# Projekt IP Location Finder

**Seminararbeit**

im Studiengang Informatik

an der Dualen Hochschule Baden-Württemberg

Stuttgart

von

**Robert Orbach**

29.03.2025

| | |
|---|---|
| **Bearbeitungszeitraum** | März 2025 bis April 2025 |
| **Matrikelnummer, Kurs** | 8489365, INF22A |
| **Bei Dozent** | Niels Riekers |

# Erklärung

Ich versiche hiermit, dass der Seminararbeit mit dem Thema *„Projekt IP Location Finder“* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben.

Robert Orbach
70174 Stuttgart, 29.03.2025

# Inhaltsverzeichnis

# 1 Einleitung

Cloud Computing hat sich in den letzten Jahren zu einem zentralen Bestandteil moderner IT-Infrastrukturen entwickelt. Unternehmen nutzen Cloud-Dienste, um ihre Anwendungen effizienter zu betreiben, Kosten zu senken und die Skalierbarkeit zu verbessern.

Im Rahmen der Vorlesung haben wir das Themenfeld erkundet und uns mit den Möglichkeiten und Herausforderungen von Cloud Computing auseinandergesetzt. Um diese selbst auszuprobieren, haben wir die Chance, ein eigenes Projekt unter anwendung der Möglichkeiten des Coud Computing zu realisieren.

In diesem Projekt wurde eine Webanwendung unter Nutzung von Cloud Computing Ansätzen entwickelt, die dem Nutzer erlaubt von IP Adressen auf die ungefähre geographische Lage zu schließen.

# 2 Theorie

## 2.1 Was ist Cloud Computing?

Klassische IT-Infrastrukturen basieren auf physischen Servern und Rechenzentren, die vor Ort betrieben werden. Da diese jedoch oft teuer zu betreiben und schwer zu skalieren sind, setzen viele Unternehmen auf Cloud Computing.

Cloud Computing bezeichnet die Bereitstellung von IT-Ressourcen wie Servern, Speicher, Datenbanken, Netzwerken und Software über das Internet. Es gibt drei Hauptmodelle:

- **Infrastructure as a Service (IaaS):** Bereitstellung von virtuellen Maschinen, Netzwerken und Speicher. Der Nutzer verwaltet die Infrastruktur selbst. Beispiele: AWS EC2, Azure Virtual Machines.

- **Platform as a Service (PaaS):** Bereitstellung einer Plattform für die Entwicklung und Bereitstellung von Anwendungen. Der Anbieter verwaltet die Infrastruktur, während der Nutzer sich auf die Anwendung konzentriert. Beispiele: Azure App Service, Google App Engine.

- **Software as a Service (SaaS):** Bereitstellung von Software über das Internet. Der Nutzer verwendet die Software, ohne sich um die Infrastruktur oder Plattform kümmern zu müssen. Beispiele: Google Workspace, Microsoft Office 365.

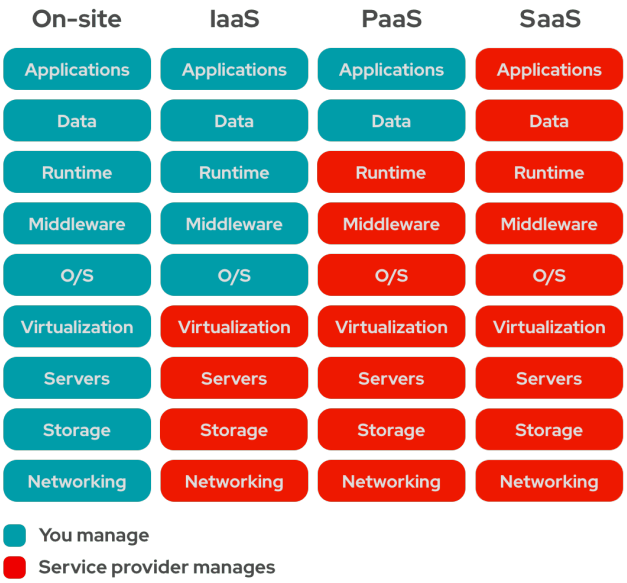


Abbildung 2.1: Cloud Computing Modelle

## 2.2 Warum Cloud Computing?

Die Cloud bietet zahlreiche Vorteile:

- **Skalierbarkeit:** Ressourcen können je nach Bedarf dynamisch angepasst werden.
- **Kostenersparnis:** Keine Investitionen in physische Hardware, Bezahlung nach Nutzung.
- **Flexibilität:** Zugriff auf Ressourcen von überall.
- **Automatisierung:** Tools wie Terraform und Ansible ermöglichen die einfache Verwaltung und Bereitstellung von Infrastruktur.

