Karlsruher Institut für Technologie

Evaluation und Vergleich von Cloud Plattformen für das Internet der Dinge

Cloud Platform Evaluation and Comparison in an Internet of Things Scenario

Bachelorarbeit
von

Nico Kuhn

An der Fakultät für Informatik
Institut für Telematik
Lehrstuhl für Pervasive Computing Systems
Forschungsgruppe TECO

Erstgutachter:	Prof. Dr. Michael Beigl
Zweitgutachter:	Prof. Dr. Hannes Hartenstein
Betreuender Mitarbeiter:	Andrei Miclaus

Bearbeitungszeit: 18. Januar 2016 – 12. Mai 2016

I declare that I have developed and written the enclosed thesis completely by myself and have not used sources or means without declaration in the text.

Karlsruhe, den 12. Mai 2016

.....

(**Nico Kuhn**)

Danksagung

An dieser Stelle möchte ich all jenen danken, die mich im Rahmen dieser Bachelorarbeit unterstützt und motiviert haben. Zuerst gebührt mein Dank Herrn Andrei Miclaus, der meine Bachelorarbeit betreut und begutachtet hat. Für die hilfreichen Anregungen und die konstruktive Kritik bei der Erstellung dieser Arbeit möchte ich mich herzlich bedanken. Ganz besonders möchte ich mich bei meinen Eltern Norbert Kuhn und Johanna Kuhn bedanken, die mir durch ihre Unterstützung mein Studium ermöglicht haben.

Zusammenfassung

Aus der aktuell steigenden Beliebtheit und dem Trend, Systeme in die Cloud auszulagern, folgt eine rasant wachsende Anzahl an Anbietern, die genau diese Bedürfnisse erfüllen wollen. Deshalb wird es auch immer wichtiger, genau diese Vielzahl an Anbietern überschauen zu können und speziell zu vergleichen. Jedoch besitzt jede dieser Plattformen unterschiedliche Architekturen, Modelle zur Konfiguration, zum Pricing und unterschiedliche vordefinierte Bausteine zum schnellen Zusammensetzen eines Cloud-Dienstes[1]. Diese Vielzahl von Ansätzen sind dem eigentlichen Nutzer gegenüber meist intransparent und somit nur sehr schwer zu vergleichen und zu optimieren. In dieser Arbeit wird ein Ansatz entwickelt, um Plattformen und speziell deren Services für das Internet der Dinge zu vergleichen.

Hierzu wurden zuerst allgemeine Key Performance Indikatoren definiert, um damit die später folgende Evaluation durchführen zu können. Auf ausgewählten Cloud Plattformen (Google Cloud Platform[2] / Microsoft Azure[3] / IBM Bluemix [4]) wurden zwei Szenarien mittels der bereitgestellten Services der Cloud-Plattformen implementiert. Zum einen die Verarbeitung und Visualisierung von streaming Daten und zum anderen analytische Berechnungen auf einem zugrunde liegenden Datensatz. Für die eigentliche Evaluation wurde ein Evaluationsframework[5] erstellt, um für die implementierten Szenarien Testdaten zu generieren und Messungen zu den KPIs durchzuführen, auszuwerten und anschaulich darzustellen. Mit den Evaluationsergebnissen der verschiedenen Plattformen wurde zum Schluss ein Modell gebildet, womit ein Decision Support System erstellt wurde. Mit dem in dieser Arbeit entstandenen Decision Support System und dem Evaluations Framework ist ein unterstützendes System zur Entscheidungsfindung der untersuchten Cloud Platformen, sowie zur Messung wichtiger KPIs entstanden. Durch die modulare und frei zugängliche Software ist es außerdem möglich, die Software zur Unterstützung weiterer Plattformen zu erweitern.

[1] Angebotener Service einer Cloud-Platform zur bearbeitung einer meist speziellen Aufgabe
[2] https://console.cloud.google.com
[3] portal.azure.com
[4] https://console.eu-gb.bluemix.net
[5] Komponentenbasiertes Softwaregerüst zur evaluation verschiedener Cloud-Plattformen

Abstract

The rising popularity of cloud platforms and the trend towards outsourcing systems to the cloud, ensues a rapidly growing number of providers meeting this users needs. Therefore, it is also becoming increasingly important to be able to accurately overlook this large number of suppliers and compare them to each other. However, each of these platforms have different architectures, models for configuration, pricing and different predefined blocks for rapidly assembling a cloud service. This variety of approaches is not transparent to the actual user and therefore difficult to compare and optimize. In this work, an approach on how to compare cloud platforms and specially their services for the Internet of things is developed.

Therefore Key Performance Indicators have to be defined in order to perfom a cross platform evaluation on different cloud plattforms. On selected Cloud-Platforms (Google Cloud Platform / IBM Bluemix / Microsoft Azure) two scenarios were implemented with the use of built in Cloud-Services. On the one hand the processing and visualization of streaming data and on the other hand analytic calculations on a predefined dataset. For the actual evaluation an evaluation framework was created to generate sample data for the implemented scenarios and to measure the key performance indicators. The results of the evaluation from the various platforms are used to build a model for a decision support system. With this system and the evaluation framework a ready to use support system for decision making of the investigated cloud platforms arose. Due to the modular and open source software it is also possible to extend this software to support additional platforms.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

CSP Cloud Service Provider

KPI Key Performance Indikator

SaaS Software as a Service

PaaS Platform as a Service

IaaS Infrastruktur as a Service

IoT Internet der Dinge

U/s Einheit pro Sekunde

API Application Program Interface

IQR Interqartile Range

1. Einleitung

1.1 Motivation

Mittels Cloud Computing ist es möglich, Ressourcen wie Hardware oder Rechenleistung über das Internet bereitzustellen. Große Investitionskosten sind daher nicht mehr nötig, da Cloud-Dienste nach dem Pay-as-you-go-Prinzip abgerechnet werden. Mit der fortschreitenden Digitalisierung der Fertigungstechnik und Logistik wird eine technologische Revolution, das Internet der Dinge, in der Industrie möglich. Es gibt bereits viele Ansätze von Cloud-Plattformen, die diese Schritte unterstützten. Cyberphysische Systeme sind bereits fester Bestandteil des alltäglichen Lebens. Um das Internet der Dinge in der Industrie zu realisieren, müssen verschiedene Aufgaben wie Datenaufnahme, Datentransport (Service-Buse und Protokolle), Datenspeicherung in Datenbanken (Historische- und Echtzeitdaten), Analytische Berechnungen, Visualisierung und die Rückkopplung mit den Anlagen (Alarm, Notifications) auf entfernte Systeme (Cloud) ausgelagert werden.

Da die meisten Cloud-Plattformen unterschiedliche Architekturen, Modelle zur Konfiguration, zum Pricing und unterschiedliche vordefinierte Bausteine zum schnellen zusammensetzten eines Cloud-Dienstes verwenden, ist es schwer sich einen Überblick über diese Vielzahl verschiedener Anbieter zu schaffen und eine Entscheidung für eine passende Plattform zu treffen. Die vorliegende Arbeit beschäftigt sich deshalb mit dem *Cloud Provider Selection Problem*. Dieses Problem beschreibt die, aufgrund der intransparenten Cloud Architekturen, schwierige Vergleichbarkeit verschiedener Cloud Service Provider (CSP). Es werden verschiedene CSP untersucht und eine Software zur Evaluation dieser Anbieter erstellt. Dabei werden Key Performance Indikatoren betrachtet und auf den untersuchten Anbietern gemessen um einen Vergleich dieser Anbieter vornehmen zu können. Mit den daraus folgenden Ergebnissen soll ein Ansatz zum Problem der Anbieterauswahl geschaffen werden. Hierfür wird eine mathematische Modellabstraktion und ein Decision Support System erstellt.

1.2 Aufbau der Arbeit

Die vorliegende Arbeit vermittelt in Kapitel 2 zunächst einen Überblick über die Grundlagen zum Thema Cloud Computing im Allgemeinen. Dafür werden wichtige Definitionen des Cloud Computing eingeführt, sowie Charakteristika, Service- und Deploymentmodelle beschrieben. Es werden außerdem wichtige Key Performance Indikatoren der Literatur

aufgeführt und eine Übersicht der für diese Arbeit relevanten Cloud Service Provider gegeben. Dafür werden die genutzten Cloud Services und das Preismodell der jeweiligen Anbieter genauer betrachtet. Kapitel 3 gibt eine Übersicht über sachverwandte Arbeiten und den aktuellen Stand der Forschung. Außerdem wird die Abgrenzung dieser Arbeit zu den bereits bestehenden Arbeiten erläutert.

In Kapitel 4 ist der Entwurf der in dieser Arbeit enstehenden Software zu finden. Dies beginnt mit der Beschreibung der abstrakten Lambda Architektur bis zum konkreten Softwareentwurf eines Evaluationsframeworks und der dazugehörigen Serverapplikation. Weiter wird der Softwareentwurf eines Decision Support Systems beschrieben, welches mit den in Kapitel 6 enstehenden Evaluationsergebnissen eine Entscheidungsstütze für das *Cloud Provider Selection Problem* darstellt. Mit dem so entstandenen Softwareentwurf wird in Kapitel 5 die konkrete Implementierung der bereits genannten Softwarekomponenten beschrieben, sowie die Implementierung der genutzten Cloud Dienste erläutert.

Kapitel 6 beschreibt die mithilfe der in dieser Arbeit entstandenen Softwarekomponenten durchgeführte Evaluation der CSP. Dabei wird das Vorgehen und die ausgeführten Tests genauer betrachtet, sowie eine Übersicht der Evaluationsergebnisse gegeben. Außerdem wird das *Cloud Provider Selection Problem* in ein abstraktes mathematisches Modell formuliert und mit den Evaluationsergebnissen eines typischen Anwendungsfalls beispielhaft durchgerechnet.

Zuletzt findet in Kapitel 7 eine kritische Reflexion der Arbeit statt. Es werden die gewonnenen Erkenntnisse und die sich daraus ergebenden Implikationen beschrieben. Dafür wird die Forschungsfrage beantwortet und eine Diskussion bezüglich des Zielereichungsgrades geführt. Die Arbeit schließt mit einem Ausblick auf den Forschungsbedarf zum Thema Cloud Computing und des *Cloud Provider Selection Problem.*

2. Grundlagen

Dieser Abschnitt gibt einen Überblick über die Grundlagen von Cloud Computing Technologien, exisiterenden Standards und Charakteristika sowie die Definitionen der unterschiedlichen Deployment und Service Modelle . Außerdem folgt eine Zusammenfassung existierender, Cloud-Computing relatirender, Key Performance Indikatoren und ein Überblick über die betrachteten CSP.

2.1 Cloud Computing

"Cloud Computing ist ein Modell, das es erlaubt, bei Bedarf jederzeit und überall bequem über ein Netz auf einen geteilten Pool von konfigurierbaren Rechnerressourcen (z. B. Netze, Server, Speichersysteme, Anwendungen und Dienste) zuzugreifen, die schnell und mit minimalem Managementaufwand oder geringer Serviceprovider-Interaktion zur Verfügung gestellt werden können" ([NIS11]). Dies ist die Definition der US-amerikanischen Standardisierungsstelle NIST (National Institute of Standards and Technology), die auch von der ENISA (European Network and Information Security Agency) genutzt wird.

Cloud-Provider stellen also ihre Hardware über ein Netz zur Verfügung. Mittels Virtualisierungstechnologien und grafischen sowie Kommandozeilen Schnittstellen wird dem Nutzer diese Physischen Infrastrukturen zur Verfügung gestellt. Der eigentliche Nutzer benötigt dabei keinerlei Wissen über die darunterliegende Hardware. Die Abrechnung dafür erfolgt typischerweise auf einem *pay-per-use* Prinzip, so dass nur die tatsächlich genutzten Ressourcen der Infrastruktur abgerechnet werden.

"Eine Cloud-Infrastruktur ist die Sammlung von Hardware und Software, die die fünf wesentlichen Merkmale des Cloud Computing ermöglicht. Die Cloud Infrastruktur beinhaltet die physikalische Schicht und die Abstraktionsschicht. Die physikalische Schicht besteht aus den Hardware-Ressourcen, die notwendig sind, um die Cloud-Dienste zur Verfügung zu stellen, dies sind in der Regel Server-, Speicher- und Netzwerkkomponenten. Die Abstraktionsschicht besteht aus der Software, welche über der physikalische Schicht eingesetzt wird und die wesentlichen Merkmale der Cloud manifestiert"([NIS11]).

Cloud Computing Charakteristika

Folgende fünf Eigenschaften charakterisieren nach der NIST-Definition ([NIS11]) einen Cloud Service:

On-demand Self Service: Die Provisionierung der unterschiedlichen Ressourcen läuft automatisch ohne zwingende Interaktion mit dem Cloud Provider ab.

Broad Network Access: Die Services sind über das Netzwerk verfügbar und ermöglichen somit eine Nutzung von sämmtlichen Client-Plattformen (Pc, Handy, Laptop Tablet, ...).

Resource Pooling: Die Ressourcen des Anbieters liegen in einem Pool vor, von dem sie den Anwendern, je nach Bedarf, zugewiesen werden. Dabei wissen die Anwender nicht, wo die Ressourcen sich befinden, sie können aber vertraglich den Speicherort, also z. B. Region, Land oder Rechenzentrum, festlegen.

Rapid Elasticity: Die Services können schnell und elastisch zur Verfügung gestellt werden, in manchen Fällen auch automatisch. Aus Anwendersicht scheinen die Ressourcen daher unendlich zu sein.

Measured Services: Die Ressourcennutzung kann gemessen und überwacht werden und entsprechend bemessen auch den Cloud-Anwendern zur Verfügung gestellt werden.

Cloud Deployment Modell

NIST unterscheidet vier Bereitstellungsmodelle (Deployment Models):

In einer **Private Cloud** wird die Cloud-Infrastruktur nur für eine Institution betrieben. Sie kann von der Institution selbst oder einem Dritten organisiert und geführt werden und kann dabei im Rechenzentrum der eigenen Institution oder einer fremden Institution stehen.

Von einer **Public Cloud** wird gesprochen, wenn die Services von der Allgemeinheit oder einer großen Gruppe, wie beispielsweise einer ganzen Industriebranche, genutzt werden können und die Services von einem Anbieter zur Verfügung gestellt werden.

In einer **Community Cloud** wird die Infrastruktur von mehreren Institutionen geteilt, die ähnliche Interessen haben. Eine solche Cloud kann von einer dieser Institutionen oder einem Dritten betrieben werden.

Werden mehrere Cloud Infrastrukturen, die für sich selbst eigenständig sind, über standardisierte Schnittstellen gemeinsam genutzt, wird dies **Hybrid Cloud** genannt.

Cloud Service Modell

Grundsätzlich können drei verschiedene Kategorien von Servicemodellen unterschieden werden:

Die höchste Ebene der Abstraktion ist das **Software as a Service (SaaS)** Modell. "Software als gehosteter Dienst implementiert mit dem Zugriff über das Internet" (nach [NIS11]). SaaS Dienstanbieter bieten Software und Applikationen an, welche auf ihrer Infrastruktur laufen. D.h. der Provider ist für die Software und die darunter liegende Infrastruktur verantwortlich. Der Nutzer kann dann über ein Webinterface die angebotene Software nutzen. Beispiel hierfür ist iCloud oder gMail.

Die zweite Ebene der Abstraktion ist das **Platform as a Service (PaaS)** Modell. Ein PaaS-Provider stellt eine komplette Infrastruktur bereit und bietet dem Kunden auf der

Plattform standardisierte Schnittstellen an, die von Diensten des Kunden genutzt werden. In dieser Umgebung kann der Nutzer eigene Applikationen entwickeln und betreiben. Der Nutzer hat dabei keinen Zugriff auf die darunterliegenden Schichten (Betriebssystem, Hardware).

Die unterste Ebene der Abstraktion heißt **Infrastruktur as a Service (IaaS)**. IaaS ist die bedarfsabhängige Bereitstellung virtueller Infrastrukturkomponenten wie z.B. Rechenleistung, Datenspeicher oder Netze durch einen Cloud Provider. Der Nutzer ist selbst für die Auswahl, die Installation und den Betrieb der Software auf den gemieteten Systemen verantwortlich.

2.2 Key Performance Indikatoren

Key Performance Indikatoren sind Leistungskennzahlen, welche die Leistung bestimmter Systeme, Dienste oder Operationen wiederspiegeln. Dieses Kapitel gibt einen Überblick der Key Performance Indikatoren nach [FRL13].

KPIs in der Literatur

Benchmark finishing time : Diese Metrik misst, wie lange die Instanz, auf der der Benchmark läuft, zur Fertigstellung benötigt. Der Benchmark stellt dabei Aufgaben, welche die wichtigsten Systemressourcen beanspruchen (CPU, memory, disk I/O).

Operation response time : Diese Metrik misst, wie lange eine Speicheroperation bis zur Fertigstellung benötigt.

Time to consistency : Diese Metrik misst die Zeit vom Schreiben eines Datums auf einen Speicherdienst, bis alle Leseoperationen des Datums konsistente und gültige Werte liefert.

Cost per operation : Diese Metrik gibt die Kosten einer Speicheroperation wieder.

Round Trip Time: Zeit, welche ein Netzwerkpaket vom Sender bis zum Empfänger und wieder zurück benötigt. Dies spiegelt die Übertragunszeit eines Paketes innerhalb der Netzwerkgrenzen wieder.

Response time : Die Antwortzeit für eine Operation ist die Zeit von dem Zeitpunkt aus, an dem der Client mit der Operation beginnt, bis das letzte Byte der Antwort den Client wieder erreicht. Hierbei ist die Zeit zur Bearbeitung der Anfrage miteinbegriffen im Gegensatz zur *Round Trip Time*. Die Art der Anforderung und das Verhalten der Verarbeitung muss dabei konkret definiert sein.

Paket Loss: Prozentsatz der verlorenen Pakete zur gesamten Übertragung.
Die Formel hierfür lautet:

$$\frac{Verlorene\_Pakete}{Anzahl\_aller\_Pakete * 100} \tag{2.1}$$

Bandwidth: Anzahl der Daten, die innerhalb einer Zeiteinheit übertragen werden kann. Dabei wird nicht die tatsächliche Kapazität angegeben, sondern die maximale Nennkapazität.

Throughput: Anzahl der übertragenen Daten pro Zeiteinheit . Nur die tatsächlich übertragenen Daten werden berücksichtigt, damit die für den Benutzer verfügbare Kapazität wiedergespiegelt wird.

Network Utilization: Anteil des Durchsatzes zur Bandbreite. Hierbei lässt sich die Auslastung der Verbindung anhand foldender Formel darstellen:

$$\frac{Throughput}{Bandwidth * 100} \tag{2.2}$$

Latency: Die Zeit zwischen dem Senden und Empfangen eines Paketes an seinem Ziel.

Average Read Speed: Im Gegensatz zu dem Durchsatz bezieht sich die durchschnittliche Lesegeschwindigkeit in der Regel auf eine einzelne Festplatte.

Average Write Speed: Ebenso wie die Lesegeschwindigkeit bezieht sich die Schreibgeschwindigkeit auf eine Festplatte .

Random Input / Outputs per second (IOPS): Anzahl der möglichen zufalls Ein- / Ausgabeoperationen pro Sekunde für verschiedene Blockgrößen . Je höher der IOPS-Wert, desto schneller die Festplatte .

Sequential Input / Outputs per second (IOPS): Anzahl der möglich sequentiellen Ein- / Ausgabeoperationen pro Sekunde für verschiedene Blockgrößen .

Free Disk Space: Verfügbare freie Kapazität in % zur gesamten Kapazität.

Backup Interval: Das Zeitintervall, in dem ein Backup durchgeführt wird.

Time To Recovery: Die Angabe der minimalen und maximalen Zeit von dem Ausfall eines Speichers, bis zur erfolgreichen Wiederherstellung einer Sicherung.

CPU Utilization. Anteil der CPU Resourcen, welche in gebrauch sind zur gesammten Resourcenkapazität pro Zeiteinheit.

VM Memory: Menge und Art der bereitgestellten Speicher. Dies kann sich auf physischen Speicher oder virtuellen Speicher beziehen.

Memory Utilization: Anteil der Memory Resourcen, welche in gebrauch sind zur gesammten Resourcenkapazität pro Zeiteinheit.

Migration Time: Zeit, die benötigt wird, um eine VM zwischen vordefinierten Ressourcen zu bewegen.

Migration Interruption Time: Maximale Zeit, in der ein Kunde keinen Zugriff auf die Ressource hat.

Mean Time Between Failure: Bei MTBF beschreibt die mittlere Lebensdauer, d.h. es wird die erwartete Servicezeit vor dem Ausfall angegeben. Die Berechnung dafür basiert dabei auf historischen Überwachungsdaten mit folgender Formel:

$$MTBF = \frac{Systemanzahl * Laufzeit}{Anzahl\ aller\ Fehler} \tag{2.3}$$

Mean Time to Repair: Mean Time To Repair (kurz: MTTR) gibt die mittlere Reparaturzeit nach einem Ausfall an.
Formel:

$$MTTR = \frac{Ausfallzeit}{Anzahl\ der\ Ausflle} \tag{2.4}$$

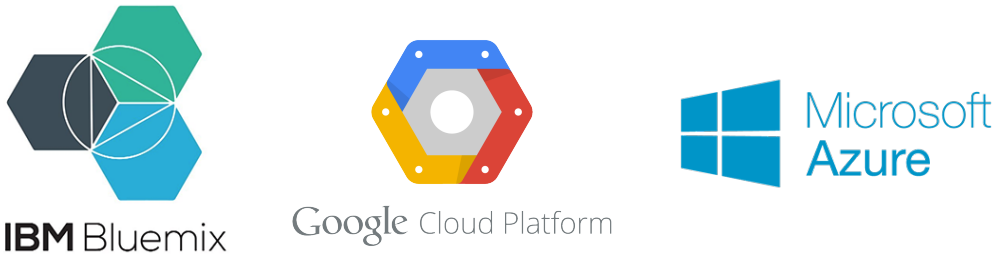
Deployment Time: Dauer, bis die entwickelte Applikation in die Plattform hochgeladen ist und zur Verfügung steht.

Zusammenfassung Die im vorherigen Kapitel aufgelisteten KPIs spiegeln bereits eine große Breite an Cloud Eigenschaften wieder. Da jedoch viele CSP nur sehr wenige Daten nach außen Preisgeben, folgt durch diese Intransparenz nur eine geringe Anwendbarkeit dieser literarischen KPIs in der Praxis. KPIs, wie zum Beispiel *MTBF*, übersteigen außerdem den zeitlichen Rahmen dieser Arbeit, da dafür ein CSP über längere Zeit beobachtet werden müsste, um aussagekräftige Resultate zu erzielen.

2.3 Cloud Service Provider (CSP) Übersicht

Dieser Abschnitt dient als Übersicht der in dieser Arbeit untersuchten CSP. Es wird ein grundlegender Überblick über die betrachteten CSP, das jeweils verwendete Service Modell (2.1), sowie die für diese Arbeit untersuchten Cloud Services gegeben. Außerdem folgt jeweils ein erster Einblick über das verwendete Preismodell der unterschiedlichen Cloud Service Provider.

Die Übersicht umfasst folgende Cloud Service Provider (CSP):



(a) IBM Bluemix[1] (b) Google Plattform[2] (c) Microsoft Azure[3]

Abbildung 2.1: Cloud Service Provider (CSP)

2.3.1 IBM Bluemix

Bluemix ist die von IBM entwickelte PaaS Cloud Lösung. Eine Cloud-basierte Plattform für die schnelle Erstellung, Verwaltung und Ausführung von Anwendungen. Bluemix ist eine Implementierung der IBM Open Cloud Architektur auf Basis von Cloud Foundry, einem Open-Source PaaS, was einem zugriff auf ein wachsendes, bereits bestehendes Ökosystem an runtime frameworks und Services bietet. Bluemix ist die jüngste in dieser Arbeit betrachtete Cloud Plattform. Die Erscheinung der ersten Beta von Bluemix fand im Jahr 2014 statt.

Services

In dem ständig erweiterten Bluemix Katalog[4] befinden sich bereits über 100 Services und Laufzeitumgebungen, welche für so ziemlich jeden Verwendungszweck einen Ansatz liefern. Da eine Auflistung aller dieser Möglichkeiten den Umfang dieser Arbeit sprengen würde, folgt hier eine Übersicht derer, welche in dieser Arbeit in Bezug auf das Internet der Dinge (IoT) betrachtet wurden:

SQL Database: Wie der Servicename bereits vermuten lässt, bietet dieser Sevice eine vollständig eingerichtete, relationale SQL-Datenbank.

DashDB: DashDB ist eine Data-Warehousing und Analyselösung. Es lassen sich ebenfalls relationale Daten speichern und mit den mitgelieferten Analysetools (prediktive Analyse, data mining, Analyse mit R und geospatiale Analyse) auswerten.

[3] Abbildung 2.1a Quelle: http://www.dbtekpro.com/wp-content/uploads/2015/07/bluemix-logo.png - abgerufen am 31.03.2016
[2] Abbildung 2.1b Quelle: http://hortonworks.com/wp-content/uploads/2015/01/Google-CloudPlatform_VerticalLockup.png - abgerufen am 31.03.2016
[3] Abbildung 2.1c Quelle: http://www.coinfox.info/images/mcrsft.png - abgerufen am 31.03.2016
[4] https://console.ng.bluemix.net/catalog/

IoT Platform: Die IoT Platform ist der zentrale Punkt sämmtlicher IoT Services in Bluemix. Hier werden Sensoren registriert und verwaltet, sodass weitere Anwendungen oder Services deren Daten konsumieren können. Dafür stehen Echtzeit- und REST-API's zur Verfügung. Die Datenübertragung zwischen den Sensoren und der IoT Platform ist jedoch auf das MQTT Protokoll beschränkt.

IoT Realtime Insights: Dieser Service ermöglicht es, die Sensordaten im Kontext zu verstehen, zu überwachen und Bedingungen der Geräte und Vorgänge zu erstellen. Mit nur wenigen Klicks kann man ein einfaches Dashboard zur Visualisierung der Sensordaten zusammenstellen oder aber Regeln für automatische Aktionen oder Benachrichtigungen erstellen.

flowthings.io: flowthings.io ist ein Drittanbieter service, welcher in Bluemix zur verfügung steht. Er dient ebenfalls zur Visualisierung von Daten, sowie der Ausführung von einfachen grundlegenden analytischen Operationen.

Auto Scaling: Mit dem Auto-Scaling Service lassen sich Richtlinien erstellen, nach denen die Anzahl der Anwendungsinstanzen dynamisch inkrementiert oder dekrementiert werden. Dies dient dazu, bei Lastspitzen oder nicht Auslastung der Instanz die entsprechenden Resourcen anzupassen.

Monitoring and Analytics: Dies ist ein Dienst zur Übersicht und Kontrolle einer Anwendung. Es lassen sich Reaktionszeiten der Anwendung, Durchsatz, CPU Auslastung und die Memory Nutzung der Anwendung anzeigen.

Preismodell

In Bluemix erfolgt die Abrechnung nach der Zeit, die eine Anwendung Läuft und der dabei benutzte Speicher, berechnet als GB-Stunden:

$$Anzahl\ der\ Instanzen * Speicher\ pro\ Instanz * Laufzeit\ der\ Applikation \quad (2.5)$$

Die in dieser Arbeit verwendete NodeJs Umgebung wird zum Beispiel aktuell [1] mit 0,0526 EUR / GB-Hour berechnet, wohingegen die ersten 375 GB-Stunden frei sind. Die zusätzlich verwendeten Service verfolgen jedoch ihre eigene Preisstruktur, wobei bei den meisten ein 'Free-Plan' verfügbar ist, unter dessen Konditionen keine Kosten erhoben werden. Der *SQL Databse* Service ist bis zu 100mb pro Instanz und Zehn konkurrierende Verbindungen kostenfrei, danach werden 301 EUR/Instanz abgerechnet.

2.3.2 Google Cloud Plattform

Google bietet mit der *Google Cloud Plattform* sowohl eine IaaS als auch eine PaaS Lösung an. Zu der schon seit 2008 Existierenden *Google App Engine*(PaaS) ist 2013 die IaaS Lösung *Google Compute Engine* dazugekommen, welche allesamt über die Google Cloud Plattform erreichbar sind. Mit dieser Plattform können Entwickler auf Googles hoch skalierbarer und zuverlässiger Infrastruktur arbeiten, welche Google intern ebenso für ihre Produkte wie 'Google Search' oder 'Youtube' verwendet.

Services

[1] 1.1.2016

Im Gegensatz zu Bluemix bietet die *Google Cloud Plattform* nur eine überschaubare Auswahl an vordefinierten Services an. Die Services sind dafür hingegen allgemeiner und modularer gehalten, um ein ebenso großes Spektrum an Anwendungsfällen abzudecken. Die folgende Auflistung beinhaltet wiederum nur die angebotenen Services, die in dieser Arbeit auf der Basis eines IoT-Workflows betrachtet werden.

Compute Engine: Wie bereits erwähnt, ist die Compute Engine Googles IaaS Lösung. Damit lassen sich anpassbare virtuelle Maschinen auf Googles Infrastruktur hosten.

Cloud SQL: Das speichern und managen von relationaler SQL-Datenbanken ist mit diesem Service möglich. Google verwaltet dabei sämtliches Management, um eine hohe Verfügbarkeit und Leistung sicherzustellen.

BigQuery: BigQuery ist ebenfalls ein Datenspeicherdienst, mit dem es möglich ist, sehr große Datensätze innerhalb von Sekunden mittels SQL-Anfragen zu durchsuchen. Skalierbar und einfach zu nutzen bietet dieser Dienst echtzeit Erkenntnisse auf großen Datensätzen.

Cloud Datalab: Cloud Datalab befindet sich aktuell[1] noch im beta Stadium, ist jedoch für unsere Anforderungen der vielversprechendste Service, den Google anbietet. Der Dienst bietet eine produktive, interaktive und integrierte Werkzeugsammlung zum Erforschen, Visualisieren, Analysieren und Transformieren gößerer Datenmengen.

