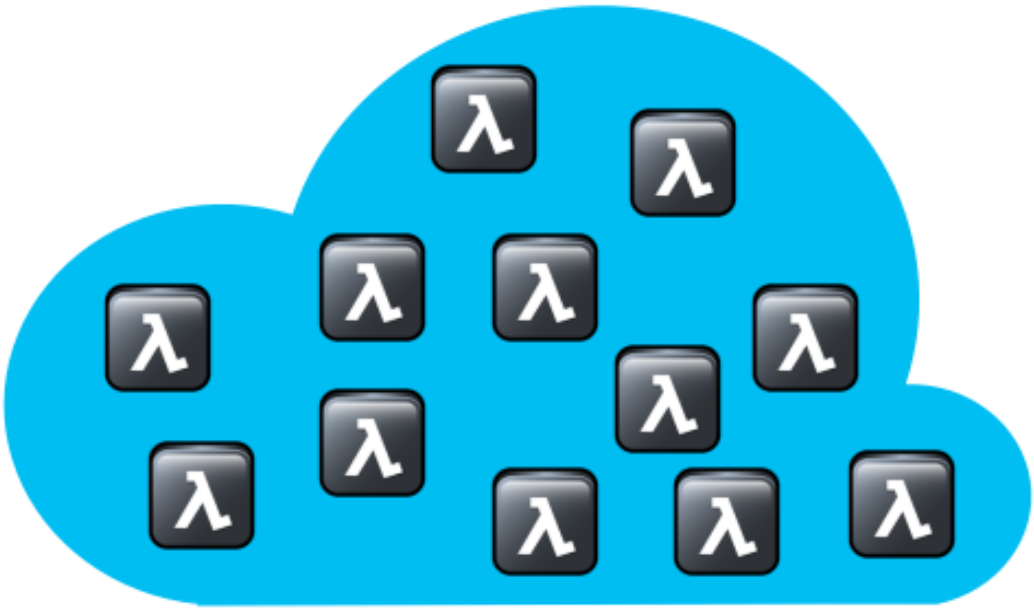
Serverless Server for Industry 4.0

November 2016



Inhaltsverzeichnis

Kapitel 1

Einleitung

Das Ziel dieses Projekt ist es, ein System zu schaffen, in dem Entwickler einfach servergebundene Software implementieren können. Derzeit ist der Entwicklungsvorgang einer servergebundenen Software eng mit dem Konfigurieren und Verwalten eines Servers verbunden. Diese Tätigkeiten brauchen viel Zeit, Geld und Arbeitskraft. Ziel ist es, Code für fast jeden Service mit minimalem Administrationsaufwand auszuführen. Es muss nur der Code hochgeladen werden und das System übernimmt alles, was zum Ausführen und Skalieren erforderlich ist.
Es ist nicht nötig, den ganzen Server mit Aufwand aufzusetzen und zu warten, was natürlich die Kosten verringert. Vorteil des Systems ist, dass es den Entwicklern ermöglicht, nur mit Code-Schreiben beschäftigt zu sein. Somit wird Entwicklern, die eine Anwendung schreiben, aber wenig Wissen über die Verteilung und Serveradministration besitzen, mit dem System das Leben erleichtert.
Zustandlose Lambda-Funktionen, die als Code von den Entwicklern hochgeladen werden können ermöglichen es, viele verschiedene Aufgaben mithilfe des Systems zu lösen und nebeneinander auszuführen.
Zudem ist es möglich, Berechtigungen zum Ausführen der Software zu delegieren, sodass die Einsatzgebiete nicht nur in Industriebereichen, sondern ebenfalls in Webanwendungen, Apps und Anwendungen des kommenden Internet of Things liegen können.
Im Folgenden werden zwei Zielgruppen unterschieden: Entwickler und Benutzer. Unter Entwickler ist die Gruppe der Menschen gemeint, die Lambda-Funktionen entwickeln und diese ausführen und ändern dürfen. Unter Benutzer ist die Gruppe gemeint, welche nur Berechtigung besitzen die Lambda-Funktion auszuführen. Somit ist der Entwickler ein Benutzer mit mehr Berechtigungen.

Kapitel 2

Zielbestimmung

2.1 Musskriterien

Musskriterien funktionaler Art

- Entwickler soll Lambda-Funktion mithilfe einer einfachen REST API hochladen können.
- Entwickler soll Lambda-Funktion mithilfe einer einfachen REST API ausführen können.
- Entwickler soll Lambda-Funktion mithilfe einer einfachen REST API löschen können.
- Entwickler soll Lambda-Funktion mithilfe einer einfachen REST API ändern können.
- Entwickler soll Lambda-Funktion mithilfe einer einfachen REST API benennen können.
- Konfigurationen (z.B. Signatur, Funktionsparameter, Ausführungsanzahl) der Lambda-Funktion sollen spezifiziert werden können.
- Entwickler sollen mithilfe eines Tokens authentifiziert werden können.
- Benutzer sollen mithilfe eines Tokens, der vom Entwickler ausgegeben wird authentifiziert werden können und mit diesem Token Lambda-Funktionen ausführen.
- Laufende Instanzen gelöschter Lambda-Funktionen sollen gestoppt werden.
- Lambda-Funktionen sollen Zustandslos sein.
- Lambda-Funktionen sollen Gekapselt werden.
- Lambda-Funktionen dürfen nur eine vom Server spezifizierte Zeit lang laufen.

Musskriterien nichtfunktionaler Art

- Das System soll in Rechner- und Speicherressourcen skalierbar sein.
- Das System soll als Mindestgrundlage Lambda-Funktionen in python2 unterstützen.

2.2 Wunschkriterien

Wunschkriterien funktionaler Art

- Lambda-Funktionen dürfen nur eine vom Benutzer spezifizierte Zeit lang laufen, sofern diese die maximale Laufzeit nicht übersteigt.
- Zu jedem Token wird ein Zähler erstellt, der für statistische Zwecke ausgelesen werden kann.
- Entwickler können ein Repository angeben und von dort die Lambda-Funktion auf den Server laden.
- Versionsverwaltung der Lambda-Funktionen.
- Der Server kann Firewall-Einstellungen für Lambda-Funktionen definieren.
- Entwickler können Firewall-Einstellungen für Lambda-Funktionen definieren, sofern sie nicht den Server-Einstellungen widersprechen.
- Sofern die Ausführungszeit einer Lambda-Funktion ein Limit überschreitet, wird die HTTP-Verbindung abgebrochen und bei Beendigung der Lambda-Funktion wird das Ergebnis bereitgestellt.
- Prepaid-Bezahlsystem für Tokens abhängig von der Ausführungszeit der zugehörigen Lambda-Funktion.
- Erwartete Laufzeit einer Lambda-Funktion bereitstellen.

Wunschkriterien nichtfunktionaler Art

- Die Ausführung einer stark ressourcenaufwändigen Lambda-Funktion beeinträchtigt nicht die Ausführung anderer Lambda-Funktionen auf dem Server.
- Das System soll um Sprachen für Lambda-Funktionen erweiterbar sein.

2.3 Abgrenzungskriterien

- Entwicklung einer graphischen Nutzeroberfläche.
- Entwicklung einer Nutzerverwaltung mit Datenbank.

Kapitel 3

Produkteinsatz

Das System erleichtert das Leben der Softwareentwickler. Es übernimmt für diese Administrationsarbeiten wie Serverwartung, Softwareverteilung (englisch software deployment) und Skalierung (auch Skalierbarkeit). Damit ist das Hauptziel, den Entwicklungsvorgang einer Software zu beschleunigen und zu verfeinern. Zudem ist das System komfortabel für Serverbetreiber, welche mit wenig Aufwand das System aufsetzen und den unterliegenden Server Warten können.

3.1 Anwendungsbereich

Das System soll für die Anwendung im Softwareentwicklungsbereich konstruiert werden. Das System wird für folgende Teilbereiche entworfen:

- Industrie 4.0
- Internet of Things
- Webanwendungen
- Apps

3.2 Zielgruppen

- Softwareentwickler

3.3 Betriebsbedingungen

- Rechner
- Internetverbindung

Kapitel 4

Produktumgebung

4.1 Software

Die folgende Software muss auf dem Server ausgeführt werden können:

- Docker als Laufzeitumgebung
- Compiler/Interpreter für die unterstützen Sprachen

Das Betriebssystem des Servers muss diese Software ausführen können. Also bieten sich vor allem Windows, Linux oder Mac OS X als Betriebssystem des Servers an.

4.2 Hardware

Der Server muss genug Ressourcen haben, damit dort Docker lauffähig ist. Ausßerdem müssen dort viele Lambda-Funktionen gleichzeitig ausgeführt werden können.

Kapitel 5

Funktionale Anforderungen

5.1 Überblick: Grundfunktionen

Nr.	Beschreibung
/F010/	Lambda-Funktionen hochladen
/F011/	Zugriffs-Token zurückgeben
/F020/	Lambda-Funktionen ausführen
/F021/	Ausführungsumgebung für Lambda-Funktionen bereitstellen
/F030/	Lambda-Funktionen löschen
/F031/	Beendigung von Instanzen gelöschter Lambda-Funktionen
/F040/	Lambda-Funktionen updaten
/F050/	Lambda-Funktionen benennen
/F060/	Lambda-Funktionen konfigurieren
/F080/	Zeitabhängige Terminierung von Lambda-Funktionen
/F090/	Daten für Lambda-Funktion einbinden
/F100/	Entwickler authentifizieren
/F110/	Entwickler soll Tokens zur Ausführung einer Lambda-Funktion erzeugen können

5.2 Überblick: Erweiterte Funktionen

Nr.	Beschreibung
/F140/	Entwickler sollen eine maximale Laufzeit spezifizieren können, die nicht die vom Server vorgegebene maximale Laufzeit überschreitet
/F150/	Zähler zu Tokens speichern
/F160/	Zähler zu Tokens auslesen
/F170/	Entwickler sollen Repositorys zum Hochladen von Lambda-Funktionen spezifizieren können
/F180/	Versionsverwaltung für hochgeladene Lambda-Funktionen
/F190/	Entwickler soll Firewalleinstellung für Lambda-Funktionen spezifizieren können
/F200/	spätere Bereitstellung des Ergebnisses für Lambda-Funktionen
/F210/	Entwickler soll Tokens kaufen können (Prepaid)
/F220/	Serverbetreiber kann Statistiken einsehen

5.3 Detaillierte Beschreibung

5.3.1 Grundfunktionen

/F010/

Kurzbeschreibung: Hochladen einer Lambda-Funktion
Vorbedingung: Entwickler besitzt eine Lambda-Funktion
Beschreibung: Ein Entwickler schickt eine Upload-Anfrage an den Server über die spezifizierte REST API. Dort wird auf mögliche Fehler geprüft, dann wird die neue Lambda-Funktion hinzugefügt, falls kein Fehler aufgetreten ist. /F021/ wird ausgeführt. /F011/ wird ausgeführt.
Nachbedingung: Die Lambda-Funktion ist auf dem Server vorhanden und kann verwendet werden.

