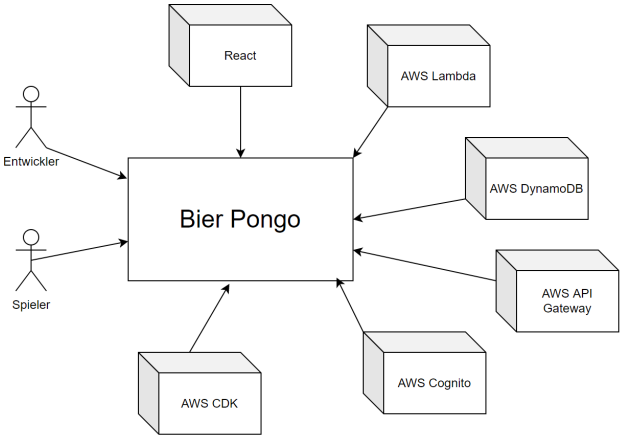
I. PROBLEMSTELLUNG

A. *Mission statement*

Bier Pongo ist eine Webanwendung, welche es Bierpong-Spielern ermöglicht, auch räumlich voneinander getrennt zusammen zu spielen und immer den aktuellen Stand des Spielfelds vor sich zu haben. Benötigt wird dafür nur ein Gerät mit Webbrowser und Internetverbindung. Bier Pongo maximiert den Spielspaß dank intuitiver Bedienung, die keiner ausführlichen Erklärung bedarf.

B. *Kontextabgrenzung*

Im Folgenden wird das Gesamtsystem als Blackbox dargestellt. Alle verwendeten externen Systeme werden als Box um die Anwendung "Bier Pongo" herum dargestellt. Der menschliche Akteur "Spieler" kann in diesem Fall als Plural aufgefasst werden, da an einem Bierpong-Spiel immer zwei Spieler beteiligt sind.



Für die Entwicklung des Frontend wird die JavaScript Library "React" verwendet. Das Backend bilden die Dienste "Lambda" und "DynamoDB" von AWS. Als Schnittstelle dient der Dienst "AWS API Gateway", um Daten im jeweilig gewünschten Format abzurufen. Für die Benutzerverwaltung wird "AWS Cognito" verwendet. Um alle Dienste von AWS per Infrastructure as Code zu orchestrieren, wird das "AWS CDK" (Cloud Development Kit) benutzt.

C. *Qualitätsziele*

Qualitätsziel	Beschreibung
Gute Benutzbarkeit	Bier Pongo ist für die Spieler intuitiv zu bedienen und ohne Erklärung spielbar.
Hohe Zuverlässigkeit	Das System steht den Spielern jederzeit zur Verfügung und bildet den Spielstand live ab.
Gute Wartbarkeit	Bier Pongo ist ohne großen Administrationsaufwand betreibbar und leicht um zusätzliche Features erweiterbar.

D. *Entscheidende Rahmenbedingungen*

Vorgabe	Beschreibung
Zeitlicher Rahmen	Das Projekt ist innerhalb von 5 Kalenderwochen abzuschließen.
Technischer Anspruch	Das Projekt muss mind. einen der Themenbereiche BigData / Cloud / NoSQL in einer hinreichend technischen Tiefe bedienen.
Einfachheit in der Anwendung	Die Anwendung soll aber so einfach gehalten werden, dass sie im gegebenen Zeitraum umgesetzt werden kann.

