BACHELOR PAPER

Thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science in Engineering at the University of Applied Sciences Technikum Wien - Degree Program Computer Science

Currying the web: A custom Java REST framework - built on functional paradigms - compared to Spring Boot: Performance, Developer Guidance and Ease of Use

Nico Lerchl
2110257236

Advisor: Dipl.-Ing. (FH) Bernhard Wallisch

May 6, 2024

Declaration

"As author and creator of this work to hand, I confirm with my signature knowledge of the relevant copyright regulations governed by higher education acts (see Urheberrechtsgesetz / Austrian copyright law as amended as well as the Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien as amended).

I hereby declare that I completed the present work independently and that any ideas, whether written by others or by myself, have been fully sourced and referenced. I am aware of any consequences I may face on the part of the degree program director if there should be evidence of missing autonomy and independence or evidence of any intent to fraudulently achieve a pass mark for this work (see Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien as amended).

I further declare that up to this date I have not published the work to hand nor have I presented it to another examination board in the same or similar form. I affirm that the version submitted matches the version in the upload tool."

Kurzfassung

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Keywords: REST, Java, Funktionale Programmierung, Spring Boot

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Keywords: REST, Java, functional programming, Spring Boot

Acknowledgements

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Contents

1 Introduction

Web development is a huge part of the software industry. Most of the time, the server part of a web application is built using the MVC pattern and object oriented programming [**?**]. Functional programming is not used as much in web development, in the last few years however, functional programming has been gaining a lot of popularity with languages such as Haskell, Scala and Clojure but also with functional concepts being added to object oriented languages like Java and C# [**?**].

Combining functional programming with web development using more widely used languages e.g. Java is a scarcely researched topic which this thesis aims to explore and shine some light on. The goal is to build a REST framework using functional paradigms and compare it to Spring Boot in terms of performance, developer guidance and ease of use.

2 Literature review

2.1 REST

Representational State Transfer (REST) is the state-of-the-art way to build the server part of a client-server-architecture and it is most likely only going to get bigger in the industry [**?**]. It was first described by Roy Fielding in his doctoral dissertation in 2000. REST is based on the following properties [**?**]:

- Client-server - The client and the server are separated and can be developed independently.
- Stateless - The server does not store any client state. Every request contains all the information the server needs to process it.
- Cache - Responses can be cached to improve performance.
- Uniform Interface - The interface between the client and the server is uniform and simple.

2.2 Functional programming

2.2.1 General

Functional programming - unlike procedural or object oriented programming - is not based on the Turing machine, but rather on lambda calculus. Lambda calculus, developed by Alonzo Church in the 1930s, is a mathematical system later proven - by Turing himself - to be equivalent to the Turing machine [**?**].

The base principles of functional programming are [**?**]:

- Immutability - Variables are not changed after they are assigned a value.
- Pure functions - Functions do not have side effects and always return the same output for the same input.
- Higher order functions - Functions can be passed as arguments to other functions.
- Referential transparency - A function call can be replaced by its return value without changing the program's behavior.

A big part of functional programmings is the concept of monads which have their roots in category theory. They allow for encapsulating side effects in a pure way. For a container to be a monad it has to abide by the laws of left identity, right identity and associativity. [**?**]

2.2.2 Web development

Yesod is a web framework for the before mentioned functional programming language Haskell. It allows developers to build entire websites using templates and widgets or RESTful web services. Additionally, Yesod offers the ability to persist data using Haskell's type system into PostgreSQL, SQLite, MySQL, and MongoDB. [**?**]

2.2.3 In Java

The introduction of lambda expressions in Java 8 brought functional programming to the Java ecosystem. Where before developers had to use anonymous classes to pass functions as arguments, they can now use lambda expressions. This also shifts the view point of passing an object that carries functionality to passing behavior itself. The concept behind these lambda expressions in Java is called functional interfaces. Functional interfaces are interfaces that have exactly one abstract method. They can be annotated with `@FunctionalInterface`.

