

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 24, 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

In the FaaS world, there exist many providers who offer serverless execution of code (functions) in the cloud. Each of them has its own specifications and definitions on how functions are deployed and executed, and how workflows are created and executed.

The Distributed and Parallel Systems group developed the "Abstract Function Choreography Language" (AFCL). It is a specification for describing serverless workflows. Also, a Java API, to describe serverless application workflows programmatically, was developed. The product which results in using that API is a workflow being described in AFCL in a text file. Until now, the workflow described in the text file has to be created manually (by editing the file directly), or a programmer has to write Java code which utilizes the API to generate the file.

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. The tool should not only be able to load, display and save workflows, but also optimize workflows for multiple FaaS provider(s) in case of quotas and limits, and also in case of performance.

Contents

Introduction	5
Background	7
1 FaaS	7
2 AFCL	7
Implementation	9
3 Requirements	9
4 Architecture	9
5 Frontend	9
5.1 Setup	10
5.2 Data model	12
5.3 Graph	13
5.4 Web Interface	14
6 Backend	17
6.1 General	17
6.2 Maven	17
6.3 Servlets	17
6.4 Persistence	17
6.5 Api	18
7 Continuous Delivery	18
Improvements	19
Conclusion	21
Bibliography	23

Introduction

”Run code, not Server” is the most recent term in the cloud computing environment. With the rise of the serverless technology during the last years, *FaaS* became more and more popular. With the serverless technology, a high level of flexibility in execution of code became possible. 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 and run a function. Many providers also offer a tool to model a workflow and execute it (Amazon Step Functions, IBM Cloud Functions, Microsoft Azure Functions, Google Cloud Functions, ...)

The distributed and parallel systems group developed a specification

Background

1 FaaS

2 AFCL

Implementation

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

3 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 Architecture

Operating mostly decoupled from one another, the application architecture consists of two main components - Backend and Frontend - which transfer data between each other by an API. The backend is responsible for converting and optimizing workflows at low-level. This is done by utilizing the AFCL API. The frontend is responsible for visualizing the data and controls the application's user interface.

5 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.

A prototyping phase was the kickoff for the development: Different frameworks and libraries have been researched, tested and selected. For the selection, the following criteria have been taken into account:

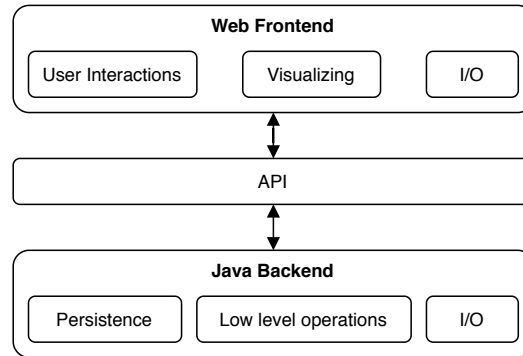


Figure 0.1: application architecture

- open-source
- good documentation
- large and active community (future-proof)
- stars on github / downloads on npmtrends
- performance
- pre-knowledge of the developer / 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 development phase was a working prototype as proof-of-concept. In the following sections, the setup and each chosen technology and frameworks are described in detail.

5.1 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 consisting of several UI components. To give the application its look and feel, CSS and some CSS Frameworks are in use. The base forms the node package manager (npm) which is used to manage and resolve the dependencies. Additionally, npm's CLI is used as tool to execute development and 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.