Preismodell

Google Cloud Plattform bietet im gegensatz zu *IBM Bluemix* keine gebündelten Preise für bestimmte vorher definierte Konfigurationen an. Es wird im Gegensatz dazu jede einzelne Service Charakteristik abgerechnet. Dieser Ansatz wird für die ausgeführten VM Instanzen, sowie für sämtliche angebotenen Services verfolgt, aus deren Summe sich dann der monatliche Endpreis berechnet.(vgl. 6.8)

2.3.3 Microsoft Azure

Azure ist die von Microsoft entwickelte Cloud Computing Lösung. Es basiert auf dem Microsoft eigenen Windows Azure Betriebssystem und ist bereits seit Anfang 2010 offiziell verfügbar. Azure bietet ähnlich wie Bluemix eine Vielzahl verschiedener internetbasiertet Dienste in ihrem Sortiment an. Mit Azure stehen sowohl IaaS als auch PaaS Module und Dienste zur Verfügung. Azure gilt außerdem als eines der führenden Unternehmen im Bereich IaaS[5].

Services

Die in dieser Arbeit von Azure genutzten Services zur Implementierung der betrachteten Szenarien sind in folgender Auflistung zu sehen:

Azure Virtual Machines: Mit Azure Virtual Machines lassen sich sehr flexibel eine Vielzahl unterschiedlicher Computinglösungen bereitstellen. Es lässt sich mit nur wenigen Klicks ein virtueller Computer mit minutengenauer abrechnung bereitstellen

MySQL Datenbank: Dieser Service bietet eine vollständig eingerichtete MySQL Datenbanklösung.

[5] https://www.datadoghq.com/blog/how-to-monitor-microsoft-azure-vms/

IoT Hub: IoT Hub ist ein vollständig verwalteter Dienst, der eine zuverlässige und sichere, bidirektionale Kommunikation zwischen den registrierten IoT Geräten ermöglicht. Dieser Service dient quasi als Knotenpunkt zur Kommunikation von IoT Geräten/Sensoren.

Stream Analytics: Stream Analytics ist ein vollständig verwaltetes, Echtzeit-Ereignisverarbeitungsmodul, mit dem Datenanalysen durchgeführt werden können. Stream Analytics erleichtert die Einrichtung analytischer Echtzeitberechnungen von Datenströmen.

Power BI: Mit Power BI lassen sich mit den erzeugten Streaming Daten als auch mit statischen Datensätzen Dashboards zur Visualisierung bilden. Mit einer großen Auswahl an Visualisierungswerkzeugen werden damit Dashboards und deren Graphen und Diagramme erzeugt.

Preismodell

Azure verfolgt ein eher IaaS-typisches Preismodell. Es wird ähnlich wie bei der Google Cloud Plattform jede Servicecharakteristik einzeln abgerechnet. Dies erfolgt jedoch im Vergleich zu den anderen betrachteten CSP sehr feingranular, sodass zum Beispiel sämtliche eingehenden, ausgehenden, sowie internen Nachrichten mit jeweiligen Preissätzen abgerechnet werden.

CSP Diskussion

Die in dieser Arbeit betrachteten CSP unterscheiden sich alle bezüglich der angebotenen Dienste, sowie des Preismodells, wobei sich speziell bei den Cloud Diensten Paralellen erkennen lassen. So werden von allen Anbietern zum Beispiel meist die selben Datenbanklösungen angeboten oder ein Zentraler Knotenpunkt zum Verwalten der IoT Geräte. Die größten Unterschiede herrschen bei den jeweiligen Preismodellen. Es wird je nach CSP und jeweiligem Dienst zum Teil pro Megabyte, Nachricht, Monat, Gigabytestunden oder einer Kombination daraus abgerechnet. Diese unterschiedlichen Abrechnungsmetriken führen zur Verwirrung der Cloud Plattform Nutzer und erschweren die Vergleichbarkeit.

2.4 Zusammenfassung

In diesem Kapitel wurden die Grundlagen des Cloud Computing erläutert. Dafür wurden die Definitionen des *National Institute of Standards and Technology* ([NIS11]) betrachtet, sowie eine Übersicht über Cloud Computing Charakteristika, Cloud Deployment Modelle und Cloud Service Modellen gegeben. Desweiteren wurde eine Zusammenstellung und Erläuterung aller in der Literatur gefundenen KPIs erstellt und einen Einblick in die für diese Arbeit relevanten CSP gegeben. Dabei wurden speziell die verwendeten Cloud Dienste und das jeweilige Preismodell betrachtet.

3. Sachverwandte Arbeiten

Dieser Abschnit beschreibt sachverwandte, und damit in Beziehung zu dieser Arbeit stehende, Arbeiten. Es gibt bereits Frameworks, welche sich auf Benchmarking und Evaluation von Cloud-Plattformen spezialisiert haben. Die für diese Arbeit herangezogenen relevanten Arbeiten, welche den aktuellen Stand der Forschung präsentieren, sind CloudCMP (vgl.[LYKZ10]),C-Meter (vgl.[YIEO09]) YCSB (vgl.[CST+10]) und Cloudstone (vgl. [SSS+05]). Einen gröberen Überblick über den aktuellen Stand der Forschung bietet das Paper [BLC+11], welches einen Zusammenfassung über sämtliche repräsentative Ansätze zur Simulation, Evaluation und Virtualisierung von Cloud Systemen bietet und ebenfalls für die folgende Übersicht der sachverwandten Arbeiten als Referenz dient.

3.1 CloudCMP

CloudCMP ist eine 2010 erstellte Sammlung von Benchmarks, welche die gängigen Services von Cloud providern vergleicht, um damit Performance und Kosten einer benutzerdefinierten Anwendung vorraussagen zu können. Implementiert wurden dafür ausgewählte Benchmarks der SPECjvm2008 Benchmark-Suite, welche für die Cloud Beschränkungen leicht modifiziert wurden (vgl [LYKZ10]). Unterstützte Provider sind dabei Microsoft Azure, Amazon AWS, Google App engine und Rackspace. Seit der Veröffentlichung 2010 sind jedoch eine Vielzahl neuer Cloud Anbieter auf dem Markt erschienen. Darunter selbst namhafte hersteller, wie die in dieser Arbeit untersuchten IBM Bluemix Cloud und Google Cloud Plattform, welche von diesem Framework noch nicht unterstützt werden. Die in der JavaVM ausgeführten Benchmarks fokussieren sich jedoch lediglich auf die Ausführungszeit und die jeweils generierten Kosten. Dies sind wichtige Indikatoren einer Cloud, die für diese Arbeit ebenfalls herangezogen werden. Da jedoch hinter aktuellen Cloud-Plattform Providern eine komplexere Preisstruktur steckt, wie es z.B. bei Bluemix oder Microsoft Azure der Fall ist, können die verursachten Kosten nicht mehr nur von der Ausführungszeit abhängig gemacht werden, da unter anderem der benutzte RAM/h sowie die CPU Auslastung eine wichtige Rolle spielen.

3.2 C-Meter

C-Meter ist ein erweiterbares, portierbares, auf Python basierendes 'easy-to-use' Framework. Entwickelt zum Erstellen von virtuellen Maschinen in unterschiedlichen Cloud Plattformen und der Ausführung von generierten Workloads auf den unterschiedlichen Virtuellen Maschinen. Für das Resultat werden dann verschiedene Parameter, basierend auf den

unterschiedlichen Konfigurationen der virtuellen Instanzen verglichen. Dieser Ansatz zur unterschiedlichen Konfiguration der VM´s bekommt in dieser Arbeit ebenfalls ein Augenmerk im erstellten Decision Support System, soll jedoch nicht direkt fokusiert werden.

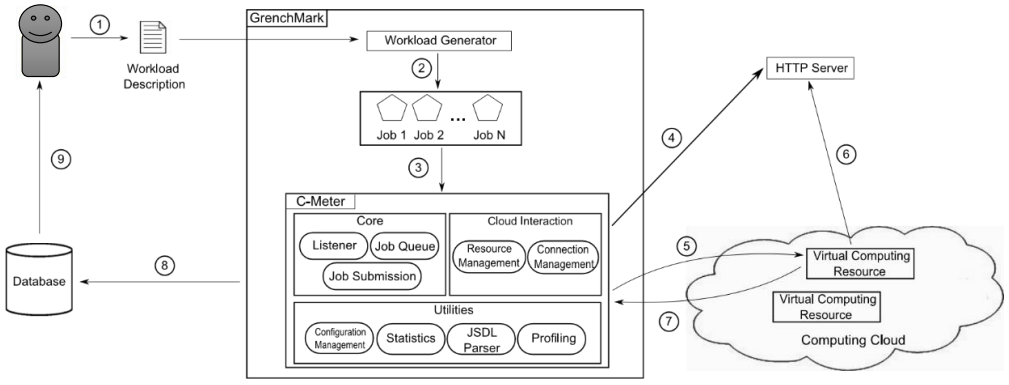


Abbildung 3.1: C-Meter Architektur

3.3 YCSB

Das Ziel des *'Yahoo Cloud Serving Benchmark'* Frameworks (YCSB) ist es, Performance-Vergleiche von Datenspeicher-Systemen zu erleichtern. Dafür stellt das open-source tool einen Kernsatz von Benchmarks für vier verbreitete Datenspeicher-Systeme zur Verfügung. Cassandra, HBase, Yahoo!'s PNUTS und eine einfache MySQL Implementierung. Untersucht werden dabei Performance, Skalierbarkeit, Replikation und Verfügbarkeit, wobei hingegen nur Perfomance und Skalierbarkeit als open-source Modul zur Verfügung stehen. Dabei wird die Latenzzeit bei zunehmendem Durchsatz gemessen und für die Skalierung werden die Performance-Werte bei zunehmender Anzahl an Maschinen untersucht(vgl [CST+10]). Gegenstand dieser Arbeit ist ebenfalls die Untersuchung der Latenz von Datentransferoperationen, wobei für das zugrunde liegende Internet of Things Szenario der Durchsatz der Sensordaten und die Anzahl der sendenden Sensoren als skalierbarer Parameter implementiert werden. Diese Arbeit ist nicht nur auf Latenz von Datenspeicheroperationen fokusiert, sondern wie in Kapitel 2.2 zu sehen, werden weitere relevante Key Performance Indikatoren betrachtet.

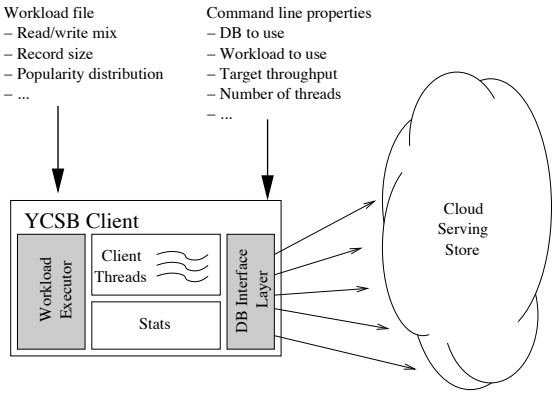


Abbildung 3.2: YCSB Client - Architektur

3.4 Cloudstone

Cloudstone besteht aus einer Open-Source-Web 2.0 Social-Anwendung (Olio) und einer Reihe von Automatisierungstools zur Erzeugung von Last und zum Messen der Performance. Außerdem wird eine Metrik eingeführt, *US-Dollar pro Benutzer pro Monat*, auf die sämtliche Resultate abgebildet werden. Dieser Ansatz zur Abbildung auf eine kostenbasierte Metrik wird in dieser Arbeit übernommen, da damit ein anschaulicher und einfach vergleichbarer Wert in dem erstellten Decision Support System (Kapitel 4.3.4) als Resultat dargestellt wird. Cloudstone setzt sich als Ziel, typische Web 2.0 Funktionalitäten unter Last zu beobachten und diesbezüglich eine Evaluation der darunterliegenden Cloud Systeme durchzuführen. An diesem Punkt differenziert sich diese Arbeit von dem in Cloudstone gelieferten Ansatz, indem wir als Basis für den in dieser Arbeit betrachteten Anwendungskontext IoT-Szenarien zugrunde legen.

3.5 Zusammenfassung

Wie in diesem Kapitel zu sehen ist, gibt es bereits einige Ansätze, welche sich mit der Evaluation von Cloud Anbietern auf verschiedene Art und Weise beschäftigen. Da in den letzten Jahren viele namhafte Hersteller wie IBM oder Google ihre eigene Lösung zum Cloud Computing auf den Markt gebracht haben, und die hier betrachteten sachverwandten Arbeiten bereits alle älter als 5 Jahre sind, finden diese dabei noch keine Beachtung. Außerdem wird für das in dieser Arbeit zugrunde liegende IoT-Szenario ein Framework erstellt, welches für eine skalierbare Anzahl an Sensoren einen variablen Durchsatz von bis zu 100 Einheiten pro Sekunde generiert. Dafür werden in dieser Arbeit mehr relevante Key Performance Indikatoren untersucht, statt wie bisher nur bestimmte Ausführungszeiten und generierte Kosten. Der Ansatz, sämtliche Evaluationsergebnisse auf eine Metrik abzubilden, wie bereits bei *Cloudstone* zu sehen, wird für das hier entstandene Decision Support System übernommen. Der Hauptaspekt an dem sich diese Arbeit zu den in diesem Kapitel untersuchten Arbeiten unterscheidet, ist jedoch die eigentliche Nutzungsart der Cloud. Wohingegen sämtliche bisher bekannten Ansätze die Cloud Plattformen nur als hosting Dienst mit skalierbarer Infrastruktur benutzen, wird das Szenario in dieser Arbeit mittels Diensten und Modulen, welche von den unterschiedlichen Plattformen angeboten werden, implementiert und in die Evaluation miteinbezogen.

4. Entwurf

4.1 Architektur

Im folgendem Abschnitt wird der Aufbau der Lambda Architektur genauer beschrieben. Es werden die einzelnen Schichten mittels Schaubild und textuell genauer betrachtet und der Bezug der Architektur zu dieser Arbeit hergestellt.

Lambda Architektur

Die *Lambda Architektur* ist eine Architektur zur Verarbeitung großer Datenmengen, entworfen von Nathan Marz ([TR14]), welche die Vorteile von Batch- und Streamprocessing nutzt. Die Schlüsselanforderungen an diese Architektur sind nach Nathan Marz:

- Fehlertoleranz gegen Hardwareausfälle und menschliche Fehler
- Unterstützung für eine Vielzahl von Anwendungsfällen, die Abfragen mit niedriger Latenz sowie Updates enthalten
- Lineare Scale-Out-Fähigkeiten, was bedeutet, je mehr Maschinen das Problem bearbeiten, desto schneller sollte die Verarbeitung einer bestimmten Datenmenge erledigt sein.
- Erweiterbarkeit, so dass das System überschaubar ist und leicht neue Funktionen aufnehmen kann.

Diese Architektur vereint Technologien zur Verarbeitung von Big Data für die Echtzeitverarbeitung von Datenströmen sowie die Verarbeitung großer Datensätze. Ziel davon ist es, historische Daten ebenso wie aktuelle Informationen aus kontinuirlichen Quellen in die Ergebnisse beliebiger Anfragen einfließen zu lassen. Wie in Abbildung 4.1 dargestellt ist die Architektur in drei Schichten unterteilt: Batch-, Serving- und Speed Layer. Die Batch- und Speed Layer sind dafür verantwortlich, die eintreffenden Daten zu verarbeiten, beim Batch Layer auf Basis des gesamten Datensatzes aller Daten und beim Speed Layer mittels des aktuellen Datenstroms. Der Serving Layer dient dazu, die durch Speed und Batch Layer vorberechneten Ansichten zur Verfügung zu stellen, sowie auf Anfragen zu reagieren.

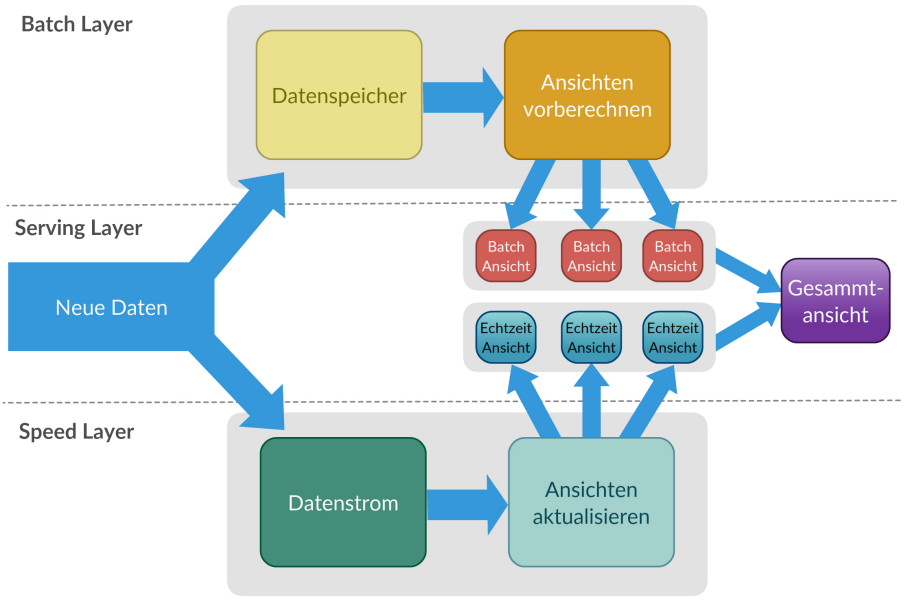


Abbildung 4.1: Lambda Architektur

Batch Layer

Der Batch Layer ist für die Datenhaltung zuständig. Die Daten werden als Rohdaten in einem unveränderlichen, '*append-only*' Dateisystem abgelegt und permanent gespeichert. Es werden zunächst mit einem verteilten Verarbeitungssystem, welches für große Datenmengen ausgelegt ist, Ergebnisse vorberechnet. Die Vorberechnung der Resultate zielt dabei auf eine perfekte Genauigkeit durch die Nutzung des kompletten historischen Datensatzes. Dadurch können ebenfalls Fehler oder Abweichungen der Berechnungen leicht behoben werden, da ständig der gesamte Datensatz zur Berechnung herangezogen wird. Mit den Resultaten werden, wie in Abbildung 4.1 zu sehen ist, sogenannte Batch Ansichten erstellt, welche dann dem Serving Layer zur weiteren Verarbeitung zur Verfügung gestellt werden. Diese Resultate werden dabei typischerweise als '*read only*' Datenbanken realisiert und bei Updates komplett erneuert.

Speed Layer

Der Speed Layer verarbeitet Datenströme in Echtzeit und ohne die Anforderungen der hohen Genauigkeit oder Vollständigkeit. Bei dieser Schicht spielt der Durchsatz eine vernachlässigbare Rolle, da darauf abgezielt wird, die Latenz der Echtzeit-Ansichten über die jüngsten Daten zu minimieren. Im Wesentlichen ist der Speed Layer dafür verantwortlich, die Lücke des Batch Layers, verursacht durch die Berechnungsdauer des großen Datensatzes, mit den Echtzeitansichten der jüngsten Daten zu füllen. Diese Ansichten können dabei von der Genauigkeit und Vollständigkeit gegenüber den Batch-Ansichten stark abweichen, jedoch sind sie dafür sehr schnell verfügbar und abrufbar, nachdem neue Daten empfangen wurden. Diese Ansichten werden typischerweise in schnellen NoSQL Datenbanken abgelegt.

Serving Layer

Die Serving Schicht ist dafür verantwortlich, die vorberechneten Ansichten zu verwalten und Anfragen an das System zu beantworten. Die Resultate des Speed und Batch layer werden in diesem Layer gespeichert, sodass auf ad-hoc Anfragen mit den vorberechneten

Ansichten oder eigens gebildeten Ansichten der verarbeiteten Daten geantwortet werden kann. Diese Architektur zur Anfragenbeantwortung ist dabei besonders robust gegenüber Berechnungsfehlern oder fehlerhaften Daten. Es werden nur Informationen aus der eigentlichen Quelle permanent gespeichert, sodass die Ergebnisse der jeweiligen Berechnungen bei jeder Iteration neu erstellt werden. Wurde bei einer Iteration also ein fehlerhaftes Ergebnis produziert, hat dieses keine weiteren Auswirkungen auf Ergebnisse weiterer Iterationen.

4.2 Anwendungskontext

Als Grundbaustein für die hier definierten Szenarien dient eine Industrieanlage, welche mit einer skalierbaren Anzahl an Sensoren Messdaten generiert und mittels Cloud Computing Daten visualisiert und analysieret. Hierfür werden in diesem Kapitel zwei Szenarien definiert, um die Aufgaben der in Kapitel 4.1 eingeführten Batch und Stream Layer der Lambda Architektur abzudecken. Diese werden mittels Cloud Services der verschiedenen CSP implementiert.

Um eine plattformübergreifende, einheitliche Evaluation der verschiedenen Plattformen erstellen zu können, werden Testdaten einer exemplarischen Flaschenabfüllanlage, wie in Abbildung 4.2 dargestellt, generiert. Dabei durchläuft eine Flasche mehrere Stationen und generiert dabei Sensordaten. Für diese Arbeit werden dabei randomisierte oder nach bestimmten mathematischen Vorschriften, wie Sinus oder Cosinus, Testdatensätze generiert, welche es dann von den verschiedenen CSP zu verarbeiten gilt.

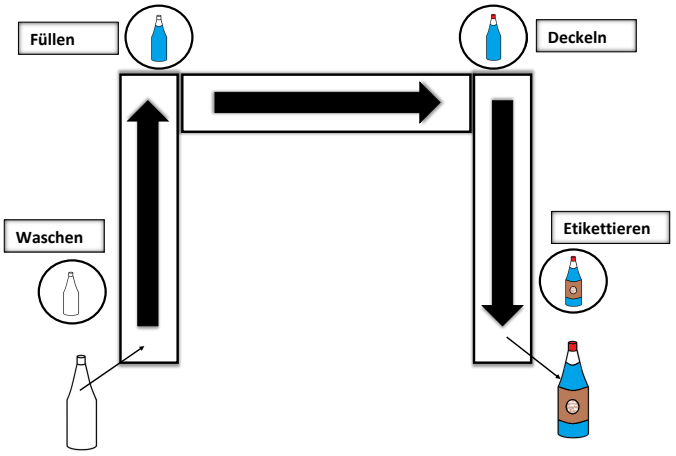


Abbildung 4.2: Exemplarische Industrieanlage

Ein Datum besteht dabei wie in Tabelle 4.1 durch ein Beispiel Datum dargestellt aus den Parametern ID, Produktname, Füllstand, Temperatur, Timestamp, Produktionsnummer und einer variabel großen Anzahl an Nutzdaten (Payload).

Tabelle 4.1: Beispiel Daten

ID	Produktname	Füllstand	Temperatur	Timestamp	Produktionsnummer	Payload
01	Bier	41,5	33,1	2016-04-23T09:32:47	0002	String
02	Bier	53.1	29.7	2016-04-23T09:32:48	0002	String
03	Bier	59.9	18.3	2016-04-23T09:32:49	0002	String

Szenario 1 (Speed Layer):
In diesem Szenario werden mit einer skalierbaren Anzahl an Sensoren Testdaten im Bereich zwischen 1 Datum/Sekunde (U/s) und 100 U/s generiert. Die jüngsten Daten sollen dabei mit den zur Verfügung stehenden Cloud Services visualisiert werden. Dabei soll nur ein definiertes Fenster jüngster Daten angezeigt werden und Daten, die aus diesem Fenster fallen in einer Datenbank für das zweite Szenario hinterlegt werden.

Szenario 2 (Batch Layer):
Im zweiten Szenario soll mittels eines bereits gespeicherten Datensatzes analytische, sowie visuelle Operationen ausgeführt werden. Die Daten stehen dafür in einer Datenbank zur Verfügung und sollen mittels bestehender Cloud Services verknüpft und analysiert werden.

4.3 Softwareentwurf

Für die Implementierung der im vorherigen Kapitel (4.2) definierten Szenarien nach der Lambda Architektur (4.1) sind mehrere Software-Komponenten notwendig. Es wird zuerst ein *Evaluations Framework* erstellt, welches zur Konfiguration der Datenparameter ein Webinterface zur Verfügung stellt, sowie zur Veranschaulichung von Messdaten. Außerdem ist dieses *Evaluations Framework* dafür zuständig, Testdaten zu generieren und diese an die verschiedenen CSP zu senden. Als Gegenstück dafür ist eine Server Application notwendig, welche auf den jeweiligen CSP läuft, die Anfragen des Evaluationsframeworks entgegennimmt und bearbeitet sowie die einzelnen Cloud Service anbindet. In Abbildung 4.5 ist ein Komponentendiagramm zu sehen, dass die Beziehung der zwei Softwarekomponenten und deren benutzten und bereitgestellten Schnittstellen zeigt.

4.3.1 Lambda Architektur im Kontext

In Bezug auf das *Internet der Dinge* wird durch die ständig wachsende Anzahl an ubiquitären Systemen eine große Masse an Daten generiert. Diese Arbeit untersucht *IoT*-Szenarien, welche mittels Cloud Computing Berechnungen auf massiven Datensätzen und Datenströmen durchführen. Hierfür wurde die Lambda Architektur gewählt, welche für ein generalisiertes IoT Szenario die zwei grundlegenden Datenverarbeitungszweige abstrahiert. Mit der Visualisierung von Live-Daten (Speed Layer) und der Berechnung und Analyse von Datensätzen (Batch Layer) wird eine Vielzahl von Anwendungsmöglichkeiten abgedeckt. Die Lambda Architektur soll in dieser Arbeit mittels vorgefertigten Services der verschiedenen CSP so gut wie möglich komplettiert werden. Dafür werden die Services der verschiedenen CSP analysiert und verknüpft, um mittels generierter Testdaten Ergebnisse im Speed und Batch Layer vorzuweisen. In Abbildung 4.3 ist die konkrete Darstellung der Architektur mit den Services der verschiedenen CSP zu sehen.

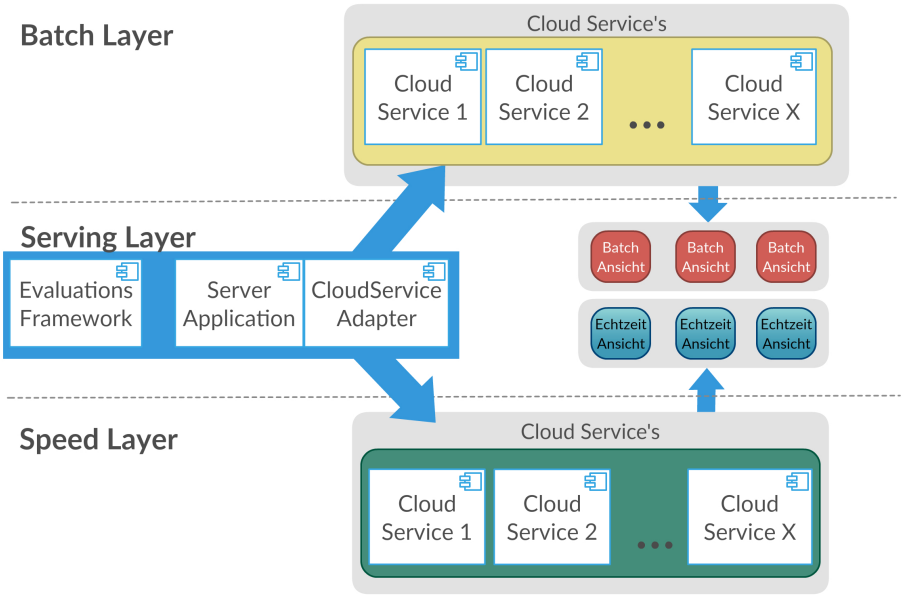


Abbildung 4.3: Lambda Architektur Komponenten

In dem Deployment Diagramm in Abbildung 4.4 ist die Verteilung der einzelnen Softwarekomponenten auf den verschiedenen Rechenknoten zu sehen. So läuft die Server Applikation auf dem jeweiligen Cloud System und das Evaluationsframework wird auf einer lokalen Maschine ausgeführt.

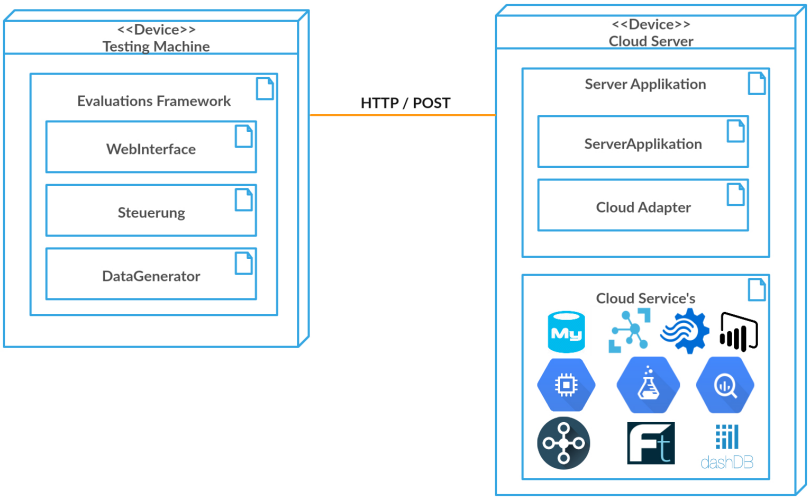


Abbildung 4.4: Software Deployment Diagramm

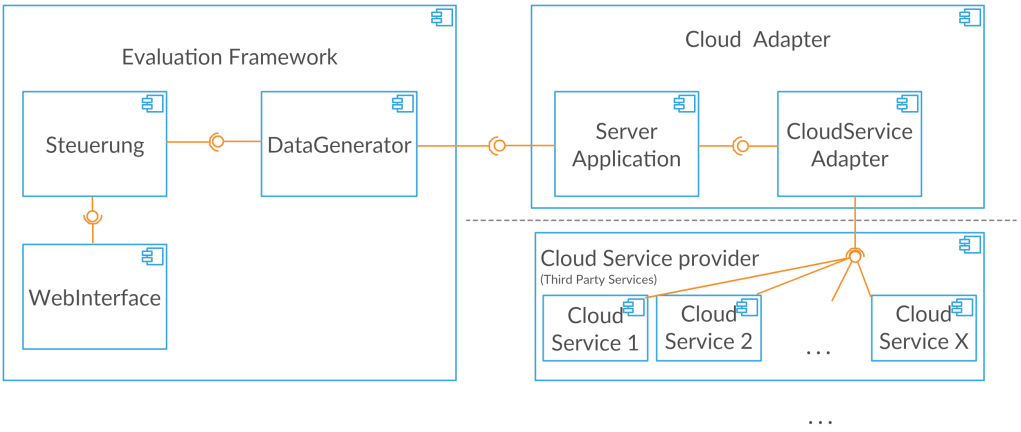


Abbildung 4.5: Software Komponenten Diagramm

4.3.2 Evaluations Framework

Das *Evaluations Framework* besteht wie in Abb. 4.5 ersichtlich aus zwei wesentlichen Komponenten. Die *dataGenerator* Komponente generiert Testdaten und sendet diese an die jeweilige Server Applikation in der Cloud. Es wird außerdem zur Konfiguration der Datenparameter wie Einheit pro Sekunde (U/s) und dem Payload ein Webinterface erstellt. Über das Webinterface lassen sich außerdem während des Testlaufs Fortschritt und live Resultate der Messungen anzeigen. In Abbildung 4.6 ist ein Mockup des Webinterface zu sehen. In verschiedene Tabs unterteilt sind dabei die unterschiedlichen Sektionen in die das Evaluations Framework gegliedert ist. In der Abbildung zu sehen ist dabei der erste Tab *Data Generation*, in dem die verschiedenen Konfigurationen der Testdatengenerierung vorgenommen werden..