Also new to Java 8 is the Streams API. It hides away the iteration over collections by offering many higher order functions. Additionally, streams are only evaluated when a terminal operation, such as collecting, counting or averaging is called, implementing the - before mentioned - functional programming principle of lazy evaluation. [**?**]

Java 8 also saw the introduction of the `Optional`-class. It is a container that may or may not contain a non-null value. TODO

2.3 Spring Boot

Spring Boot is a framework for building stand-alone web applications and RESTful web services in Java. Unlike Spring Framework there is zero requirement for XML configuration. It can be deployed using an internal web server or going the classic route of deploying a war file onto an external web server. [**?**]

3 Curryful

Curryful tries to combine Java's simplicity with functional paradigms to build better performant and unwanted behavior diminishing REST APIs, whilst offering a great developer experience through easy use. As of now, when developing a REST API using Curryful, one will require two dependencies: Curryful-commons and curryful-rest. Having a dependency just for common functional programming tools in Java allows for building any kind of application using functional paradigms or enriching existing REST frameworks with functional utilities. Curryful is hosted under a GitHub organization and can be found at `https://github.com/Curryful`.

3.1 Curryful-commons

Curryful-commons offers a basis to writing functional Java code. It provides a common abstract class for Monads and implementations for the Maybe and Try monads. Since Java contains the before mentioned monadic Optional class, which shares striking resemblance to the Maybe monad, the Maybe monad offers a function to create a Maybe from an Optional. The Try monad allows for elegantly handing exceptions in a functional way instead of throwing and catching them.

Though Java is always pass-by-value, anything but primitive types essentially behave like pass-by-reference as object's references are passed by value. [**?**] This allows for objects to be mutated, without the caller knowing, causing unwanted behavior and it also goes against the functional principle of purity. To avoid this, curryful-commons offers mutable and immutable variants of Java's `ArrayList` and `HashMap` where the mutable collections inherit from the immutable ones. This way, the mutable ones can be passed in place of their immutable counterparts, making it impossible for the called function to mutate the parameters. This concept of "immutability downcasts" is also referenced in "Javari: Adding Reference Immutability to Java" [**?**] by Matthew S. Tschantz and Michael D. Ernst from the University of Cambridge.

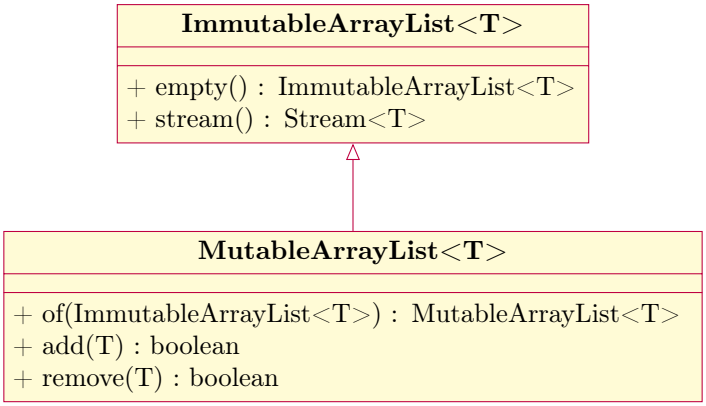

Figure 1: (Incomplete - for clarity) class diagrams of ImmutableArrayList and MutableArrayList and their relationship

Additionally, curryful-commons provides a higher order function Y-combinator to allow for recursion when using lambda expressions. This is necessary because Java does not allow referencing a lambda expression before it has been fully defined, e.g. from within itself.

```
Y(readHttp).apply(ImmutableArrayList.empty());
```

Listing 1: Example of using the Y-combinator

3.2 Curryful-rest

In curryful-rest, everywhere possible, lambda expressions are used to define functionality. This is done in a way where each lambda expression only takes one input. This allows for currying whenever desired.