II. Lösungsstrategie

A. *Lösungsansätze*

Die zuvor benannten Qualitätsziele wurden beim Projekt Bier Pongo folgendermaßen berücksichtigt:

Qualitätsziel	Lösungsansatz
Gute Benutzbarkeit	• Dank der simpel gehaltenen React-Oberfläche kann die Anwendung direkt ohne Tutorial o.Ä. benutzt werden. • Kennt man die Regeln des "Real-Life"-Spiels kommt man ohne Einweisung direkt zurecht. • Die Benutzer benötigen lediglich einen Webbrowser.
Hohe Zuverlässigkeit	• Da sämtliche Infrastruktur durch Dienste von AWS bereitgestellt wird, ist die Anwendung praktisch ohne zu erwartende Ausfälle verfügbar. • Den einzigen Flaschenhals stellt die Internetverbindung des Benutzers dar.
Gute Wartbarkeit	• Durch Nutzung des AWS Clous Development Kit können etwaige Änderungen an der Infrastruktur im Code getätigt und anschließend deployed werden. • Zusätzliche Features können für das separate React-Frontend entwickelt werden, ohne dadurch das Backend auf AWS zu beeinflussen.

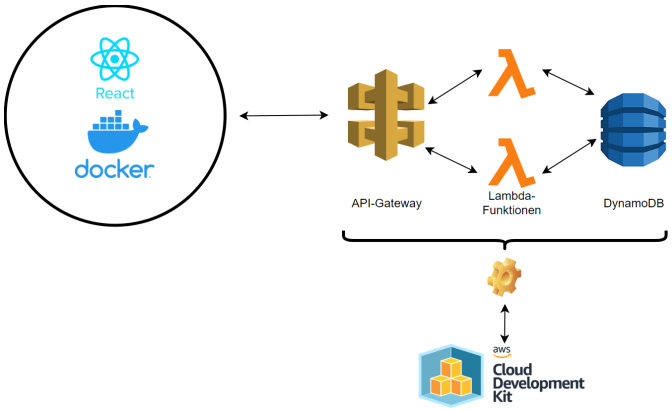
B. *Technologiestack*

Das Frontend von Bier Pongo besteht aus einer React-Anwendung in einem Docker-Container.

Die Backend-Infrastruktur besteht aus Diensten von AWS. Die Brücke zwischen React und AWS bildet eine REST-API, realisiert durch AWS API-Gateway. Die zur Verfügung gestellten Endpoints nehmen API-Aufrufe von Seiten React entgegen und bedienen diese, indem AWS Lambda-Funktionen aufgerufen werden. Durch die Lambda-Funktionen werden die Spiel-Daten aus der AWS DynamoDB gelesen und geschrieben.

Für die Verwaltung der AWS Infrastruktur wird per Infrastructure as Code das AWS Cloud Development Kit verwendet.

Zur besseren Übersicht wird hier nochmals die Struktur veranschaulicht:



III. Lösungsdetails

A. *Architekturentscheidung*

Die Wahl des Frontend-"Frameworks" fiel auf die JavaScript-Library React, da sich mit ihr leicht moderne User Interfaces erstellen lassen und im Entwicklerteam schon Erfahrungen mit React und JavaScript im Allgemeinen vorhanden waren. Für React sprach auch die Modularität mittels Components. Diese sind wiederverwertbar und austauschbar und konnten unabhängig voneinander entwickelt werden.

Damit die Applikation von allen Entwicklern ohne Kompatibilitätsprobleme auf z.T. unterschiedlichen Betriebssystemen gebaut, entwickelt und getestet werden kann, lag es nahe dafür einen Docker-Container zu verwenden. Auch nach der Entwicklungszeit ist geplant, den Docker-Container auf AWS Elastic Container Service zu hosten.

Das Backend wird komplett durch Dienste von AWS gebildet. Sämtliche Services werden über das AWS Cloud Development Kit per Infrastructure as Code definiert.

Das AWS API-Gateway stellt verschiedene Endpoints zur Verfügung, damit die React-App auf Spiel-Daten zugreifen kann. Für die API-calls wird die Bibliothek axios verwendet. Das API-Gateway ruft abhängig vom Typ der Anfrage die entsprechende Lambda-Funktion auf.

Alle Lambda-Funktionen sind in der Programmiersprache Python geschrieben, da das Team hier auch schon Erfahrung hatte. Mithilfe der Python-Bibliothek boto3 werden die Spieldaten, welche in einer Datenbanktabelle der AWS DynamoDB liegen, in der Lambda-Funktion gelesen und geschrieben.

