FireForceDefense
Technical Report

Cameron Barbee, Tim Hoffmann, Christian Piffel, Tobias Schotter,
Sebastian Schuscha, Philipp Stangl, Thomas Stangl

I. Einführung und Ziele

Im vorliegenden Projekt soll das Tower-Defense-Spiel „FireForceDefense“ erweitert werden. Ziel ist es, das Spiel um eine einfache Benutzerverwaltung, d.h. Registrierung und Anmeldung, zu ergänzen. Außerdem soll eine Rangliste geschaffen werden, wofür eine Spielstand-Speicherung notwendig ist.

II. Bausteinsicht

Diese Sicht zeigt die statische Zerlegung des Systems in Bausteine sowie deren Beziehungen.

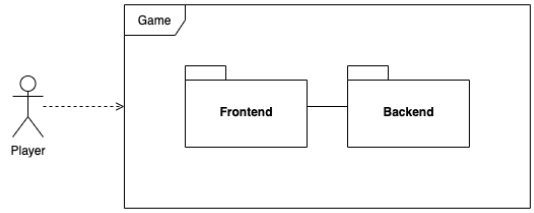

Fig. 1. Kontextabgrenzung

A. Gesamtsystem

Die Anwendung basiert auf einer Client-Server-Architektur (1). Frontend und Backend kommunizieren über eine RESTful-API.

B. Frontend

Dieser Abschnitt beschreibt die client-seitige Frontend-Architektur. Das Frontend wird unter Zuhilfenahme des Frameworks Vue.js realisiert.

Es ist selbst in mehrere Unter-Bausteine zerlegt:

1) Model: Dieser Baustein enthält die Logik für den Problembereich im Frontend. Die eigentlichen Anzeigekomponenten besitzen Referenzen auf die Instanzen der Klassen des Model-Bereichs, sodass Eingaben an das Model weitergegeben und dort verarbeitet werden können. Schließlich besitzt das Model Schnittstellen, mittels denen Informationen über den aktuellen Zustand abgefragt werden können, um basierend darauf die Anzeige anzupassen.

Im Model ist beispielsweise die Spiellogik und die Schnittstelle zum Speichern und Abrufen von Spielständen verortet.

2) Components: In diesem Modul sind die Vue-Komponenten gesammelt, mit denen die eigentliche Anzeige im Webbrowser realisiert wird. Die Komponenten sind dabei hierarchisch geordnet, so besteht ein Level beispielsweise aus einer Sidebar und der LevelMap, die LevelMap wiederum enthält einzelne Zellen und so weiter.

Die Komponenten behandeln alle Eingaben und senden bei Bedarf entsprechende Nachrichten an das Model.

3) SCSS: Hier werden zentral genutzte Stile im SCSS-Format abgelegt.

4) Lang: Dieser Baustein ist für die Internationalisierung, genauer gesagt die Übersetzung, zuständig. Aktuell ist lediglich eine deutsche Sprachvariante hinterlegt.

5) Levels: Dieses Modul enthält die Level-Definitionen. Eine Level-Definition beschreibt den initialen Aufbau des Spielfelds und die auftretenden Effekte. Jedes Level muss anhand der jeweiligen Definition im LevelManager des Model-Bereichs registriert werden.

6) Cells, Contents und Effects: In diesen drei Bausteinen sind die verschiedenen, konkreten Zell-, Inhalts- bzw. Effekt-Typen samt ihren jeweiligen Eigenschaften hinterlegt.

C. Backend

Dieser Abschnitt beschreibt die server-seitige Backend-Architektur.

1) Datenbank: Die Speicherung der Daten erfolgt im dokumentenorientierten NoSQL-Datenbankmanagementsystem *MongoDB*. Zusätzlich wird die Bibliothek *mongoose* für das Object Data Modeling (ODM) verwendet.

2) Laufzeitumgebung: JavaScript-basierte Plattform Node.js mit dem serverseitigen Webframework ExpressJS.

III. Verteilungssicht

Das Verteilungssicht beschreibt die Verteilung des Gesamtsystems, wichtige Begründungen für diese Verteilungsstruktur und die Zuordnung von Softwareartefakten zu Bestandteilen der Infrastruktur. Zentrale Bestandteile der Verteilungsstruktur sind (A) das Kubernetes Cluster, (B) das Datenbank-Cluster und (C) die GitLab CI/CD-Pipeline.