/F011/

Kurzbeschreibung: Rückgabe eines Tokens zum Zugriff auf eine Lambda-Funktion
Vorbedingung: Die Lambda-Funktion ist auf dem Server vorhanden.
Beschreibung: Der Server gibt einen Token an den Entwickler zurück, mit dem er auf die Lambda-Funktion zugreifen kann.
Nachbedingung: Der Token kann zum Zugriff auf die Lambda-Funktion verwendet werden.

/F020/

Kurzbeschreibung: Ausführung einer Lambda-Funktion
Vorbedingung: Die Lambda-Funktion ist auf dem Server vorhanden und der Benutzer hat den benötigten Token. Dies ist entweder der Token des Entwicklers oder ein vom Entwickler für Benutzer erzeugter Token (siehe /F110/).
Beschreibung: Der Server startet die Ausführung einer Lambda-Funktion und übergibt ihr die vom Benutzer übergebenen Daten. Nach der Ausführung liegt ein Ergebnis vor, das wieder an den Benutzer zurückgegeben wird.
Nachbedingung: Der Server hat danach den gleichen Zustand wie vor der Ausführung.

/F021/

Kurzbeschreibung: Bereitstellung einer Umgebung in der eine Lambda-Funktion ausgeführt werden kann.
Vorbedingung: Die Lambda-Funktion ist auf dem Server vorhanden.
Beschreibung: Der Server erstellt eine Ausführungsumgebung für eine Lambda-Funktion, die z.B. Bibliotheken, die zur Ausführung benötigt werden, Software oder Daten beinhaltet. Außerdem werden so die Ausführungen gekapselt.
Nachbedingung: Die Ausführungsumgebung wird gelöscht, wenn sie nicht mehr benötigt wird.

/F030/

Kurzbeschreibung: Löschen einer Lambda-Funktion
Vorbedingung: Die Lambda-Funktion ist auf dem Server vorhanden und der Entwickler hat den benötigten Token zur Authorisierung.
Beschreibung: Ein Entwickler benutzt seinen Token um eine Lambda-Funktion vom Server zu entfernen. /F031/ wird ausgeführt.
Nachbedingung: Die Lambda-Funktion ist nicht mehr auf dem Server vorhanden und laufende Instanzen werden beendet.

/F031/

Kurzbeschreibung: Beendigung aller laufender Instanzen einer gelöschten Lambda-Funktion
Vorbedingung: Die Lambda-Funktion wurde gelöscht.
Beschreibung: Der Server beendet alle laufenden Instanzen einer Lambda-Funktion die von ihrem Entwickler gelöscht wurde.
Nachbedingung: Auf dem Server laufen keine Instanzen der gelöschten Lambda-Funktion mehr.

/F040/

Kurzbeschreibung: Ändern einer bereits vorhandenen Lambda-Funktion
Vorbedingung: Die Lambda-Funktion ist auf dem Server vorhanden und der Entwickler hat den benötigten Token zur Authorisierung.
Beschreibung: Ein Entwickler benutzt seinen Token um eine auf dem Server vorhandene Lambda-Funktion zu ändern (z.B. zur Fehlerkorrektur oder zur Erweiterung).
Nachbedingung: Die neue Version der Lambda-Funktion ist auf dem Server vorhanden.

/F050/

Kurzbeschreibung: Benennen einer Lambda-Funktion
Vorbedingung: Entwickler besitzt gewünschte Benennung einer Lambda-Funktion.
Beschreibung: Fall 1: Lambda-Funktion noch nicht auf dem Server vorhanden.
Ein Entwickler spezifiziert in der Konfiguration der Lambda-Funktion die Benennung der Lambda-Funktion
Fall 2: Lambda-Funktion auf dem Server vorhanden.
Ein Entwickler benutzt seinen Token um eine auf dem Server vorhandene Lambda-Funktion zu benennen. (siehe /F040/)
Nachbedingung: Die Lambda-Funktion kann in Zukunft über den vergebenen Namen angesprochen werden.

/F060/

Kurzbeschreibung: Konfigurieren einer Lambda-Funktion
Vorbedingung: Die Lambda-Funktion ist auf dem Server vorhanden und der Benutzer hat den benötigten Token zur Authorisierung.
Beschreibung: Ein Benutzer benutzt seinen Token um für eine auf dem Server vorhandene Lambda-Funktion eine Konfiguration zur Ausführung festzulegen. /F020/ wird ausgeführt.
Nachbedingung: Ausführungs-Instanzen der Lambda-Funktion verwenden die andere Konfigurationen.

/F080/

Kurzbeschreibung: Zeitabhängiges Beenden einer Lambda-Funktion
Vorbedingung: Auf dem Server ist eine Maximallaufzeit definiert.
Beschreibung: Der Server beendet jede Lambda-Funktion, deren Ausführung die Maximallaufzeit überschreitet, damit der Server nicht von zu lange laufenden Instanzen der Lambda-Funktionen blockiert wird.
Nachbedingung: Die laufende Instanz von der Lambda-Funktion wird nach Überschreitung der Maximallaufzeit terminiert.

/F090/

Kurzbeschreibung: Einbindung von Daten für eine Lambda-Funktion
Vorbedingung: Die Lambda-Funktion ist auf dem Server vorhanden und der Benutzer hat den benötigten Token zur Authorisierung.
Beschreibung: Ein Benutzer verwendet einen Token um festzulegen, mit welchen Daten eine Lambda-Funktion ausgeführt werden soll. Der Server holt diese Daten und bindet sie in die Ausführungsumgebung ein (siehe /F021/).
Nachbedingung: Die Lambda-Funktion kann mit den angegebenen Daten ausgeführt werden.

/F100/

Kurzbeschreibung: Authentifizierung der Entwicklers
Vorbedingung: Die Lambda-Funktion ist auf dem Server vorhanden und der Entwickler hat den benötigten Token zur Authorisierung.
Beschreibung: Ein Entwickler authentifiziert sich mit einem vorher erworbenen Token beim Server.
Nachbedingung: Dem Entwickler ist es nun möglich Operationen auf Lambda-Funktionen durchzuführen (z.B. ändern, löschen).

/F110/

Kurzbeschreibung: Erzeugung von Tokens zur Ausführung einer Lambda-Funktion
Vorbedingung: Die Lambda-Funktion ist auf dem Server vorhanden.
Beschreibung: Ein Entwickler erzeugt Tokens für Benutzer, mit denen diese die Berechtigung haben eine bestimmte Lambda-Funktion auszuführen.
Nachbedingung: Die Lambda-Funktion kann mit den erzeugten Tokens ausgeführt werden.

5.3.2 Erweiterte Funktionen

/F140/

Kurzbeschreibung: Festlegung einer Maximallaufzeit für eine Lambda-Funktion
Vorbedingung: Die Lambda-Funktion ist auf dem Server vorhanden und der Entwickler hat den benötigten Token zur Authorisierung.
Beschreibung: Ein Entwickler kann für eine Lambda-Funktion eine Maximallaufzeit festlegen nach der die Lambda-Funktion automatisch abgebrochen wird. Diese Maximallaufzeit soll unter der vom Server definierten liegen.
Nachbedingung: Laufende Instanzen der Lambda-Funktion werden zukünftig nach dieser Maximallaufzeit beendet.

/F150/

Kurzbeschreibung: Erzeugung von Zählern zu Tokens
Vorbedingung: Ein Token ist vorhanden.
Beschreibung: Zu jedem Ausführungs-Token wird ein Zähler gespeichert, der bei der Ausführung der Lambda-Funktion startet und nach deren Beendigung stoppt.
Nachbedingung: Zu jedem Ausführungs-Token wird ein Zähler gespeichert, der angibt wie lange die Laufzeit der Ausführung war.

/F160/

Kurzbeschreibung: Auslesen der Zähler zu Tokens
Vorbedingung: Ein Token mit Zähler ist vorhanden.
Beschreibung: Ein Benutzer kann die Zähler zu seinen Tokens auslesen.
Nachbedingung: Benutzer weiß, wie lange die Lambda-Funktion gelaufen ist.

/F170/

Kurzbeschreibung: Hochladen einer Lambda-Funktion aus einem Repository
Vorbedingung: Token zur Authentifizierung ist beim Entwickler vorhanden.
Beschreibung: Ein Entwickler schickt eine Upload-Anfrage an den Server. Der Prozess läuft ähnlich ab wie in /F010/, mit dem Unterschied, dass die Lambda-Funktion aus einem Repository geladen wird.
Nachbedingung: Die Lambda-Funktion ist auf dem Server vorhanden und kann verwendet werden.

/F180/

Kurzbeschreibung: Versionsverwaltung bei Lambda-Funktionen
Vorbedingung: Eine Lambda-Funktion wurde geändert.
Beschreibung: Die Änderungen an einer Lambda-Funktion werden vom Server protokolliert.
Nachbedingung: Die Lambda-Funktionen liegen auf dem Server in verschiedenen Versionen vor.

/F190/

Kurzbeschreibung: Für Lambda-Funktionen sollen Firewalleinstellungen definiert werden können.
Vorbedingung: Die Lambda-Funktion ist auf dem Server vorhanden.

Beschreibung: Ein Entwickler kann für eine Lambda-Funktion eine Liste von Web-Adressen angeben, auf die diese zugreifen darf.
Nachbedingung: Die Lambda-Funktion kann auf die definierten Web-Adressen zugreifen.

/F200/

Kurzbeschreibung: Spätere Abholung der Ergebnisse einer Lambda-Funktion
Vorbedingung: Die Lambda-Funktion ist auf dem Server vorhanden.
Beschreibung: Ein Benutzer kann bei Lambda-Funktionen mit langer Laufzeit das Ergebnis zu einem späteren Zeitpunkt abfragen, um nicht die ganze Zeit eine Verbindung aufrecht erhalten zu müssen.
Nachbedingung: Das Ergebnis muss bis zur Abholung gespeichert werden.

/F210/

Kurzbeschreibung: Erwerb von Tokens für Entwicklers
Vorbedingung: keine
Beschreibung: Ein Entwickler kann per Prepaid-Kauf Tokens erwerben, mit denen er sich beim Server authentifizieren kann.
Nachbedingung: Der Entwickler soll sich mit dem erworbenen Token authentifizieren können (siehe /F100/).

