universität innsbruck

High-Level Modeling and Low-Level Adaption of Serverless Function Choreographies

Benjamin Walch

Bachelor Thesis

Supervisor:
Dr. Shashko Ristov
Department of Computer Science
Universität Innsbruck

Innsbruck, February 13, 2020

Eidesstaatliche Erklärung

Ich erkläre hiermit an Eides statt durch meine eigenhändige Unterschrift, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Alle Stellen, die wörtlich oder inhaltlich den angegebenen Quellen entnommen wurden, sind als solche kenntlich gemacht.

Ich erkläre mich mit der Archivierung der vorliegenden Bachelorarbeit einverstanden.

Datum	Unterschrift

Abstract

"Run code, not Server" is the most recent term of cloud computing providers. With the rise of the serverless technology during the last years, *FaaS* became more and more popular. The Distributed and Parallel Systems Group from University of Innsbruck are doing research in this topic. One of the results of this research is the (generic) specification of the "Abstract Function Choreography Language" (AFCL). Also, a Java API, to describe serverless application workflows programmatically. The product which results in using that API is the workflow being described in AFCL in a generated YAML or JSON file. This (textual) workflow definition can then be further processed by (other) machines.

The aim of this bachelor project is to develop a visual workflow editor, which makes modeling of workflows possible at a high level of abstraction. Additionally, composed workflows can be saved, reopened and edited. The tool should also be able to optimize given workflows for multiple *FaaS* provider(s) in case of quotas and limits, and also in case of performance.

Contents

1 Introduction

With the rise of *FaaS*, a high level of flexibility in execution of code appeared. Global Players like Amazon, Google, IBM and Microsoft jumped on the train and provide their infrastructure to Developers, able to deploy functions and execute them in the cloud. Each system has its own definitions on how to define, deploy, run and execute code.

1.1 Motivation

2 Background

2.1 FaaS

2.2 AFCL

3 Overview

4 Implementation

The Implementation of a Graphical User Interface as well as the optimization for Workflows were the main goals of this bachelor thesis.

4.1 Requirements

- design for the overall application [coreUI]
- clean and intuitive UI [bootstrap]
- ability to add components/menu points easily [modularity]
- good user experience [SPA]
- frameworks: open source, large community (future-safe)
- cross-browser [?]
- performance [?]

4.2 Frontend

One of the main parts of the application is the frontend. Since a web based frontend which, runs in the user's web browser, is a hard requirement, the core technologies are limited to HTML, CSS and JavaScript. Before the implementation - in a prototyping phase - different frameworks and libraries have been researched, tested and selected. For the selection, the following criteria have been taken into account:

- open-source
- good documentation
- large and active community (future-proof)
- stars on github / downloads on npmtrends
- performance
- learning curve

This criteria ensure the usage of well-documented and open-source software and makes it easier for others (dps) to extend the application later. The result of this phase was a working prototype as proof-of-concept. In the following sections, the setup and each technology and corresponding frameworks are described in detail.

Setup

The setup of the frontend application is a selection of tools and technologies for modern and complex web development. Under the hood, JavaScript and several JavaScript Frameworks are in action to control the application with all its UI components. To give the application its look and feel, CSS and some CSS Frameworks are in use. As a basis, Node.js' package manager, npm, is used to manage and resolve the dependencies. Additionally, npm's CLI is used as tool to execute development and/or build tasks.
Webpack does much of heavy-lifting, by bundling all sources with the assets into single files, and execute additional build steps by utilizing so-called 'loaders'. Babel is used as such a webpack loader, for compiling the ECMAScript and React JSX source code to browser-compatible JavaScript. The same applies for the CSS extension SASS, which is also integrated as a webpack loader, and is used to make the process of styling the user interface more efficient.