In der AWS DynamoDB Datenbank liegt lediglich eine Tabelle, welche für jedes Spiel die Spiel-ID sowie den Spielstand (getroffene Becher) speichert. Die Daten gelangen anschließend über das API-Gateway zurück zur aufrufenden React-App. Entsprechend gilt der beschriebene Weg für das Updaten eines Spiels.

B. Beschreibung der Lambda-Funktionen

Um einen detaillierteren Eindruck der Backend-Funktionalitäten zu erhalten, werden die einzelnen Funktionen in AWS Lambda kurz beschrieben:

POST-Lambda:

Request-Typ:	POST-Request
Aufgabe:	Erstellen eines neuen Spiels
Beschreibung:	Die Funktion erstellt eine eindeutige GameID mit 8 Zeichen und schreibt anschließend ein neues Item in die Datenbank, das wie folgt aufgebaut ist:

```
{
    "GameId": game_id,
    "State": "",
    "playerCount": 1
}
```

Response:	200 OK, 500 Error bei Spielerstellung

PUT-Lambda:

Request-Typ:	PUT-Request
Aufgabe:	Anpassung des Spielstands nach jeder Runde
Beschreibung:	Die Funktion erhält eine GameID und den State-String der letzten Spielrunde und fügt diesen zum State-String in der Datenbank hinzu.
Response:	200 OK, 400 Invalide GameID, 500 Error beim updaten des Spiels

GET-Lambda:

Request-Typ:	GET-Request
Aufgabe:	Aktuellen Spielstand abfragen
Beschreibung:	Die Funktion liefert anhand der GameID den State-String des Spiels zurück
Response:	200 OK, 404 Spiel nicht gefunden

JOIN-Lambda:

Request-Typ:	GET-Request
Aufgabe:	Lässt andere Spieler oder Zuschauer einem vorhandenen Spiel beitreten
Beschreibung:	Die Funktion nimmt eine GameID entgegen, inkrementiert dann „playercount“ in der Datenbank und gibt dann den aktuellen Wert von „playercount“ sowie die GameID zurück
Response:	200 OK, 404 Spiel nicht gefunden, 500 Error beim Beitritt

C. Verhalten

Im Folgenden wird der Ablauf eines Spiels beispielhaft in Stichpunkten beschrieben:

- Spieler 1 öffnet die URL und erstellt ein Spiel.
- Über den "post"-Endpoint wird das API-Gateway dazu aufgefordert, die entsprechende post-Lambda-Funktion aufzurufen.
- Die Lambda-Funktion erstellt die eindeutige GameID und speichert diese in der Datenbanktabelle ab (siehe POST-Lambda oben)
- Spieler 2 öffnet die URL und tritt dem von Spieler 1 erstellten Spiel durch Eingabe der Spiel-ID bei
- Über den "join"-Endpoint wird das API-Gateway dazu aufgefordert, die entsprechende join-Lambda-Funktion aufzurufen.
- Die Lambda-Funktion erhöht die Spieler-Anzahl des entsprechenden Spiels um 1
- Nun nutzen beide Spieler die gleiche URL, welche die Spiel-ID enthält, um zu spielen
- Spieler 1 trifft beispielsweise 2 Becher, klickt diese an und beendet anschließend seinen Zug
- Der State-String der letzten Runde wird per API-call dem API-Gateway übergeben. Beispiel: "1:52"
- Das bedeutet, dass Spieler 1 die beiden Becher mit den Nummern 2 und 5 getroffen hat
- Der API-Gateway ruft die put-Lambda-Funktion auf, um die Spieldaten in der Datenbank zu aktualisieren
- Dabei wird dem State (Spielstand) der String angehängt
- Spieler 1 und 2 spielen abwechselnd, bis eine Seite alle Becher getroffen hat
- Für die jeweiligen Spieleupdates wird dann entsprechend die PUT-Lambda verwendet

IV. Fazit und Ausblick

Zum aktuellen Stand bietet Bier Pongo den Spielern alle grundsätzlichen Funktionalitäten, um gemeinsam spielen zu können. Es kann ein Spiel angelegt, sowie einem bereits angelegten Spiel beigetreten werden. Beide Seiten können ihre getroffenen Becher über die Benutzeroberfläche auswählen und so den Spielstand aktualisieren.