/F220/

Kurzbeschreibung: Ein Serverbetreiber soll Statistiken einsehen können.
Vorbedingung: Die Daten über die Ausführung von Lambda-Funktionen müssen gespeichert werden.
Beschreibung: Ein Serverbetreiber kann Statistiken über die Ausführung von Lambda-Funktionen einsehen wie z.B. die Anzahl an Ausführung einer Lambda-Funktion oder die durchschnittlich verbrauchte Rechenzeit (auch Laufzeit).
Nachbedingung: keine

Kapitel 6

Nichtfunktionale Anforderungen

6.1 Überblick: Grundanforderungen

Nr.	Beschreibung
/NF010/	Skalierbarkeit in Rechner- und Speicherressourcen
/NF040/	Nach einem vom Server spezifizierten zeitlichen Intervall nach dem letzten Aufruf einer Lambda-Funktion wird diese gelöscht.
/NF050/	Die Zeit zwischen der Ankunft der HTTP-Anfrage und dem Beginn der auszuführenden Lambda-Funktion soll höchstens 10 Sekunden dauern.
/NF060/	Die Zeit zwischen dem Ende der auszuführenden Lambda-Funktion und dem Beginn des Sendens des HTTP-Rückgabewerts soll höchstens 5 Sekunden dauern.
/NF070/	Das System soll Lambda-Funktionen in python2 unterstützen.

6.2 Überblick: Erweiterte Anforderungen

Nr.	Beschreibung
/NF080/	Die Ausführung einer stark ressourcenaufwändigen Lambda-Funktion beeinträchtigt nicht die Ausführung anderer Lambda-Funktionen auf dem Server.
/NF090/	Das System soll um Sprachen erweiterbar sein.

6.3 Detaillierte Beschreibung

6.3.1 Grundanforderungen

/NF010/

Kurzbeschreibung: Skalierbarkeit in Rechner- und Speicherressourcen
Vorbedingung: Server mit größeren Rechner- und Speicherressourcen als Minimalressourcen
Beschreibung: Veränderung (Vergrößerung, Verkleinerung, allgemeine Veränderung) der Rechner- und Speicherressourcen. Bei Verkleinerung muss sichergestellt werden, das die Ressourcen für die urprünglichen Lambda-Funktionen ausreichen.
Nachbedingung: Alle vorher unterstützten Lambda-Funktionen sind werden weiterhin unterstützt.

/NF040/

Kurzbeschreibung: Nach einem vom Server spezifizierten zeitlichen Intervall nach dem letzten Aufruf einer Lambda-Funktion wird diese gelöscht.
Vorbedingung: Möglichkeit zur Festsetzung eines zeitlichen Intervall bis zur Löschung einer Lambda-Funktion für Serverbetreiber in der Software
Beschreibung: Serverbetreiber setzt Limit (z.B. 31.553.280) fest. 31.553.280 Sekunden (365,2 Tage mit 24 Stunden pro Tag) nach dem letzten Aufruf einer Lambda-Funktion wird diese vom System gelöscht.
Nachbedingung: Gelöschte Lambda-Funktion

/NF050/

Kurzbeschreibung: Die Zeit zwischen der Ankunft der HTTP-Anfrage und dem Beginn der auszuführenden Lambda-Funktion soll höchstens 10 Sekunden dauern.
Vorbedingung: Abgeschlossene HTTP-Übertragung der Information
Beschreibung: Die Verarbeitung der Information, das Erstellen einer Ausführungsumgebung und das Laden serverinterner Ressourcen darf nicht länger als 10 Sekunden dauern. Nach diesen 10 Sekunden soll die Lambda-Funktion ausführbar sein. Ausgenommen ist hierbei das Nachladen benötigter Ressourcen (z.B. Bibliotheken).
Nachbedingung: Ausführbare Lambda-Funktion auf dem Server

/NF060/

Kurzbeschreibung: Die Zeit zwischen dem Ende der auszuführenden Lambda-Funktion und dem Beginn des Sendens des HTTP-Rückgabewerts soll höchstens 5 Sekunden dauern.
Vorbedingung: Terminierte oder abgebrocheneLambda-Funktion
Beschreibung: Nach dem Terminieren/Abbruch einer Lambda-Funktion soll in höchstens 5 Sekunden eine Rückmeldung an den Benutzer formuliert werden und abgesendet oder zur Verfügung gestellt werden.
Nachbedingung: Fertige Rückmeldung an den Benutzer

/NF070/

Kurzbeschreibung: Das System soll Lambda-Funktionen in python2 unterstützen.
Vorbedingung: Docker-Container unterstützen python2 .
Beschreibung: Die Lambda-Funktion in python2 ist ausführbar und kann mit dem System genutzt werden, indem in den Docker-Containern die Umgebung unterstützt wird.
Nachbedingung: Die Lambda-Funktion in python2 ist ausführbar und kann mit dem System genutzt werden.

6.3.2 Erweiterte Anforderungen

/**NF080**/

Kurzbeschreibung: Die Ausführung einer stark ressourcenaufwändigen Lambda-Funktion beeinträchtigt nicht die Ausführung anderer Lambda-Funktionen auf dem Server.
Vorbedingung: Hohe Serverbelastungsgrenzen, Abschottung der Lambda-Funktionen, Ausführung einer stark ressourcenaufwändigen Lambda-Funktion
Beschreibung: Bei der Ausführung einer stark ressourcenaufwändigen Lambda-Funktion werden andere Lambda-Funktionen auf dem Server nicht beeinträchtigt in ihrer Ausführung. Die Ressourcen werden gekapselt und gemanaged.
Nachbedingung: Parallele Ausführung von stark ressourcenaufwändigen Lambda-Funktionen ohne Beeinträchtigungen aufgrund der Serverbelastungsgrenzen

/**NF090**/

Kurzbeschreibung: Das System soll um Sprachen erweiterbar sein.
Vorbedingung: keine
Beschreibung: Das System soll objektorientiert sein und somit das Erweitern um Sprachen vereinfacht werden.
Nachbedingung: Das System ist einfach um Sprachen erweiterbar.

Kapitel 7

Produktdaten

7.1 Systemdaten

Nr.	Beschreibung
/D10/	Berechtigungen, welcher Benutzer welche Lambda-Funktion verwenden darf (Token)
/D20/	Rechenzeit (auch Laufzeit), die von Benutzern verbraucht wurde

7.2 Benutzerdaten

Nr.	Beschreibung
/D100/	Hochgeladene Lambda-Funktionen eines Entwicklers und passende Konfiguration dazu
/D110/	Anzahl der Ausführungen einer Lambda-Funktion
/D120/	Verbrauchte Rechenzeit (auch Laufzeit) für die Ausführung von Lambdas (Zähler)
/D130/	Noch verfügbare Rechenzeit (auch Laufzeit)

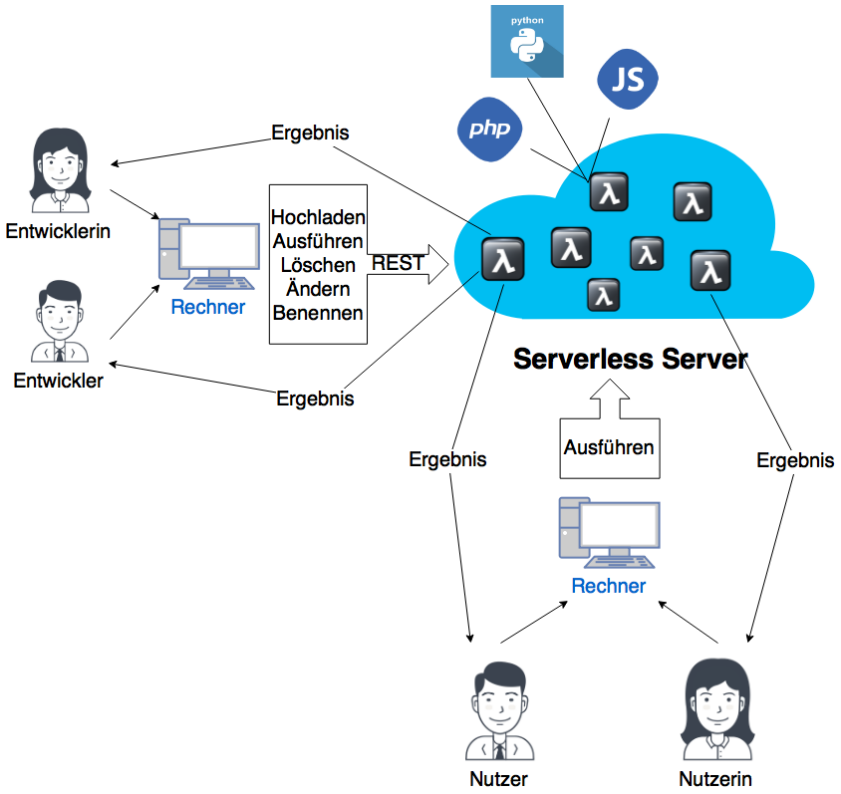
Kapitel 8

Systemmodell

8.1 Einleitung

Zum Verständnis des Systemaufbaus wird zunächst eine Übersicht der zur Verfügung gestellten Lambda-Funktionen des Serverless Server dargeboten, danach erfolgt die Beschreibung eines Anwendungsfalls in einer Schokoladenfabrik. Zu dieser Beschreibung werden einige Beispielszenarien dargestellt und aufgezeigt.

8.2 Übersicht



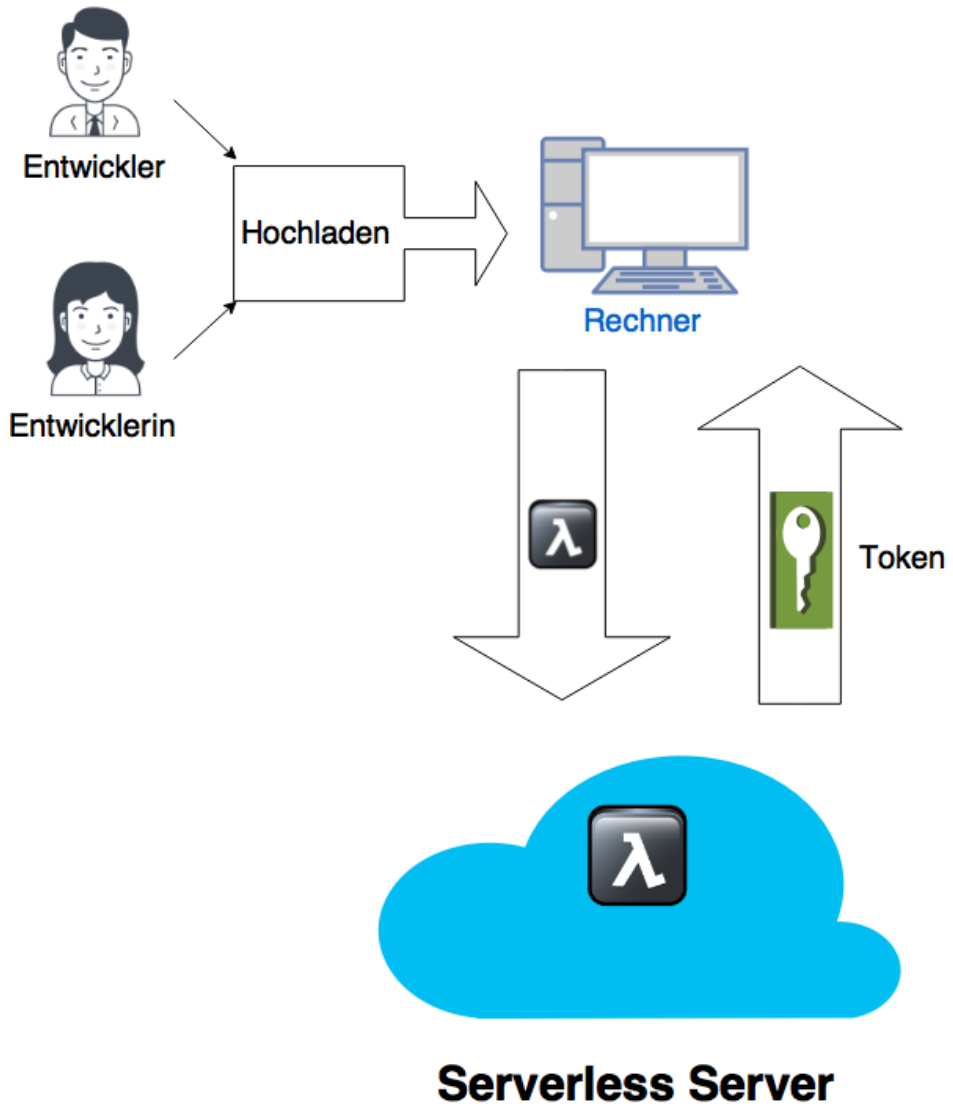
Das System lässt sich in drei große Bestandteile aufteilen: **Gesamtumgebung** (Cloud), **REST API** und **Docker-Container** (Lambdas).