3.2.1 HTTP

Curryful-rest implements its own HTTP handling to build functionally from the ground up. The `HTTP`-Class offers methods to parse incoming requests and serialize outgoing responses. All fetching of information from the request such as method, path, headers, etc. is done using regular expressions. As visible in figure **??** and **??**, Headers, path and query parameters are stored in immutable collections to mitigate unwanted behavior. All other types used do not allow for mutation as their fields are `final` and no mutating methods are provided.

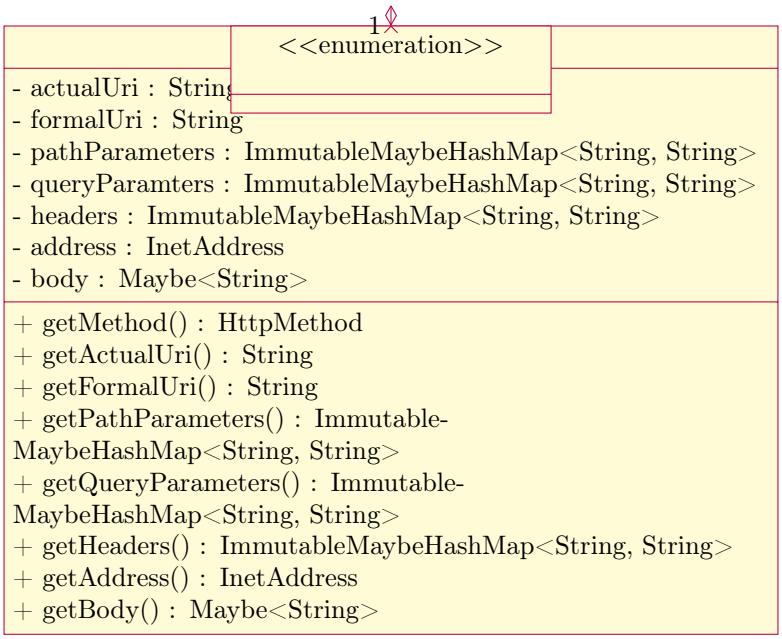

Figure 2: (Incomplete - for clarity) class diagrams of classes representing HTTP requests in curryful-rest

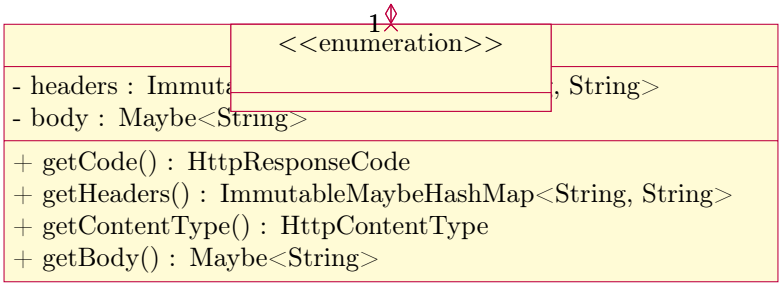

Figure 3: (Incomplete - for clarity) class diagrams of classes representing HTTP responses in curryful-rest

3.3 Routing

The `Router`-Class provides one function that can be curried to register middleware and endpoints. The curried function can then be called for each connection and its raw HTTP. It is here where the `HTTP`-Class' functions are used. To find the formal endpoints for the actual endpoint, each registered formal endpoint's URI is turned into a regular expression and the actual endpoint's URI matched against, as well as checking if the endpoint' HTTP methods match. A formal URI might looks like this: `/hello/:name` where `hello` is a static part and `:name` is a dynamic part in the form of a path parameter. `/hello/:name` then gets turned into `/hello/(?<name>[^/?]+)`. Query parameters are not part of the formal URI and therefore removed from the actual URI before matching. If no formal endpoint is found, a 404 response is sent. If a formal endpoint is found, the same regex can be used to extract the path parameters, also the query parameters are extracted and all other relevant data is stored in a `HttpContext` object to then be passed to the endpoint's rest function. Figure **??** shows the classes used to register endpoints, this will however be further explained - with code - later.