A. Kubernetes Cluster

Für ein Kubernetes-Cluster wurde sich entschieden, aufgrund der Tatsache, das mehrere virtuelle Cluster (sog. Namespaces) auf demselben physischen Cluster unterstützt werden. Dadurch kann die CI/CD Pipeline im *Gitlab-managed-apps* Namespace laufen, währenddessen die Anwendung auf einem anderen Namespace bereitgestellt wird.

Das Kubernetes Cluster wird über den Elastic Kubernetes Service des Cloud-Providers Amazon Web Services bereitgestellt. Aus datenschutzrechtlichen Gründen werden nur Cloud Server, die der Verfügbarkeitszone Frankfurt (eu-central-1) angehörig sind, verwendet.

B. Datenbank-Cluster

Das Datenbank-Cluster besteht aus drei Replikationen. Es stellt jeweils eine Datenbank für den Entwicklungs- und Produktionsbetrieb bereit. Näheres zur Datenbank wird im Abschnitt II-C.1 erklärt.

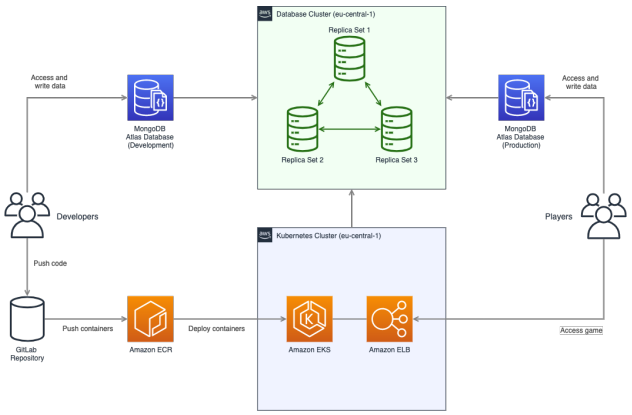

Fig. 2. Cloud Infrastruktur

C. CI/CD-Pipeline

Für die Bereitstellung der CI/CD Pipeline ist ein Kubernetes-Executor[1] auf dem Kubernetes Cluster verantwortlich. Dieser verbindet sich mit der Kubernetes Cluster API und erstellt einen Pod für jeden GitLab CI Job. Dieser Pod besteht aus einem Build-Container und einem zusätzlichen Container für jeden Service. Die Pipeline umfasst das Erstellen, Testen und Bereitstellen der Anwendung. Folgende vier Phasen durchläuft jeder Push in das Gitlab-Repository:

1) Dependencies: Installation der npm-Pakete in das Verzeichnis node_modules.

2) Lint: Durchführung statischer Code-Analysen.

3) Build: Alle notwendigen Bau-/Vorbereitungsaufgaben der Anwendung werden ausgeführt.

4) Test: Durchführung von Unit Tests, um die Korrektheit des Codes zu validieren.

Die letzten beiden Phasen werden nur im main-Zweig des Repositories durchlaufen, um die Anwendung für den Produktionsbetrieb bereitzustellen.

5) Dockerize: Wenn alle Unit Tests bestanden sind, wird für Front- und Backend jeweils ein Docker-Image gebaut. Anschließend werden beide Images im Docker-Registry abgelegt, damit sie für die nächste Stage (Deploy) verfügbar sind.

6) Deploy: In der letzten Phase werden die Docker Images aus der Docker-Registry auf dem Kubernetes Cluster als Container bereitgestellt. Die Anwendung ist dann, bis einschließlich dem Tag der Präsentation, öffentlich zugänglich.

IV. Querschnittliche Konzepte

Dieser Abschnitt beschreibt übergreifende, prinzipielle Regelungen und Lösungsansätze, die an mehreren Stellen relevant sind.

A. <Konzept 1>

<Erklärung>

B. <Konzept 2>

<Erklärung>

References

[1] GitLab. Gitlab Runner: Kubernetes executor. `https://docs.gitlab.com/runner/executors/kubernetes.html`