Gesamtumgebung Die Gesamtumgebung regelt das Zusammenspiel der Docker-Container mit der REST API, den Lambda-Funktionen und dem unterliegenden Server.

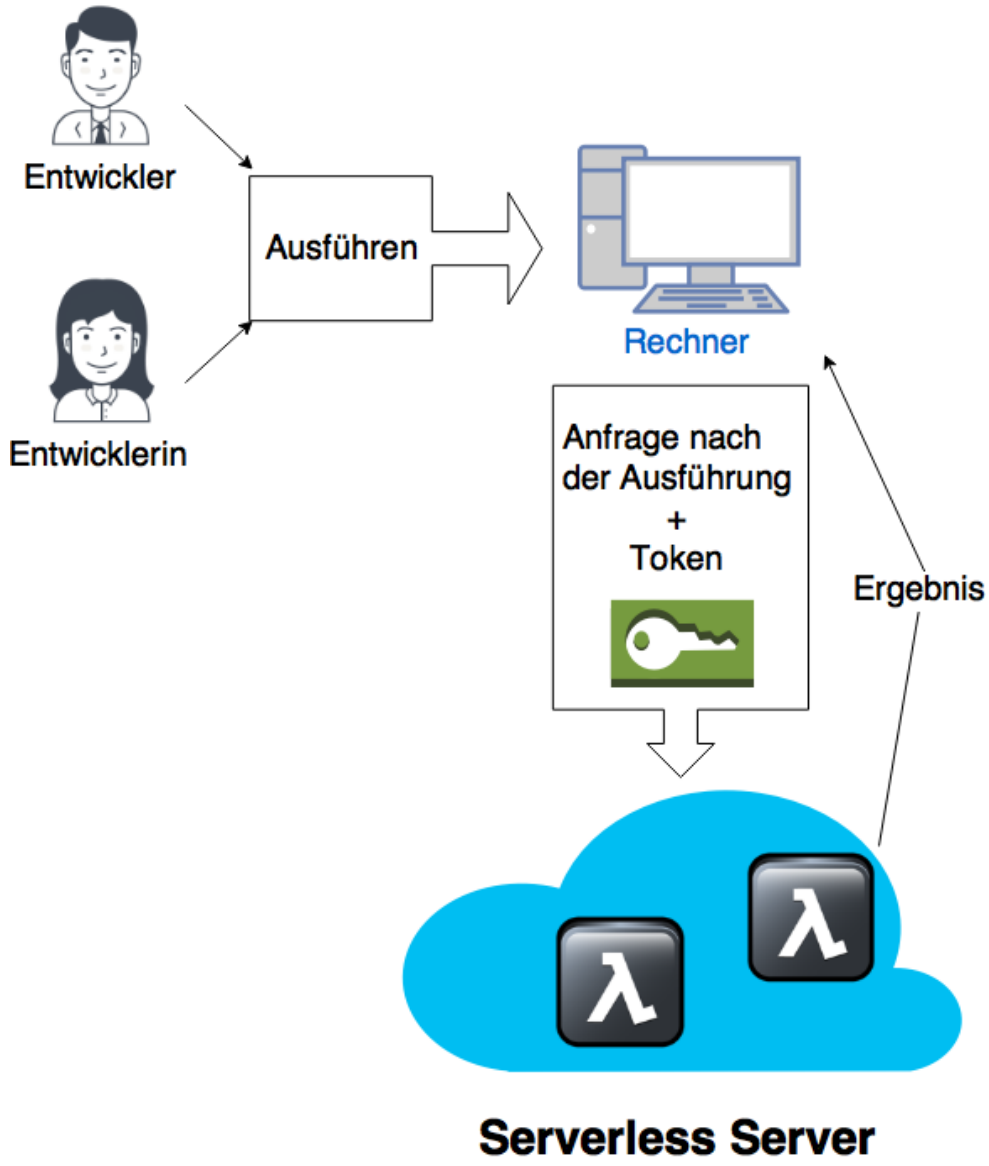
REST API Die REST API definiert die Kommunikation der Entwickler und Benutzer mit dem System. Die Hauptfunktionen, die dadurch angesprochen werden sind das Hochladen, Ausführen, Löschen und Ändern der Lambda-Funktionen. Die genaue Schnittstelle findet sich unter dem Kapitel zur Benutzerschnittstelle.

Docker-Container Die Docker-Container enthalten die Umgebung für die hochgeladenen Lambda-Funktionen. Durch diese wird die Kapselung der Lambda-Funktionen geschaffen.