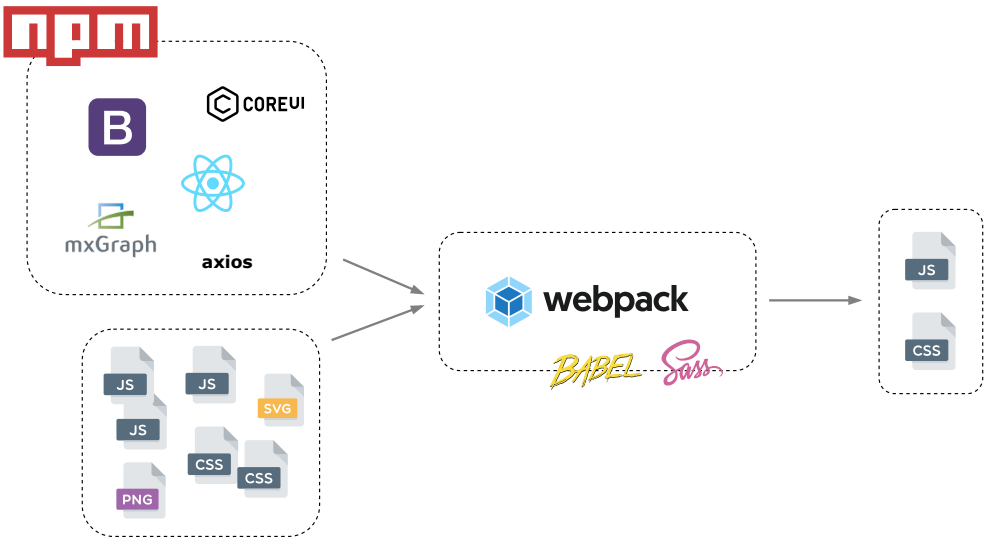

Figure 0.1: frontend development workflow

npm

The node package manager[1] is the world's largest software registry, where open-source software packages of developers and companies are shared all over the world. Over the last years, npm became a de-facto standard for package management in JavaScript development.

webpack

webpack is a module bundler, its main purpose is to bundle JavaScript code for usage in a browser.[2] In particular, multiple modules (often hundreds of) with dependencies [to each other] are processed and bundled into a few files. To be able to process other types of files than JavaScript or JSON, webpack offers the opportunity to configure a **loader**. In this application, the following loaders are configured:

- babel-loader, to transform ECMAScript and React JSX to browser-compatible JavaScript
- sass-loader, to transform SASS to CSS

[1] https://www.npmjs.org
[2] https://webpack.js.org

- css-loader, to transform CSS to CommonJS
- file-loader, to handle static resources like images and fonts

ECMAScript

The scripting language specification ECMAScript (ES) was created to standardize JavaScript. With the release of ES6 (also known as ECMAScript 2015), features like class declarations, module imports and arrow function expressions became possible. After ES6, every year a new edition of the ECMAScript standard was finalized and released, offering new features. Worth mentioning here is the rest/spread operator released with ES9, which occurs a lot in the source code of this thesis.
Since current browsers only have partial support of ECMAScript, a compile 'transcompiler' (or transpiler) is needed to transform the ECMAScript source code to JavaScript common browsers are capable of interpreting. This process of compiling is done with Babel[3], which is configured to not only compile ECMAScript, but also JSX to JavaScript.

SASS

CSS, in its pure form, reaches its limits when one thinks about using variables, functions or nested rules. SASS[4] is a stylesheet language, which is compiled to CSS and offers the mentioned and even more features. A lot of CSS Frameworks also offer its source code in SASS with a large variable set which makes it easy to customize.

4.3 Web Interface

The layout of the web interface is based on coreUI[5], a admin panel template, built on top of the web toolkit Bootstrap[6]. A clean and nested structure of components, which is typical for a React app, guarantees modularity and forms the whole application. The main component has four sub-components - where each of them consists again of multiple sub-components represent the core features of the app.

Dashboard

The dashboard is the entry point where the user lands after accessing the app. The purpose of this component is to give the user a quick overview of the application, provide short informational texts and links to the specific modules.

[3]https://babeljs.io
[4]https://sass-lang.com
[5]https://coreui.io
[6]https://getbootstrap.com

Bibliography

[1] (2020, February) Webpack. [Online]. Available: https://webpack.js.org