In Abbildung 4.7 ist ein Sequenzdiagramm zu sehen, welches die Interaktion eines Nutzers

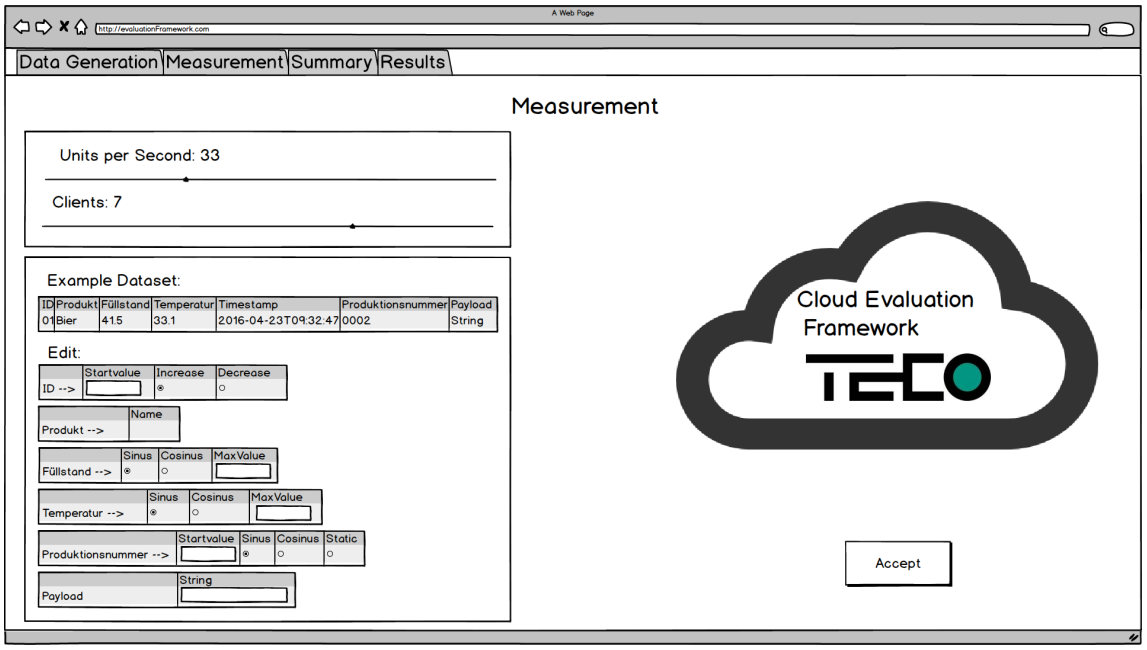


Abbildung 4.6: Evaluation Framework Mockup

mit dem Webinterface zur Ausführung eines Testlaufs zeigt. Der erste *post Request* Aufruf erfolgt durch das Definieren/Ändern der Datenparameter. Als optional gekennzeichnet ist der zweite Aufruf. Durch die nicht zwingende Automatisierung mehrerer sequentiell ablaufender Testläufe erfolgt dieser Aufruf nur wahlweise. Der eigentliche Kern der Anwendung ist die Ausführung des Testlaufes. Dabei wird zuerst unterschieden, ob eine Automatisierung mehrerer Testläufe definiert wurde. Beim eigentlichen Testlauf werden durch die Funktion *requestGenerator* Anfragen generiert. Die *genNextDate* Funktion generiert nach dem vorgegebenen Datenparametern ein neues Datum pro Anfrage. Die *statusCallback2socket* Funktion hat dabei die Aufgabe, Parameter des Testlaufes aktuell zu halten, welche dann über eine Socket Schnittstelle auf dem Webinterface während des Testlaufes angezeigt werden. Der letzte Aufruf, welcher im Sequenzdiagramm zu sehen ist, dient dem Zurücksetzten des Frameworks auf seine Ausgangstellung und der Rekonfiguration auf die Standardparametern.

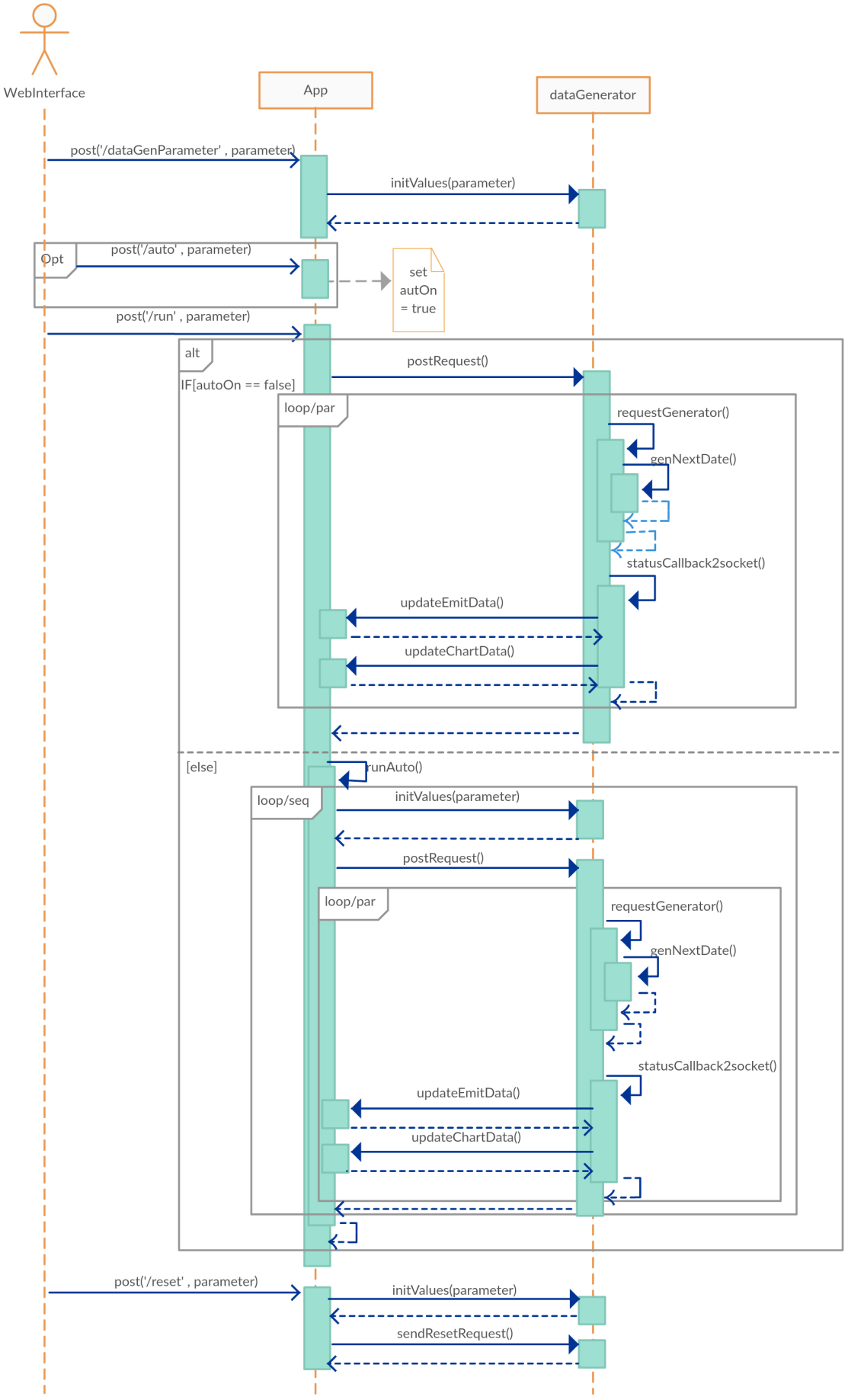


Abbildung 4.7: Evaluation Framework Sequenz Diagramm

4.3.3 Cloud Adapter Applikation

Die Cloud Adapter Applikation ist das Gegenstück zum Evaluationsframework. Diese Anwendung soll später auf den verschiedenen CSP laufen und die generierten Testdaten des Evaluations Frameworks entgegen nehmen und verarbeiten. Außerdem soll es die Schnittstelle zu den unterschiedlichen Cloud Services darstellen und die empfangenen Testdaten des Evaluations Frameworks genau diesen bereitstellen. Die Schnittstelle zu den verschiedenen Cloud Serives soll dabei modular Implementiert werden, um das Verhalten der Applikation leicht auf den jeweiligen CSP anpassen zu können. In Abbildung 4.8 wird das Verhalten dieser Applikation dargestellt. Im Body der Eingehenden "Post Requests" sind die von dem Evaluations Framework generierten Testdaten enthalten. Diese werden über einen asynchronen Aufruf der Funktion *insertDate* an die jeweiligen *CloudAdapter* Instanz weitergereicht. Hier werden die Testdaten dann den jeweiligen Cloud Services des CSP zur Verfügung gestellt. Mit dem *reset* request werden sämtliche Konfigurationen der Applikation wieder auf Standardwerte gesetzt. Außerdem werden Cloud Services wie der Datenspeicher zurückgesetzt, um gleiche Testbedingungen für folgende Testläufe zu gewährleisten. In der *ServerApplikation* läuft außerdem noch eine Endlosschleife, die einmal pro Sekunde Telemetrie Daten über das System und das Betriebssystem sammelt und diese ebenfalls über einen "Post Request" an das Evaluations Framework sendet.

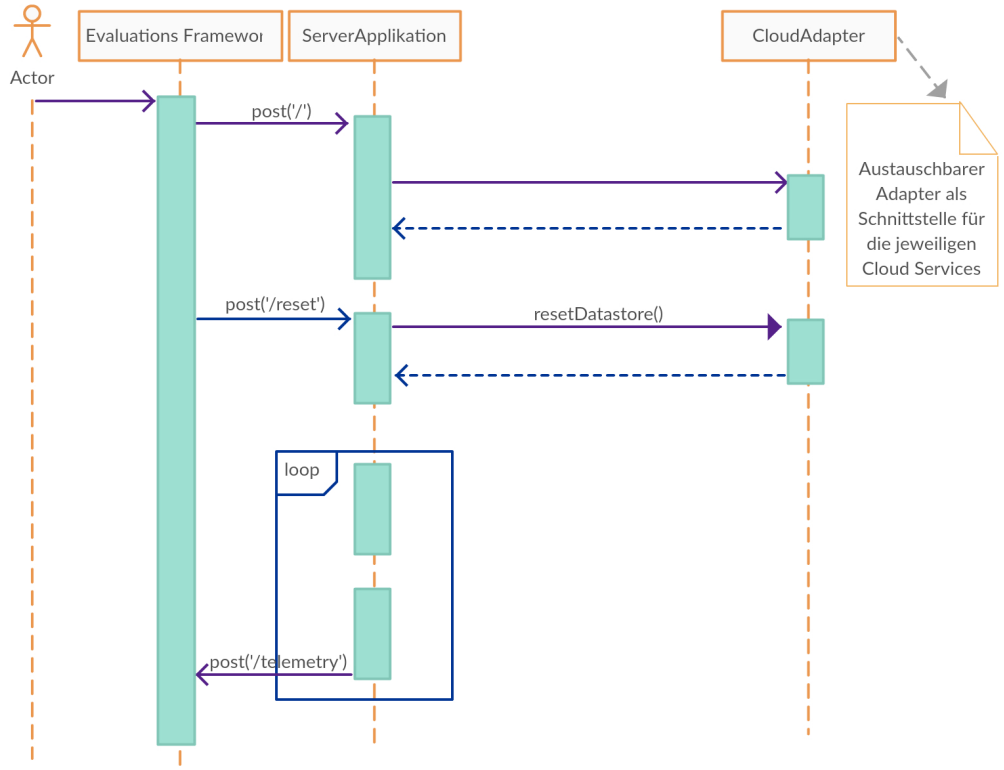


Abbildung 4.8: Cloud Adapter Sequenz Diagramm

4.3.4 Decision Support System

Das Decision Support System soll als Entscheidungsstütze zum *Cloud Provider Selection Problem* dienen. Mit den aus der Evaluation gewonnenen Erkenntnissen wird ein auf Javascript basierte interaktive Website erstellt, welche auf einem mit Evaluationsergebnissen gefühlten Modell basiert. Es werden Graphen zu CPU-Usage, Memmory-Usage Latenzzeiten und Preisentwicklung dargestellt. Durch Anpassen bestimmter Attribute wird das dazu passende Verhalten der einzelnen CSP in den Diagrammen angepasst. Außerdem wird eine mögliche Aktualisierung der Preisinformationen oder eine Modellanpassung als Erweiterbarkeit dieser Softwarekomponente angedacht, weshalb bereits ein Platzhalter dafür entworfen wird. In Abbildung 4.9 ist ein Mockup des Decision Support Systems zu sehen. Dabei dient der Konfigurations Tab der bereits erwähnten Erweiterbarkeit der Software. Im zu sehenden Ausschnitt ist der Aufbau des Visualisierungsparts zu sehen. Dabei sind im unteren Teil der Seite die Graphen angebracht und der obere Teil dient der Konfiguration.

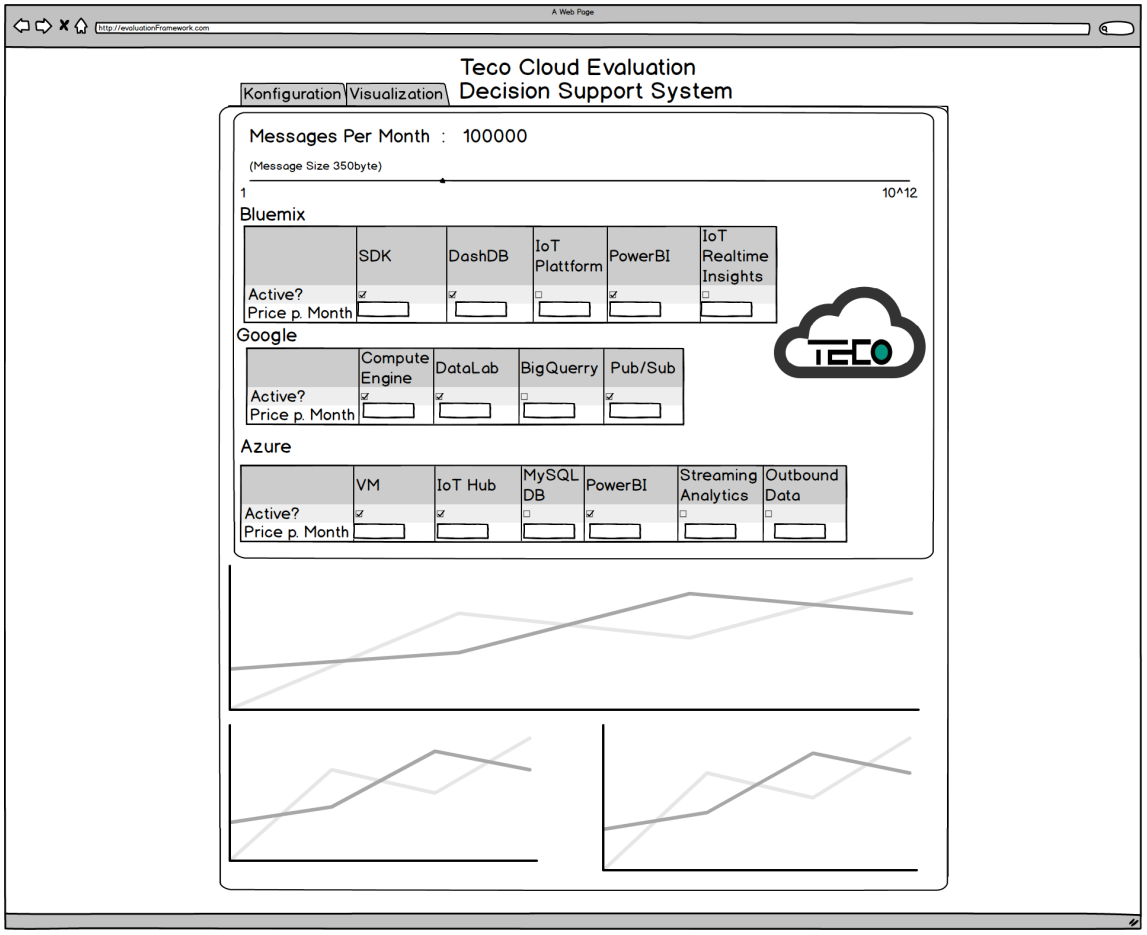


Abbildung 4.9: Decision Support System Mockup

5. Implementierung

In diesem Kapitel wird die Implementierung der Client- Server Architektur des Evaluations Frameworks und der Cloud Adapter Applikation beschrieben. Es werden die benutzten Bibliotheken sowie die Codestruktur beschrieben. Der komplette Programmcode ist außerdem auf der dieser Arbeit beigelegten DVD zu finden.

Mit Ausnahme des HTML Gerüsts im WebInterface wurden sämtliche Softwarekomponenten dieser Arbeit in der open-source Programmiersprache *NodeJs* geschrieben. *NodeJs* basiert auf der Javascript-Laufzeitumgebung V8 und bietet daher eine ressourcensparende Architektur, welche eine besonders große Anzahl simultaner Netzwerkverbindungen ermöglicht. Es wird dabei ein eventgetriebenes, nicht-blockierendes I/O - Modell verwendet, welches den Durchsatz und die Skalierbarbeit in Web-Applikationen mit vielen Ein- und Ausgabeoperationen optimiert. *NodeJs* Applikationen operieren in einem einzigen Thread mittels nicht-blockierenden I/O-Aufrufen, was bei vielen gleichzeitigen Verbindungen die teuren Kosten des Kontextwechsels erspart.

Als Entwicklungsumgebung wurde dabei Microsoft's Visual Studio[1] sowie Notepad++[2] verwendet. Verwendete Browser, in denen das WebInterface untersucht wurde, waren die aktuellen Versionen von Google Chrome(V 49.0.2623), Firefox (V 45.0) sowie Safari (V 9.0.3).

5.1 Evaluations Framework

Das Evaluations Framework soll ein WebInterface bereitstellen, mit dem sich Parameter der Messdaten bearbeiten, sowie Testläufe ausführen und Messungen dazu anzeigen lassen. Außerdem sollen Testdaten mit einer skalierbaren Frequenz und dynamisch anpassbarer Anzahl an Clients erstellt und an die Cloud Adapter Applikation gesendet werden.

WebInterface

Das WebInterface wurde als Single-page-Webanwendung mit dem Standard NodeJs server Framework *Express.js* erstellt. Der damit erstellte Server stellt eine einfache HTML Seite zur Verfügung, welche mit Javascript und CSS die Interaktion mit dem Nutzer durchführt. Unterteilt wird diese Seite, wie in Abbildung 5.1 zu sehen, mit der vom *Bootstrap Framework*[3] angebotenen Tabs-Vorlage, womit der Seiteninhalt in verschiedene Tabs unterteilt

[1] https://www.visualstudio.com/
[2] https://notepad-plus-plus.org/
[3] http://getbootstrap.com/

und nur benötigte Inhalte angezeigt werden. Die Kommunikation des WebInterfaces mit der Cloud Adapter Applikation erfolgt bidirektional über die *socket.io*[4] Bibliothek. Wie der Name schon vermuten lässt, wird dabei das WebSocket Protokoll verwendet. Dabei werden zum Beispiel die Live Telemetrie Daten der Cloud Adapter Applikation sowie die Zustände aktueller Messungen erneuert oder Daten für die Live Grafiken bereitgestellt. Zu sehen ist dies zum Beispiel in dem *Data*-Panel in Abbildung 5.2 sowie dem *Chart*-Panel. Die Live Grafiken bedienen sich der *chartist-js* Bibliothek, welche eine einfache und leicht zu bedienende Bibliothek zum Erstellen von Diagrammen in HTML und CSS ist.

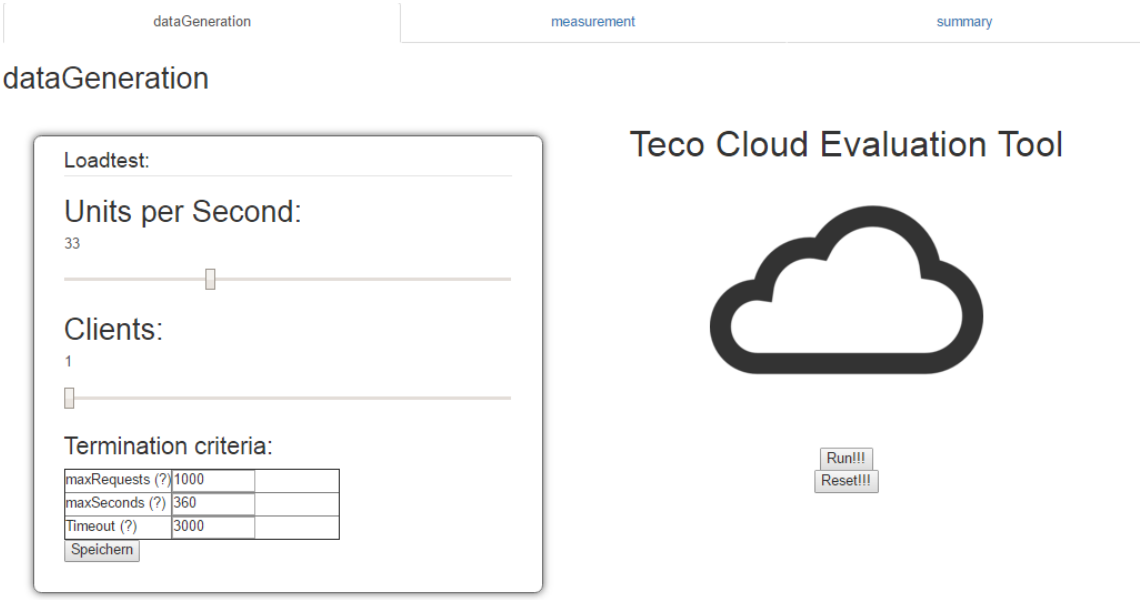


Abbildung 5.1: WebInterface

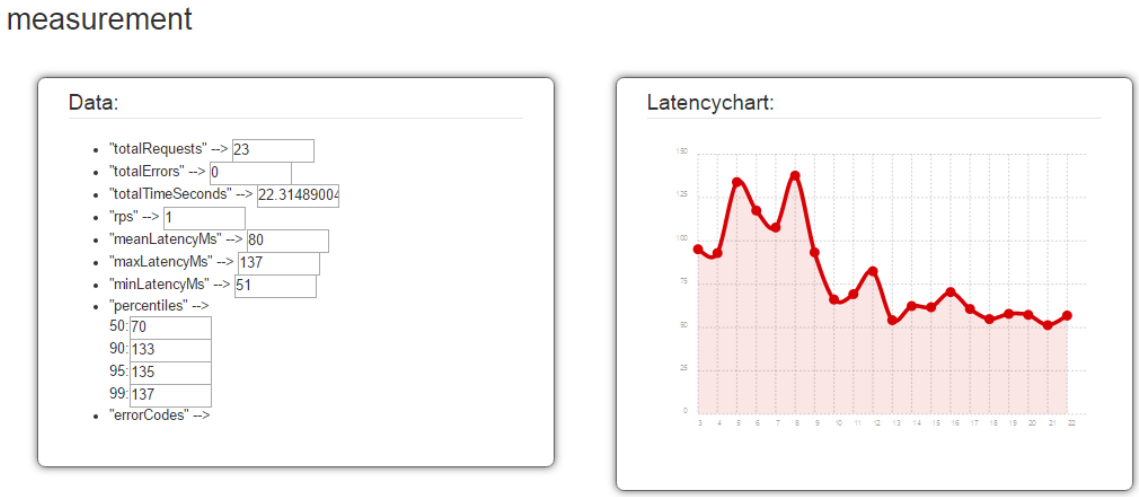


Abbildung 5.2: Measurement Tab Ansicht

[4] http://socket.io/

DataGenerator

Die *DataGenerator*-Komponenten dienen hauptsächlich dazu, für die Testläufe Datenpackete nach den verschiedenen Konfigurationen zu generieren und diese über das HTTP-Protokoll via *POST-Request's* an die Server Applikation zu senden.

Ein Datum enthält dabei die in Tabelle 4.1 zu sehenden Parameter. In der *DataGenerator*-Klasse wird dafür erst ein Standard JSON-Object (Abbildung 5.3) erstellt, was später mit den entsprechenden Daten gefüllt und gesendet wird.

```
var requestData = {
    "idflaschen": 01,
    "Produktname": "Bier",
    "Fuellstand" : 41.5,
    "Temperatur" : 33.1,
    "Timestamp" : new Date().toJSON(),
    "Produktionsnummer" : 0002
};
```

Abbildung 5.3: JSON Dataset Object

Dieses Objekt wird später im Body eines *POST-Requests* an die entsprechende Cloud Adapter Applikation gesendet. Dafür wird die *loadtest*[5] Bibliothek zum Senden der Anfragen an die Serverapplikation verwendet.

Die *loadtest*[5] Bibliothek bietet eine API für automatisierte Lasttests auf HTTP oder WebSocket URLs. Das Verhalten des Lasttests lässt sich dabei über einige Parameter in einem Options JSON Object definieren (Abbildung 5.4). Speziell die Möglichkeiten, die U/s und Anzahl der Clients definieren zu können, sind für die in dieser Arbeit betrachteten Szenarien nützlich. Außerdem wird mit der *statusCallback* Funktion eine Methode bereitgestellt, um aktuelle Messwerte während der Ausführung des Tests zu bekommen, welche für die Anzeige auf dem WebInterface genutzt wird. Das Ergebnis des Tests sowie sämtliche Zwischenergebnisse im *statusCallback* werden als JSON Objekt mit den Parametern totalRequests, percentiles, rps, totalTimeSeconds, meanLatencyMs, maxLatencyMs, totalErrors und errors bereitgestellt.

```
var options = {
        url: serverUrl,
        method: "POST",
        maxRequests: maxReq,
        maxSeconds: maxSec,
        timeout: +tout,
        concurrency: clients,
        requestsPerSecond: ups,
        statusCallback: statusCallback2socket
};
```

Abbildung 5.4: loadtest options

[5]https://www.npmjs.com/package/loadtest

In der *DataGenerator*-Klasse sind, wie in Abbildung 5.5 zu sehen, sämtliche Parameter der Datenpakete beherbergt sowie Parameter, welche das Generieren der Testdaten beeinflussen. Außerdem werden folgende Funktionen bereitgestellt:

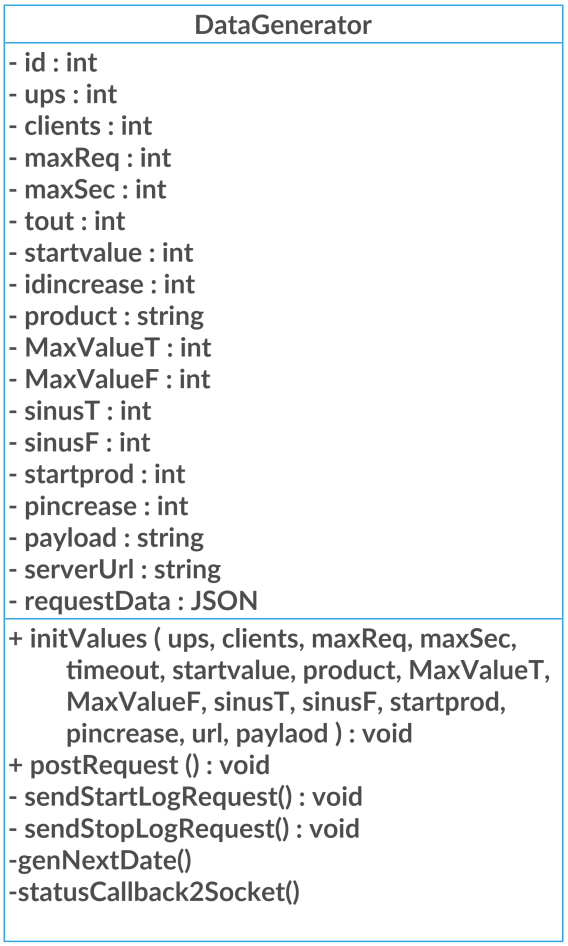


Abbildung 5.5: DataGenerator Klassen Diagramm

- *initValues():* Diese Funktion wird bei einer Änderung der Parameter aufgerufen, um den Zustand der in dieser Klasse gehaltenen Parameterwetern äquivalent mit den angezeigten Parametern im WebInterface zu halten.
- *sendStartLogRequest():* Initiiert die Log-Funktion der Server Applikation mittels eines *POST-Request's* an die Route *\startLog*.
- *sendStopLogRequest():* Stoppt die Log-Funktion der Server Applikation mittels eines *POST-Request's* an die Route *\stopLog*.
- *genNextDate():* Diese Funktion wird vor jedem *POST-Request's* aufgerufen und generiert ein individuelles Datum nach den vom Nutzer vorgegebenen mathematischen Richtlinien, welche im WebInterface definiert wurden und in den Klassenvariablen dieser Klasse hinterlegt sind.
- *statusCallback2socket:* Da diese Funktion nach jeder Anfrage aufgerufen wird und die aktuellen Zwischenergebnisse als Parameter übergeben bekommt, ist sie dafür zuständig, die Resultate der Main Klasse bereitzustellen sowie in die Log Datei zu schreiben.
- *postRequest():* Hier wird der eigentliche loadtest initiiert. Es wird zuerst der LogRequest an die Cloud Adapter Applikation gesendet, danach eine eigene Log-Funktion für die Ergebnisse der Latenzmessung erstellt, sowie das options Objekt für den Loadtest definiert (Abbildung 5.4) und anschließend der loadtest gestartet.