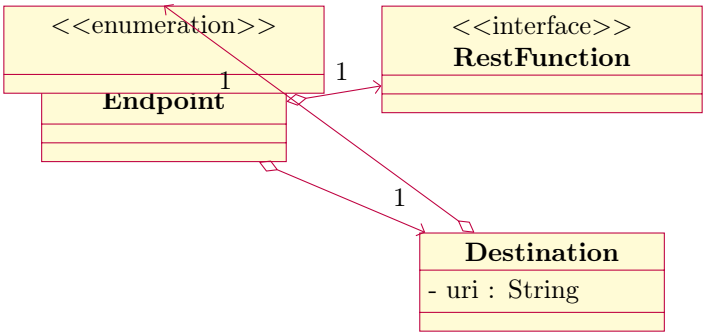

Figure 4: (Incomplete - for clarity) class diagrams of classes representing HTTP requests in curryful-rest

3.4 Middleware

Middleware can be registered as `PreMiddleware` or `PostMiddleware` where the former receives the same `HttpContext` as the endpoint's rest function and returns an `HttpContext`, allowing for manipulation for general use cases such as logging or authentication. The latter also receives the `HttpContext` but returns an `HttpResponse`, allowing for the same freedom of manipulating the response, returned by the endpoint's rest function, as well as the `HttpContext`.

`PreMiddleware` is executed as soon as the `HttpContext` can be created, i.e. after finding the formal `Destination` to be able to extract the path parameters. `PostMiddleware` is executed after the endpoint's rest function has been called and returned an `HttpResponse`.

3.5 Server

The `Server`-Class provides the entry point to the REST application. This is where a Socket is opened, connections accepted and then handed off to the `Router` to be processed. The `Server` not only passes through the Middleware created by the developer but also registers its own Middleware in the form of a `PreMiddleware` to log incoming requests and a `PostMiddleware` to log outgoing responses.

Each new connection is accepted and handled in a new thread to provide resilience against failure. If any errors where to occur during routing or the executing of developer-implemented `RestFunction`s, the server will respond with a 500 status code but continue running to accept new connections.

4 Research questions and hypotheses

4.1 Research questions

All of the following questions will be answered by comparing Curryful to Spring Boot.

1. Will building a REST API using functional paradigms, from the ground up, result in a more performant application?
2. Will building a REST API using functional paradigms naturally guide the developer to eliminate unwanted behavior?
3. Will the developer experience benefit from developing a REST API using functional paradigms from the ground up?

4.2 Hypotheses

1. Building a REST API using functional paradigms, from the ground up, will result in a more performant application. Functional programming's keenness on mutability and mitigation of side effects makes concurrency and parallelism disregard the need for locks or synchronization. In the context of FaaS, startup times are lower and the cold start problem is eased.

2. Building a REST API using functional paradigms will naturally guide the developer to eliminate unwanted behavior. Common pitfalls of object oriented programming such as mutable state, side effects, null references and unchecked exceptions will be avoided. Null values are practically omitted and exceptions handled gracefully through the use of monads. The stateless design functional programming promotes will also synergize with REST's statelessness principle.

3. The developer experience will benefit as error handling becomes more natural and testing becomes less of a burden because functions will always produce the same output for the same input and not rely on external factors. Additionally, the declarative nature of functional programming will increase conciseness and expressiveness directly leading to less lines required to achieve similar results.

5 Methodology

ChatGPT 4 was prompted to generate two simple REST APIs. One for a todo list application and one for playing Yahtzee. Each API was generated twice, once using Curryful and once using Spring Boot. The prompts were kept as identical as possible besides, having to provide more information about Curryful, as ChatGPT does not know about this new framework, resulted in changes that had to be made.

