# Microservices -Projektarbeit

Wintersemester 2020/2021

Metz, Patrick

# Einleitung

Im Folgenden wird, die von mir angefertigte Lösung zur Projektarbeit des Kurses „Projektseminar – Microservices“ (Wintersemester 2020/21, gehalten von Dr. Thomas Soddemann), vorgestellt.

Zunächst wird erläutert, wie die Infrastruktur hochgefahren, getestet und heruntergefahren werden kann. Danach werden die einzelnen Komponenten vorgestellt.

# Inbetriebnahme

Die anzufertigende Service-Infrastruktur, kann unter diesem Link als Git-Repository betrachtet, und von diesem Link als Zip-Datei (ca. 112MB) heruntergeladen werden.

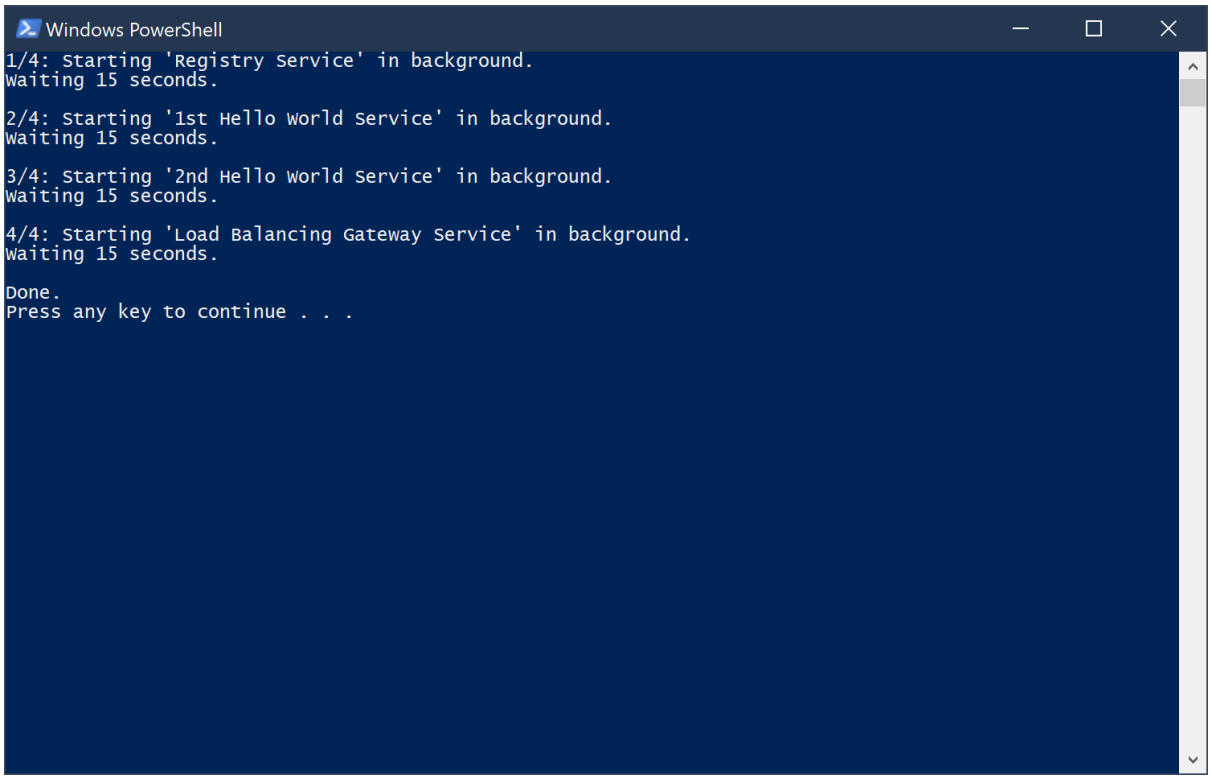
Zum einen beinhaltet die Zip-Datei die Services als Quellcode im Klartext, sowie als kompilierte Jar-Dateien, welche Java 15 zur Ausführung benötigen.