A. Herausforderungen während der Entwicklung

Folgende Schwierigkeiten stellten uns vor zusätzliche Probleme, die in der Summe viel Zeit in Anspruch genommen haben:

AWS Policies:

Oft kamen wir an den Punkt, dass fehlende Berechtigungen in AWS verschiedene Probleme verursacht haben. Beispielsweise muss das API-Gateway auf Lambda zugreifen können und die Lambda-Funktionen müssen auf die DynamoDB zugreifen können. Hier die richtigen

Berechtigungen zu finden damit alles funktioniert, hat relativ viel Recherche erfordert.

CDK Policies definieren

Aus dem vorherigen Problem ergab sich ein Folgeproblem, da wir die Policies im AWS CDK entsprechend per Code definieren mussten nicht die übersichtliche graphische Oberfläche von AWS zur Verfügung haten

Online-Ressourcen auf TypeScript ausgelegt

Da unsere Wahl für die Definition der Infrastruktur per CDK auf Python fiel, mussten wir die überwiegend auf TypeScript ausgelegten Erklärungen und Codebeispiele auf Python ummünzen. Davon betroffen waren im Besonderen Methoden und Methodensignaturen aus Bibliotheken, welche für den Zugriff auf Services von AWS benötigt werden.

CORS

Beim ersten vollständigen Deploy musste das API-Gateway angepasst werden, um die CORS-Richtlinien für das jeweilige Teilsystem nicht zu verletzen. Beispielsweise wurden Mock-Endpoints eingefügt, welche aus die HTTP-Methode „Options“ antworten.

Remapping Template

Frägt man Daten per GET-Request über die API an werden die nötigen Informationen für die Lambdas verschachtelt in einem Feld „params“ zur Verfügung gestellt. Hier mussten die Lambdas mit dem Request-Typ GET (GET- und JOIN-Lambda) angepasst werden, um die Informationen aus dem der Funktion übergebenen Parameter „event“ richtig auszulesen. Dafür wurde in AWS ein Remapping Template definiert.

Docker

Die vom API-Gateway bereitgestellte Backend-URL musste Docker bereits während dem Docker Build übergeben werden und nicht während Docker Run.

React-Hooks

Das Problem bestand darin, den State-String (Spielstand) über mehrere React-Komponenten hinweg zu speichern. Gelöst wurde dies durch Verwendung der useContext()-Hook.

B. Ausblick

Aus Kostengründen wird die Anwendung derzeit nicht auf AWS gehostet, sondern wird lokal als Docker-Container betrieben. Für die Zukunft ist aber das Szenario denkbar, den Docker-Container auf dem AWS Elastic Container Service zu hosten.

In naher Zukunft ist geplant, eine Benutzerverwaltung mittels AWS Cognito zu realisieren, damit sich Spieler bei der Anwendung registrieren und anmelden können, um beispielsweise die persönlichen gespielten Spiele im Profil zu hinterlegen. Geplant ist dies durch Implementierung einer Spielhistorie. Damit soll es den Spielern ermöglicht werden, bereits abgeschlossene Spiele einsehen und nachverfolgen zu können. Außerdem soll es für die Anwendung zukünftig eine Passwort-Vergessen-Funktionalität geben, um ggf. das Passwort zurückzusetzen.

Denkbar für die Anwendung ist auch, eine Statistik-Funktionalität einzubetten. Damit könnten die Spieler beispielsweise ihre persönlichen Werte wie Gewinnrate, allgemeine Trefferrate und Trefferrate für bestimmte Becher einsehen oder ihre Leistung auf einem zentral geführten Highscore-Board mit anderen Spielern vergleichen.