Jedoch gibt es Nachteile bei der Nutzung von Cloud Computing:

- **Abhängigkeit:** Unternehmen sind von Cloud-Anbietern abhängig.
- **Sicherheit:** Schutz sensibler Daten und Zugriffskontrolle sind kritisch.
- **Kosten:** Fehlende Kontrolle über die Ressourcennutzung kann zu hohen Kosten führen.

## 2.3 Terraform

Terraform ist ein Open-Source-Tool zur Infrastrukturautomatisierung. Es ermöglicht die deklarative Beschreibung von Infrastruktur als Code (IaC). Mit Terraform können Ressourcen wie virtuelle Maschinen, Netzwerke und Datenbanken in verschiedenen Cloud-Anbietern wie AWS, Azure oder Google Cloud bereitgestellt werden.

Vorteile von Terraform:

- Plattformunabhängigkeit.
- Wiederholbare und konsistente Bereitstellung.
- Versionskontrolle der Infrastruktur.

## 2.4 Ansible

Ansible ist ein Open-Source-Tool zur Konfigurationsverwaltung und Automatisierung. Es wird verwendet, um Software zu installieren, Konfigurationen zu ändern und Anwendungen bereitzustellen.

Folgende Features machen Ansible attraktiv:

- Agentenlos: Es benötigt keine zusätzliche Software auf den Zielsystemen.

- Einfachheit: Konfigurationen werden in YAML geschrieben.
- Skalierbarkeit: Kann große Umgebungen effizient verwalten.

# 3 Umsetzung

## 3.1 Architektur

Die Architektur des Projekts umfasst folgende Komponenten:

1. **Virtuelles Netzwerk:** Ein Azure Virtual Network mit Subnetzen für die Anwendung und die Datenbank.

2. **Datenbank:** Eine PostgreSQL Flexible Server Instanz, die als PaaS-Dienst bereitgestellt wird.

3. **Webanwendung:** Mehrere virtuelle Maschinen (VMs) mit einer Load Balancer-Konfiguration, die eine Next.js-Anwendung hosten.

4. **Monitoring:** Eine Monitoring Instanz über die mittels pgAdmin, Prometheus und Graphana die Überwachung der Anwendung erfolgt.

5. **Automatisierung:** Terraform wird verwendet, um die Infrastruktur zu erstellen, und Ansible, um die VMs zu konfigurieren und die Anwendung bereitzustellen.