Und zum anderen enthält sie, entsprechend benannte, ausführbare Shell-Skripte für Linux, sowie Batch-Skripte für Windows, mit denen die Service-Jar-Dateien ausgeführt, getestet und beendet werden können.
Diese benötigen, um zu funktionieren, das Kommandozeilen-Programm cURL, sowie die freien lokalen Ports 8080 und 8081.

Die folgenden Anweisungen wurden exemplarisch auf einem Windows-System ausgeführt. Die mitgelieferten Linux-Skripte funktionieren analog.

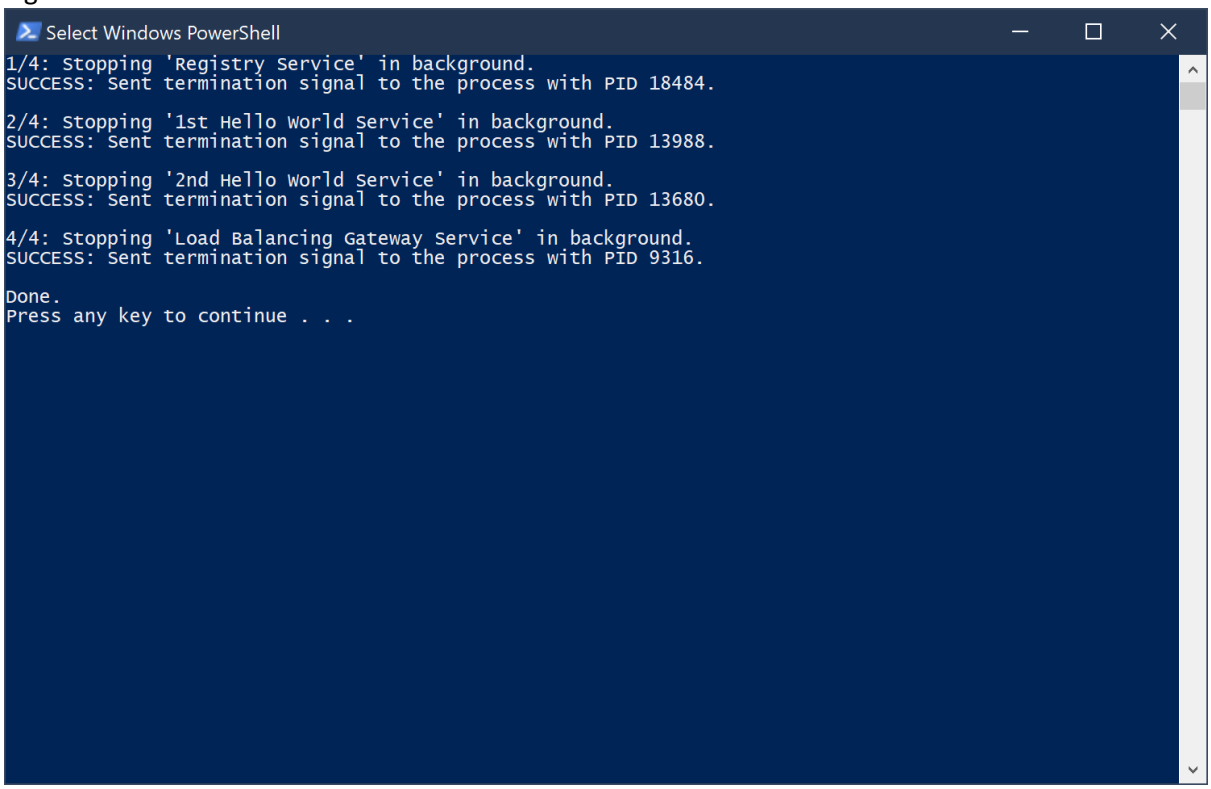
## Starten der Services

Mittels Aufruf von *„start_windows_services.bat“* werden alle vier beteiligten Services gestartet.




## Beenden der Services

Durch *„stop_windows_services.bat“* werden alle noch aktiven Services, mittels Betriebssystem-Signalen beendet.