Steuerung

Die *Steuerungs*-Komponente dient zur Verwaltung sämtlicher Klassen und Funktionen. Diese Klasse ist der Startpunkt des Evaluations Frameworks. Es wird mittels *Express*[6] ein Server erstellt, auf dem das zuvor beschriebene WebInterface läuft. Außerdem werden sämtliche Routen verwaltet, mit denen das WebInterface Interaktionen mit der Anwendung durchführt. Es werden außerdem die vom *DataGenerator* bereitgestellten Zwischenergebnisse mit der *socket.io*[7] Biliothek an das WebInterface zur Darstellung übertragen. Im Folgenden sind die dafür bereitgestellten Funktionen erläutert:

- *runAuto():* In dieser Funktion steckt die eigentliche Logik der automatisierten sequentiellen Ausführung mehrerer Tests. Es werden nacheinander Lasttests mit verschiedenen Parametern ausgeführt, welche im Webinterface spezifiziert wurden.
- *updateEmitData():* Empfängt Zwischenergebnisse von laufenden Lasttests und speichert diese zur Übermittlung an das Webinterface.
- *updateChartData():* Empfängt Zwischenergebnisse für die Graphdarstellung von laufenden Lasttests und speichert diese zur Übermittlung an das Webinterface.
- *post():* Die verschiedenen post-Funktionen verwalten die Routen, auf denen das WebInterface mit der Applikation kommuniziert.

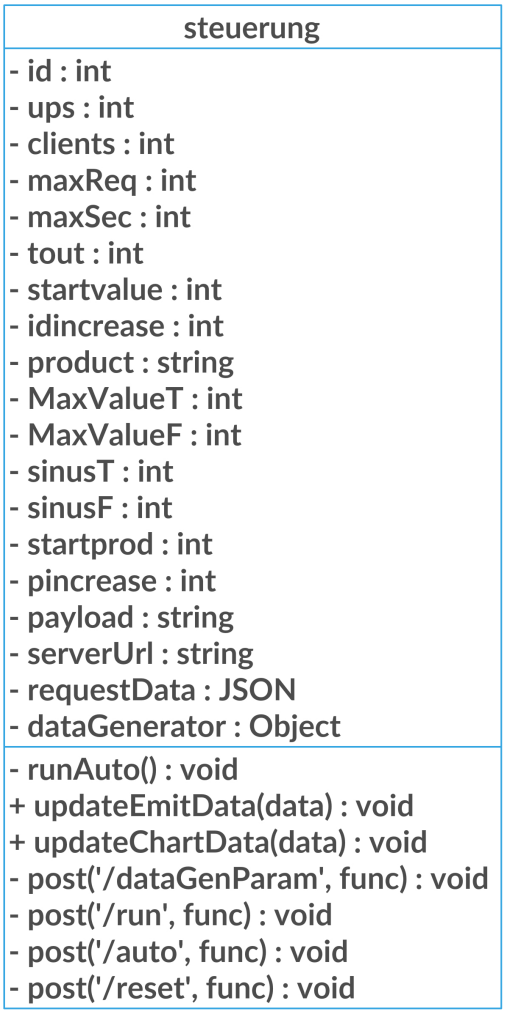


Abbildung 5.6: Steuerung Klassen Diagramm

[6] http://expressjs.com/
[7] http://socket.io/

5.2 Cloud Adapter Applikation

Die Cloud Adapter Applikation wird auf den verschiedenen CSP gehostet und empfängt Daten und Instruktionen des Evaluations Frameworks. Dabei sollen die empfangenen Daten an die von den CSP bereitgestellten Services übergeben und verarbeitet werden. Dafür sind die verschiedenen Adapter Klassen zuständig, welche im folgenden Kapitel genauer erläutert werden und von dieser Klasse benutzt werden. Die Anwendung ist außerdem dafür zuständig, Telemetriedaten des Cloudsystems zu messen und an das Evaluationsframework zu senden.

Für die Messung der Systeminformationen wird die nodejs Bibliothek *syteminformation*[8] verwendet. Dies ist eine einfach zu verwendende Bibliothek zum Auslesen von System und Betriebssystem Informationen.

Im Attribut *frameworkIp* wird die IP oder der Domain Name des Evaluations Frameworks vom Nutzer eingegeben. Dies wird für das Senden der Telemetriedaten benötigt. Das Cloud Objekt ist der Schnittstellenadapter für die Cloud Services. Hier muss vom Nutzer die jeweilige Adapterklasse angegeben werden je nachdem auf welchem CSP die Applikation gehostet wird. Die folgenden Funktionen sind Bestandteil der Cloud Adapter Applikation:

- *gatherData():* Wird einmal pro Sekunde aufgerufen und speichert Systeminformationen im *requestData* Objekt.
- *startCSVlog():* Startet die Logging funktion wichtiger Indikatoren während der Lasttests. Wird durch das Evaluationsframework durch einen POST-Request ausgelöst.
- *stopCSVlog():* Stoppt die Logging Funktion. Wird durch das Evaluationsframework durch einen POST-Request ausgelöst.
- *sendTelemetry():* Wird einmal pro Sekunde ausgeführt und sendet die im *requestData* Objekt gespeicherten Systeminformationen an das Evaluations Framework.
- *post():* Die verschiedenen post Funktionen verwalten die Routen, auf denen das Evaluationsframework mit der Applikation kommuniziert.

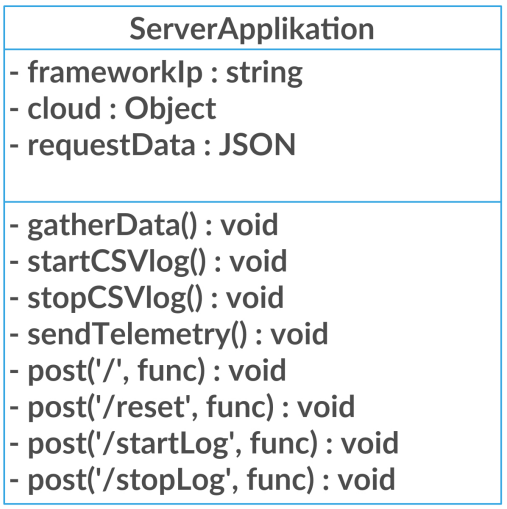


Abbildung 5.7: Server Applikation Klassen Diagramm

Die angebotenen Cloud Services der verschiedenen CSP verwenden meist unterschiedliche Authentifizierungsmethoden sowie verschwiedenen Übertragungsprotokolle. Deshalb wurde eine modulare Schnittstellenarchitektur dafür implementiert. Es wurde für jeden CSP eine eigene Adapter Klasse geschrieben, die die Spezifischen CSP und Service Anforderungen beachtet. Die jeweilige Klasse muss wie bereits erwähnt vom Nutzer in das cloud Objekt dieser Klasse eingetragen werden.

[8] https://github.com/sebhildebrandt/systeminformation

Cloud Adapter Klasse

Die Cloud Adapter Klassen werden speziell für jeden CSP implementiert und müssen die Methoden, wie in dem Interface (Abbildung 5.8) dargestellt, bereitstellen. Diese Klasse dient dazu, das empfangene Datum der Cloud Adapter Applikation an den Datenspeicherservice des CSP weiterzureichen und gegebenenfalls an weitere Services, falls die direkte Anbindung dieser 'Third-Party-Services' mit dem Datenspeicherservice nicht möglich ist. In der *cloudLib* Variable wird die vom CSP bereitgestellte Bibliothek zur Kommunikation mit den Services geladen, wie in folgender Abbildung exemplarisch für die Google Cloud Plattform zu sehen ist.

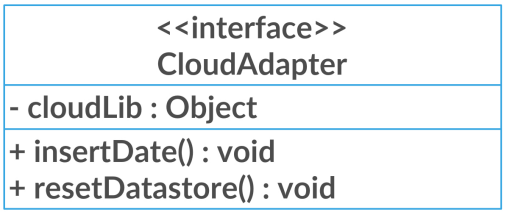


Abbildung 5.8: Cloud Adapter Klassen Diagramm

```
1 var cloudLib = require('gcloud');
```

Abbildung 5.9: Google Cloud Plattform API

Die benötigten Funktionen in diesen Klassen dienen hauptsächlich der Interaktion mit dem Datenspeicherservice, da die meisten weiteren Services eine cloudinterne Anbindung zu diesem Service besitzen.

- *insertDate():* Speichert das empfangene Datum in einem Puffer Objekt, welches in bestimmten Intervallen dann in den Datenspeicherservice geschrieben wird.
- *resetDatastore():* Setzt den Datenspeicherservice zurück.

5.3 Cloud Service Anbindung

Um die von den unterschiedlichen CSPs angebotenen Cloud Services zu nutzen, müssen diese angepasst, konfiguriert und an die *ServerApplikation* angebunden werden. In diesem Kapitel werden dafür die vorgenommenen Konfigurationen der benutzten Services, sowie die nötigen Implementierungsmaßnahmen zur Verbindung dieser Services erläutert.

5.3.1 IBM Bluemix

dashDB

Für den Service *dashDB* muss zuerst eine Verbindung zu der integrierten Datenbank hergestellt werden. Diese Verbindung kann über das Internet oder über den internen Bluemix Bus hergestellt werden. Dafür wird, nachdem der Service über das Dashboard verknüpft wurde, ein Eintrag in der lokalen Umgebungsvariable *VCAP_SERVICES* angelegt, in der sämtliche für den Zugang zum Service nötigen Informationen hinterlegt sind. Im Programmcode der Adapterklasse für Bluemix muss diese dann nur noch wie in Abbildung 5.10 dargestellt, ausgelesen werden.

Für das Erstellen einer Tabelle in diesem Service ist lediglich mittels *Data Definition Language* eine Struktur vorzugeben. Für die Analyse der in der Datenbank hinterlegten Daten bietet *dashDB* die Möglichkeit mit der für statistische und analytische Berechnungen ausgelegten Programmiersprache *R* den erstellten Datensatz zu analysieren.

```
if (process.env.VCAP_SERVICES) {
    var env = JSON.parse(process.env.VCAP_SERVICES);
    if (env['dashDB']) {
        hasConnect = true;
        db2 = env['dashDB'][0].credentials;
    }
}
```

Abbildung 5.10: dashDB VCAP_SERVICES

Internet of Things Foundation / Realtime Insights

Zur Visualisierung der Sensordaten dient der Service *IoT Realtime Insights*. Als Voraussetzung dafür muss jedoch der Service *IoT Foundation* zur Verwaltung von Sensorknoten ebenfalls instanziiert werden. Für die Verbindung zwischen der *IoT Foundation* und *IoT Realtime Insights* wird ein API Schlüssel und Access Token im *IoT Foundation* Service erstellt, welcher im *IoT Realtime Insights* Service eingegeben werden muss. Für die Verbindung der Sensoren (in dieser Arbeit das Evaluations Framework) zum *IoT Realtime Insights* Service wird ebenfalls ein Token generiert, welcher im Programmcode hinterlegt werden muss. Die Verbindung erfolgt über das *MQTT* Protokoll (vgl Abbildung 5.11).

```
var client = mqtt.connect(
    'tcp://cdja8q.messaging.internetofthings.ibmcloud.com:1883',
            {
        clientId: 'd:cdja8q:Test:001BB9AFB1E7',
        username: 'use-token-auth',
        password: 'yNSE?3b20tk)9s3?RQ'
    }
);
```

Abbildung 5.11: MQTT IoT Foundation

Flowthings.io

Flowthings ist eine Visualisierungsalternative eines Drittanbieter, die ebenfalls im IBM Katalog angeboten wird. Für die authentifizierte Verbindung zu den API Aufrufen dient ein Accesstoken, welcher zusammen mit dem Account Name im Code hinterlegt werden muss. Die Daten werden dann als sogenante *Drops* innerhalb von Flowthings repräsentiert. Der in Abbildung 5.12 zu sehende Programmcode zeigt das Erstellen eines Drops in Flowthings mit den zu repräsentierenden Daten.

```
    flowthingsWs.drop("f569ba14568056d1fd2b1078e").create( {
       elems: { Temperatur: parseFloat(dataBuffer3[0].
       Temperatur) } }, function () {})
```

Abbildung 5.12: Flowthings Drop

Die somit erstellten *Drops* lassen sich dann auf dem Flowthings Dashboard auf verschiedene Art und Weise visualisieren.

So ergibt sich für Bluemix die in Abbildung 5.13 dargestellte Lambda Architektur.

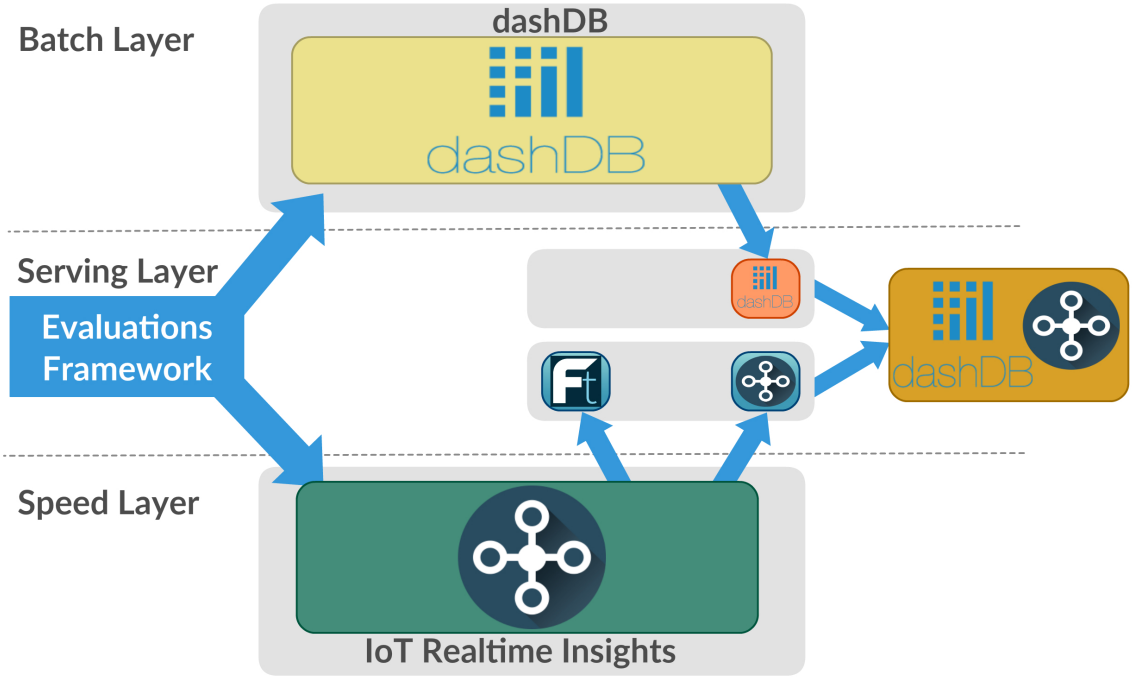


Abbildung 5.13: Implementierung der Lambda Architektur auf Bluemix

5.3.2 Google Cloud Plattform

BigQuerry

Die in dieser Arbeit benutzte Datenspeicherlösung der Google Cloud Plattform nennt sich *BigQuerry*. Beim Erstellen muss im Programmcode lediglich ein Schema der Tabelle angegeben werden. Alle weiteren Interaktionen erfolgen über die Google API mit Befehlen ähnlich normalen SQL Befehlen.

Datalab

Der Google Cloud *Datalab* Dienst bietet eine produktive, interaktive und integrierte Werkzeugsammlung zum Erforschen, Visualisieren, Analysieren und Transformieren von göße-ren Datenmengen. Es vereint die Vorteile von Python, SQL und der Google Cloud Plattform. In diesem Dienst lassen sich mittels der Programmiersprache *iPython* sogenannte *Notebooks* implementieren. Dies sind interaktive Computerumgebung, in der sich die Ausführung von Code, Text, Mathematik und medialen Interaktionen kombinieren lassen. Für diese Arbeit wurden dafür zwei Notebooks erstellt, welche nach den Konventionen der Lambda Architektur die Aufgaben des Speed- und Batchlayer abdecken (siehe Abbildung 5.16 / Kapitel 4.1).

Eine Abfrage des Datenspeichers und die Visualisierung und Analyse dieser Daten findet im *Batchlayer-Notebook* statt. Wie in Abbildung 5.15 zu sehen, erfolgt eine Datenanfrage an den *Bigquerry* Dienst. Die Daten werden danach visualisiert und zur Veranschaulichung mit einfachen mathematischen Funktionen analysiert.

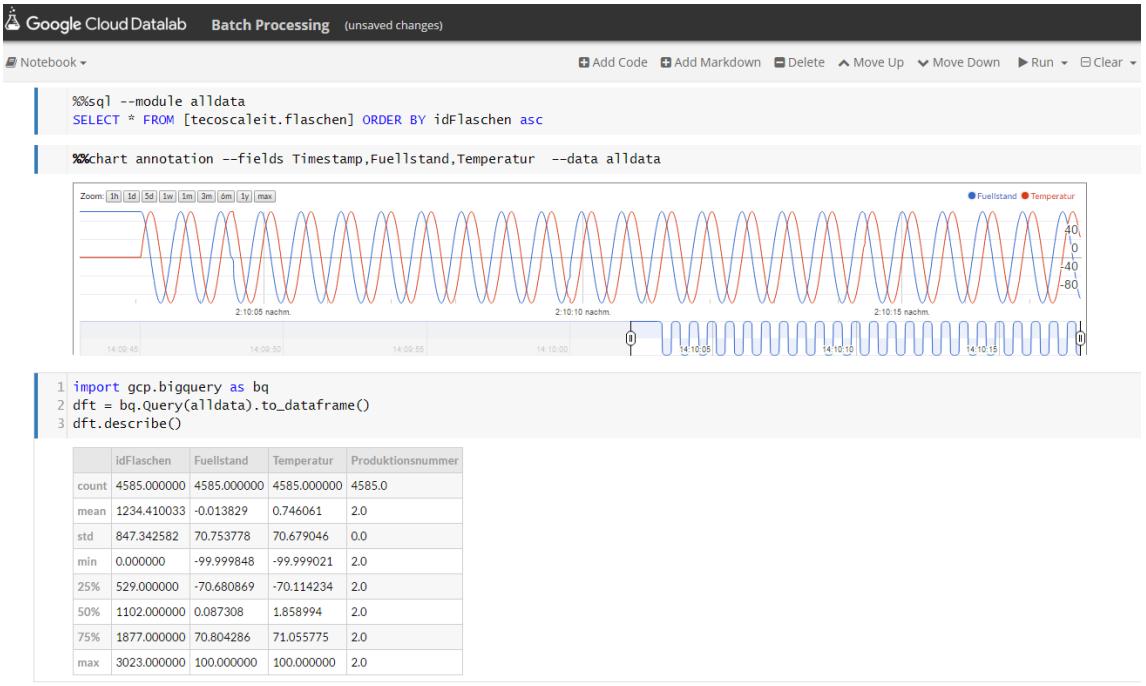
Für das *Speedlayer-Notebook* wurde mittels *iPython* und der *matplotlib.pyplot* Bibliothek eine Live Visualisierung der neuesten Sensordaten implementiert. Der Code dafür ist in Abbildung 5.14 zu sehen.

Mit den Google Cloud Services wird die in Abbildung 5.16 dargestellte Lambda Architektur implementiert.

```
import gcp.bigquery as bq
import time
import numpy as np
import matplotlib.pyplot as plt
from IPython import display

plt.ion()
plt.show()
df = bq.Query(limitdata).to_dataframe()
y1 = df['Temperatur'].values
y2 = df['Fuellstand'].values
x = df['idFlaschen'].values
while 1==1 :
    df = bq.Query(limitdata).to_dataframe()
    y1 = df['Temperatur'].values
    y2 = df['Fuellstand'].values
    x = df['idFlaschen'].values
    plt.gca().cla()
    plt.xlabel("Zeit", fontsize=20)
    plt.ylabel("Temperatur\Fuellstand", fontsize=20)
    plt.plot(x,y1,label='Temperatur')
    plt.plot(x,y2,label='Fuellstand')
    plt.legend()
    display.clear_output(wait=True)
    display.display(plt.gcf())
```

Abbildung 5.14: Datalab Speed Layer



	idFlaschen	Fuellstand	Temperatur	Produktionsnummer
count	4585.000000	4585.000000	4585.000000	4585.0
mean	1234.410033	-0.013829	0.746061	2.0
std	847.342582	70.753778	70.679046	0.0
min	0.000000	-99.999848	-99.999021	2.0
25%	529.000000	-70.680869	-70.114234	2.0
50%	1102.000000	0.087308	1.858994	2.0
75%	1877.000000	70.804286	71.055775	2.0
max	3023.000000	100.000000	100.000000	2.0

Abbildung 5.15: Datalab Batch Layer

Da Google nur einen kleinen Katalog an Cloud Diensten zur Verfügung stellt, diese jedoch abstrakt gehalten sind, ist die Funktionalität des Speed und Batch Layers der Lambda Architektur (Abb. 5.16) mit diesen Services und eigener Implementierungen entstanden. So ist die Funktionalität des Speedlayers im Datalab Service mit eigenem iPython Code programmiert, welcher dann innerhalb des Services ausgeführt wird.

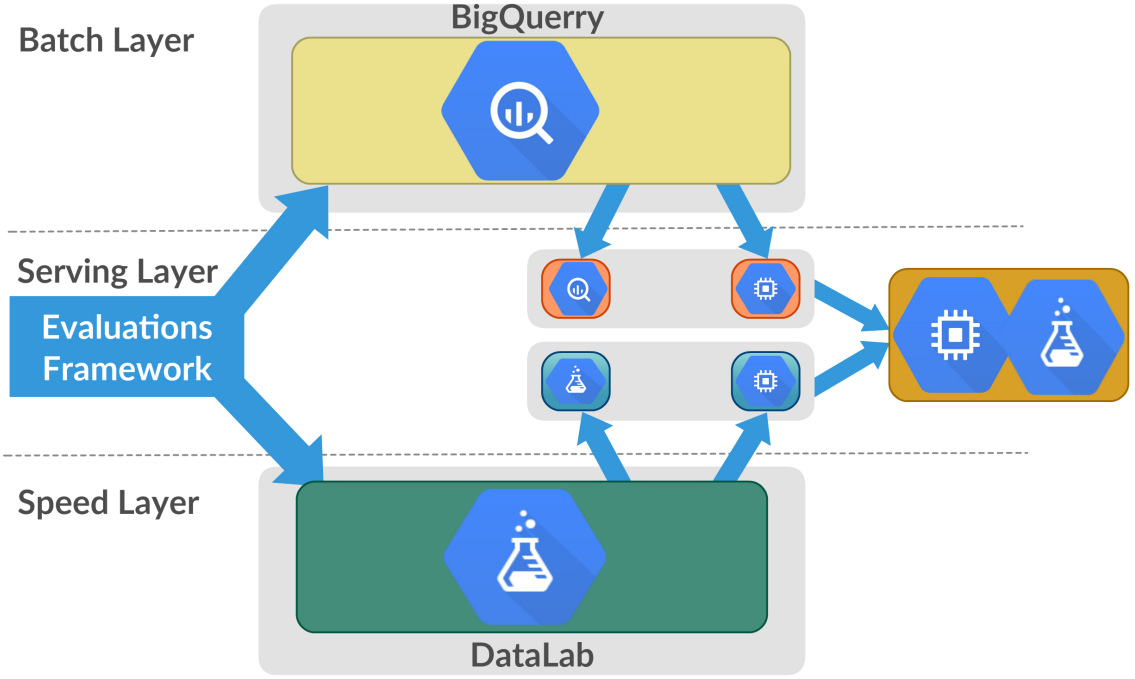


Abbildung 5.16: Implementierung der Lambda Architektur auf der Google Cloud Plattform

5.3.3 Microsoft Azure

MySQL Database: Die bei Azure genutzt Datenspeicherlösung wird mit dem MySQL Database Dienst realisiert. So werden die übertragenen Nachrichten an Azure mittels HTTP direkt in dieser Datenbank gespeichert.

IoT Hub: Der IoT Hub Dienst wird mittels der vorhandenen NodeJs API angesteuert. Um eine Nachricht zu erstellen, werden die Attribute in einen JSON-String verpackt und wie in Abbildung 5.18 zu sehen mit den nötigen API calls an den IoT Hub geschickt.

```
1     var data = JSON.stringify(dataBuffer2[0]);
2     var message = new Message(data);
3     message.properties.add('myproperty', 'myvalue');
4     console.log('Sending message: ' + message.getData());
5     client.sendEvent(message, function (err) {
6       if (err) console.log(err.toString());
7     });
```

Abbildung 5.17: IoT Hub API

Stream Analytics: Für den StreamAnalytics Dienst müssen sogenannte Jobs erstellt werden, welche mittels Structured Query Language Daten eines Inputs verarbeiten und an einen Output Dienst liefern. So wurde für diese Arbeit je ein Job für das Batch und Stream Processing erstellt. Als Input Dienst wird der IoT Hub gesetzt und als Output der Visualisierungsdienst PowerBI. In Abbildung 5.18 ist exemplarisch die Jobdefinition für den Speedlayer zu sehen.

```
1       SELECT System.Timestamp as WindowEnd, idflaschen, COUNT(*)
            as CallCount, max(Temperatur) as maxTemp, max(
           Fuellstand)
2       INTO Ausgabealias
3       FROM Eingabealias
4       GROUP BY TUMBLINGWINDOW(ms, 1), idflaschen
```

Abbildung 5.18: IoT Hub API

Power BI: Für den PowerBI Dienst müssen keine Implementierungsmaßnahmen vorgenommen werden, da die Daten bereits durch den StreamAnalytics Dienst den Weg zu PowerBI finden und das Erstellen von Diagrammen intuitiv mit Mausgesten zusammengeklickt werden können.

Die Lambda Architektur in Abbildung 5.19 zeigt die einzelenen Services und ihre Funktion in der Architektur.