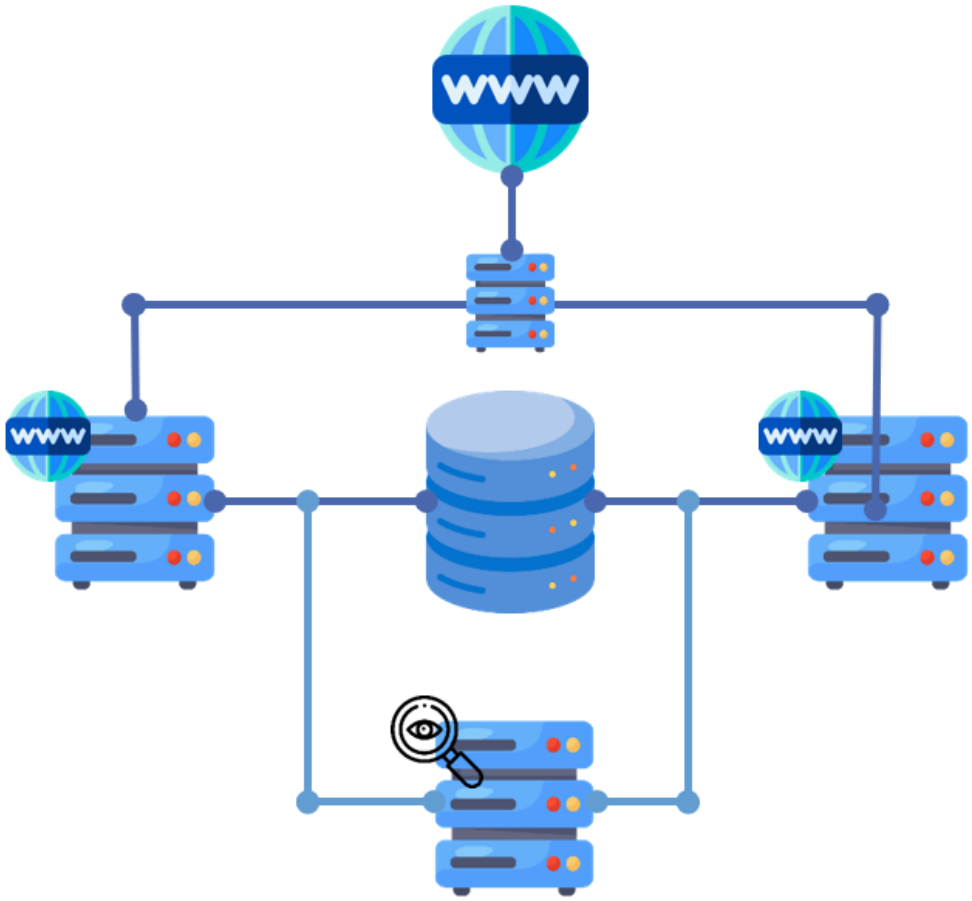


Abbildung 3.1: Architektur des Projekts

## 3.2 Aufbauablauf

### 3.2.1 Schritt 1: Infrastruktur mit Terraform

Zuerst müssen die netzwerke und Server mit Terraform erstellt werden. Hierzu wird ein Azure Resource Group erstellt, in der die Ressourcen gruppiert werden. Anschließend wird ein Virtual Network mit Subnetzen für die Anwendung und die Datenbank erstellt. Danach wird eine PostgreSQL Flexible Server Instanz mit privatem Netzwerkzugriff bereitgestellt. Daruf werden zwei VMs für die Webanwendung erstellt und ein Load Balancer konfiguriert. Abchließend wird ein Monitoring Server erstellt.

### 3.2.2 Schritt 2: Konfiguration mit Ansible

Konfiguration der App VMs

- Installation von Node.js und PostgreSQL-Client auf den VMs.
- Bereitstellung der Next.js-Anwendung aus einem Git-Repository.
- Konfiguration der Umgebungsvariablen für die Datenbankverbindung.
- Erstellung eines Systemd-Dienstes, um die Anwendung automatisch zu starten.

Konfiguration des Monitoring Servers

- Installation von pgAdmin, Prometheus und Grafana auf dem Monitoring Server.
- Konfiguration von pgAdmin zur Verwaltung der Datenbank.
- Konfiguration von Prometheus und Grafana zur Überwachung der Anwendung.
- Einrichtung von Dashboards in Grafana zur Visualisierung der Metriken.

# 4 Fazit

## 4.1 Wann ist die Lösung sinnvoll?

Die gewählte Lösung ist besonders geeignet für:

- Anwendungen, die eine hohe Skalierbarkeit und Verfügbarkeit erfordern.
- Projekte, die von der Automatisierung der Infrastruktur profitieren.
- Teams, die eine konsistente und wiederholbare Bereitstellung benötigen.

## 4.2 Was ist der Aufwand?

Der Aufwand für die Implementierung der Lösung umfasst:

- **Initiale Einrichtung:** Die Erstellung der Terraform- und Ansible-Skripte erfordert Zeit und Fachwissen.
- **Wartung:** Änderungen an der Infrastruktur oder der Anwendung können einfach durch Anpassung der Skripte vorgenommen werden.
- **Kosten:** Die Nutzung von Cloud-Diensten ist abhängig von der Ressourcennutzung und kann bei falscher Planung teuer werden.

## 4.3 Vorteile der Lösung

- Automatisierung reduziert menschliche Fehler.

- Skalierbarkeit und Flexibilität der Cloud-Dienste.
- Sicherheit durch private Netzwerke und SSL-Verbindungen.

## 4.4 Nachteile der Lösung

- Abhängigkeit von Cloud-Anbietern.
- Komplexität bei der initialen Einrichtung.
- Laufende Kosten für die Cloud-Dienste.

# 5 Zusammenfassung

Das Projekt zeigt, wie moderne Cloud-Technologien wie Terraform und Ansible genutzt werden können, um eine skalierbare und sichere Infrastruktur für eine Webanwendung bereitzustellen. Die Kombination aus IaaS und PaaS ermöglicht eine effiziente Nutzung der Cloud-Ressourcen, während die Automatisierung die Verwaltung vereinfacht. Die Lösung ist ideal für Anwendungen, die hohe Verfügbarkeit und Flexibilität erfordern, und bietet eine solide Grundlage für zukünftige Erweiterungen.