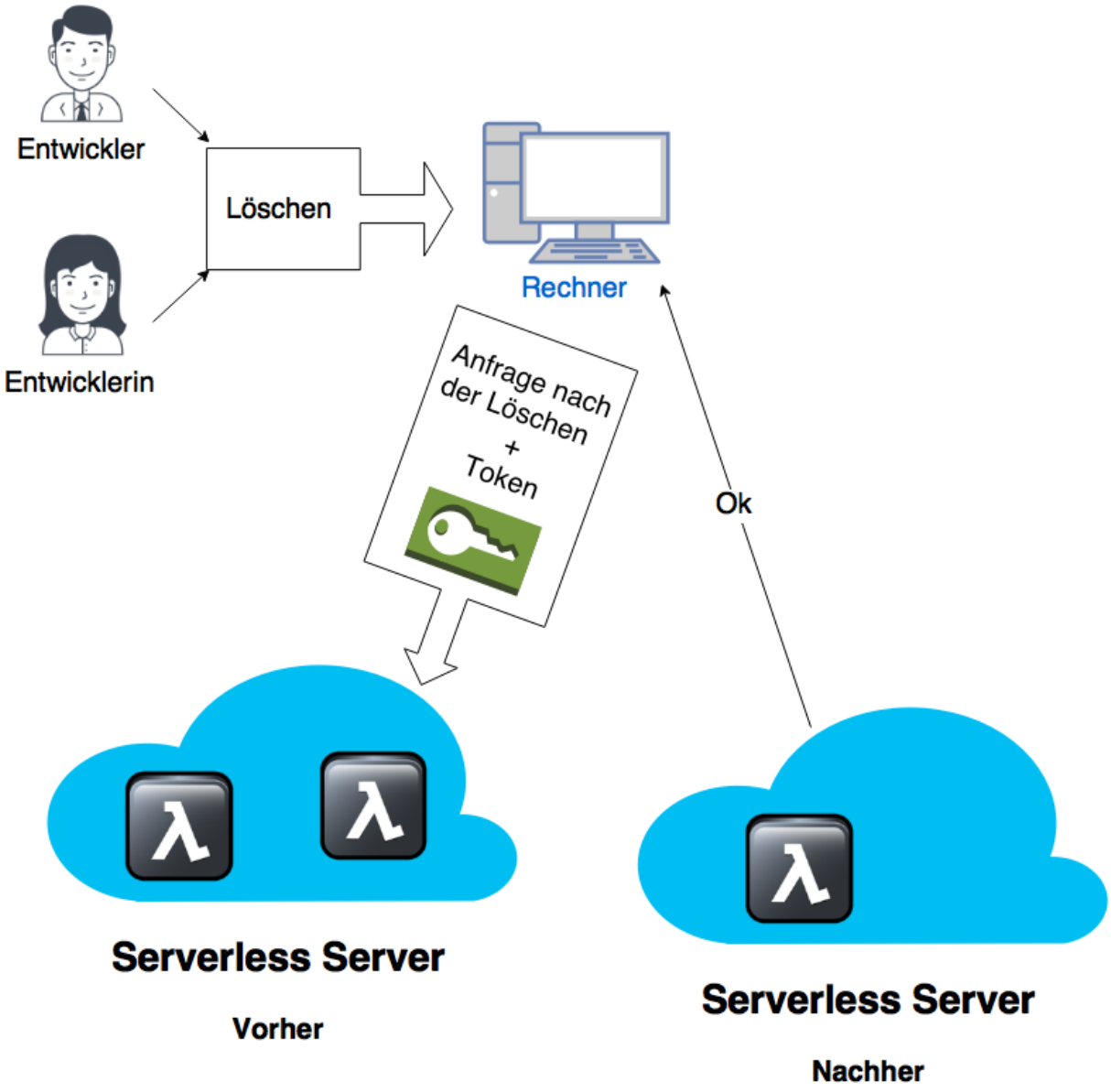
8.2.1 Hochladen



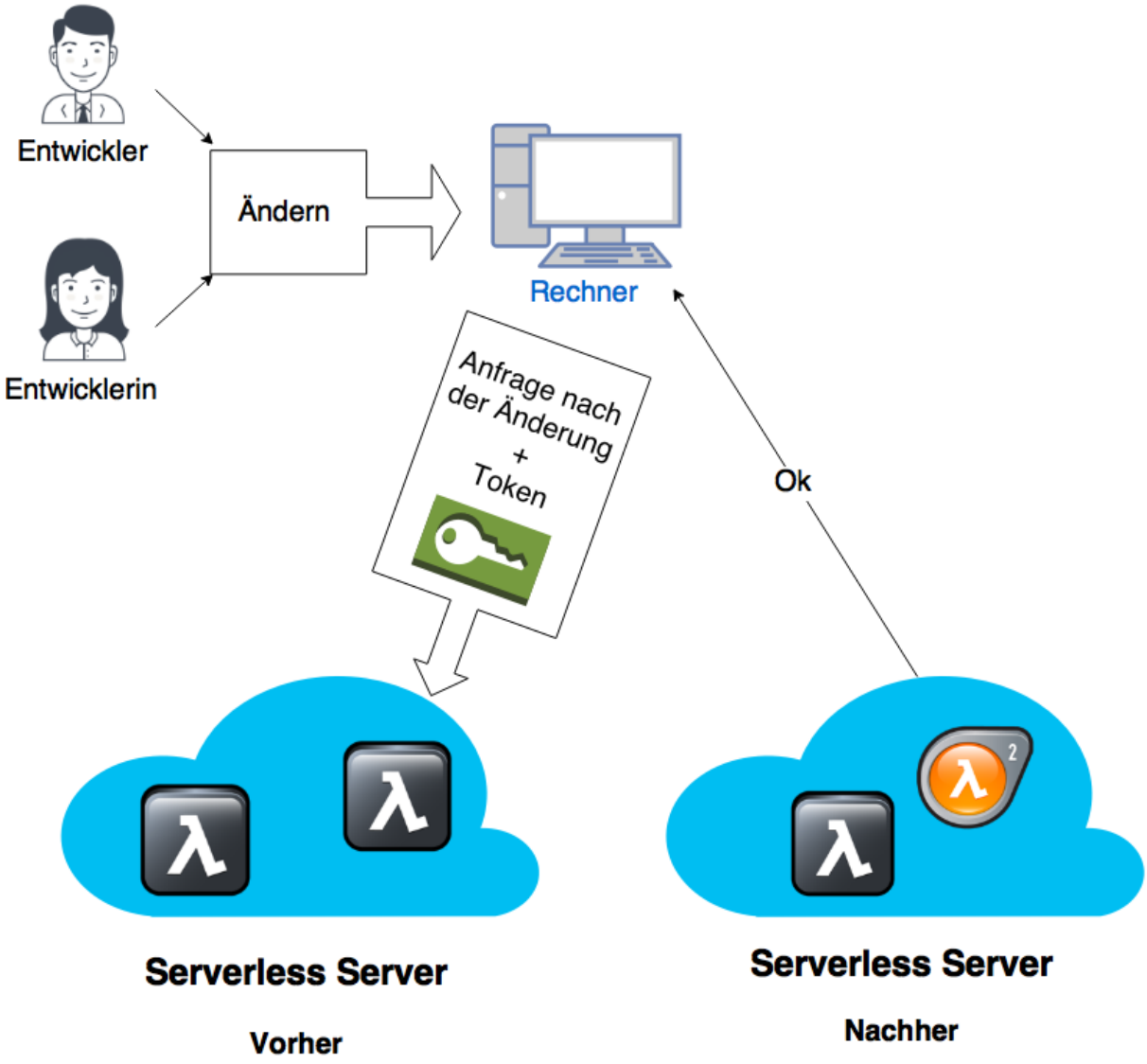
8.2.2 Ausführen



8.2.3 Löschen


Entwickler
Entwicklerin
Löschen
Rechner
Anfrage nach der Löschen + Token
Ok
Serverless Server
Vorher
Serverless Server
Nachher

8.2.4 Ändern

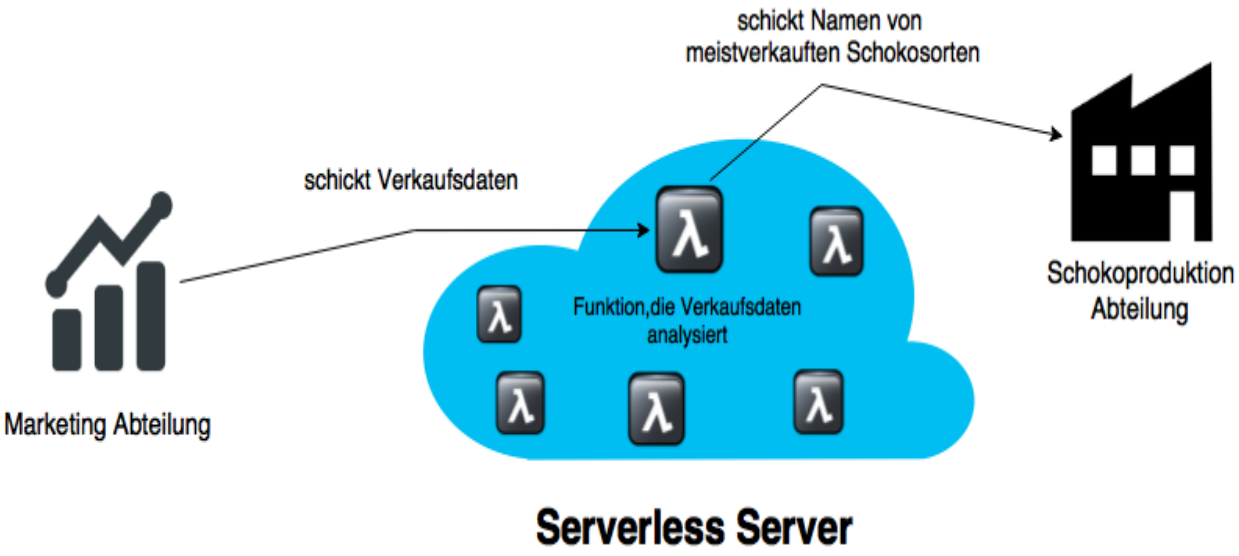

Entwickler
Entwicklerin
Ändern
Rechner
Anfrage nach der Änderung + Token
Ok
Serverless Server
Vorher
Serverless Server
Nachher

8.3 Anwendungsfall: Schokoladenfabrik

Es gibt eine intelligente Fabrik, die Schokoladenerzeugnisse produziert. Alle ihre Abteilungen sind mit Hilfe vom Serverless Server miteinander verbunden. Jede Abteilung schickt Daten an die Lambda-Funktionen, die sich auf dem Serverless Server befinden. Dort werden die Daten verarbeitet und anschließend die Ergebnisse zurück an die aufrufende oder andere Abteilungen geschickt.

8.4 Detailszenario: Marketing

Die Marketingabteilung schickt Verkaufsdaten an eine Lambda-Funktion auf dem Serverless Server. Diese Lambda-Funktion analysiert die Daten und berechnet, welche Schokoladensorten am besten verkauft werden und schickt die Namen der erfolgreichsten Sorten an die Produktionsabteilung. Die Marketingabteilung wird über das Ende des Vorgangs benachrichtigt.



Im Folgenden werden die Aufgaben der Bestandteile erklärt:

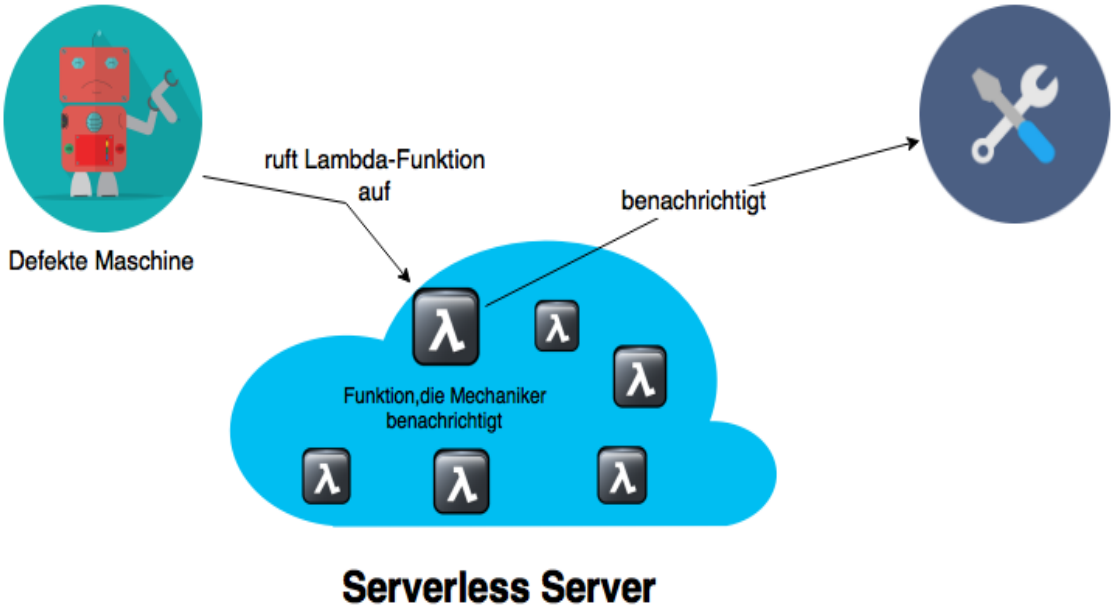
Gesamtumgebung Authentifiziert die Marketing-Abteilung, verarbeitet über REST API gesendete Daten der Marketingabteilung und holt sich aus bestimmten angegebenen Quellen benötigte Ressourcen. Danach startet sie die zugehörige Lambda-Funktion in einem Docker-Container mit der verarbeiteten Konfiguration, wartet auf die Beendigung und schickt über die REST API die Fehlermeldung oder das Ergebnis.

REST API Nimmt die Konfiguration für die Ausführung und den Authentifizierungstoken der Marketingabteilung entgegen und schickt ihr nach der Ausführung eine Rückmeldung.

Docker-Container Im Container wird die Lambda-Funktion, die die Verkaufsdaten analysiert und die meistverkauften Namen an die Schokoproduktion schickt, ausgeführt.

8.5 Detailszenario: Maschinenwartung

Wenn eine Maschine defekt ist, wird eine Nachricht an eine Lambda-Funktion auf dem Serverless Server geschickt. Diese verarbeitet die Nachricht und bestellt z.B die Ersatzteile für die Maschine nach und benachrichtigt Menschen für die Reperatur. Die Maschine wird über das Ende des Vorgangs benachrichtigt.



Aufgaben analog zum Detailszenario: Marketing.

Kapitel 9

Benutzerschnittstelle

9.1 Einleitung

Die Benutzer können die REST API zur Interaktion mit dem System verwenden. Im Folgenden wird zunächst eine Beschreibung der REST API gegeben, die JSON Schnittstelle beschrieben und danach auf ein Beispiel eingegangen, welches das System mithilfe des Programms cURL benutzt. Die Schnittstellen beschreiben dabei nur die Grundfunktionen aus dem Kapitel über Funktionale Anforderungen.

9.2 REST API

Die REST API unterstützt die neun Befehle GET, POST, PUT, PATCH, DELETE, HEAD, OPTIONS, CONNECT und TRACE. Die folgenden Befehle werden zum Datenaustausch mit dem System verwendet.

9.2.1 GET

Mit GET ist es möglich, Daten von einem Server zu empfangen. Damit kann der Benutzer beispielsweise die Ergebnisse der Ausführung einer Lambda-Funktion vom Server abrufen.