5.4 Decision Support System

Das Decision Support System ist eine auf Javascript basierende interaktive Website, welche mittels eines einfachen Nodejs Web-Server zur Verfügung gestellt wird. Für die graphische Darstellung und der Tab Ansicht werden die bereits beim Evaluations Framework verwendeten Bibliotheken Bootstrap und jQuery verwendet. Für das Erstellen der Graphen wird die Highcharts Bibliothek verwendet. Mit dieser ist es möglich aus HTML Tabellen eine Chart-Repräsentation zu generieren. Die in der Evaluation dieser Arbeit entstehenden Datensätze werden dafür in Excel als HTML Tabelle exportiert und auf der Website

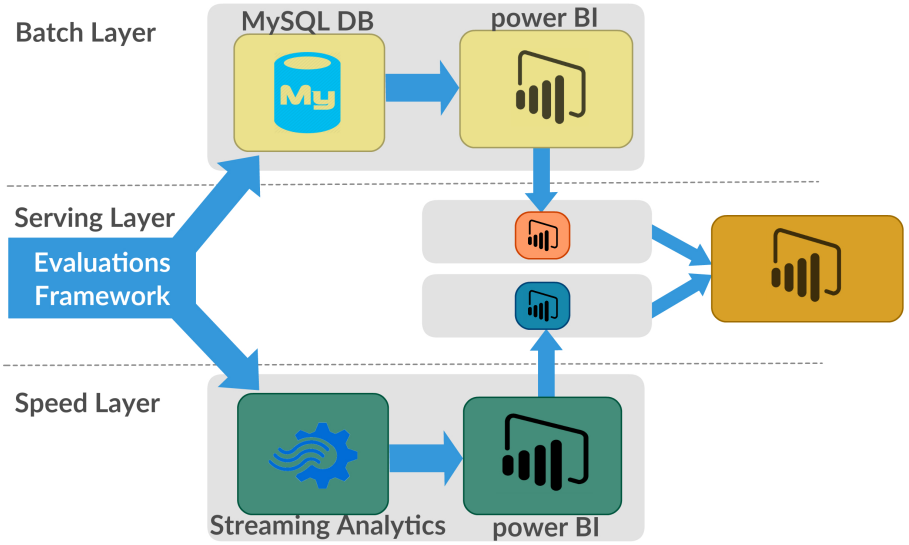


Abbildung 5.19: Implementierung der Lambda Architektur auf Azure

Visualisiert. Dafür muss im HTML <table> Tag die in Abb. 5.20 zu sehende Klasse *Highchart* hinzugefügt werden. Mit dem Style Attribut lässt sich die Tabelle ausblenden, sodass lediglich der Graph angezeigt wird.

```
1     <table id="myID" style="display:none" class="highchart">
```

Abbildung 5.20: Highcharts Table Tag

Der dazugehörige Javascript Code dient der Initialisierung und Konfiguration der Charts. Wie in Abb. 5.21 zu sehen ist wird dafür ein Options Objekt erstellt welches die von Highchart definierten Attribute und die Datenquelle enthält. Beim Aufruf der highchart() Funktion wird dieses Objekt übergeben und der Graph erstellt. Desweiteren ist der Javascript Code zuständig bei einer Änderung der Modellparameter die passende HTML Tabelle auszuwählen und die Graphen mit den passenden Daten zu aktualisieren. Der komplette Code ist auf der beigelegten CD dieser Arbeit zu finden. In Abbildung 5.22 ist das Resultat der Visualisierung zu sehen.

```
1     var options = {
2         data: {
3           table: 'memory1'
4         },
5         chart: {
6           type: 'line'
7         },
8         yAxis: {
9           allowDecimals: false,
10          title: {
11            text: 'Memory Usage in mb'
12          }
13        }
14      }
15
16      $('#chartmem').highcharts(options);
```

Abbildung 5.21: Highcharts Javascript

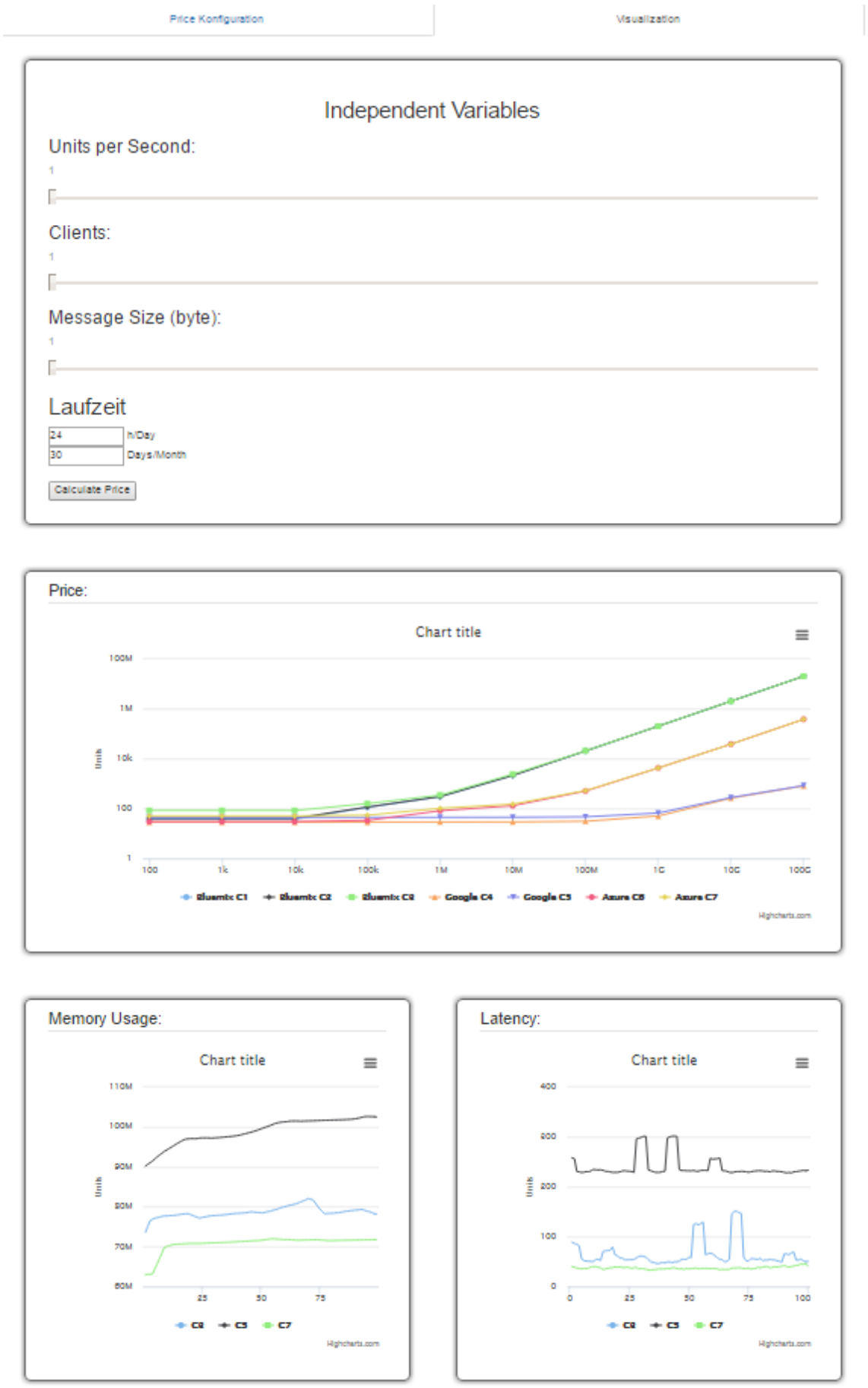


Abbildung 5.22: Decision Support System

6. Evaluation

In diesem Abschnitt wird mit den in den vorherigen Kapiteln erstellten Software Komponenten eine Evaluation der betrachteten CSP durchgeführt. Dafür werden mit dem Evaluations Framework Testdaten generiert, welche auf den verschiedenen CSP empfangen und bearbeitet werden sollen. Während dieser Testläufe werden verschiedene KPIs gemessen, um basierend darauf ein Vergleich der CSP vornehmen zu können.

6.1 Setting und Evaluationsbeschreibung

Für die Evaluation der CSP wird das Evaluationsframework auf einer Virtuellen Maschine mit den in Abb. 6.1 zu sehenden Spezifikationen gehostet, welche über eine ausreichend stabile Bandbreite für die definierten Testszenarien dieser Arbeit verfügt. Damit ist die Ausführung der Lasttests auf einer Generalisierten Testumgebung gewährleistet.

Die Server Applikation wird auf den verschiedenen CSP mit jeweils verschiedenen Maschinenkonfigurationen gehostet. Dies ist in dem Deployment Diagramm in Abbildung 6.1 dargestellt. Dabei werden die in folgender Tabelle 6.1 aufgeführten Maschinenkonfigurationen betrachtet:

Tabelle 6.1: Cloud Service Provider Settings

	CSP:	CPU:	Memory:
Minimal Configurations	Bluemix (C1)	∼[2]	64mb
	Google (C4)	1 shared vCPU	0.6gb
	Azure (C6)	0.25 Kern	0.75gb
Scenario-Fit Configurations	Bluemix (C3)	∼[2]	1.75gb
	Google (C5)	1 vCPU	1.75gb
	Azure (C7)	1 Kern	1.75gb
Comparable Configurations	Bluemix (C2)	∼[2]	600mb

Minimal Configurations: Die von den jeweiligen CSP angebotene Konfiguration mit den wenigsten Resourcen.

Scenario-Fit Configurations: Eine angemessene Maschinenkonfiguration zum Verarbeiten der für diese Arbeit definierten Szenarien.

[2]Die CPU spezifikation werden von Bluemix nicht preisgegeben. Die CPU Skalierung erfolgt automatisch von Bluemix und lässt sich vom Nutzer nicht anpassen.

```
1  CPU(s):                2
2  Architecture:          x86_64
3  CPU op-mode(s):        32-bit, 64-bit
4  processor  QEMU Virtual CPU version 0.12
5  processor  QEMU Virtual CPU version 0.12
6  Vendor ID:             GenuineIntel
7  CPU family:            6
8  CPU MHz:               3499.996
9  BogoMIPS:              6999.99
10 L1d cache:             32K
11 L1i cache:             32K
12 L2 cache:              4096K
13 MemTotal:            1024344 kB
14
15 Networktest (speedtest-cli)
16   Testing from Karlsruhe Institute of Technology
        (129.13.170.100)...
17   Hosted by Vodafone DE (Frankfurt) [125.50 km]: 9.683 ms
18   Testing download speed
        ....................................
19   Download: 118.67 Mbit/s
20   Testing upload speed
        ............................................
21   Upload: 105.89 Mbit/s
```

Abbildung 6.1: VM Specs

Comparable Configuration: Eine weitere Maschinenkonfiguration für Bluemix, um die vergleichbarkeit mit den anderen CSP sicherzustellen.

Es wird jeweils die kleinste verfügbare Maschinenkonfiguration der CSP instanziiert (C1, C4, C6), sowie je eine Instanz mit für die in dieser Arbeit betrachteten Szenarien passenden Parametern (C3, C5, C7). Da jedoch bei den einzelnen CSP verschiedene Schrittweiten zur Skalierung der Maschine angeboten werden, wird eine weitere Konfiguration (C2) zur vergleichbarkeit betrachtet. Anzumerken hierbei ist jedoch die Prozessorkonfiguration, welche je nach Anbieter abweicht. Bei Bluemix lässt sich die Prozessorkonfiguration nicht anpassen, wobei Azure und Google die Entscheidungsmöglichkeiten *shared CPU*, vCPU und unterschiedlicher Anzahl an Kernen anbieten. Für die eigentliche Evaluation soll das Verhalten abhängiger Variablen bei verschiedener Konfiguration von unabhängigen Variablen untersucht werden. In dieser Arbeit sind die betrachteten abhängigen Variablen die Key Performance Indikatoren, dargestellt in Abbildung 6.3, und die unabhängigen Variablen die in Tabelle 6.2 dargestellten Konfigurationensmöglichkeiten des Evaluations Frameworks. In Tabelle 6.2 sind 5 Testszenarien definiert, welche auf jeder der verschiedenen Maschinenkonfigurationen C1 - C7 ausgeführt werden. Dabei wird ein LOG-File erstellt was gemessene KPIs während des Testlaufs protokolliert.

Tabelle 6.2: Test Definitions

Unabhängige Variablen				
Test	Units/s	Clients	Message Size	Message Limit
1	1	1	350 byte	100
2	1	1	500kb	100
3	100	1	350 byte	10000
4	100	10	350 byte	10000
5	1 - 100	1	350 byte	15000

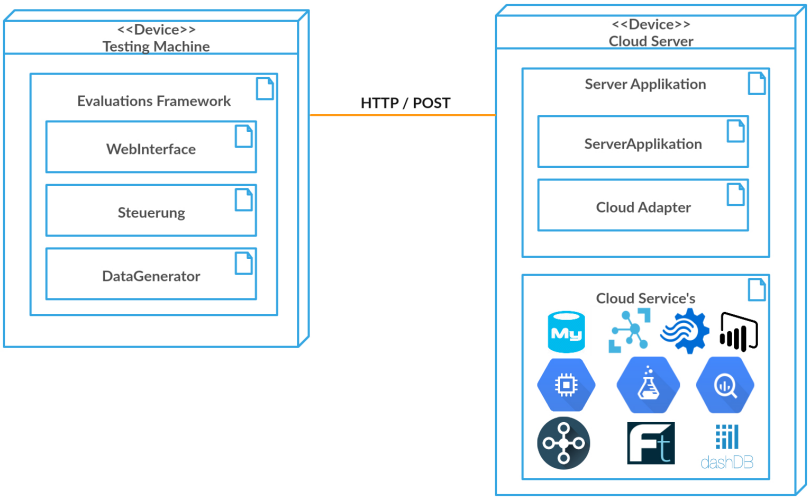


Abbildung 6.2: Software Deployment Diagramm

6.2 Angewandte KPIs

Die in Kapitel 2.2 eingeführten KPIs in der Literatur können in ihrer Anzahl und Genauigkeit nicht von den einzelnen CSP betrachtet werden. Da die einzelnen Cloudanbieter intransparent sind, was ihre zugrundeliegende Architektur und Infrastruktur angeht, ist es sehr schwer, an nützliche Informationen zum bestimmen der verschiedenen Indikatoren zu kommen. Außerdem muss für einen Vergleich verschiedener CSP der jeweilige KPI auf allen zu vergleichenden Anbietern messbar sein. In folgender Abbildung 6.3 sind die abhängigen Variablen zu sehen, welche auf den untersuchten Plattformen messbar sind. Dabei wird der Packet Loss Indikator als Quality of Service dargestellt um Fehlinterpretationen der Zehnerskala mit der eigentlichen Güte des Indikators zu vermeiden.

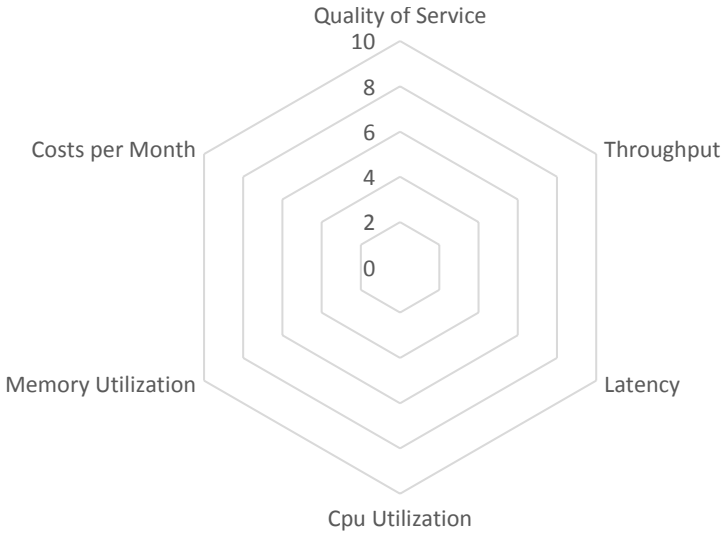


Abbildung 6.3: Abhängige Variablen (KPI)

6.3 Cloud Provider Selection Problem - Modell

Ein wichtiger erster Schritt bei der Entscheidungsfindung ist das Entscheidungsproblem in eine formale und rigorose Form zu bringen. Ein guter Ansatz dafür liefert die Veröffent-

lichung "Towards Multi-Criteria Cloud Service Selection"[uRHH11], welche grundlegende Elemente bei der Entscheidungsfindung von CSP in ein mathematisches Modell abstrahiert.

Im folgenden Abschnitt präsentieren wir das *Cloud Provider Selection* Problem in einer Generalisierten und abstrakten mathematischen Form und bilden unsere eigentliche Evaluation basierend darauf.

Modell

Sei C die Menge der verschiedenen CSP die für diese Arbeit betrachtet werden, wobei gilt $C = \{c_1, c_2, \dots c_l\}$ mit $l \geq 2$.

Desweiteren sei $K = \{k_1, k_2, \dots k_m\}$ die Menge der zu messenden *Key Performance Indicators*, wobei jedes $k_i \in K$ einen KPI repräsentiert, welcher zur Auswahl der CSP betrachtet wird.

Zu jedem KPI $k_i \in K$ korrespondiert eine eindeutige Funktion $f_i \in F$, welche angewandt auf einem $c_i \in C$ bezüglich eines $k_i \in K$ einen Wert $p_i \in P$ liefert. p_i beschreibt dabei die Leistung des CSP bezüglich des KPI. Die Menge aller dieser Funktionen wird als $F = \{f_1, f_2, \dots f_m\}$ definiert. Dabei ist F die Menge der Funktionen, die die Leistungen der einzelnen Maschinenkonfigurationen für alle $k_i \in K$ auf eine Skala von 0 - 10 abbilden. Es gilt

$$\forall f_i \in F : f_i = \frac{x - worst(c_j(k_i))}{best(c_j(k_i)) - worst(c_j(k_i))} * 10 \tag{6.1}$$

,was eine einfache Skalierung der Messwerte auf der Vergleichsskala darstellt. Dabei beschreibt $p_i = c_j(k_i)$ den gemessenen KPI k_i zur Maschinenkonfiguration c_j. Die Funktion $best()$ gibt den besten und $worst()$ den schlechtesten Wert unter allen betrachteten Maschinenkonfigurationen aus.

Außerdem definieren wir die Zeilenvektoren (1 x n Matrix) D_i wobei jedes Element in $d_j \in D_i$ die Leistung eines CSP $c_i \in C$ unter einem KPI $k_j \in K$ wiederspiegelt. In anderen Worten ist $D_i = \{d_1, d_2, \dots d_m\}$, wobei $d_j = f_j(c_i)$. Dieser Vektor muss außerdem für die später folgende Gegenüberstellung normiert werden.

Die Leistungsvektoren D_i können zeilenweise zur Entscheidungsmatrix A (l x l Matrix) zusammengefasst werden.

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{l,1} & a_{l,2} & \dots & a_{l,n} \end{pmatrix} \tag{6.2}$$

Wobei jedes $a_{i,j}$ die Auswertung des CSP c_i bezüglich des KPIs k_j darstellt.

Gleich dem Vektor D_i wird ein weiterer Vektor $R = \{r_1, r_2, \dots r_m\}$ definiert, wobei jedes r_i die Mindesanforderung des Benutzers an den KPI $k_i \in K$ darstellt. Für die später folgenden Gegenüberstellung des Anforderungsvektors R zu den Leistungsvektoren D_i muss dieser ebenfalls normiert werden.

Ein Möglichkeit, die KPIs unterschiedlich zu gewichten, soll in der Evaluation dieser Arbeit nicht betrachtet werden, wird jedoch vollständigkeitshalber in dieser Modellabstraktion

dennoch eingeführt. Es sei der Gewichtsvektor $W = \{w_1, w_2, \dots w_m\}$, wobei jeds w_i das Gewicht zum KPI k_i repräsentiert. In dieser Arbeit definieren wir $W = \{1, 1, \dots 1_m\}$, um sämtlichen KPIs das selbe Gewicht zu geben.

Das Ziel der CSP-Auswahl ist es, den passendsten Anbieter von der Menge aller Anbieter auszuwählen. Dieser Prozess ist, in Betracht des soeben definierten Modells, der Vergleich des Vektors R zu allen Zeilen der Entscheidungsmatrix A und des finden einer Zeile A_i, welche am ehesten mit dem Anforderungsvektor R übereinstimmt. Die Operationen zur Ähnlichkeitsberechnung erfolgt auf dem Anforderungsvektor und der Entscheidungsmatrix anstatt der individuellen Leistungsvektoren aufgrund der Notationserleichterung und Einfachheit.

Es gibt verschiedene Methoden zur Ähnlichkeitsbestimmung von Vektoren wie die *Euklidische Distanz*, *Pearson's Correlation* oder *Cosine Similarity*. Bei diesen Methoden wird jedoch die Ähnlichkeit zweier Vektoren berechnet ohne Berücksichtigung des Vorzeichen des Resultats. Es kann also nicht unterschieden werden, ob ein CSP die Anforderungen des Nutzers übersteigt oder nicht erwartungsgemäß erfüllt. Die in dieser Arbeit verwendete Methode ist die in [uRHH11] definierte *Exponential Weighted Difference (EWD)*. Dabei wird zuerst der Anforderungsvektor R von jeder Reihe der Entscheidungsmatrix substrahiert und das Resultat mit dem Gewichtsvektor multipliziert. Die Matrix wird mit dem Skalar -1 multipliziert und jedes Element als Exponent der Exponentailfunktion genommen und Zeilenweise Addiert, was zum Ergebnisvektor E führt:

$$E = \begin{pmatrix} e^{-(a_{1,1}-r_1)w_1} + e^{-(a_{1,2}-r_2)w_2} + \ldots + e^{-(a_{1,m}-r_m)w_m} \\ e^{-(a_{2,1}-r_1)w_1} + e^{-(a_{2,2}-r_2)w_2} + \ldots + e^{-(a_{2,m}-r_m)w_m} \\ \vdots \\ e^{-(a_{l,1}-r_1)w_1} + e^{-(a_{l,2}-r_2)w_2} + \ldots + e^{-(a_{l,m}-r_m)w_m} \end{pmatrix} \tag{6.3}$$

Das Element d_i mit dem niedrigsten Wert ist demnach dann der passendste CSP c_i zum Anforderungsvektor R

6.4 Evaluationsdurchführung

Die Durchführung der Evaluation besteht aus dem Ausführen der einzelnen Tests mit dem Evaluations Framework auf den jeweiligen CSP. Es wird jeder der fünf Tests auf sämtlichen Maschinenkonfigurationen ausgeführt. Um Ausreißer und irreguläres Verhalten zu kompensieren, wird jeder Test mit der selben Parametriesierung dreimal nacheinander auf jeder Maschinenkonfiguration ausgeführt und davon der Mittelwert berechnet. Somit werden beim Durchführen der Evaluation

$$7\ Maschinenkonfigurationen\ *5\ Tests\ *3\ Durchlaeufe\ =\ 105 \tag{6.4}$$

Datensätze erstellt, welche es zu analysieren gilt. Damit das Auftreten einzelner Ausreißer im Datensatz keine große Gewichtung im Endergebniss findet, wird der Mittelwert mit dem Sliding Window Verfahren und einer Fenstergröße von 5 berechnet, und anschließen der Mittelwert über die Resultate der drei Durchläufe gebildet, womit für das Ergebnis aussagekräftige geglättete Werte entstehen.

6.5 Evaluationsergebnisse

Die Ergebnisse der in dieser Arbeit ausgeführten Evaluation wurden mittels Liniendiagramen visualisiert und aufbereitet. Durch die große Menge an erstellten Datensätzen zu den jeweiligen Maschinenkonfigurationen und verschiedenen Tests werden in diesem Abschnitt nur die Diagramme mit aussagekräftigen Daten und interessanten Eigenschaften aufgeführt. Es sind jedoch alle Diagramme im Anhan dieser Arbeit hinterlegt. Nach der Visualisierung und Diskussion der Ergebnisse in diesem Abschnitt wird das abstrakte Evaluationsmodell zum *Cloud Provider Selection Problem* auf die Evaluationsergebnisse angewendet.

Test 1

Im ersten Test wurden lediglich 100 Einheiten des standarddatums ohne Payload-Anhang mit 1 U/s an die verschiedenen Maschinenkonfigurationen gesendet. Das jeweils gesendete Datum beträgt mit leerem Payload eine Größe von 350 byte. Dabei ist zu sehen, dass der Memory verbauch bei allen Konfigurationen größtenteils konstant bleibt, da diese geringe Auslastung keine große Last für die Maschinen darstellt. Interessant ist jedoch, dass die Menge des genutzten Memory in diesem niedrigen Lastzustand je nach CSP unterschiedlich hoch ist. Die Instanzen der Google Cloud Plattform vebraucht bei diesem Test ∼100mb, wohingegen Bluemix ∼80mb und Azure ∼70mb verbrauchen.

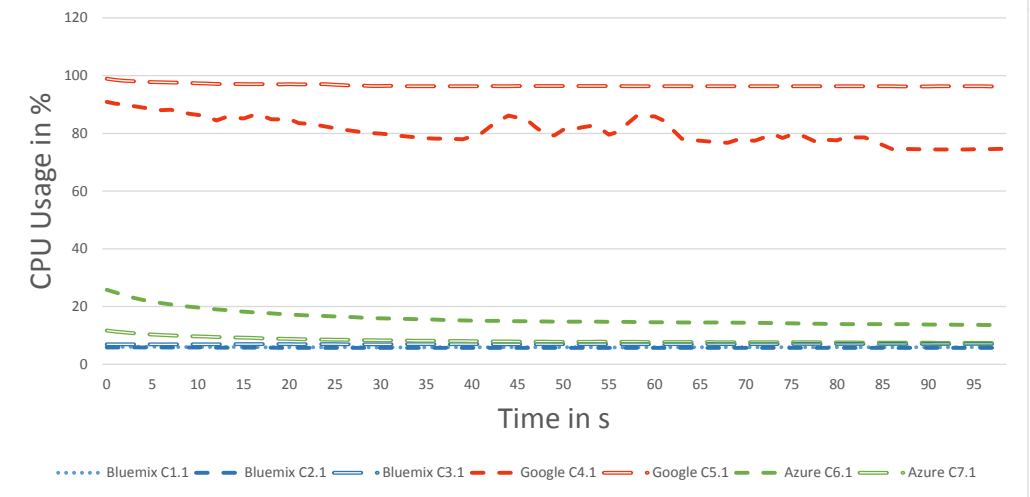


Abbildung 6.4: CPU Verhalten bei Test 1

Außerdem ist zu sehen, dass die Minimal Konfiguration von Bluemix mit einem Memory Limit von 64mb konstant am Limit ist. Bezüglich der Latenz gibt es ebenfalls große Unterschiede je nach Anbieter. Wo die beiden Google Instanzen (C5, C4) Latenzen von ∼250ms und ∼400ms aufweisen, sind die Latenzen der Azure- und Bluemixinstanzen, mit außnahme einzelner Peaks, unter 100ms. Wobei bei Azure eine Konstanz von ca 50ms zu sehen ist, tauchen bei Bluemix generell ziemlich viele unregelmäßigkeiten in den Werten auf. Die Werte von Bluemix befinden sich jedoch, mit Ausnahme einzelner Ausreiser, im Bereich von 60 - 70ms befinden. Was die CPU-Auslastung angeht, gibt es leider keine vergleichbaren Werte zwischen den einzelnen CSP, da Informationen über die physische CPU von den anbietern Intransparent gehalten werden. So lässt sich zum Beispiel bei Bluemix keine Konfiguration der CPU vornehmen, wobei bei Google die Minimal Konfiguration eine shared CPU beinhaltet und die höheren Klassen vCPU's. Bei Azure ist die Angabe zur Minimalkonfiguration 0.25 Kerne und bei der Scenario-fit Konfiguration ein Kern. So ergeben sich, wie in Abbildung 6.4 zu sehen je nach Anbieter sehr unterschiedliche Werte.

Speziel zu erwähnen ist hierbei das unterschiedliche Verhalten der beiden Google Instanzen C5 und C4. Wo das Verhalten von C5 durch eine Monotone Kurve dargestellt werden kann, herrscht bei der C4 Konfiguration ein Flattern der Kurve. Dies ist vermutlich auf die shared CPU bei C4 und die vCPU Konfiguration bei C5 zurückzuführen.

Tabelle 6.3: Übersicht der Testergebnisse für Test 1

	Bluemix C1	Bluemix C2	Bluemix C3	Google C4	Google C5	Azure C6	Azure C7
U/s	1						
Clients	1						
Message Size	350 byte						
Total Messages	100						
Total Errors	0	0	0	0	0	0	0
Mean Latency	70 ms	53 ms	65 ms	385 ms	239 ms	37 ms	37 ms
Mean Cpu Load	5.91 %	5.72 %	6.98 %	80.91 %	96.62 %	15.87 %	8.24 %
Mean Memory	62.22 MB	77.80 MB	78.63 MB	99.74 MB	98.92 MB	70.94 MB	70.84 MB

Test 2

In Test 2 wird die unabhängige Variable Message Size auf 500 kbyte erhöht. Dafür wurde ein String als Payload an die eigentliche Nachricht gehängt, sodass ein 500 kbyte großes Datenpaket entsteht. Die weiteren unabhängigen Parameter bleiben im Vergleich zu Test 1 unverändert.

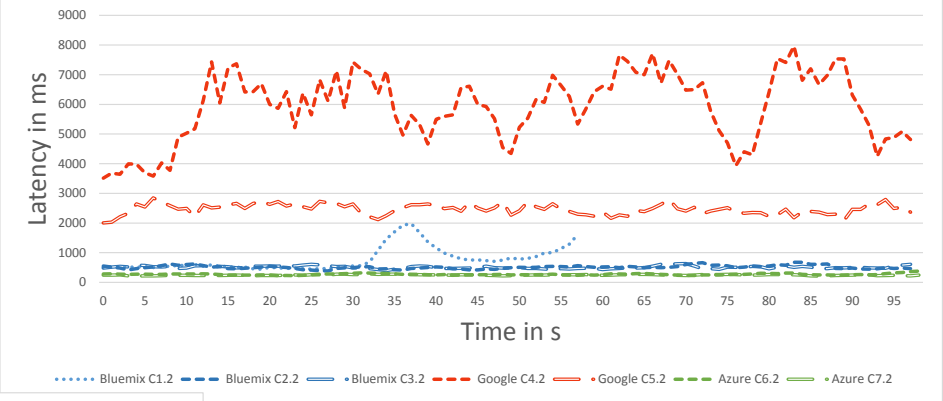


Abbildung 6.5: Latenz Verhalten bei Test 2

Beim Memory Verhalten ist bei allen Maschinenkonfigurationen erstmals ein Anstieg zu verzeichnen. Die Google Instanzen nähern sich beide asymptotisch an ∼110mb an. Die Instanzen von Azure und Bluemix C2 und C3 haben jeweils einen niedrigeren Ausgangspunkt, wobei die Bluemixinstanz einen Anstieg bis ∼105mb aufweisen und danach konstant bleiben. Die Azure Instanzen übersteigen nach ca 50 Sekunden sogar die Kurven der Google Instanzen und nähern sich dann ∼120mb an. Ein komplett unerwartetes Verhalten liegt dabei bei Bluemix C1 vor. Dies ist die 64mb RAM-Konfiguration von Bluemix, welche ständig an dieser 64mb Grenze ist und nach ca 55 Sekunden abstürzt. Dies ist ebenfalls in Abbildung 6.5 zu sehen, bei der nach dem Absturz keine weiteren Latenzwerte mehr gemessen werden konnten. In dieser Abbildung sind ebenfalls wieder die höheren Latenzen der Google Instanzen zu sehen, wobei C4 mit dem shared CPU außerdem noch ein ziemlich großes Flattern aufweist, wohingegen Googles C5 konstanter bleibt. Bei Azure und Bluemix ist dieses Flattern garnicht zu beobachten. Bluemix bleibt konstant bei ca. 510 ms und Azure bei 255 ms. Das CPU-Verhalten bei diesem Test ist relativ äquivalent

zur Test 1 Abbildung 6.4, wobei dabei ebenfalls der Absturz von C1 zu erkennen ist und die Azure Instanzen im Vergleich zu Test 1 anfangs einen leichten Anstieg verzeichnen.

Tabelle 6.4: Übersicht der Testergebnisse für Test 2

	Bluemix C1	Bluemix C2	Bluemix C3	Google C4	Google C5	Azure C6	Azure C7
U/s	1						
Clients	1						
Message Size	500000 byte						
Total Messages	100						
Total Errors	32	0	0	0	0	0	0
Mean Latency	780 ms	511 ms	512 ms	5916 ms	2466 ms	273 ms	247 ms
Mean Cpu Load	7.42 %	6.47 %	6.78 %	79.99 %	95.35 %	29.03 %	19.43 %
Mean Memory	65.72 MB	95.10 MB	95.02 MB	106.40 MB	104.24 MB	105.09 MB	105.02 MB

Test 3

In diesem Test wird der Durchsatz der Einheiten auf 100 U/s gesetzt. Die Message size ist wiederum die Minimalgröße von 350 byte. Bei diesem Test werden dabei 10000 Nachrichten an die CSP gesendet.

