FOM Hochschule für Oekonomie & Management

university location Bonn

Bachelor Thesis

in the study course Wirtschaftsinformatik

to obtain the degree of

Bachelor of Science (B.Sc.)

on the subject

Microservice Monitoring - Identification of Demands and a prototypical Implementation

by

Sophie Neck

Advisor:	Prof. Dr. Peter Steininger
Matriculation Number:	538289
Submission:	October 13, 2022

Contents

Index of Figures

Index of Tables

Index of Abbreviations

Index of Code Listings

1 Abstract

Abstract

2 Theory

2.1 Lexer

The first part of a code generator is the lexer. A lexer gets a file or in this case a string as input and divides this input into a series of tokens. So the input `@contains:apple:` becomes the tokens: '@', 'contains', ':', 'apple' and ':'. These tokens are not interpreted yet but are only being recognized as separate characters. To achieve this in code the crate logos is used, to avoid writing redundant code. To understand the code written in lexer.rs what follows is a short explanation of how this crate is used in the context of this prototype. To define tokens, Logos can be added to the derive statement of an enumeration and a matching rule can be defined using a literal string or a regular expression. For example, in line 73 of code listing 1, a literal string is used to recognize the colon token, and line 33 uses a regular expression to recognize decimals between 0 and 1. It also calls an arbitrary function to_float (code 1, 17-19) to define that in this case the data should be cast into the datatype f64. Logos also requires an error type (code 1, 78-80), which is also used to skip whitespaces (cf. '3 Layers of Monitoring | Keytorc Software Testing Services' n.d., n.p.).

Code Listing 1: Token defintions

```
16
17 @Slf4j
18 @Configuration
19 @Data
⋮
26         this.environment = environment;
27     }
```

Source: AppConfiguration.java

These tokens are then compiled in a list and passed over to the parser as the work of the lexer is done.

3 Summary

Summary

Appendix

Appendix 1: AppConfiguration.java

```java
package de.telekom.bonicheckprototype.configuration;

import lombok.Data;
import lombok.extern.slf4j.Slf4j;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.core.env.Environment;

import javax.annotation.PostConstruct;

/*
For loading System Environment variables after Spring or Test is started
And Bean Configuration used by the Application.
Reading the variables from the CONFIG_LOCATION (set in ApplicationInit)
 */

@Slf4j
@Configuration
@Data
@PropertySource("file:${CONFIG_LOCATION}")
public class AppConfiguration {

    private final Environment environment;

    public AppConfiguration(Environment environment) {
        this.environment = environment;
    }

    public String oAuth_Issuer_Uri;


    @PostConstruct
    public void appConfigInit() {
```

```
        log.info("---------Set Configuration after Spring Boot
            Start----------");
        oAuth_Issuer_Uri = environment.getProperty("
            OAuth_Issuer_Uri");

        log.info(this.toString());
        ApplicationInit.getPropertyLoader().printValues();
        log.info("---------Set Configuration after Spring Boot
            is set----------");

    }

}
```

Appendix 2: Criteria Catalog.pdf

Criteria Catalog

- Based on requirements for the prototypical implementation -

The aim of this criteria catalog is to verify a successful prototypical implementation of a monitoring solution.

During a workshop with the business side and the operations team, the following criteria regarding the prototypes and the monitoring solution were defined, and the system environment was specified.

No.	Description	Criteria	Fulfilled?
1.	Prototypes		
1.1.	Close-to-production implementation with Spring Webflux	The Spring Webflux Framework is used as the basis for the prototypes.	
1.2.	Close-to-production implementation with the modules developed internally	The following internally developed modules are used: - Security Module - Mongo Module - Objectmapper Module - Validation Module	
1.3.	Trace simulation across multiple microservices	Two microservices were created that can simulate a trace across multiple microservices.	
1.4.	Database connection	- Service uses the MongoDB Docker container locally and can access it via a repository.	
		- Service uses the existing MongoDB in the cloud and can access it via a repository.	
1.5.	Kibana (APM) connection	- Service runs locally with Docker container for Kibana APM	
		- Service runs in the cloud with existing Kibana.	
		- Data is sent correctly through microservices to Kibana.	
2.	System environment		
2.1.	Local development	The prototypes can be run in the local system environment for development purposes.	
2.2.	Delivery to the cloud	The existing CI/CD solution is used to build, to test and to deploy the prototypes.	
2.3.	Runs in the cloud	The prototypes can be run in the cloud system environment.	

3.	Monitoring values		
3.1.	Runtime request	The duration of the request to the partner system is measured.	
3.2.	Response duration	The duration to process the response is measured.	
3.3.	Request/response successful	The successful requests/responses are measured.	
3.4.	Request/response with error	The unsuccessful requests/responses are measured.	
3.5.	Number of occurring errors	The number of occurring errors is measured.	
3.6.	Number of occurring warnings	The number of occurring warnings is measured.	
3.7.	Microservice availability	The availability of the system can be measured.	
3.8.	GUI (Graphical User Interface) availability	The availability of the GUI can be measured. *-Only applicable to microservices with GUI-*	
3.9.	CPU usage in %	The CPU usage is measured in %.	
3.10.	Memory usage in %	The memory usage is measured in %.	

Bibliography

3 Layers of Monitoring | Keytorc Software Testing Services. en. In: (), p. 8

Declaration in lieu of oath

I hereby declare that I produced the submitted paper with no assistance from any other party and without the use of any unauthorized aids and, in particular, that I have marked as quotations all passages which are reproduced verbatim or near-verbatim from publications. Also, I declare that the submitted print version of this thesis is identical with its digital version. Further, I declare that this thesis has never been submitted before to any examination board in either its present form or in any other similar version. I herewith agree that this thesis may be published. I herewith consent that this thesis may be uploaded to the server of external contractors for the purpose of submitting it to the contractors' plagiarism detection systems. Uploading this thesis for the purpose of submitting it to plagiarism detection systems is not a form of publication.

Rheinbreitbach, 13.10.2022	
(Location, Date)	(handwritten signature)