The prompts, also found in the appendix, and any changes that had to be done to the generated code are available in the projects' repositories:

- Todo list in Curryful
- Todo list in Spring Boot
- Yahtzee in Curryful
- Yahtzee in Spring Boot

5.1 Performance

5.1.1 Response time

To measure response time, a folder or requests was sent to both applications with 100 iterations using Postman. From the results, the total duration and average response time of both applications can be compared to determine which application is more performant regarding response times.

5.1.2 Cold start

To measure cold start time, the applications were each started 100 times using a bash script. As both Curryful and Spring Boot log their startup time, it can be read by the script and the average cold start time calculated.

5.2 Provoking unwanted behavior using invalid requests

The prompts were held short and only ask for necessary implementation details, leaving room to play with for the AI generating the code.

Using Postman, both application generated by ChatGTP were sent requests, trying to provoke unwanted behavior. The applications were restarted after each request to empty the in-memory storage, which was one of the implementation details.

The requests can be found in the form of JSON, exported by Postman as a collection v2.1 here

5.2.1 Todo list

The requests are:

- POST request where "completed" is a string
- POST request where a car is added

- POST request where the body is empty
- GET request for id -1
- PUT request for id -1
- POST request to toggle completed for id -1
- DELETE request for id -1
- GET request for id test
- GET request for id 999999999999999999999

5.2.2 Yahtzee

5.3 Static code analysis

The generated code was analyzed using SonarCloud to determine both cyclomatic and cognitive complexity. The cyclomatic complexity is a measure of the number of linearly independent paths through a program's source code and therefore also represents the number of test cases required to reach a coverage of 100%. Cognitive complexity describes how hard it is for a person to understand the code. (TODO: Might need citation)

An additional measure to determine developer experience is the number of lines of code and statements, which Sonar also provides. To guarantee a fair comparison, all projects were formatted according to the same rules:

- lines must not be longer than 120 characters
- each added part of a method chain should be in a new line, unless the entire chain is not longer than 120 characters

6 Results

6.1 Performance

6.1.1 Response time

6.1.2 Cold start

The scripts were run on a laptop, AMD Ryzen 5 5600U, 8.0GiB memory running Ubuntu 23.10 and on a desktop computer, Intel Core i9 9990k, 32GiB memory running Ubuntu 20 through WSL2.

Curryful Laptop: 239.74ms

6.2 Provoking unwanted behavior using invalid requests

6.2.1 Todo list

The project using Curryful responded with the expected status code seven out of nine times. The two times it did not respond with the expected status code, the application crashed and did not respond at all. This is an oversight by ChatGTP which tried parsing the id path parameter to an integer, without checking if it is actually an integer:

```
context.getPathParameters().get("id").map(Integer::parseInt)
```

More than just an oversight by ChatGPT, this is a massive error in the Curryful framework itself.

The project using Spring Boot responded with the expected status code four out of nine times. The POST request adding a car created a todo without a title. All requests trying to access a non-existent todo with the id -1, returned 200, making it seem like the todo actually exists.

Request	Expected status code	Actual status code	
		Curryful	Spring Boot
POST request where "completed" is a string	400	400	400
POST request where a car is added	400	400	200
POST request where the body is empty	400	400	400
GET request for id -1	404	404	200
PUT request for id -1	404	404	200
POST request to toggle completed for id -1	404	404	200
DELETE request for id -1	404	404	200
GET request for id test	400	-	400
GET request for id 999999999999999999999	400	-	400

Table 1: Results of the invalid requests

6.2.2 Yahtzee

6.3 Static code analysis

7 Discussion

8 Conclusion

9 Future work

10 Self-review

List of Figures

Listings

List of Tables

A Prompts

A.1 Todo list

A.1.1 Curryful

I have provide you with: - The curryful-commons library, which curryful-rest builds upon - The curryful-rest library, which is a simple rest framework, leaving out the utility classes Http and Uri, you don't need to know about those - An example application using curryful-rest

The framework utilizes the concepts of functional programming, so use functional programming principles you know about as well as the principles already applied in the example code.