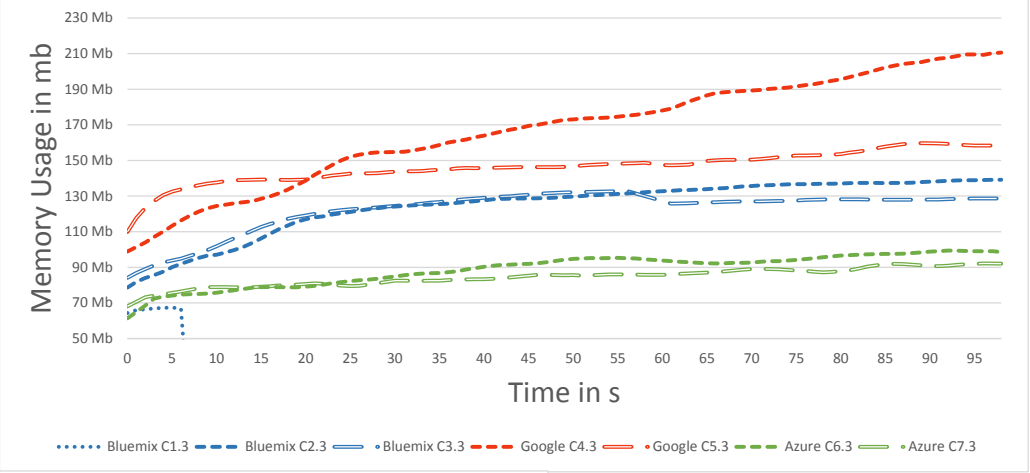


Abbildung 6.6: Memory Verhalten bei Test 3

Beim Memory Verhalten ist dabei, wie in Abbildung 6.6 zu sehen ein stetiger Anstieg bei allen Instanzen zu beobachten. Man erkennt dabei wiederum eine obere Grenze für jeden Anbieter, an die sich die Kurven annähern. Wie in den bereits vorangegangenen Tests sind dabei die Instanzen der Google Cloud Plattform am höchsten, wobei sich C5 an ∼160mb anschmiegt, wobei C4 weiterhin stetig wächst. Die Bluemix Instanzen, mit Ausnahme von C1, nähern sich asymptotisch an ∼135mb und beide Azure Instanzen an ∼95mb RAM. C1 sürzt in diesem Test dabei bereits im Schnitt nach ca 9 Sekunden ab. Das Latenzverhalten ähnelt dem bereits in Test 2 beschriebenen Verhalten. Herausstechend ist das hohe Flattern bei Googles C4 und die generell höheren Werte bei beiden Google Instanzen C4 und C5. Bei der CPU Auslastung ist ebenfalls ein Anstieg bei den Instanzen von Bluemix und Azure zu erkennen und eine recht konstantes Verhalten am oberen Rand der Instanzen der Google Cloud Plattform. Die größte Abweichung im Vergleich zu Test 2 ist bei den beiden Instanzen von Azure zu erkennen. Während bei Test 2 C6 noch bei ca 30 % und C7 bei 20 % Auslastung lagen, sind bei diesem Test nun Werte von 70% und 40% zu verzeichnen, was ungefähr einer Verdoppelung entspricht.

Tabelle 6.5: Übersicht der Testergebnisse für Test 3

	Bluemix C1	Bluemix C2	Bluemix C3	Google C4	Google C5	Azure C6	Azure C7
U/s	100						
Clients	1						
Message Size	350 byte						
Total Messages	10000						
Total Errors	8705	0	0	0	0	0	0
Mean Latency	515 ms	101 ms	333 ms	675 ms	234 ms	85 ms	66.26 ms
Mean Cpu Load	8.32 %	10.39 %	8.92 %	82.15 %	92.43 %	56.15 %	34.73 %
Mean Memory	66.44 MB	124.50 MB	122.44 MB	167.30 MB	146.33 MB	88.76 MB	84.51 MB

Test 4

Test 4 ist bis auf die Anzahl der Clients identisch mit Test 3. Bei diesem Test werden ebenfalls mit 100 U/s 10000 Nachrichten gesendet, wobei jedoch zehn konkurierende Verbindungen aufgebaut werden.

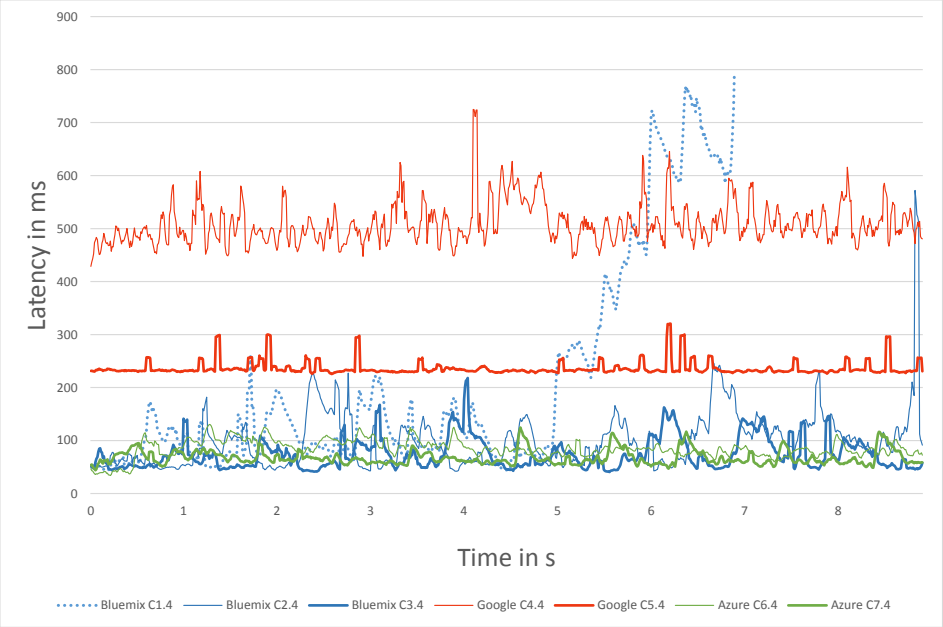


Abbildung 6.7: Latenz Verhalten bei Test 4

Das Memory Verhalten ist dabei annähernd identisch mit dem Verhalten aus Test 3, ebenfalls das CPU Verhalten und die Latenzen. In Abbildung 6.7 zu sehen sind die ersten 900 Latenzen der einzelnen Maschinen unter Test 4. Hierbei sieht man erneut das Versagen von C1, sowie das Flattern von C4 sowie die generell höheren Werte der Google Cloud Plattform. Jedoch steigt die Latenz der Bluemix Maschinen stetig bei diesem Test, was im Diagramm nicht mehr ersichtlich ist. Deshalb ist die Mean Latency von Bluemix C3 höher als Googles C4. Die kompletten Messdaten sind ebenfalls im Anhang dieser Arbeit zu finden. Im unteren Latenzbereich halten sich die Azure- und Bluemixinstanzen auf. Hier ist wiederum anzumerken, dass Azure eher konstante Latenzen hat und Bluemix leichtes Flattern und mehrere Ausreißer aufweist.

Tabelle 6.6: Übersicht der Testergebnisse für Test 4

	Bluemix C1	Bluemix C2	Bluemix C3	Google C4	Google C5	Azure C6	Azure C7
U/s	100						
Clients	10						
Message Size	350 byte						
Total Messages	10000						
Total Errors	8175	0	0	0	0	0	0
Mean Latency	241 ms	360 ms	617 ms	565 ms	235 ms	102 ms	78 ms
Mean Cpu Load	7.73 %	11.14 %	11.30 %	81.70 %	91.56 %	63.64 %	47.13 %
Mean Memory	65.65 MB	120.37 MB	123.30 MB	172.62 MB	150.79 MB	89.59 MB	84.38 MB

Test 5

Bei Test 5 wird das Verhalten der unterschiedlichen Konfigurationen bei einer ansteigenden Nachrichtenzahl betrachtet. Zu Beginn wird der Test mit 1 U/s, 1 Client und einer Message Size von 350 Byte gestartet.

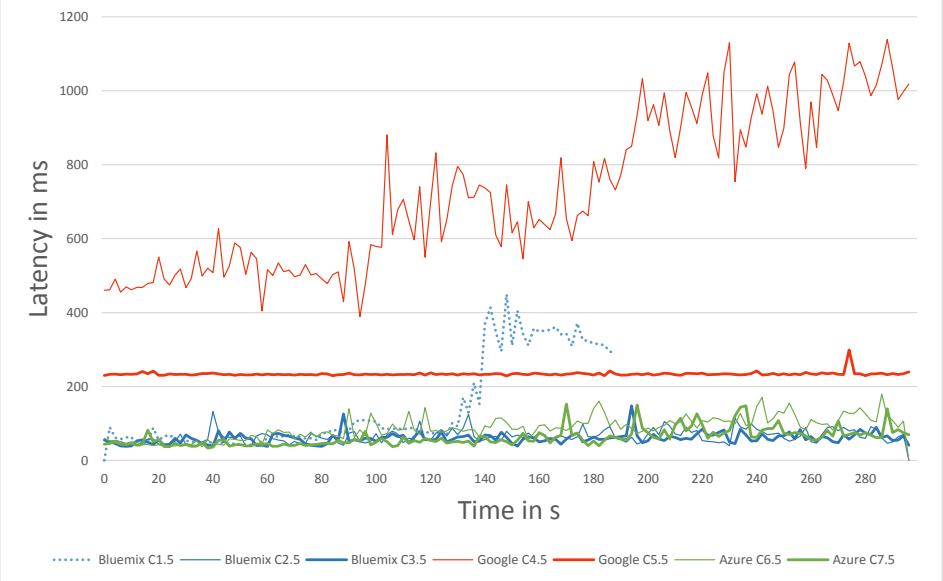


Abbildung 6.8: Latenz Verhalten für Test 5

Nach jeder abgeschlossenen Minute erhöhen sich die Units/s um +25 bis nach fünf Minuten der Test mit 100 U/s abgeschlossen ist. Abbildung 6.8 ist dabei das Latenzverhalten zu sehen, bei dem vor allem bei Googles C4 ein Anstieg über die Dauer des Testlaufs zu beobachten ist. C1 fängt dabei ab der zweiten Minute an bemerkbar Längere Latenzen aufzuweisen und stürzt dann ab der dritten Minute nach der Erhöhung auf 50 U/s komplett ab. Die Restlichen Instanzen weisen dagegen nur kleine Schwankungen auf. Für die CPU Auslastung ergibt sich bei Azure für beide Instanzen ein bemerkbarer Anstieg von ∼20 % auf ∼50 %. Google und Bluemix sind dagegen konstant, jedoch befinden sich die Google Instanzen konstant im >70% Bereich und Bluemix im <10% Bereich.

Tabelle 6.7: Übersicht der Testergebnisse für Test 5

	Bluemix C1	Bluemix C2	Bluemix C3	Google C4	Google C5	Azure C6	Azure C7
U/s	1 - 100						
Clients	1						
Message Size	350 byte						
Total Messages	15000						
Total Errors	13040	0	0	0	0	0	0
Mean Latency	145 ms	64.98 ms	59.95 ms	724.60 ms	233.71 ms	87.60 ms	61.88 ms
Mean Cpu Load	6.81 %	8.61 %	7.78%	76.97 %	93.81%	30.21 %	24.30 %
Mean Memory	63.61 MB	107.10 MB	107.17 MB	177.94 MB	130.54 MB	85.73 MB	81.78 MB

Maschinenkonfigurationen

Das Verhalten der einzelnen Instanzen ist für die unterschiedlichen CSP sehr unterschiedlich. Speziell die Minimalkonfiguration bei Bluemix (C1) sticht dabei heraus. Wie in Abbildung 6.9 zu sehen, sind bis auf Test 1 Abstürze dieser Maschinenkonfiguration zu verzeichnen.

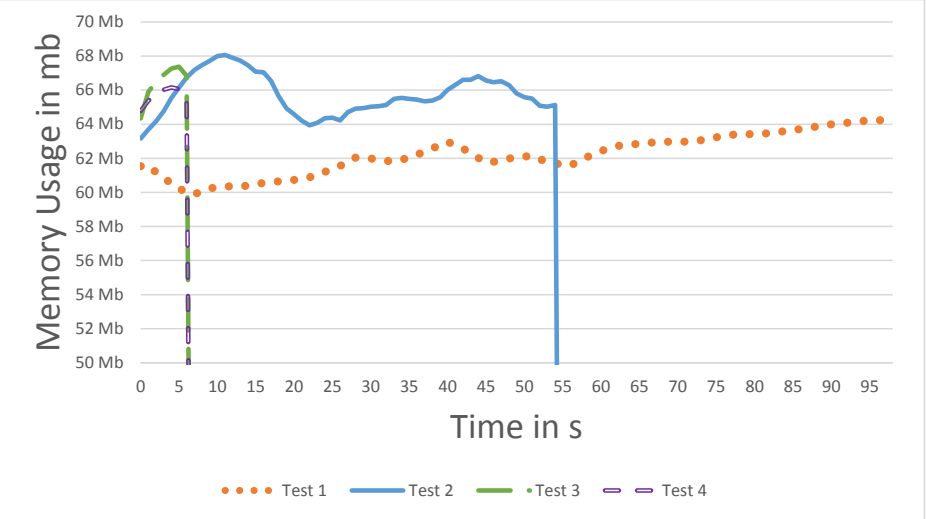


Abbildung 6.9: Memory Verhalten für C1

Das Verhalten der CPU Auslastung weist während der Durchführen der Tests für die Instanzen von Bluemix und Azure, wie zu erwarten war, von Test 1 bis 4 je eine größere Ansteigung der Auslastung als zum vorangegangenen Test auf. Dies ist hier Exemplarisch in Abbildung 6.12 für die Azure Instanz C7 zu sehen.

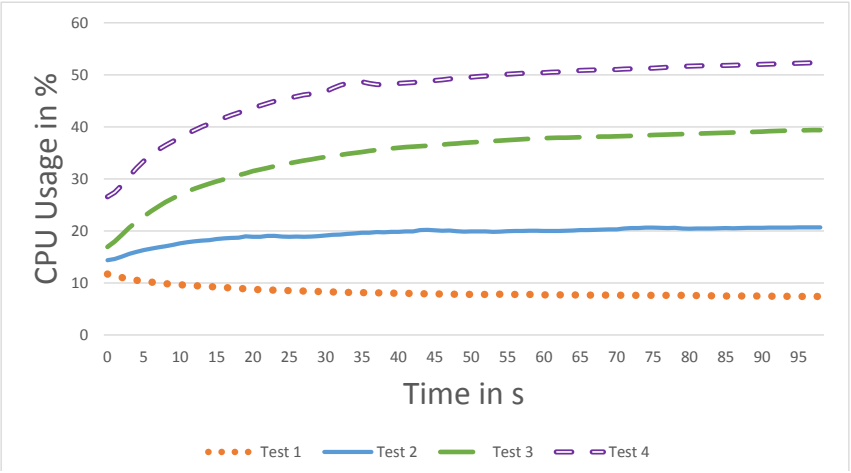


Abbildung 6.10: CPU Verhalten für C7

Ein im Vergleich zu Azure und Bluemix abweichendes Verhalten ist dabei bei den Google Instanzen C4 und C5 zu erkennen. In C4 kommt eine shared CPU zum Einsatz, wohingegen in C5 eine eigene vCPU eingesetzt wird. Das resultierende Verhalten ist wie in Abbildung 6.11 stark schwankend für C4 und relativ konstant für C5.

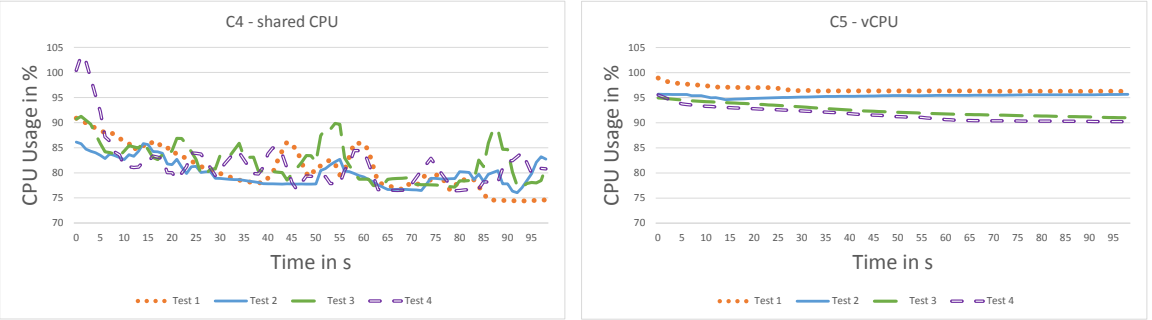


Abbildung 6.11: shared CPU vs vCPU in Googles Cloud Plattform

Beim Latenzverhalten unterscheiden sich Bluemix und Azure wiederum nur sehr gering. Jedoch ist bei den Instanzen von Google eine generell viel höhere Latenz zu erwarten. In Abbildung 6.12 ist das Verhalten für Azures C7 beim Durchführen aller Tests zu sehen. Hier ist wie erwartet die höhere Latenz für Test 2 deutlich zu sehen, da dort das Datenpaket 500kb statt den standardmäßigen 350byte groß ist.

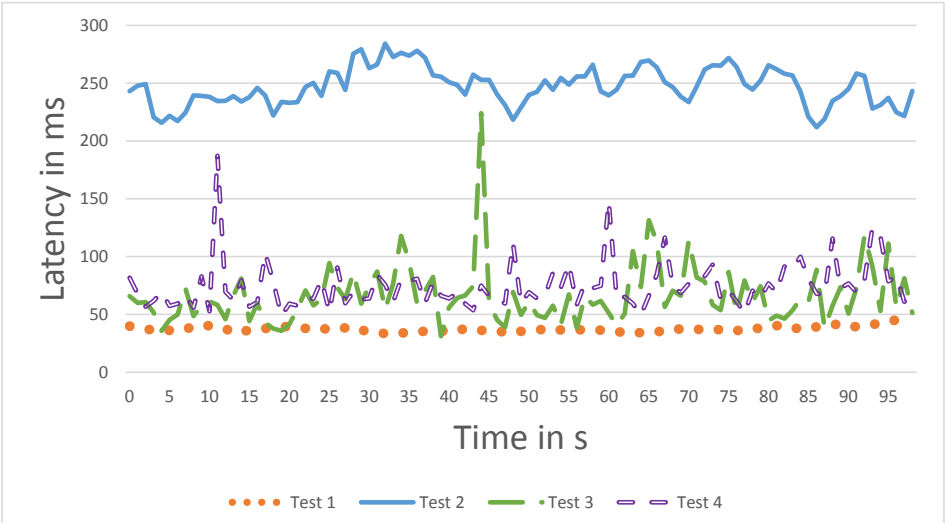


Abbildung 6.12: Latenz Verhalten für C7

Ergebnisanalyse

Da in diesem Kapitel die Auswertung einer großen Anzahl an Datensätzen stattfindet, jedoch die Diagramme teilweise nur einen Ausschnitt der Daten zeigen, wird in diesem Abschnitt die Verteilung der einzelnen Merkmale anhand von Boxplots visualisiert und untersucht. In diesem Kapitel werden exemplarisch die Metriken Latenz, Memory Usage und CPU Load während Test 4 betrachtet. Weitere Boxplots sind im Anhang 7 zu finden.

In dieser Darstellungsform lassen sich Median, erstes Quartil, zweites Quartil, Mittelwert und Max/Min Werte ablesen. Die Box entspricht dabei dem Bereich in dem die mittleren 50% der Daten liegen. Sie wird durch das obere und untere Quartil begrenzt. Die komplette länge der Box entspricht dem Interquartilsabstand IQR, was ein Maß der Streuung der Daten darstellt. Die durchgehende Trennlinie in dieser Box ist der Median. Über die Lage des Medians innerhalb der Box lassen sich Informationen über die Schiefe der Verteilung ableiten. Ist der Median im unteren Teil der Box, so ist die Verteilung rechtsschief, und umgekehrt. Berechnet wird die Schiefe mit der Karl Pearson definition:

$$S = \frac{Mittelwert - Median}{Standardabweichung} \tag{6.5}$$

Der Wertebereich von S ist das Intervall [-1,1]. Für symmetrische Verteilungen ist S = 0. Rechtsschiefe Verteilungen besitzen ein positives S und Linksschiefe Verteilungen ein Negatives S. Der nicht durchgezogene Balken innerhalb der Box ist der Mittelwert. Durch die Antennen (Whiskers) werden die außerhalb der Box liegenden Werte dargestellt. Die Begrenzung ist dabei der Maximal- und Minimalwert.

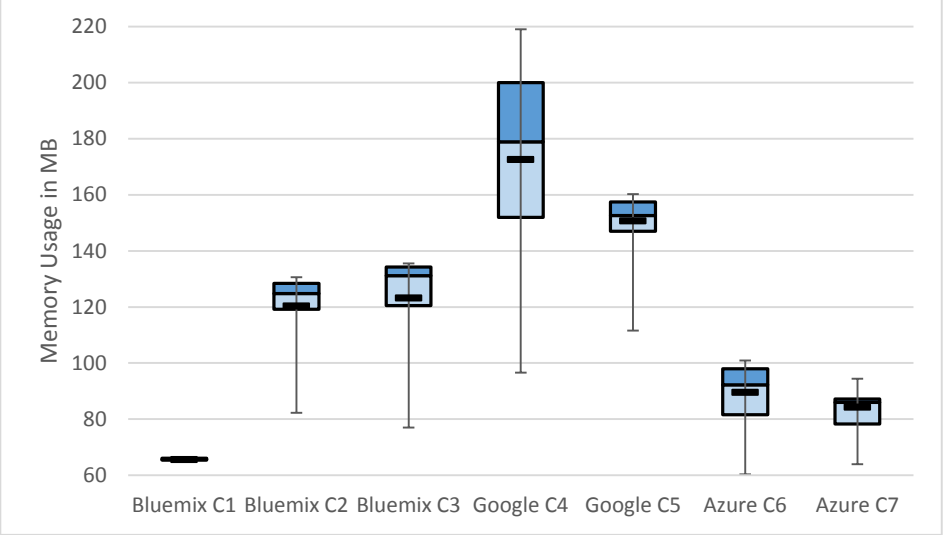


Abbildung 6.13: Boxplot Memory Behaviour Test 4

Die in Abbildung 6.13 dargestellten Boxplots zeigen die Verteilung des Memory Merkmals der verschiedenen Maschinenkonfigurationen während Test 4. Die Ergebnisse von C1 sind dabei zu vernachlässigen da diese Instanz bereits nach 8 Sekunden abgestürzt ist. Im Anhang ist das zu diesem Test zugehörige Diagramm zum Memory Verhalten zu finden. Dort ist der bei Googles C4 stärker steigende Memory verbraucht zu beobachten. Dies erklärt die in Abb. 6.13 zu sehende große Distanz der Whiskers bei Google C4. Es lässt sich außerdem eine generell leichte Linksschiefe Verteilung bei dieser Metrik erkennen. Der Median sowie der Mittelwert sind bei Google höher als die äquivalenten Konfigurationen bei Bluemix und Azure. Wie auch im Liniendiagramm im Anhang zu entnehmen, erfahren die Instanzen von Azure den geringsten Memory Anstieg während dieses Tests.

Interessanterweise lassen sich auch ohne das Betrachten der Beschriftungen der X-Achse Klassifikationen der unterschiedlichen Konfigurationen vornehmen.

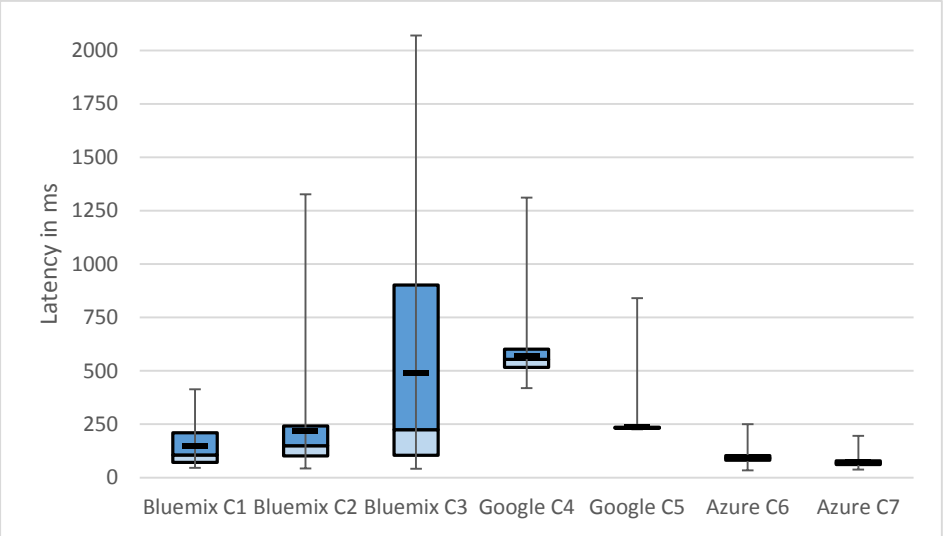


Abbildung 6.14: Boxplot Latency Behaviour Test 4

In Abb. 6.14 ist der Boxplots der Latenzzeiten für Test 4 dargestellt. Die Messwerte von C1 sind dabei ebenfalls nur bis zum Crash der Instanz nach 8 Sekunden Protokolliert. In Abbildung 6.7 ist dabei eine Darstellung der Latenzwerte der ersten 9 Sekunden zu sehen. Hieraus lässt bereits der höhere Median der Google Instanzen erkennen. Speziell die Konfiguration C4 mit der shared CPU weist dabei den größten Median und ein generell instabiles Verhalten der Messwerte auf. Jedoch erfolgt bei der Instanz C3 von Bluemix im laufe des Tests ein Anstieg der Latenzwerte was in dem Ausschnitt in Abb. 6.7 nicht zu erkennen ist. Dies ist jedoch im Boxplot in Abb. fig:BPLat deutlich zu sehen, da der Maximalwert bei 2070 ms liegt und die Box stark rechtsschief ist (S = 0.59). Erwähnenswert ist auch die bereits im Latenzdiagramm zu sehende Konstanz der Azure Instanzen. Der IQR ist dabei gering und die Maximal- und Minimalausschläge sind im Vergleich zu den anderen Instanzen wesentlich geringer.

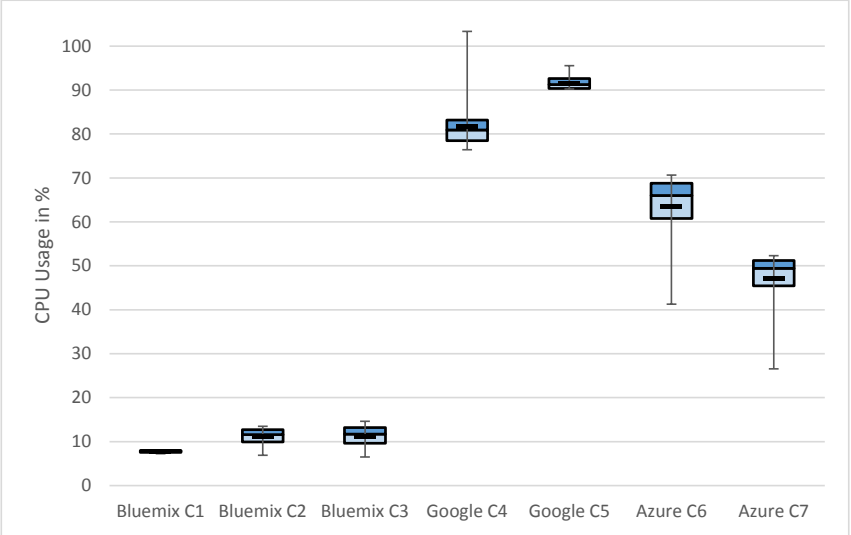


Abbildung 6.15: Boxplot CPU Behaviour Test 4

Ein weiteres betrachtetes Merkmal ist die CPU-Usage. Abb. 6.15 zeigt dabei die Verteilung

dieses Merkmals ebenfalls für Test 4. Im Vergleich zu den Latenz und Memory Verteilungen fällt dabei sofort die generell geringere IQR auf. Ohne die Legende der X-Achse zu betrachten lässt sich hier wieder sehr gut eine Klassifikation der Instanzen erkennen. So ist der Bereich in dem sich die CPU-Usage bei Bluemix bewegt im Intervall von 5 % bis 15 %. Google bewegt sich dabei mit beiden Instanzen am oberen Bereich der Skala. Wobei auch hier der unterschied der shared CPU bei C4 und der vCPU von C5 durch den Abstand der Whiskers erkennbar ist. Bei Azure ist als einziges der Sprung zur besseren Hardwarekonfiguration gut zu sehen.

Wie die Untersuchung gezeigt hat gibt es je nach Anbieter bemerkenswerte Unterschiede der einzelnen Metriken. Eine interessante Information ist dabei die Abrechnung dieser Metriken. So wird bei Bluemix z.B. das niedrige CPU Verhalten nicht abgerechnet, wohingegen bei Azure und Google für ausreichen CPU bezahlt werden muss.