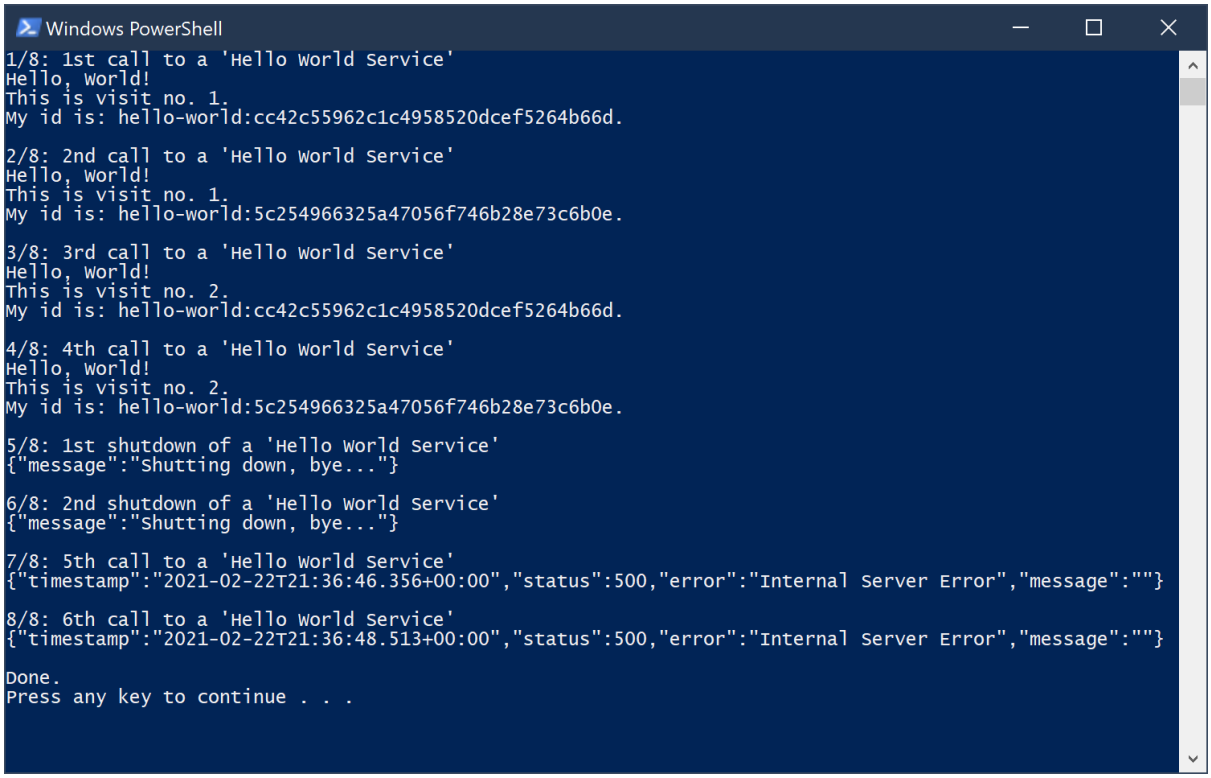
# Funktionstest

Via *„test_windows_services.bat"* werden die gestarteten Services in acht Schritten auf die geforderte Funktionalität überprüft.

- Schritte 1 bis 4
  - Vier sukzessive identische HTTP/GET-Anfragen an „*localhost:8080/hello"*
  - Diese werden abwechselnd an die beiden gestarteten „Hello World"-Services vermittelt – erkennbar an den alternierenden Service-IDs, sowie daran, dass deren Besucherzähler nur alle zwei Aufrufe hochzählen.

- Schritte 5 bis 6
  - Zwei identische aufeinanderfolgende HTTP/POST-Anfragen an „*localhost:8080/actuator/shutdown"*
  - Auch diese werden auf die „Hello World"-Services verteilt, worauf jene herunterfahren.