9.2.2 DELETE

Der Befehl DELETE ermöglicht es, Daten vom Server zu löschen. Entwickler können so nicht mehr benötigte oder fehlerhafte Lambda-Funktionen vom Server entfernen.

9.2.3 POST

POST können Entwickler verwenden, um dem System neue Lambda-Funktionen hinzuzufügen. Außerdem werden über diesen Befehl die benötigten Daten zur Ausführung einer Lambda-Funktion auf den Server hochgeladen.

9.2.4 PUT

Mit PUT ist es möglich Daten auf dem Server anzupassen. So kann ein Entwickler eine neuere Version einer Lambda-Funktion auf den Server laden und ausführen.

API

REST API for uploading, executing, deleting, changing and naming Lambdas.

Version 1.0.0

Paths

/lambdas

POST /lambdas

Summary

Upload Lambda Function.

Description

Uploads a Lambda function and config as JSON. If successful (200), it returns a master token for accessing, executing, deleting, changing the Lambda Function and which can be used to generate child tokens with specific needs to give execution authorization to third party. Also a link to the executable lambda function is described. If not successful because of an error parsing the JSON body, 422 will be returned with a specific error message. If not successful because a lambda function already exists, 500 will be returned with a description of options how to resolve the problem. Else the default error handling returns a triple (errorcode, message, fields).

Parameters

Name	Located in	Required	Schema
config	body	Yes	⇄ string (json)

Responses

Code	Description	Schema
201	Successful, return Token and Link.	⇄ ▼ { token: string link: string }
422	Error with JSON body.	⇄ ▼ string *specific error message*
500	Lambda Function under this name exists.	⇄ ▼ string *how to solve*
default	Unexpected Error	⇄ ▼ **Error** { code: integer message: string fields: string }

Try this operation

/lambdas/{name}

PUT /lambdas/{name}

Summary

Update Lambda Function.

Description

Updates a Lambda function or config with a JSON file. If successful, it returns 200. If not successful because of a wrong or no authentification token, 401 will be returned with a description of options how to resolve the problem. If not successful because of a missing lambda function, 404 will be returned with a description of options how to resolve the problem. If not successful because of an error parsing the JSON body, 422 will be returned with a specific error message. Else the default error handling returns a triple (errorcode, message, fields).

Parameters

Name	Located in	Required	Schema
token	header	Yes	⇄ string (token)
config	body	Yes	⇄ string (**json)**
name	path	Yes	⇄ string (string)

Responses

Code	Description	Schema
200	Successful.	
401	Authentication failed.	⇄ ▼ string *how to solve*
404	Not found.	⇄ ▼ string *how to solve*
422	Error with JSON body.	⇄ ▼ string *specific error message*
default	Unexpected Error	⇄ ▼ **Error {** code: integer message: string fields: string }

Try this operation

GET /lambdas/{name}

Summary

Show configuration of Lambda function.

Description

Shows configuration (JSON uploaded on creation or update) of Lambda function. Returns configuration as JSON if successful (200). If not successful because of a wrong or no authentification token, 401 will be returned with a description of options how to resolve the problem. If not successful because of a missing lambda function, 404 will be returned with a description of options how to resolve the problem. Else the default error handling returns a triple (errorcode, message, fields).

Parameters

Name	Located in	Required	Schema
token	header	Yes	⇄ string (token)
name	path	Yes	⇄ string (string)

Responses

Code	Description	Schema
200	Configuration of the Lambda Function	⇄ ▼ string (json) *current config*
401	Authentication failed.	⇄ ▼ string *how to solve*
404	Not found.	⇄ ▼ string *how to solve*
default	Unexpected error	⇄ ▼ **Error {** code: integer message: string fields: string }

Try this operation

DELETE /lambdas/{name}

Summary

Delete Lambda Function

Description

Deletes Lambda Function, returns 204 if successful. If not successful because of a wrong or no authentification token, 401 will be returned with a description of options how to resolve the problem. If not successful because of a missing lambda function, 404 will be returned with a description of options how to resolve the problem. Else the default error handling returns a triple (errorcode, message, fields).

Parameters

Name	Located in	Required	Schema
token	header	Yes	⇄ string (token)
name	path	Yes	⇄ string (string)

Responses

Code	Description	Schema
204	Deletion successful. No content.	
401	Authentication failed.	⇄ ▼ string *how to solve*
404	Not found.	⇄ ▼ string *how to solve*
default	Unexpected error	⇄ ▼ **Error {** code: integer message: string fields: string }

Try this operation

`POST /lambdas/{name}/execute`

Summary

Execute Lambda Function

Description

Executes Lambda Function with a specific configuration in JSON, returns result if successful (200). If not successful because of a wrong or no authentification token, 401 will be returned with a description of options how to resolve the problem. If not successful because of a missing lambda function, 404 will be returned with a description of options how to resolve the problem. If not successful because of an error parsing the JSON body, 422 will be returned with a specific error message. Else the default error handling returns a triple (errorcode, message, fields).

Parameters

Name	Located in	Required	Schema
token	header	Yes	⇄ `string (token)`
config	body	Yes	⇄ `string (json)`
name	path	Yes	⇄ `string (string)`

Responses

Code	Description	Schema
200	Results of the Lambda Function	⇄ ▼ { *results* }
401	Authentication failed.	⇄ ▼ string *how to solve*
404	Not found.	⇄ ▼ string *how to solve*
422	Error with JSON body.	⇄ ▼ string *specific error message*
default	Unexpected error	⇄ ▼ **Error** { code: integer, message: string, fields: string }

Try this operation

GET /lambdas/{name}/token

Summary

Generates a child-token to execute Lambda Function for third parties

Description

Generates a child-token to execute Lambda Function for third parties, returns token if successful (200). If no parameters are specified in JSON, a token is generated with no time or execution limit. If not successful because of a wrong or no authentification token, 401 will be returned with a description of options how to resolve the problem. If not successful because of a missing lambda function, 404 will be returned with a description of options how to resolve the problem. If not successful because of an error parsing the queue in JSON, 422 will be returned with a specific error message. Else the default error handling returns a triple (errorcode, message, fields).

Parameters

Name	Located in	Required	Schema
token	header	Yes	⇄ string (token)
parameters	query	No	⇄ string (json)
name	path	Yes	⇄ string (string)

Responses

Code	Description	Schema
200	Successful, returns Child Token.	⇄ ▼ string (token) *child token*
401	Authentication failed.	⇄ ▼ string *how to solve*
404	Not found.	⇄ ▼ string *how to solve*
422	Error with JSON.	⇄ ▼ string *specific error message*
default	Unexpected error	⇄ ▼ **Error {** code: integer message: string fields: string }

Try this operation

Models

Error

▼ **Error {**
code: integer
⇄ message: string
fields: string
}

9.3 Kurzüberblick: JSON Schnittstelle

Ein Kurzüberblick über die JSON Parameter wird hier gegeben. Eine längere Liste mit ausführlichen Eigenschaften wird im Entwurfsdokument des Systems zu finden sein.

9.3.1 Hochladen/Anpassen von Lambda-Funktionen

Der Absatz beschreibt das Format des Parameters "config" beim Hochladen der Lambda-Funktion.

```
{
   "name":"<Name_of_Lambda>", #required, id, e.g. lel
   "runtimeAttributes":{
         "language":"<Language_of_Lambda>", #required, programming language, e.g. python3
         "libraries":["library0", "library1"], #required if lamda uses libraries, libraries, e.g.
             lxml
         "code":"<Code>" #required, json escaped code, e.g. print("Hello world")
   }
}
```

Beim Anpassen der Lambda-Funktionen gelten dieselben Regeln wie beim Hochladen. Zu beachten ist dabei allerdings, dass ausschließlich die zu ändernden Parameter angegeben werden und andere nicht im Format auftauchen. Leer angegebene Parameter werden auch so interpretiert und der vorherige Stand mit einer leeren Version überschrieben.

9.3.2 Ausführung von Lambda-Funktionen

Der Absatz beschreibt das Format des Parameters "config" beim Ausführen der Lambda-Funktion. Diese darf leer bleiben, sofern keine Parameter benötigt werden. Eine leere Konfiguration sorgt für die einfache Ausführung der Lambda-Funktion ohne Parameter.

```
{
   "times": <number_of_times_running> #not required, number of times running the lambda function
       (0,maxtimes],
   "parameters_input":[<input0>,<input1>] #required if the lambda function needs arguments to
       run, e.g. "yolo", "swag", 3
}
```

9.4 Beispiel: REST API mit cURL

cURL ist ein Kommandozeilen-Programm zum Übertragen von Dateien in Rechnernetzen.

```
curl
```

9.4.1 Lambdas hochladen

Beispiel: Hochladen als Plaintext

```
curl -X POST -d '{"name": "lel", "runtimeAttributes":{"language":"python3","code":"print(\"Hello
    world\")"}}' http://server.less/lambdas
```

```
HTTP/1.1 201 Created
Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiw
ibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMH
rHDcEfxjoYZgeFONFh7HgQ
Location: /lambdas/lel
```

9.4.2 Lambdas verändern

Beispiel: Verändern als Plaintext

```
curl -X PUT -d '{"code":"print(\"Bye world\")"}' -H 'token: eyJhbGciOiJIUzI1NiIsInR5cC
I6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibm
FtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.T
JVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ' http://server.less/lambdas/lel
```

```
HTTP/1.1 200 OK
```

9.4.3 Lambdas anzeigen

Beispiel: einfache Anzeige

```
curl -H 'token: eyJhbGciOiJIUzI1NiIsInR5cC
I6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWR
taW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ' http://server.less/lambdas/lel
```

```
HTTP/1.1 200 OK
{"name": "lel",
"runtimeAttributes":{
   "language":"python2",
   "code":"print(\"Bye world\")"
}
}
```

9.4.4 Lambdas löschen

Beispiel: Löschen

```
curl -X DELETE -H 'token: eyJhbGciOiJIUzI1NiIsInR5cC
I6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWR
taW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ'
http://server.less/lambdas/lel
```

```
HTTP/1.1 204 No content.
```

9.4.5 Lambdas ausführen

Beispiel: 2x ausführen

```
curl -X POST -H 'token: eyJhbGciOiJIUzI1NiIsInR5cC
I6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWR
taW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ'
-d '{"times":"2"}' http://server.less/lambdas/lel/execute
```

```
HTTP/1.1 200
Bye world
Bye world
```

9.4.6 Tokens generieren

Beispiel: Tokens generieren

```
curl -X GET
token: eyJhbGciOiJIUzI1
NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4g
RG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ'
http://server.less/lambdas/lel/token
```

```
HTTP/1.1 200 OK eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkw
IiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2
cBab30RMHrHDcEfxjoYZgeFONFh7HgQ
```

Kapitel 10

Qualitätsbestimmungen

Produktqualität	**Sehr gut**	**Gut**	**Normal**	**Nicht relevant**
Funktionalität				
Angemessenheit		X		
Richtigkeit				X
Interoperablilität	X			
Ordnungsmäßigkeit				X
Sicherheit		X		
Konformität	X			
Zuverlässigkeit				
Fehlertoleranz		X		
Verfügbarkeit		X		
Benutzbarkeit				
Verständlichkeit			X	
Erlernbarkeit		X		
Bedienbarkeit		X		
Effizienz				
Zeitverhalten		X		
Verbrauchsverhalten		X		
Übertragbarkeit				
Anpassbarkeit	X			
Installierbarkeit	X			
Austauschbarkeit		X		

Tabelle 10.1: Übersicht der Qualitätsbestimmungskriterien

10.1 Funktionalität

10.1.1 Angemessenheit: gut

Definition
Eignung von Lambda-Funktionen für spezifizierte Aufgaben, zum Beispiel aufgabenorientierte Zusammensetzung von Lambda-Funktionen aus Teilfunktionen.