Preise

Für die Preiskalkulation wurden die implementierten Cloud Dienste für das Stream und Batch Szenario sowie die Maschinen auf denen die Server Applikation gehostet wurde, abgerechnet. Generell kann man für jeden CSP sagen, dass die zur Verfügung gestellten Dienste den größten Anteil zur monatlichen Endabrechnung beigetragen haben. Speziell die Dienste, die nach verbrauchten Megabytes abregechnet werden, fallen sehr tief in das Gewicht. So verlangt der IoT Realtime Insights Dienst von Bluemix zum Beispiel 0.75€/MB. Die vereinzelt berechneten Preise pro verbrauchten Mengeneinheiten bei Google fallen im Vergleich zu Bluemix wesentliches geringer aus. So wird beim Speicherdienst BigQuerry ein Satz von 0.009€/200mb berechnet. Azure setzt bei der Preiskalkulation vieler Services auf das Bundling, jedoch werden viele benutzten Resourcen minutengenau abgerechnet, sodass angegebene monatliche Kosten auf den Erfahrungen mit anderen Nutzern beruhen.

Für die Preiskalkulation wurden die aktuellen[1] Preise der unterschiedlichen CSP herangezogen. Da die unterschiedlichen Cloud Dienste meist unterschiedliche Preisstufen haben, folgt im folgenden abschnitt eine Übersicht aller möglichen Stufen:

[1] 6.5.2016

Tabelle 6.8: Pricing Informationen

Service Name	Stufe/Spezifikation	Preis
Bluemix		
Nodejs SDK:	375gb Gratis pro Monat	0.0526€/GBh
Auto Scaling:	-	Gratis
Monitoring/Analytics:	-	Gratis
DashDB:	Entry - 1GB frei 20 GB maximum	37,61 €/Month
	Enterprise 64.1 - Empfohlen bis 1TB	1.129,00 €/Instance
	Enterprise 256.4 - Empfohlen bis 4TB	4.536,00 €/Instance
	Enterprise 256.12 - Empfohlen bis 12TB	7.112,00 €/Instance
IoT Plattform:	Max. 20 Devices 100mb	Gratis
	Standard - Unliminted Devices, 100mb Frei	0.0075€/MB
IoT RealtimeInsights:	Lite - 25mb Frei	Gratis
	Bronze - 100mb Frei	75.20 €/Inst.+0.75€/MB
	Silver - 500mb Frei	376 €/Inst.+0.57€/MB
	Gold - 2500mb Frei	1128 €/Inst.+charges/MB
Azure		
VM:	750mb RAM, 0,25 Kern	11.29€/Monat
	1,75gb RAM, 1 Kern	32€/Monat
IoTHub:	bis 240.000 Pakete	Gratis
	bis 12.000.000 Pakete	42.17€/Monat
	bis 180.000.000 Pakete	421.65€/Monat
MySslDB:	bis 20mb	Gratis
	bis 250mb	3.50€/Monat
	bis 1gb	9.99€/Monat
	bis 2gb	49.99€/Monat
	bis 10gb	99.99€/Monat
PowerBI:	bis 1gb	Gratis
	bis 10gb	8.40€/Monat
Streaming Analytics:	-	0.0008€/GB
	pro Streaming Einheit	18.80€/Einheit
Google		
Compute Engine	Micro - 600mb RAM, shared CPU	5.08€/Monat
	Standard - 1.75gb RAM, 1vCPU	20.79€/Monat
DataLab	3.75gb RAM, 1vCPU	23.87€/Monat
BigQuerry	Streaming Data	0.009€/200mb
	Data Store	0.018€/GB
	Query - 1TB Frei	4.38€/TB

Für die Preisberechnung werden die Kosten der jeweiligen Services addiert womit sich folgende Formeln ergeben:

$$Kosten_{Bluemix} = Kosten_{SDK} + Kosten_{DashDB} + Kosten_{IoT-P} + Kosten_{IoT-RI} \quad (6.6)$$

$$Kosten_{Google} = Kosten_{BigQuerry} + Kosten_{Datalab} + Kosten_{ComputeEngine} \quad (6.7)$$

$$\begin{aligned} Kosten_{Azure} = {} & Kosten_{VM} + Kosten_{IoT-Hub} + Kosten_{MySqlDB} \\ & + Kosten_{PowerBI} + Kosten_{StreamingAnalytics} + Kosten_{OutboundData} \end{aligned} \quad (6.8)$$

Im folgenden wird eine Exemplarische Preisberechnung für die jeweilige Szenario-Fit-Konfiguration der einzelnen CSP vorgenommen. Dabei wird von einer Anzahl an Nachrichten pro Monat von 259200000 und einer Nachrichtengröße von 350 byte ausgegangen,

was einem Durchsatz von 100 U/s entspricht. Dies ergibt einem Monatlichen Datentransfer von ∼90 GB. Das sind die, in dem für dieser Arbeit definierten Test 3, generierten Werte mit einer Maschinenlaufzeit von 30 Tagen á 24 Stunden.

Tabelle 6.9: Exemplarische Preisberechnung

Plattform	**Service Name**	**Konfiguration**	**Estimated Costs**
Bluemix	Nodejs SDK	(1417.5GBh - 375GBh)*0.0526€	54.84€
	DashDB	Entry	37.61€
	IoT Plattform	Standard (89900MB*0.0075€)	674.25€
	IoT Realtime Insights	Silver (89500MB*0.57€)	376€ 51015€
		Kosten pro Monat:	**52157.70€**
Azure	VM	1.75GB RAM, 1 Kern	32€
	IoT Hub	bis 12.000.000 Pakete	421.65€
	MySQLDB	9*10GB	899.91€
	PowerBI	9*10GB	75.60€
	Streaming Analytics	1 Einheit 90GB*0.0008€	18.80€ 0.07€
	Outbound Data	90GB*0.07€	6.30€
		Kosten pro Monat:	**1454.33€**
Google	BigQuery	(90000/2)*0.009€ Datastore 90GB*0.018€	405€ 1.62€
	Compute Engine	Standard	20.79€
	Datalab(Beta)	Billed as Compute Engine	23.87€
		Kosten pro Monat:	**451.28€**

In Tabelle 6.16 sind die monatlichen Kosten der unterschiedlichen Maschinenkonfigurationen mit verschiedenen Anzahlen an gesendeten Nachrichten pro Monat zu sehen. Bei nur 100 Einheiten pro Monat resultiert ein monatlicher Datenfluss von 3,5 kb. Bei dieser Anzahl an Nachrichten setzt sich der Großteil des Preises mit den Kosten der Virtuellen Maschinen für die Server Applikation zusammen. Bei den Instanzen von Bluemix ist ab einer monatlichen Nachrichtenzahl >100000 eine Ansteigung am Preis zu erkennen, welche auch bei weiterer Erhöhung der Nachrichtenzahl rasant in die Höhe steigt. So kommt man bei 3.5 GB pro Monat schon in den vierstelligen bereich und im Terrabytebereich werden bei Bluemix schon Preise in Millionenhöhe berechnet. Azure ist bei kleinen Nachrichtenzahlen pro Monat bei jeder Konfiguration leicht billiger als Bluemix. In den vierstellige Euro Bereich gelangt man erst ab einem monatlichen Datenfluss von 350 GB. Selbst bei 35 tb ist ein großer Unterschied zu Bluemix zu erkennen. Google sticht mit seinen niedrigen Preisen bei den in dieser Arbeit betrachteten CSP sichtlich heraus. Jedoch glänz Google dafür auch nicht mit der Vielzahl an verfügbaren Cloud Diensten wie Azure oder Bluemix. Die bei Azure und Bluemix teuer zu Buche schlagenden Cloud Dienste sind bei Google schlichtweg nicht vorhanden. Dort werden eher abstraktere Cloud Services angeboten, welche dann meist mit eigenem Programmieraufwand die Funktionen der bei Azure und Bluemix *Out-of-the-box* funktionierenden Dienste ausführen.

In Abbildung 6.16 ist eine grafische Darstellung des Preisverhaltens je einer ausgewählten Instanz pro CSP zu sehen. Dabei für jeden CSP nach einer gewissen Anzahl an Nachrichten pro Monate eine Lineare Skalierung des Preises zu beobachten. Diese Lineare Skalierung beginnt bei Bluemix bereits nach 10000 Nachrichten pro Monat, wobei Azure und Google noch einen Konstanten Preis bis 100000 bzw 10000000 Nachrichten aufweisen.

Tabelle 6.10: CSP Preise

MessageSize/Month	Message/Month	Bluemix C1	Bluemix C2	Bluemix C3	Google C4	Google C5	Azure C6	Azure C7
3,5 kb	10^2	37,61 €	40,60 €	84,11 €	28.98 €	44,69 €	30,16 €	50,87 €
350 kb	10^3	37,61 €	40,60 €	84,11 €	28.98 €	44,69 €	30,16 €	50,87 €
3,5mb	10^4	37,61 €	40,60 €	84,11 €	28.98 €	44,69 €	30,16 €	50,87 €
35mb	10^5	112,81 €	115,80 €	159,31 €	28.98 €	44,69 €	33,66 €	54,37 €
350mb	10^6	302,19 €	305,18 €	348,69 €	28.98 €	44,69 €	82,32 €	103,03 €
3,5gb	10^7	2.377,11 €	2.380,10 €	2.423,61 €	29.18 €	44,69 €	130,93 €	151,64 €
35gb	10^8	20.771,75 €	20.774,74 €	20.818,25 €	31.11 €	46,82 €	508,30 €	529,01 €
350gb	10^9	202.684,25 €	202.687,24 €	202.730,75 €	50.55 €	66,26 €	4.270,19 €	4.290,90 €
3,5tb	10^{10}	2.026.104,25 €	2.026.107,24 €	2.026.150,75 €	262,47 €	278,18 €	38.636,06 €	38.656,77 €
35tb	10^{11}	20.219.930,25 €	20.219.933,24 €	20.219.976,75 €	833.87 €	833,87 €	380.064,56 €	380.085,27 €

Abbildung 6.16: Preisverhalten

Cloud Provider Selection Problem

Mit den aus der Evaluation gewonnen Informationen wird nun das abstrakte Modell zum *Cloud Provider Selection Problem* konkretisiert. Die Menge C ist die Menge der betrachteten CSP und den jeweiligen Maschinenkonfigurationen und ist definiert durch

$$C = \{C1, C2, C3, C4, C5, C6, C7\} \tag{6.9}$$

Die Definitionen der einzelnen Maschinenkonfigurationen C1 - C7 sind der Abbildung 6.1 zu entnehmen. Die Menge K enthält die in der Evaluation gemessenen Key Performance Indikatoren

$$K = \{QoS, Throughput, Latency, CpuUtil, MemoryUtil, Costsp.Month\} \tag{6.10}$$

Für die Entscheidungsvektoren werden die Evaluationsergebnisse von Test 4 benutzt und eine Message Anzahl von 1.000.000 pro Monat angenommen, da dieses Verhalten einem möglichen Verhalten einer IoT Industrieanlage entspricht. Somit entstehen folgende Entscheidungsvektoren:

$$d_1 = \begin{pmatrix} 1.9 \\ 1.8 \\ 7 \\ 10 \\ 10 \\ 1.5 \end{pmatrix}^T \quad d_2 = \begin{pmatrix} 10 \\ 10 \\ 4.8 \\ 9.6 \\ 4.9 \\ 1.4 \end{pmatrix}^T \quad d_3 = \begin{pmatrix} 10 \\ 10 \\ 0 \\ 9.6 \\ 4.6 \\ 0 \end{pmatrix}^T \quad d_4 = \begin{pmatrix} 10 \\ 10 \\ 1 \\ 1.2 \\ 0 \\ 10 \end{pmatrix}^T \quad d_5 = \begin{pmatrix} 10 \\ 10 \\ 7.1 \\ 0 \\ 2 \\ 9.6 \end{pmatrix}^T \quad d_6 = \begin{pmatrix} 10 \\ 10 \\ 9.6 \\ 3.3 \\ 7.8 \\ 8.5 \end{pmatrix}^T \quad d_7 = \begin{pmatrix} 10 \\ 10 \\ 10 \\ 5.3 \\ 8.2 \\ 7.8 \end{pmatrix}^T \tag{6.11}$$

und die Entscheidungsmatrix A mit den normierten Leistungsvektoren:

$$A = \begin{pmatrix} 0.118 & 0.112 & 0.436 & 0.622 & 0.622 & 0.093 \\ 0.541 & 0.541 & 0.260 & 0.520 & 0.265 & 0.076 \\ 0.565 & 0.565 & 0 & 0.542 & 0.260 & 0 \\ 0.575 & 0.575 & 0.058 & 0.069 & 0 & 0.575 \\ 0.537 & 0.537 & 0.381 & 0 & 0.107 & 0.516 \\ 0.479 & 0.479 & 0.460 & 0.158 & 0.373 & 0.407 \\ 0.468 & 0.468 & 0.468 & 0.248 & 0.384 & 0.365 \end{pmatrix} \tag{6.12}$$

Zur beispielhaften Lösung des *Cloud Provider Selection Problem* wird ein Anforderungsvektor R wie folgt definiert:

$$R = \begin{pmatrix} AnforderungQualityOfService \\ AnforderungThroughput \\ AnforderungLatency \\ AnforderungCpuUtil. \\ AnforderungMemoryUtil. \\ AnforderungCostp.Month \end{pmatrix}^T = \begin{pmatrix} 10 \\ 7 \\ 8 \\ 4 \\ 6 \\ 7 \end{pmatrix}^T \qquad \frac{1}{|R|} = \begin{pmatrix} 0.564 \\ 0.395 \\ 0.451 \\ 0.226 \\ 0.339 \\ 0.395 \end{pmatrix}^T \tag{6.13}$$

Mit der in [uRHH11] definierten *Exponential Weighted Difference (EWD)* Methode folgt der Ergebnissvektor E:

$$E = \begin{pmatrix} 6.68 \\ 6.3 \\ 6.71 \\ 6.72 \\ 6.38 \\ 6.02 \\ 5.98 \end{pmatrix} \tag{6.14}$$

Somit ist der niedrigste Wert in der Ergebnissmatrix bei e_7 zu verzeichnen, da die Maschinenkonfiguration C7 den Mindestanforderungen des Anforderungsvektos am ehesten entspricht.

In Abbildung 6.17 zu sehen sind die Leistungsvektoren der einzelnen CSP in Netzdiagrammen visualisiert. Sprich, die gemessenen KPIs auf der 10er Skala für jeden CSP. Wobei 10 der beste gemessene Wert und 0 der schlechteste gemessene Wert zum jeweiligen KPI ist. Es wurden dabei speziell für die Maschinenkonfiguration C1 nur die gemessenen Werte bis zum Absturz der Maschine betrachtet.

Abbildung 6.17: Netzdiagramme der CSP

6.6 Ergebnisdiskussion

Wie die Untersuchungen in dieser Arbeit gezeigt haben, gibt es mit vergleichbaren Konfigurationen und Szenarien unterschiede, was das Verhalten, die angebotenen Service und vor allem die Abrechnung angeht. Wie im diesem Kapitel ausführlich behandelt wurde, ist bei einer vergleichbaren Konfiguration der CSP ein unterschiedliches Verhalten zu beobachten. In dieser Arbeit wurde gezeigt wie das Verhalten der gemessenen KPIs auf den verschiedenen Plattformen variiert.

Speziell Google generell höhere Latenzzeiten auf. Speziell mit der shared CPU Konfiguration ist zudem eine größere Varianz dieser Werte zu verzeichnen. Auch die Konfigurationsmöglichkeiten weichen stark voneinander ab. So lässt sich bei Bluemix die CPU Konfiguration nicht anpassen. Jedoch wird bei Bluemix das CPU Verhalten nicht abgerechnet. Eine interessante Erkenntnis dabei ist, dass die nicht abgerechnete Bluemix CPU in dieser Arbeit konstant unter 25% Auslastung liegt, wobei die abgerechnete Azure CPU während den Lasttests auf bis zu 80% ansteigt und Google sogar Konstant über 80% ist.

Ein weiterer Gesichtspunkt der Cloud-Plattformen sind die angebotenen Cloud-Dienste. So war es Ziel dieser Arbeit ein einheitliches Szenario auf den verschiedenen CSP zu Implementieren. Dafür wurden die zur Verfügung stehenden Cloud Dienste verwendet. Der Versuch vergleichbare Cloud-Dienste zu finden und zu verwenden war dabei nur bedingt möglich. So sticht speziell die Google Cloud Plattform aus der Menge heraus, da dort nur eine geringe Menge an Cloud-Diensten zur Verfügung steht, welche meist abstrakter als die von Azure und Bluemix angebotenen Dienste sind. Jedoch war eine Implementierung des geplanten Stream- und Batch-Processing damit auf allen Plattformen möglich.

Im Fokus der Überlegungen standen auch die unterschiedlichen Preisarchitekturen. So wurde anhand den in dieser Arbeit implementierten Szenarien die Monatlich generierten Kosten der Plattformen und deren Diensten mit variabler Nachrichtenzahl untersucht. Betrachtet man die Einstiegspreise, so befinden sich alle Anbieter im mittleren zweistelligen Bereich. Die Preissteigung der verschiedenen Anbieter verhält sich jedoch stark unterschiedlich mit steigender Nachrichtenzahl. Bei einer Monatlichen Übertragung im Terrabyte Bereich fallen bei Google Kosten im dreistelligen Bereich, bei Azure im sechsstelligen Bereich und bei Bluemix sogar im zweistelligen Millionenbereich an. Die größten Kosten werden dabei beim Verarbeiten von Streaming Daten verursacht. Durch den Fokus dieser Arbeit auf das Internet der Dinge ist dies ein nicht zu vernachlässigbarer Faktor beim Cloud Computing.

6.7 Zusammenfassung

In diesem Kapitel wurde zuerst das Setting für die Evaluation beschrieben, indem auf die unterschiedlichen Tests und die untersuchten Maschinenkonfigurationen der unterschiedlichen CSP genauer beschrieben wurden. Weiter wurde das *Cloud Provider Selection Problem* als abstraktes mathematisches Modell aufgefasst und mit den gemessenen Daten der Evaluation für ein Beispielszenario durchgerechnet. Nach der Beschreibung der Evaluationsdurchführung folgte die Diskussion der Evaluationsergebnisse, welche ebenfalls mittels verschiedener Graphen visualisiert dargestellt wurden. Durch die Vielzahl an Evaluationsdaten wurden nur aussagekräftige oder beispielhafte Graphen in dieses Kapitel übernommen und alle weiteren im Anhang dieser Arbeit aufgeführt.

7. Fazit und weiterer Forschungsbedarf

Zum Abschluss dieser Thesis wird in diesem Kapitel eine Zusammenfassung über die in dieser Arbeit erbachten Leistungen gegeben, sowie eine Diskussion und ein Resümee bezüglich des Zielerreichungsgrades der Forschungsfrage geführt. Weiter wird ein Ausblick über mögliche weitere Forschungsarbeit an dem in dieser Arbeit betrachteten Problem gegeben.

In dieser Arbeit wurde zu Beginn eine Übersicht der zum Thema Cloud Computing nötigen Grundlagen, sowie der bereits bestehenden sachverwandten Arbeiten gegeben. Dabei wurden wesentliche Begriffe eingeführt sowie wichtige Definition zu Cloud Computing Charakteristika, Deployment Modelle und Serice Modelle betrachtet. Bei den sachverwandten Arbeiten wurde eine Abgrenzung zu dieser Arbeit diskutiert. Hierbei wurde der Fokus dieser Arbeit auf zugrundeliegende IoT-Szenarien gesetzt, welche mittels bereits existierender Cloud Dienste der einzelnen CSP implementiert werden.

Weiter ist für diese Arbeit ein Evaluationsframework entstanden, welches aus einer Anwendung zum Generieren von Testdaten und Visualisieren der gemessenen Key Performance Indikatoren dient, sowie eine Server Applikation, welche dazu dient, die generierten Daten zu empfangen und in der Cloud mit den verschiedenen Cloud Diensten zu verarbeiten.

Als Architektur dieser Softwarekomponenten wurde die Lambda-Architektur gewählt, welche zusammen mit dem Softwareentwurf in Kapitel 4 dokumentiert wird. Bestandteil dieser Arbeit ist ebenfalls die Dokumentation zur Implementierung dieser Komponenten. Dabei wurde speziell Bezug auf die benutzten Libraries genommen, sowie die für diese Arbeit benutzten Cloud Dienste genauer betrachtet.

Bei der eigentlichen Evaluation wurde das Evaluationsframework auf drei ausgewählten Cloud Service Providern ausgeführt und die Messdaten protokolliert. Die für diese Arbeit betrachteten CSP sind die Google Cloud Plattform, IBM Bluemix und Microsoft Azure. Mit den Evaluationsergebnissen wurde ein Decision Support System erstellt, welches ein, mit den Evaluationsergebnissen gefülltes, Modell zur Entscheidungsuntersützung von Cloud Service Providern bietet und mit Konfigurierbaren Anforderungen die unterschiedlichen Indikatoren der CSP visualisiert.

Das *Cloud Provider Selection Problem* wurde in ein abstraktes mathematisches Modell transformiert und mit den Evaluationsergebnissen auf den in dieser Arbeit betrachteten Anwendungsfall konkretisiert. Desweiteren wurden die Evaluationsergebnisse anhand verschiedener Graphen visualisiert und Erkenntnisse zu den einzelnen CSP daraus abgeleitet.

Die gewonnenen Ergebnisse dieser Arbeit geben einen Einblick über das Verhalten der betrachteten CSP und zeigen deren Stärken und Schwächen auf. In diesem Abschnitt folgt eine Diskussion der in dieser Arbeit gewonnenen Forschungsergebnisse.

Ziel dieser Arbeit war es einen Ansatz zum Vergleich verschiedener Cloud Anbieter, auf Basis von IoT-Szenarien, zu Entwickeln und diesen an den Plattformen von Google, Microsoft und IBM zu testen. Dieses Forschungsziel konnte mit einigen Einschränkungen bezüglich der Key Performance Indikatoren erreicht werden. Die bereits zu Beginn dieser Arbeit aufgestellte Behauptung der Intransparenz der CSP konnte in dieser Arbeit bestätigt werden. In der Literatur ist eine Vielzahl von KPIs zu finden, welche jedoch in diesem Ausmaß auf keine der Untersuchten Plattformen anwendbar waren. Preisgegebene Informationen sind schwer zu finden und sind sehr Komplex Strukturiert. So ergeben sich vor allem für die Preisarchitekturen komplexe strukturierte Modelle und Angaben.

Die dargestellten Ergebnisse rechtfertigen die zu Beginn dieser Arbeit getroffene Aussage zur Intransparenz der betrachteten Cloud Plattformen. So konnte nur eine geringe Menge der in der Literatur gefundenen Key Performence Indikatoren gemessen werden, und diese oftmals auch nur mit eigenem Programmieraufwand. Die von den CSP freiwillig preisgegebenen Informationen waren oft schwer zu erreichen oder schlecht dokumentiert. Wie die Untersuchung der unterschiedlichen CSP gezeigt hat, sind teils enorme Unterschiede der angebotenen Leistungen und den dafür berechneten Preisen zu verzeichnen. So sind vor allem Latenzen und die CPU Konfigurationen Gesichtspunkte, welche sich von Anbieter zu Anbieter stark unterscheiden. Ebenfalls die angebotenen Cloud Dienste und die dafür berechneten Preise variieren sehr stark.

Zielsetzung der vorliegenden Arbeit war, mit den Untersuchungen und den erstellten Softwarekomponenten eine Hilfestellung zum *Cloud Provider Selection Problem* zu geben. Dies wurde auf den betrachteten CSP erfolgreich durchgeführt, jedoch entstehen in diesem schnelllebigen Markt häufig neue CSP oder Neustrukurierungen der Preismodelle, sodass die Ergebnisse dieser Arbeit möglicherweise im Laufe der nächsten Jahre nicht mehr als aussagekräftig angesehen werden können. Durch die Modularisierung und die leichte Anpassbarkeit der Softwarekomponenten lassen sich jedoch auf eine einfache Art und Weise neue Anbieter hinzufügen und betrachten.

Außerdem bestätigen die dargestellten Ergebnisse die Aussage, dass für das Internet der Dinge im Bereich Cloud Computing weitere Fortschritte gemacht werden müssen. Vor allem das Preisverhalten macht die Auslagerung in die Cloud für viele Anwendungsfälle noch nicht Interessant. Jedoch ist der in dieser Arbeit entstandene Ansatz zum Vergleich verschiedener Anbieter und die Ergebnisse der Evaluation ein weiterer Schritt, um die Cloud Plattformen und das Internet der Dinge einander näher zu bringen.

Als weitere Arbeiten an diesem Projekt sollte eine Ausweitung der unterstützten CSP und Cloud Diensten durchgeführt werden. Desweiteren gilt es weitere Key Performance Indikatoren zu finden, welche bei aktuellen Anbietern auslesbar sind und einen wichtigen Faktor zum *Cloud Provider Selection Problem* liefern. Außerdem sollten weitere Verbesserungen der entstandenen Software bezüglich Performance und Stabilität durchgeführt werden und eine verbesserte Logging-Funktionalität implementiert werden.

Literaturverzeichnis

[Ant12] A. Antoniou, "Performance Evaluation of Cloud Infrastructure using Complex Workloads," p. 79, 2012.

[BG10] T. Brunzel and D. D. Giacomo, "Cloud Computing Evaluation: how it differs to traditional IT outsourcing," no. May, p. 70, 2010. [Online]. Available: http://hj.diva-portal.org/smash/record.jsf?pid=diva2:328402

[BLC+11] X. Bai, M. Li, B. Chen, W. T. Tsai, and J. Gao, "Cloud testing tools," *Proceedings - 6th IEEE International Symposium on Service-Oriented System Engineering, SOSE 2011*, no. Sose, pp. 1–12, 2011.

[BYV08] R. Buyya, C. S. Yeo, and S. Venugopal, "Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities," in *2008 10th IEEE International Conference on High Performance Computing and Communications*. IEEE, sep 2008, pp. 5–13. [Online]. Available: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=4637675

[CST+10] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," *Proceedings of the 1st ACM symposium on Cloud computing - SoCC '10*, pp. 143–154, 2010.

[FM] S. Frey and M. Maier, "SLA-Richtliniendokument für Cloud Dienstleistungen Autonomic SLA Management as a Service (ASLAMaaS) Autonomic SLA Management as a Service."

[FRL13] S. Frey, C. Reich, and C. Lüthje, "Key Performance Indicators for Cloud Computing SLAs," *EMERGING 2013, The Fifth International ...*, no. c, pp. 60–64, 2013. [Online]. Available: http://www.thinkmind.org/index.php?view=article{&}articleid=emerging{_}2013{_}3{_}30{_}40082

[GB10] A. Goscinski and M. Brock, "Toward dynamic and attribute based publication, discovery and selection for cloud computing," *Future Generation Computer Systems*, vol. 26, no. 7, pp. 947–970, 2010. [Online]. Available: http://dx.doi.org/10.1016/j.future.2010.03.009

[GVB12] S. K. Garg, S. Versteeg, and R. Buyya, "A framework for ranking of cloud computing services," *Future Generation Computer Systems*, vol. 29, no. 4, pp. 1012–1023, 2012. [Online]. Available: http://dx.doi.org/10.1016/j.future.2012.06.006

[HK11] C. N. Höfer and G. Karagiannis, "Cloud computing services: taxonomy and comparison," *Journal of Internet Services and Applications*, vol. 2, no. 2, pp. 81–94, jun 2011. [Online]. Available: http://www.springerlink.com/index/10.1007/s13174-011-0027-x

[Jac14] M. Jacoby, "Andrei - Enabling Domain-Specific Rule-Based Automation With Semantic Stream Technology," 2014.

[JJj10] X. Jing and Z. Jian-jun, "A Brief Survey on the Security Model of Cloud Computing," in *2010 Ninth International Symposium on Distributed Computing and Applications to Business, Engineering and Science*. IEEE, aug 2010, pp. 475–478. [Online]. Available: http://www.computer.org/csdl/proceedings/dcabes/2010/4110/00/4110a475-abs.html

[KKDD] D. Kulessa, A. Kurz, V. Der, and V. E. Der, "Redemittel Wissenschaftliches Schreiben," pp. 8–11.

[KNtY12] E. Kuiper, S. Nadjm-tehrani, and D. Yuan, "A framework for performance analysis of geographic delay-tolerant routing," pp. 1–17, 2012.

[KR13] N. Khanghahi and R. Ravanmehr, "Cloud Computing Performance Evaluation : Issues and Challenges," *... Journal on Cloud Computing: ...*, vol. 3, no. 5, pp. 29–41, 2013. [Online]. Available: http://search.ebscohost.com/login.aspx?direct=true{&}profile=ehost{&}scope=site{&}authtype=crawler{&}jrnl=22315853{&}AN=91987036{&}h=jDJh34FN0rPQ4NMSlsWOXzpMW3rzi2/05IAV6hKNcYlo11mUeTAyjZ8at3zmV+XYrIRIbvmkHxAj3fsyHXR3KA==={&}crl=c

[KR14] K. Kaur and A. K. Rai, "A Comparative Analysis : Grid , Cluster and Cloud Computing," vol. 3, no. 3, pp. 5730–5734, 2014.

[Lan11] C. Langguth, "Cloud Computing : Meet the Players . Performance Analysis of Cloud Providers," 2011.

[LY10] A. Li and X. Yang, "CloudCmp : Shopping for a Cloud Made Easy," *Framework*, p. 5, 2010. [Online]. Available: http://portal.acm.org/citation.cfm?id=1863108

[LYKZ10] A. Li, X. Yang, S. Kandula, and M. Zhang, "CloudCmp: Comparing Public Cloud Providers," *Proceedings of the 10th annual conference on Internet measurement - IMC '10*, p. 1, 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1879143$\delimiter"026E30F$nhttp://portal.acm.org/citation.cfm?doid=1879141.1879143

[MVM11] Madurima, Vandana, and Madhulika, "Windows Azure Platform : an Era for Cloud Computing," *International Journal of Computer Science and Information Technologies*, vol. 2, no. 2, pp. 621–623, 2011.

[NIS11] NIST, "The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology," *Nist Special Publication*, vol. 145, p. 7, 2011. [Online]. Available: http://www.mendeley.com/research/the-nist-definition-about-cloud-computing/

[Pel10] C. Pelletingeas, "Performance Evaluation of Virtualization with Cloud Computing Submitted in partial fulfilment of the requirements of Edinburgh Napier University for the Degree of MSc Advanced Networking School of Computing December 2010," no. December, p. 91, 2010.