I want you to write a simple rest api for a todo list. The todo list should be stored in memory and should be accessible through the following endpoints:

- GET /todos - POST /todos - GET /todos/:id - PUT /todos/:id - DELETE /todos/:id - POST /todos/:id/toggle

A todo is described by the following json object: "id": 1, "title": "Buy milk", "completed": false

As you can see, the framework does not handle json on its own. Please use Jackson to parse json to objects and objects to json.

A.1.2 Spring Boot

I want you to write a simple rest api for a todo list using spring boot. The todo list should be stored in memory and should be accessible through the following endpoints:

- GET /todos - POST /todos - GET /todos/:id - PUT /todos/:id - DELETE /todos/:id - POST /todos/:id/toggle

A todo is described by the following json object: "id": 1, "title": "Buy milk", "completed": false

A.2 Yahtzee

A.2.1 Curryful

A.2.2 Spring Boot

B Cold startup time measuring script

B.1 Curryful

!/bin/bash

Define the location of the Maven wrapper and the jar file MVNW$_PATH =$ "$./mvnw$" $JAR_FILE =$ "$target/todo-0.0.1-SNAPSHOT.jar$" $LOG_DIR =$ "$target$" $LOG_FILE =$ "LOG$_DIR/log.txt$"

Ensure Maven wrapper is executable chmod +x $MVNW_PATH$

Build the project $MVNW_PATH clean package$

Array to hold all the start times declare -a startTimes

Loop to start the application 100 times for i in 1..100 do echo "Iteration i"

Start the application in the background and redirect output to log file java -jar $JAR_FILE >$LOG$_FILE 2> 1 PID =$!

Wait for a few seconds to let the application initialize sleep 0.5

Kill the application kill $PID wait$PID

Read the last line from the log file lastLine=$(tail -1$LOG$_FILE) REGEX =$ "$Curryful server started in (.*) ms$"

Check if the line matches the expected startup message if [[$lastLine =$ REGEX]]; then startTimes+=$(BASH_REMATCH[1]) else echo$ "$Failed to extract start time in iteration$i" fi

Clear the log file to avoid confusion in the next iteration $> LOG_FILE done$

Calculate the average start time total=0 count=$startTimes[@]$

for time in "$startTimes[@]$" $do total =$(echo "$total$+time" | bc) done

if [$count - gt 0]; then average =$(echo "scale=2; $total$/count" | bc) echo "Average cold start time: $average ms$" $else echo$ "$No valid start times recorded.$" fi

B.1.1 Spring Boot

!/bin/bash

Define the location of the Maven wrapper and the jar file MVNW$_PATH$ = "./mvnw" JAR_FILE = "target/todo − 0.0.1 − SNAPSHOT.jar" LOG_DIR = "target" LOG_FILE = "LOG$_DIR$/log.txt"

Ensure Maven wrapper is executable chmod +x $MVNW_PATH$

Build the project $MVNW_PATH clean package$

Array to hold all the start times declare -a startTimes

Loop to start the application 100 times for i in 1..100 do echo "Iteration i"

Start the application in the background and redirect output to log file java -jar JAR_FILE >LOG$_FILE$2 > 1PID =!

Wait for a few seconds to let the application initialize sleep 2

Kill the application kill $PID wait$PID

Read the last line from the log file lastLine=($tail$−1LOG$_FILE$)$REGEX$ = "$Started Application in (.*) seconds$"

Check if the line matches the expected startup message if [[$lastLine =$ REGEX]]; then startTimes+=($BASH_REMATCH$[1])$else echo "Failed to extract start time in iteration$i" fi

Clear the log file to avoid confusion in the next iteration > $LOG_FILE done$

Calculate the average start time total=0 count=$startTimes$[@]

for time in "$startTimes$[@]" $do total$ =(echo "$total$+time" | bc) done

if [$count - gt 0$]; $then average$ =(echo "scale=2; $total$/count" | bc) echo "Average cold start time: $average seconds$" $else echo$ "$No valid start times recorded.$" fi