Begründung
Da das Ziel ist, einen möglichst allgemein benutzbaren Server zum Hochladen und Ausführen von Lambda-Funktionen zu entwickeln, müssen die Lambda-Funktionen des Servers für verschieden spezifizierte Aufgaben genügen.

10.1.2 Richtigkeit: nicht relevant

Definition
Liefern der richtigen oder vereinbarten Ergebnisse oder Wirkungen, zum Beispiel die benötigte Genauigkeit von berechneten Werten.

Begründung
Da die auf dem Server ausgeführten Lambda-Funktionen von externen Entwicklern stammen, sind diese für die Richtigkeit der zurückgegebenen Werte verantwortlich.

10.1.3 Interoperabilität: sehr gut

Definition
Fähigkeit, mit vorgegebenen Systemen zusammenzuwirken.

Begründung
Da das Produkt vor allem auf die Verwendung als externer Rechner abzielt, muss vor allem die Schnittstelle, aber auch das Gesamtprodukt sehr gute Interoperabilität besitzen.

10.1.4 Ordnungsmäßigkeit: nicht relevant

Definition
Merkmale von Software, die bewirken, dass die Software anwendungsspezifische Normen oder Vereinbarungen oder gesetzliche Bestimmungen und ähnliche Vorschriften erfüllt.

Begründung
Da die auf dem Server ausgeführten Lambda-Funktionen von externen Entwicklern stammen, sind diese für die Ordnungsmäßigkeit verantwortlich.

10.1.5 Sicherheit: gut

Definition
Fähigkeit, unberechtigten Zugriff, sowohl versehentlich als auch vorsätzlich, auf Programme und Daten zu verhindern.

Begründung Da in dem Produkt mehrere Lambda-Funktionen verschiedener Entwickler nebeneinander auf einem Server liegen, muss sichergestellt werden, dass weder Programm noch Mensch auf nicht auf Lambda-Funktionen Zugriff haben, wenn sie nicht berechtigt sind.

10.1.6 Konformität: sehr gut

Definition
Fähigkeit des Softwareprodukts, Standards, Konventionen oder gesetzliche Bestimmungen und ähnliche Vorschriften bezogen auf die Funktionalität einzuhalten.

Begründung
Da die Interoperabilität schwer ohne Konformität zu erreichen ist, ist die Einhaltung dieses Kriteriums wichtig.

10.2 Zuverlässigkeit

10.2.1 Fehlertoleranz: gut

Definition
Fähigkeit, ein spezifiziertes Leistungsniveau bei Software-Fehlern oder Nicht-Einhaltung ihrer spezifizierten Schnittstelle zu bewahren. Konformität: Grad, in dem die Software Normen oder Vereinbarungen zur Zuverlässigkeit erfüllt.

Begründung
Da verschiedene Lambda-Funktionen von verschiedenen Entwicklern gleichzeitig auf dem Server liegen, muss eine gewisse Fehlertoleranz geschaffen werden, da sonst z.B. bei einem Absturz viele Benutzer betroffen wären.

10.2.2 Verfügbarkeit: Gut

Definition
Wahrscheinlichkeitsmaß, dass das System bestimmte Anforderungen zu bzw. innerhalb eines vereinbarten Zeitrahmens erfüllt.

Begründung
Die Verfügbarkeit ist ein wichtiges Kriterium, dass mit der Begründung zur Fehlertoleranz einhergeht.

10.3 Benutzbarkeit

10.3.1 Verständlichkeit: normal

Definition Aufwand für den Benutzer, das Konzept und die Anwendung zu verstehen.

Begründung
Da das Hauptklientel des Produktes Entwickler sind, welche ein Verständnis für die Produktteile haben, ist der Aufwand normal hoch, um sich mit dem Produkt vertraut zu machen.

10.3.2 Erlernbarkeit: gut

Definition Aufwand für den Benutzer, die Anwendung zu erlernen.

Begründung
Da das Hauptklientel des Produktes Entwickler sind, welche ein Verständnis für die geläufigen Schnittstellen haben, ist der Aufwand gering, um sich mit den Schnittstellen vertraut zu machen.

10.3.3 Bedienbarkeit: gut

Definition
Aufwand für den Benutzer, die Anwendung zu bedienen.

Begründung
Da REST sehr geläufig und einfach ist und es viele Anleitungen und Bedienungsmöglichkeiten dazu existieren, ist der Aufwand sehr gering.

10.4 Effizienz

10.4.1 Zeitverhalten: gut

Definition
Antwort- und Verarbeitungszeiten sowie Durchsatz bei der Funktionsausführung.

Begründung
Da die Antwort- und Verarbeitungszeiten von der jeweiligen Lambda-Funktion abhängen ist diese von den Entwicklern und Benutzern abhängig. Das Übertragen und die Vorbereitungen für die Ausführung sollten allerdings minimal gehalten werden.

10.4.2 Verbrauchsverhalten: gut

Definition
Anzahl und Dauer der benötigten Betriebsmittel bei der Erfüllung der Lambda-Funktionen.

Begründung
Da das Verbrauchsverhalten ebenso mehrheitlich von den jeweiligen Lambda-Funktion abhängen, ist nur die Minimierung der Betriebsmittelbeschaffung zu beachten.

10.5 Übertragbarkeit

10.5.1 Anpassbarkeit: sehr gut

Definition
Fähigkeit der Software, diese an verschiedene Umgebungen anzupassen.

Begründung
Da das System auf verschiedenen Umgebungen laufen muss, ist die Anpassbarkeit ein wichtiges Kriterium.

10.5.2 Installierbarkeit: sehr gut

Definition
Aufwand, der zum Installieren der Software in einer festgelegten Umgebung notwendig ist.

Begründung
Da das System auf verschiedenen Umgebungen laufen muss, ist die Minimierung des Installationsaufwands ein wichtiges Kriterium.

10.5.3 Austauschbarkeit: gut

Definition
Möglichkeit, diese Software anstelle einer spezifizierten anderen in der Umgebung jener Software zu verwenden, sowie der dafür notwendige Aufwand.

Begründung
Da das System zur Ausführung von Lambda-Funktionen durch andere Systeme ersetzbar sein sollte, muss die Austauschbarkeit vereinfacht werden.

Kapitel 11

Testfälle und Szenarien

11.1 Testfälle

11.1.1 Funktionen

Nr.	Beschreibung	Kriterium
/T010/	Entwickler lädt eine Lambda-Funktion hoch und bekommt ein Zugriffs-Token zurück.	/F010/, /F011/
/T020/	Entwickler führt die Lambda-Funktion mit seinem Zugriffs-Token aus.	/F020/
/T021/	Entwickler stellt die Ausführungsumgebung für die Lambda-Funktion bereit.	/F021/
/T030/	Entwickler gibt Tokens für Benutzer aus.	/F110/
/T040/	Entwickler authentifiziert sich.	/F100/
/T050/	Entwickler löscht seine Lambda-Funktion.	/F030/, /F031/
/T051/	Entwickler ändert seine Lambda-Funktion.	/F040/
/T052/	Entwickler benennt seine Lambda-Funktion.	/F050/
/T053/	Entwickler konfiguriert seine Lambda-Funktion.	/F050/
/T054/	Entwickler bindet Daten für seine Lambda-Funktion ein.	/F050/
/T060/	Benutzer führt eine Lambda-Funktion mit einem Token aus.	/F020/
/T061/	Benutzer gibt an, wie oft die Lambda-Funktion ausgeführt werden muss.	

/T070/	Gleichzeitiger Zugriff auf eine hochgeladene Lambda-Funktion von mehreren Rechnern aus.	
/T080/	Terminierung einer Lambda-Funktion.	/F080/
/T090/	Eine Lambda-Funktion, die länger läuft als vom Server vorgegeben, wird terminiert.	/NF030/

11.1.2 Testfalle für unzulässige Aktionen

Nr.	Beschreibung	Kriterium
/UT010/	Benutzer ruft eine nicht-existierende Lambda-Funktion auf.	
/UT020/	Benutzer ruft eine Lambda-Funktion mit einem falschen Token auf.	/F020/
/UT030/	Entwickler gibt ein zu langes Zeitintervall für eine Funktionsausführung an.	bezieht sich auf /WT010/

11.1.3 Testfalle für Wünschkriterien

Nr.	Beschreibung	Kriterium
/WT010/	Entwickler gibt eine Höchst-Laufzeit für seine Lambda-Funktion an.	/F140/
/WT020/	Ein Zähler zu den Tokens wird erstellt.	/F150/
/WT021/	Entwickler ließt den Token-Zähler aus.	/F160/
/WT030/	Entwickler spezifiziert Firewalleinstellungen für seine Lambda-Funktion.	/F190/
/WT040/	Entwickler gibt ein Repository an.	/F170/
/WT050/	Entwickler lädt eine Lambda-Funktion von dem angegebenen Repository auf den Server.	/F170/
/WT060/	Benutzer oder Entwickler spezifiziert die Zeit für die Bereitstellung des Ergebnisses der Lambda-Funktion.	/F200/
/WT070/	Entwickler kauft die Tokens für seine Lambda-Funktion.	/F210/

/WT080/	Entwickler ersetzt die aktuelle Lambda-Funktion durch ihre frühere Version.	/F180/
/WT090/	Serverbetreiber greift auf die Statistiken für die Lambda-Funktionen zu.	/F220/

11.2 Testszenarien

11.2.1 Entwickler lädt seine Lambda-Funktion hoch

Ein Entwickler lädt eine Lambda-Funktion mit ihrem Namen, Angabe der Sprache, ggf. Adresse von zusätzlichen Dateien und dem Code selbst mit dem Befehl POST hoch. Falls die Eingabe korrekt ist, bekommt er eine Bestätigung, den Speicherort der erstellten Lambda-Funktion und den Zugriffs-Token als Antwort, sonst bekommt er eine Fehlermeldung mit dem Fehlercode zurück.