[PZL+09] J. Peng, X. Zhang, Z. Lei, B. Zhang, W. Zhang, and Q. Li, "Comparison of Several Cloud Computing Platforms," in *2009 Second International Symposium on Information Science and Engineering*. IEEE, dec 2009, pp. 23–27. [Online]. Available: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=5447227

[Rep13] J. Repschläger, "Entscheidungsfindung im Cloud Computingâ€"Konzeption und Analyse eines Modells zur Anbieterauswahl," *Opus4.Kobv.De*, 2013. [Online]. Available: http://opus4.kobv.de/opus4-tuberlin/frontdoor/deliver/index/docId/3898/file/repschlaeger{_}jonas.pdf

[SK11] N. Sadashiv and S. M. D. Kumar, "Cluster, grid and cloud computing: A detailed comparison," in *2011 6th International Conference on Computer Science & Education (ICCSE)*. IEEE, aug 2011, pp. 477–482. [Online]. Available: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=6028683

[SSS+05] W. Sobel, S. Subramanyam, A. Sucharitakul, J. Nguyen, H. Wong, S. Patil, A. Fox, and D. Patterson, "Cloudstone: Multi-Platform, Multi-Language Benchmark and Measurement Tools for Web 2.0," *Benchmarking*, 2005.

[SSS12] E. Siham, C. Schlereth, and B. Skiera, "Price comparison for infrastructure-as-a-service," no. March 2016, pp. 1–12, 2012. [Online]. Available: http://aisel.aisnet.org/ecis2012/161/

[TR14] B. Twardowski and D. Ryzko, "Multi-agent Architecture for Real-Time Big Data Processing," *2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, no. JUNE 2015, pp. 333–337, 2014. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6928203

[uRHH11] Z. ur Rehman, F. K. Hussain, and O. K. Hussain, "Towards Multi-criteria Cloud Service Selection," in *2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*. IEEE, jun 2011, pp. 44–48. [Online]. Available: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=5976164

[WCB10] B. Wickremasinghe, R. N. Calheiros, and R. Buyya, "CloudAnalyst: A CloudSim-Based Visual Modeller for Analysing Cloud Computing Environments and Applications," *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, pp. 446–452, 2010. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5474733

[Wes14] C. Westhoff, "Übersicht über die Frameworks für den Einsatz von Cloud- Computing im Gesundheitswesen," no. September, pp. 1–51, 2014.

[YIEO09] N. Yigitbasi, A. Iosup, D. Epema, and S. Ostermann, "C-Meter: A framework for performance analysis of computing clouds," *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID 2009*, pp. 472–477, 2009.

[ZCZH10] S. Zhang, X. Chen, S. Zhang, and X. Huo, "The comparison between cloud computing and grid computing," in *2010 International Conference on Computer Application and System Modeling (ICCASM 2010)*, vol. 11. IEEE, oct 2010, pp. V11–72–V11–75. [Online]. Available: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=5623257

Anhang

Evaluationsdiagramme

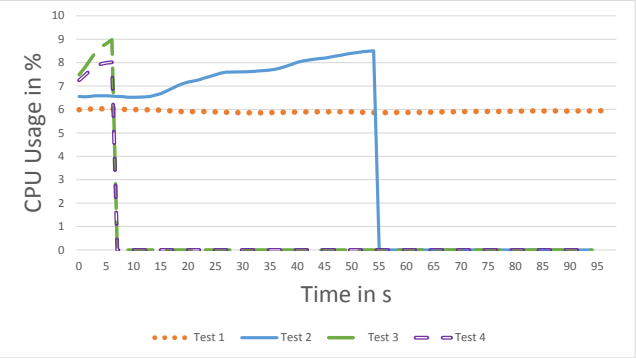


Abbildung 7.1: CPU Usage C1

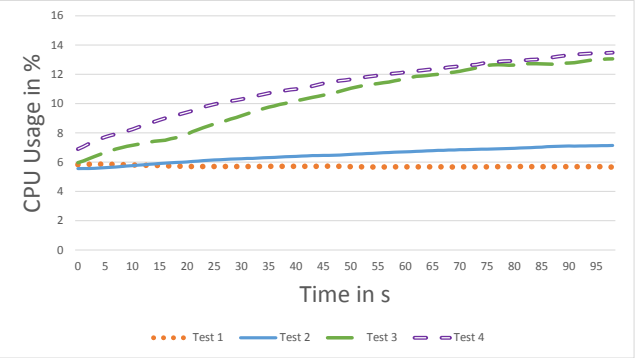


Abbildung 7.2: CPU Usage C2

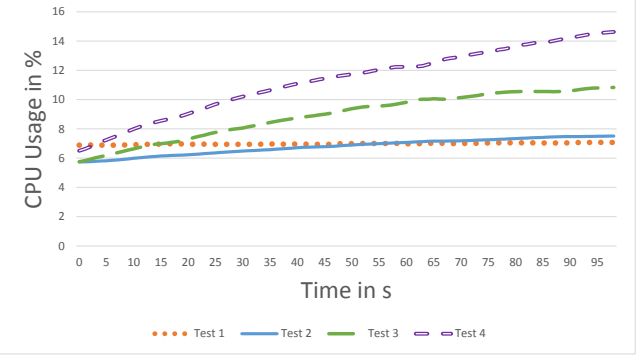


Abbildung 7.3: CPU Usage C3

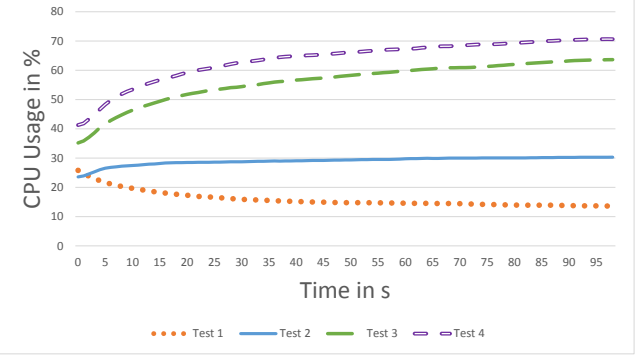


Abbildung 7.4: CPU Usage C6

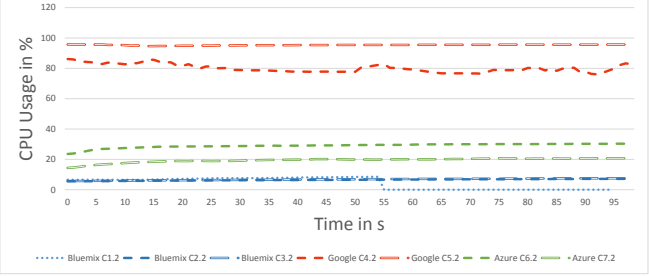


Abbildung 7.5: CPU Usage on Test 2

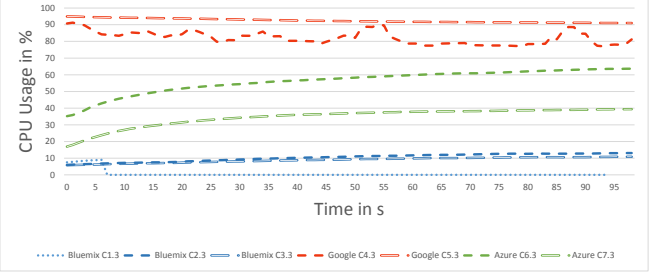


Abbildung 7.6: CPU Usage on Test 3

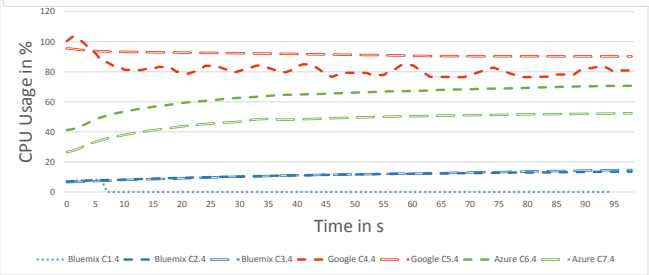


Abbildung 7.7: CPU Usage on Test 4

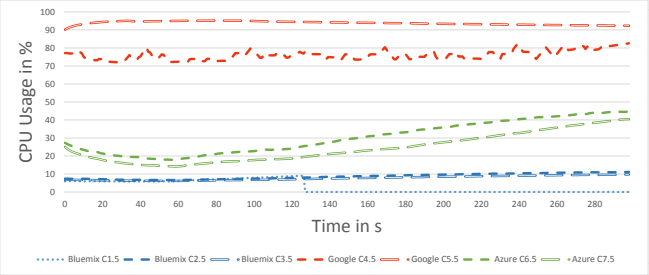


Abbildung 7.8: CPU Usage on Test 5

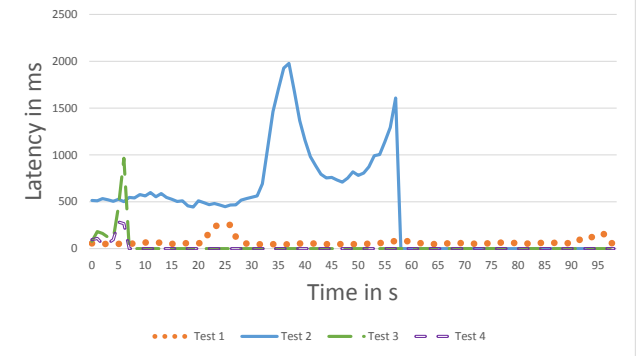


Abbildung 7.9: Latency C1

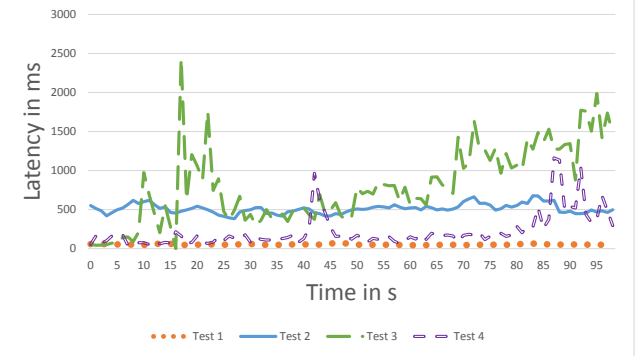


Abbildung 7.10: Latency C2

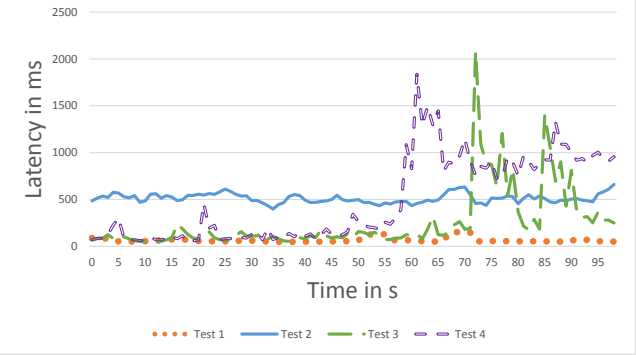


Abbildung 7.11: Latency C3

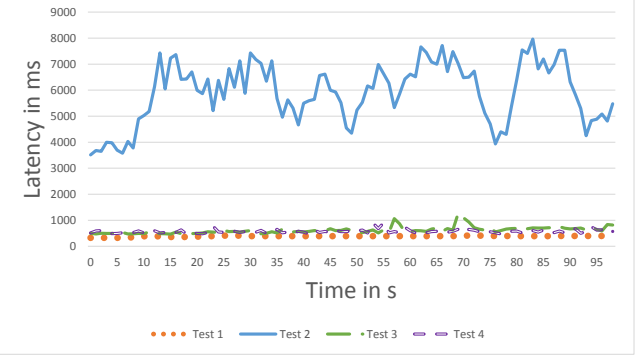


Abbildung 7.12: Latency C4

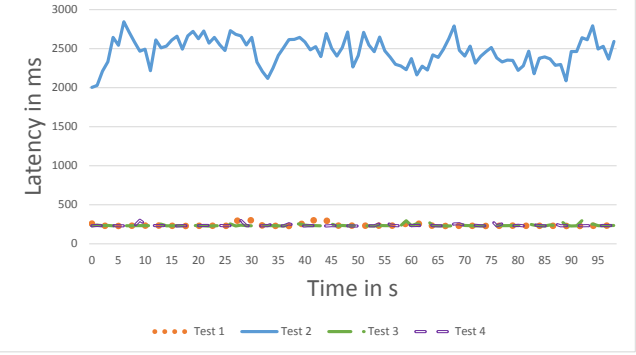


Abbildung 7.13: Latency C5

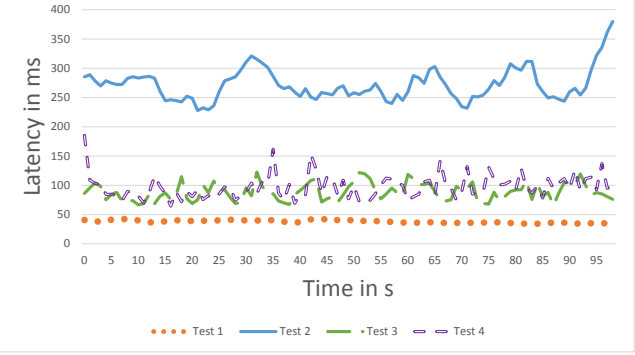


Abbildung 7.14: Latency C6

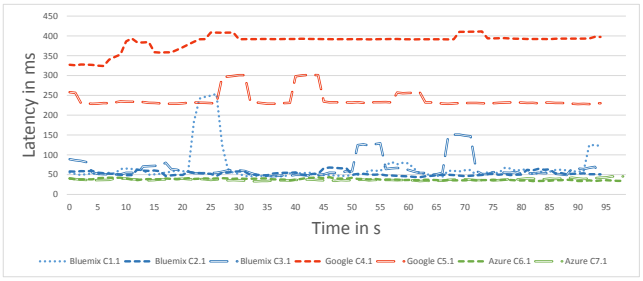


Abbildung 7.15: Latency on Test 1

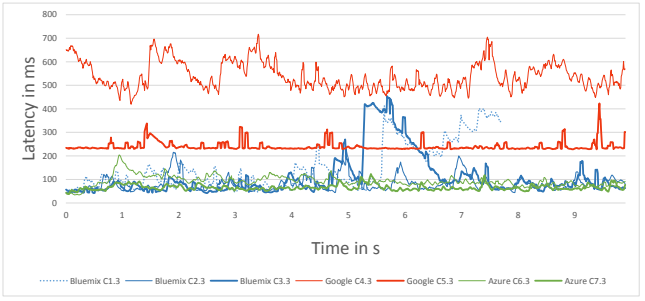


Abbildung 7.16: Latency on Test 3

Abbildung 7.17: Latency on Test 4

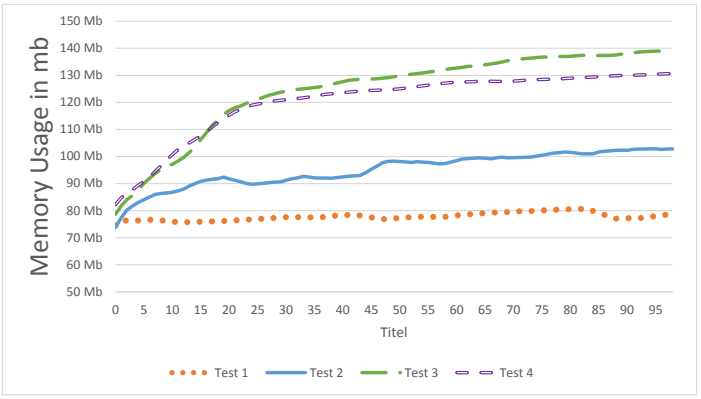


Abbildung 7.18: Memory Usage C2

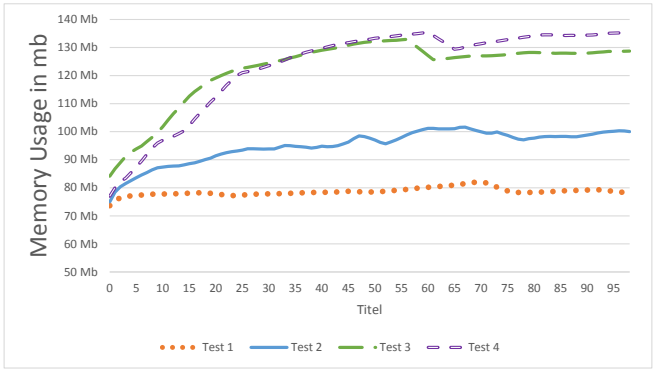


Abbildung 7.19: Memory Usage C3

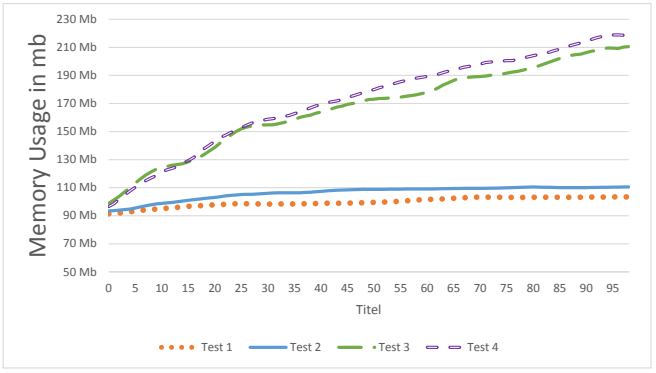


Abbildung 7.20: Memory Usage C4

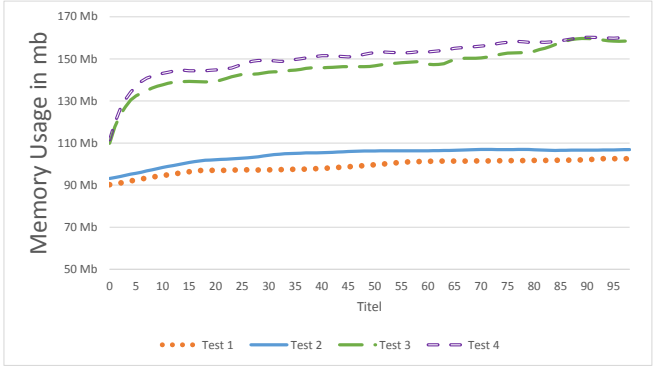


Abbildung 7.21: Memory Usage C5

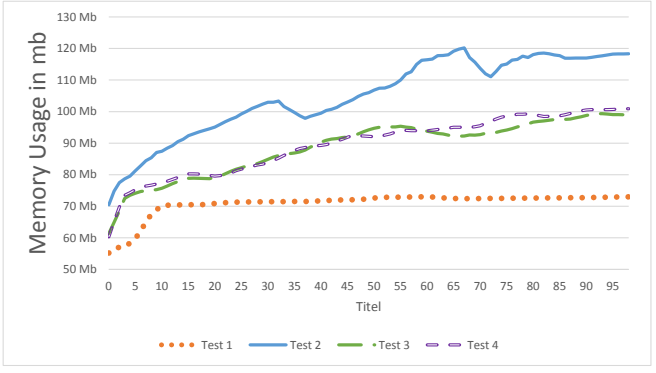


Abbildung 7.22: Memory Usage C6

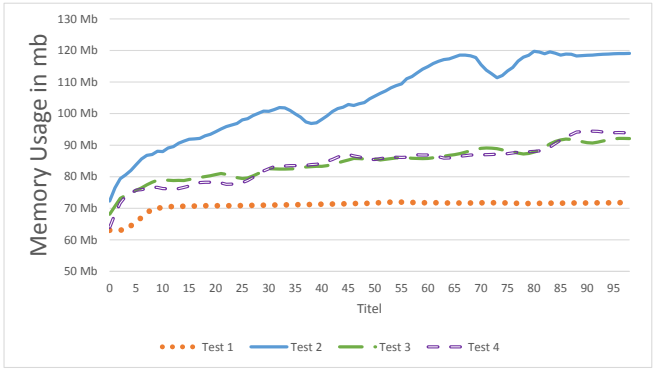


Abbildung 7.23: Memory Usage C7

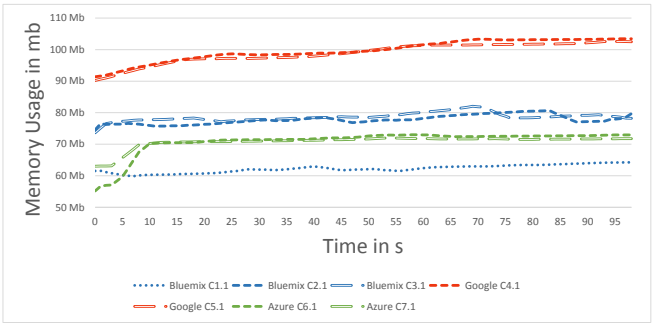


Abbildung 7.24: Memory Usage on Test 1

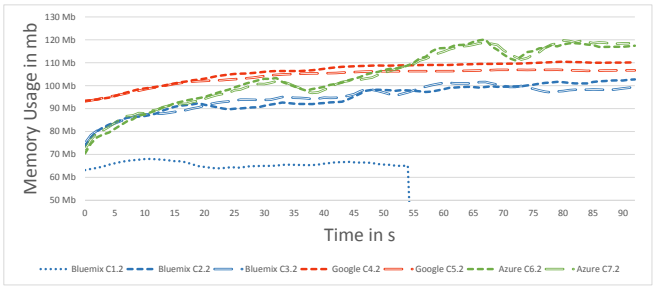


Abbildung 7.25: Memory Usage on Test 2

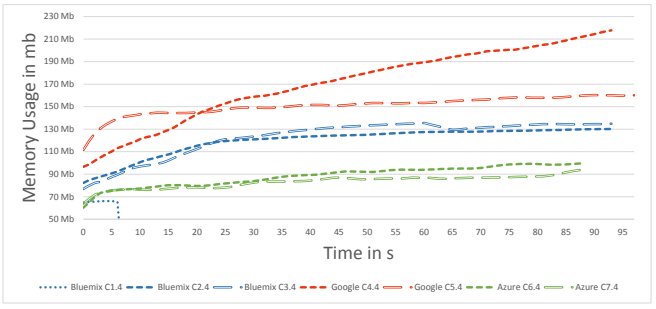


Abbildung 7.26: Memory Usage on Test 4

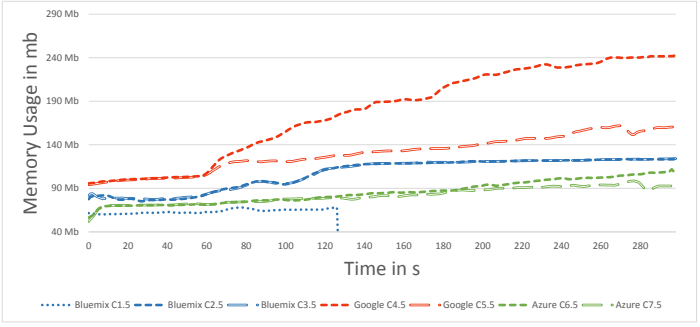


Abbildung 7.27: Memory Usage on Test 5

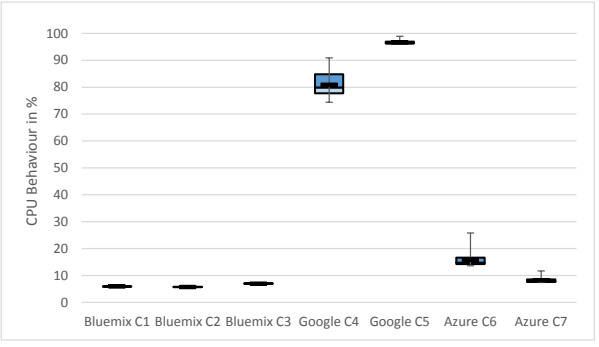


Abbildung 7.28: CPU Boxplot on Test 1

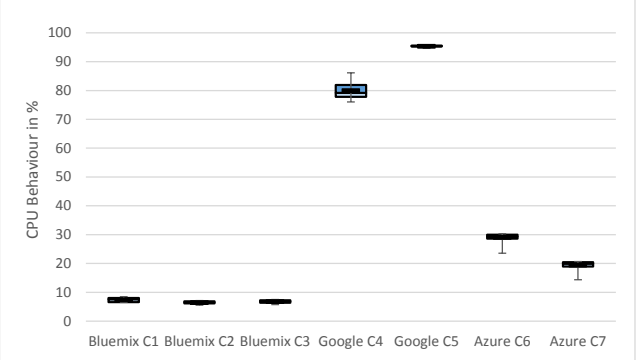


Abbildung 7.29: CPU Boxplot on Test 2

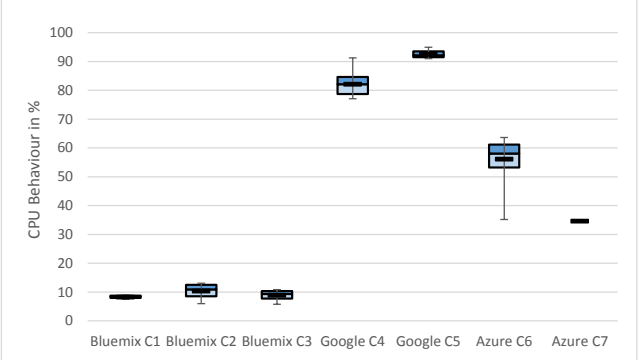


Abbildung 7.30: CPU Boxplot on Test 3

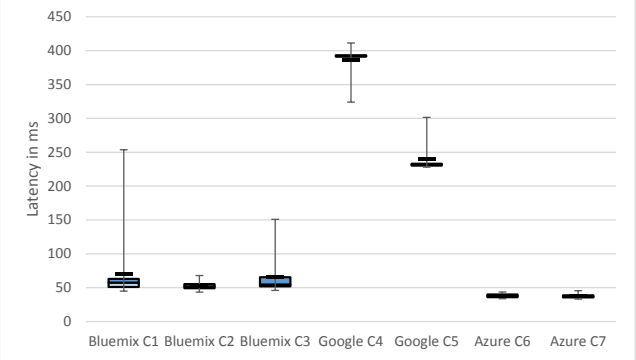


Abbildung 7.31: Latency Boxplot on Test 1

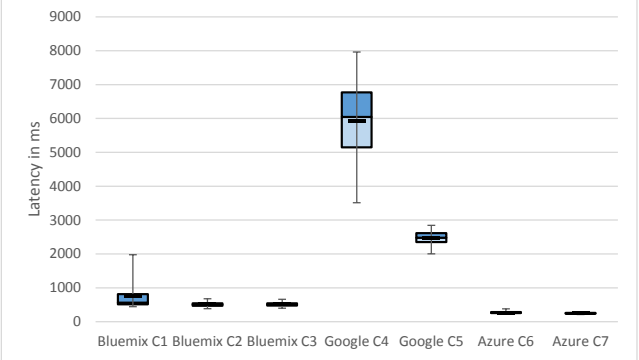


Abbildung 7.32: Latency Boxplot on Test 2

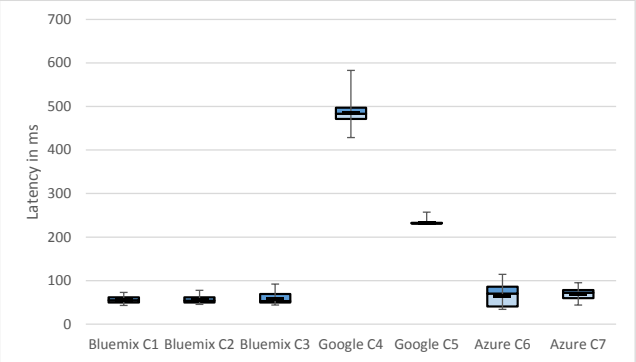


Abbildung 7.33: Latency Boxplot on Test 3

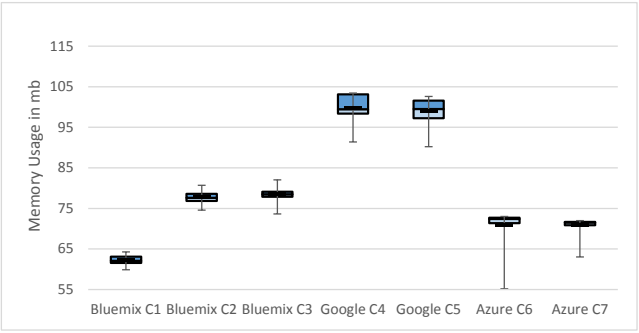


Abbildung 7.34: Memory Boxplot on Test 1

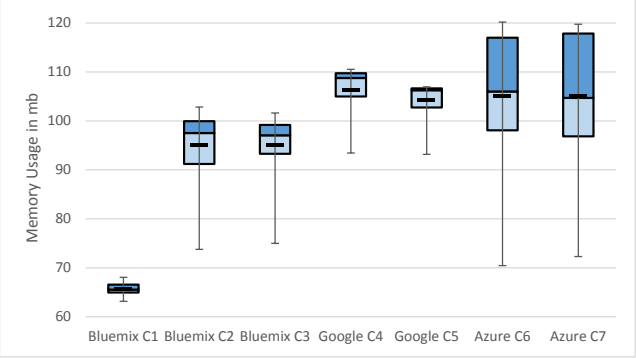


Abbildung 7.35: Memory Boxplot on Test 3

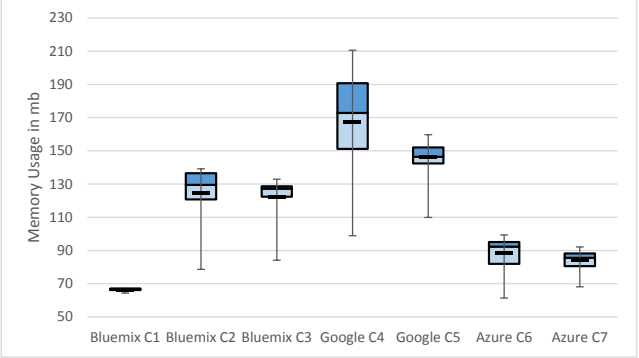


Abbildung 7.36: Memory Boxplot on Test 4