- Schritte 7 bis 8
  - Zwei erneute HTTP/GET-Anfragen an „*localhost:8080/hello"*
  - Diese können, nun erkennbar, nicht mehr vermittelt werden.
  - Somit sind alle Tests abgeschlossen und die verbleibenden Services können, wie unter „Beenden der Services" geschildert, heruntergefahren werden.



# Architektur

## Allgemein

Es handelt sich um eine, auf Microservices basierende, Cloud-Infrastruktur, welche mittels Komponenten des Java-Frameworks *Spring Boot*, sowie *Netflix Zuul* realisiert wurde.

Bei der implementierten Variante greifen Clients von außerhalb der Infrastruktur auf, ihnen bekannte, API-Endpunkte zu.

Die zugehörigen HTTP-Anfragen gehen bei einem zentralen Gateway-Service ein, welcher bei einem Registrierungsservice, im inneren der Infrastruktur, geeignete Services anfragt, die die Client-Anfragen bedienen können.

Diese geeigneten Services werden vom Gateway sowohl lastverteilt als auch transparent zum Bearbeiten bzw. Beantworten der Anfragen an die Clients vermittelt.

In den folgenden Abschnitten werden die beteiligten Microservices, bezüglich ihrer Funktionalität, vorgestellt.

## Registry- und Discovery-Service

Als sogenannter *Registry- und Discovery-Service* kommt *Netflix Eureka* zum Einsatz. Bei diesem registrieren sich die *Hello-World-Service*-Instanzen, die schlussendlich die Client-Anfragen bedienen. Andere Services können, die hier registrierten Services, abfragen und deren bereitgestellten Endpunkte zum Erledigen beliebiger Aufgaben nutzen.

Unter *„localhost:8081“* steht die Instanz dieses Services zur Verfügung, und lässt unter dieser Adresse auch Einsicht, auf die angemeldeten *Hello-World-Service*-Instanzen und den abfragenden *Gateway-/Load-Balancer-Service* nehmen.

## Gateway- / Load-Balancer-Service

Eine *Netflix-Zuul*-Instanz übernimmt sowohl die Aufgabe des, bereits beschriebenen, Gateways, als auch die des Lastausgleichs. Sie fragt in regelmäßigen Intervallen beim *Registry- und Discovery-Service* an, welche *Hello-World-Services* gerade zur Verfügung stehen, und vermittelt deren „*/hello“*-, sowie „*/actuator/shutdown“*-Endpunkte an anfragende Clients.

Als Lastausgleich ist ein sogenanntes *Round-Robin*-Verfahren konfiguriert, dass die *n* verfügbaren *Hello-World-Services* reihum, d.h. sukzessive mit jeder Anfrage abwechselnd, an die Clients vermittelt, so dass jeder *Hello-World-Service* eine Last von *1/n* Anfragen trägt.

Seine Instanz stellt ihre vermittelnde Tätigkeit unter „*http://localhost:8080“* als Einstiegspunkt für externe Clients zur Verfügung.

## Hello-World-Service

Die Instanzen des *Hello-World-Service* registrieren sich, nach dem sie gebootet wurden, beim *Registry- und Discovery-Service* und werden fortan vom *Gateway-/Load-Balancer-Service* zum Abarbeiten von Client-Anfragen eingesetzt.

Er ist so konfiguriert, dass jede gestartete Instanz zufallsbasierte Ports und eine ebenfalls randomisierte Identität erhält, und stellt zwei Endpunkte für Clients zur Verfügung:

- */hello* via HTTP/GET
    - Ausgabe einer Begrüßung inklusive der Instanzidentität sowie eines instanzeigenen Aufrufzählers.

- */actuator/shutdown* via HTTP/POST
    - Abmeldung vom *Registry- und Discovery-Service*, sowie anschließendes Herunterfahren der Instanz

## Quellcode-Komentare

Im Quellcode der Services befinden sich noch Kommentare, für die wichtigsten Annotationen, in den *„*Controller.java“*-Dateien, sowie die wichtigsten Konfigurationsdirektiven in den *„application.yml“*-Dateien.