Wunschfunktion 1: der Entwickler gibt anstatt von Code die Adresse von eines Repository an, in welchem die Lambda-Funktion gespeichert ist.

Wunschfunktion 2: der Entwickler kann zusätzlich eine maximale Laufzeit und Firewall-Einstellungen für seine Lambda-Funktion angeben.

1. /T010/ Entwickler lädt eine Lambda-Funktion hoch und bekommt ein Zugriffs-Token zurück.
2. /T021/ Entwickler stellt die Ausführungsumgebung für die Lambda-Funktion bereit.
3. /T040/ Entwickler authentifiziert sich.
4. /T052/ Entwickler benennt seine Lambda-Funktion.
5. /T053/ Entwickler konfiguriert seine Lambda-Funktion.
6. /T054/ Entwickler bindet Daten für seine Lambda-Funktion ein.
7. /WT040/ Entwickler gibt ein Repository an.
8. /WT010/ Entwickler gibt eine Laufzeit für seine Lambda-Funktion an.
9. /WT030/ Entwickler spezifiziert Firewalleinstellungen für seine Lambda-Funktion.

11.2.2 Entwickler ändert seine Funktion

Ein Entwickler lädt eine veränderte Lambda-Funktion mit ihrem Namen, dem Zugriffs-Token und dem Code selbst mit dem Befehl PUT hoch. Falls die Eingabe korrekt ist, bekommt er eine Bestätigung als Antwort, sonst bekommt er eine Fehlermeldung mit dem Fehlercode zurück.

Wunschfunktion 1: der Entwickler gibt anstatt von dem geänderten Code die Adresse eines Repository an, in welchem die veränderte Lambda-Funktion gespeichert ist.

1. /T051/ Entwickler ändert seine Lambda-Funktion.
2. /T040/ Entwickler authentifiziert sich (mittels eines Zugriffs-Token).
3. /WT050/ Entwickler lädt eine veränderte Lambda-Funktion von dem angegebenen Repository auf den Server.
4. /WT080/ Entwickler ersetzt die aktuelle Lambda-Funktion durch ihre frühere Version.

11.2.3 Entwickler ruft die Einstellungen seiner Funktion auf

Ein Entwickler gibt den Befehl GET, den Namen der Lambda-Funktion und den Zugriffs-Token ein. Falls die Eingabe korrekt ist, bekommt er eine Bestätigung und die Einstellungen der Lambda-Funktion als Antwort. Im anderen Fall bekommt er eine Fehlermeldung mit dem Fehlercode zurück.

11.2.4 Entwickler löscht seine Funktion

Ein Entwickler gibt den Befehl DELETE, den Namen der Lambda-Funktion und den Zugriffs-Token ein. Falls die Eingabe korrekt ist, bekommt er eine Bestätigung, und alle laufende Instanzen der Lambda-Funktion werden folgend beendet (nach /F031/). Im anderen Fall bekommt er eine Fehlermeldung mit dem Fehlercode zurück.

1. /T050/ Entwickler löscht seine Lambda-Funktion.
2. /T040/ Entwickler authentifiziert sich (mittels eines Zugriffs-Token).

11.2.5 Benutzer ruft eine Lambda-Funktion auf

Ein Benutzer authentifiziert sich mittels eines Tokens und gibt den Befehl POST mit der Adresse der zu ausführenden Funktion, den möglichen Parametern und der Anzahl der Ausführungen ein. Die Lambda-Funktion wird ausgeführt, falls sie unter gegebener Adresse existiert, und wird eventuell nach einer vom Server oder vom Benutzer spezifizierten Zeit abgebrochen. Als Antwort bekommt der Benutzer eine Bestätigung und das Ergebnis der Ausführung(en), oder er bekommt eine Fehlermeldung mit dem Fehlercode zurück.

Wunschfunktion 1: der Benutzer kann die Zeit angeben, wann die Ergebnisse der Lambda-Funktion bereitgestellt werden.

1. /T060/ Benutzer führt eine Lambda-Funktion mit einem Token aus.
2. /T061/ Benutzer gibt an, wie oft die Lambda-Funktion ausgeführt werden muss.
3. /T080/ Terminierung einer Lambda-Funktion.
4. /T090/ Eine Lambda-Funktion, die länger läuft als vom Server vorgegeben, wird terminiert.
5. ggf. /T070/ Gleichzeitiger Zugriff auf eine hochgeladene Lambda-Funktion von mehreren Rechnern aus.
6. /UT020/ Benutzer ruft eine nicht-existierende Lambda-Funktion auf.
7. /UT030/ Benutzer ruft eine Lambda-Funktion mit einem falschen Token aus.
8. /WT060/ Benutzer spezifiziert die Zeit für die Bereitstellung des Ergebnisses der Lambda-Funktion.
9. /WT010/ Abbruch von Lambda-Funktionen.

11.2.6 Entwickler generiert einen Child-Token für Benutzer

Ein Entwickler gibt seinen Zugriffs-Token, die Funktionsparameter und die Adresse der Lambda-Funktion mit dem Befehl GET ein. Als Antwort erhält er eine Bestätigung und einen Child-Token für die zukünftigen Benutzers oder er bekommt eine Fehlermeldung mit dem Fehlercode zurück.

Wunschfunktion 1: der Entwickler kann zusätzliche Tokens für seine Lambda-Funktion kaufen.

Wunschfunktion 1: der Entwickler kann einen Zähler für seine Child-Tokens erstellen.

1. /T030/ Entwickler gibt Tokens für Benutzers aus.
2. /T040/ Entwickler authentifiziert sich.
3. /WT070/ Entwickler kauft die Tokens für seine Lambda-Funktion.
4. /WT020/ Entwickler erstellt einen Zähler zu den Tokens.

Kapitel 12

Entwicklungsumgebung

12.1 Betriebssysteme

- Mac OS X
- Windows
- Linux

12.2 Entwicklung

- Entwicklungsumgebung IntelliJ für die Java-Entwicklung.
- Entwicklungsumgebung Eclipse für die Java-Entwicklung.

12.3 Versionsverwaltung

- Git

12.4 Sonstige verwendete Software

- LaTeX für die zu erstellenden Dokumente (z. B. Pflichtenheft)
- Docker

12.5 Hardware

- Standard-PCs
- Standard-Laptops
- Server zum Testen der Software

Glossar

API ist ein Programmteil, der von einem Softwaresystem anderen Programmen zur Anbindung an das System zur Verfügung gestellt wird..

App ist ein Computerprogramm, das eine für den Anwender nützliche Lambda-Funktion ausführt.

Benutzer ist ein User, der Lambdas aus ihrem entsprechenden Docker-Container mit Parameter aufruft.

Container ist ein lauffähiges, virtuelles Betriebssystem..

Docker ist eine Open-Source-Software, die dazu verwendet werden kann, Anwendungen mithilfe von Betriebssystemvirtualisierung in Containern zu isolieren.

Eclipse ist eine Entwicklungsumgebung der Firma Eclipse Foundation, die vor allem für Java-Entwicklung verwendet wird, aber auch viele andere Sprachen unterstützt.

Entwickler ist ein User, der Lambdas durch die geeignete API-Schnittstelle hochlädt, verändert, löscht oder ausführt.

gekapselt isoliert.

Industrie 4.0 ist ein Begriff, der auf die Forschungsunion der deutschen Bundesregierung und ein gleichnamiges Projekt in der Hightech-Strategie der Bundesregierung zurückgeht, er bezeichnet ebenfalls eine Forschungsplattform. Die industrielle Produktion soll mit moderner Informations- und Kommunikationstechnik verzahnt werden. Technische Grundlage hierfür sind intelligente und digital vernetzte Systeme. Mit ihrer Hilfe soll eine weitestgehend selbst-organisierte Produktion möglich werden: Menschen, Maschinen, Anlagen, Logistik und Produkte kommunizieren und kooperieren in der Industrie 4.0 direkt miteinander. Durch die Vernetzung soll es möglich werden, nicht mehr nur einen Produktionsschritt, sondern eine ganze Wertschöpfungskette zu optimieren. Das Netz soll zudem alle Phasen des Lebenszyklus des Produktes einschließen – von der Idee eines Produkts über die Entwicklung, Fertigung, Nutzung und Wartung bis hin zum Recycling.

IntelliJ ist eine Entwicklungsumgebung speziell für Java von der Firma JetBrains, die die wichtigsten Java-Technologien unterstützt.

Internet of Things ist die Zusammenarbeit von physischen Geräten, Fahrzeugen (auch als "connected devices" und "smart devices" bezeichnet), Gebäuden und anderen - ausgestattet mit Elektronik, Software, Sensoren, Motoren und Netzwerkverbindung, was es den Objekten ermöglicht, Daten zu sammeln und auszutauschen.

Konfiguration ist eine Menge von Einstellungen, die eine Lambda-Funktion zur Ausführung benötigt.

Lambda-Funktion ist eine zustandslose Funktion.

python2 ist eine universelle, üblicherweise interpretierte höhere Programmiersprache..

Rechenzeit (auch Laufzeit) ist die Zeit, die ein Programm bzw. eine Lambda-Funktion zur Ausführung benötigt oder benötigt hat.

Repository ist ein verwaltetes (Online-) Verzeichnis zur Speicherung und Beschreibung von digitalen Objekten für ein digitales Archiv.

REST (Representional State Transfer) ist eine einfache Alternative zu anderen Verfahren zur Kommunikation zwischen Maschinen. Der größte Teil der von REST benötigten Infrastruktur ist bereits vorhanden, was die Verwendung sehr einfach macht.

Server ist ein Computerprogramm oder ein Computer, der Computerfunktionalitäten wie Dienstprogramme, Daten oder andere Ressourcen bereitstellt, damit andere Computer oder Programme („Clients“) darauf zugreifen können, meist über ein Netzwerk.

Serverbetreiber ist die Person, die die Software auf einem Server ausführen lässt und anderen Benutzern die Funktionalität der Software bereitstellt.

Serverless ist ein Konzept in der Softwaretechnik in dem Entwickler für das Managen und Provisionieren der Server nicht mehr selbst verantwortlich sein müssen.

skalierbar siehe Skalierung.

Skalierung (auch Skalierbarkeit) ist die Fähigkeit eines Systems aus Hard- und Software, die Leistung durch das Hinzufügen von Ressourcen – z. B. weiterer Hardware – in einem definierten Bereich proportional (bzw. linear) zu steigern.

Softwareverteilung (englisch software deployment) nennt man Prozesse zur Installation von Software auf Rechnern.

Token ist eine Komponente zur Identifizierung und Authentifizierung von Benutzern. In diesem Fall in Form von einer Textdatei..

Webanwendung ist ein Anwendungsprogramm, das beim Benutzer in einem Webbrowser abläuft bzw. dargestellt wird.

zustandslos ist ein System oder Protokoll, welches keine Zustandsinformationen speichert.