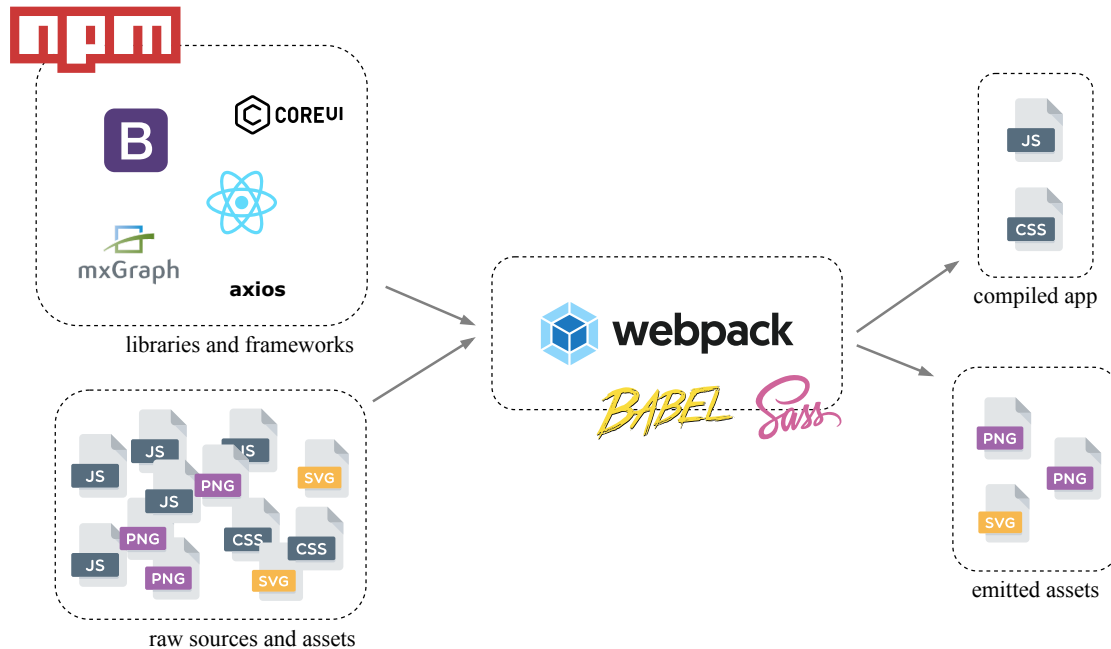


Figure 0.2: frontend development toolchain

npm

The node package manager¹ 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 the 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:

¹<https://www.npmjs.org>

²<https://webpack.js.org>

- babel-loader, to transform ECMAScript and React JSX to browser-compatible JavaScript
- sass-loader, to transform SASS to CSS
- 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 'transcompiler' (or transpiler) is needed to transform the ECMAScript source code to JavaScript, which common browsers are capable of interpreting. This compiling process is done with Babel³, 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. SASS⁴ is a stylesheet language, which is - similiar to ES - compiled to CSS and offers the mentioned and even more features. A lot of CSS Frameworks also provide their source code in SASS, often served with a large variable set which makes it easy to customize it. The transpiler used in this thesis to transform SASS to CSS is node-sass⁵.

5.2 Data model

All AFCL Java model classes have been ported to ECMAScript, to have all classes and properties available in the frontend, similiar like they are in the backend. Although plain JavaScript objects would also do the job, this adds a kind of type-safety and improves readability of the code. Also, the data exchange with the backend and the encoding of the graph benefit from this approach, since the XML node names map to constructor names.

³<https://babeljs.io>

⁴<https://sass-lang.com>

⁵<https://github.com/sass/node-sass>

5.3 Graph

A workflow can be visualized as a directed graph. The graph library used in this thesis to visualize a workflow is the JavaScript portion of `mxGraph`⁶. It is actively maintained and developed (during the development of the frontend, three bugfix releases and one minor release have been made public) and has an outstanding documentation. Additionally, it has a lot of demos and example code which show a lot of use cases and extensive features. Furthermore it integrates easily with React.

Basically, vertices and edges are represented by an `mxCell` model, which stores information about the cell's position, dimension, geometry and style. Additionally, a user object can be attached to a `mxCell`. User objects give the graph its context, they store the business logic associated with the visual part. In this application, the cell's user objects are instances of AFCL objects.

Thus, the graph visualizes the execution sequence of the workflow.

Validation

Encoding

Alternatives

One of the more difficult tasks was to choose a proper library for graph drawing. Since the graph drawing and visualization is one of the most important features of the frontend, a careful choice has to be made. Of course there exist multiple excellent JavaScript graph drawing libraries.

When searching a JavaScript library, a good starting point is

Below is a list of tested and researched JavaScript graph libraries with additional information why they were insufficient for the project.

- **jsPlumb** is a library of great quality and looks like it would fit all the requirements for graph drawing and user interaction out of the box. It has a very good documentation, and a lot of examples. Furthermore its animations and drag and drop handling are outstandingly good. Unfortunately this software is not open source and a license costs a few thousand dollars.
- The same goes for **Rappid** (formerly known as JointJS) and **GoJS**, which is even more expensive.

⁶<https://jgraph.github.io/mxgraph/>

- **diagram-js** is one of the newer diagram libraries. It seems to be a good candidate to draw BPMN⁷ graphs. The lack of an appropriate documentation - even after 4 years of existence (there exists an open issue on github⁸ for this) - is an issue.
- **Cytoscape.js** is older than mxGraph but the development and the community is not less active. This library has a very good documentation and a lot of demos are provided. But it offers fewer examples and "out-of-the-box" features for modeling flowcharts than mxGraph.
- **vis.js** is the star on npmtrends. It has a very smooth animations when manipulating the graph. There exists only one example of graph manipulation on its documentation.
- **Sigma JS** is a tiny library with a small documentation. It clearly focuses on displaying and visualizing graphs, not on modeling them. This part would have to be developed manually. Furthermore, the last activity on this project on github was 2 years ago.

5.4 Web Interface

The layout of the web interface is based on coreUI⁹, an admin panel template, built on top of the web toolkit Bootstrap¹⁰. A clean and nested structure of components, which is typical for a React app, guarantees modularity and forms the whole frontend application. One of the benefits when using React is that the data model displayed to the user maps most of the time nicely to UI components. Basically, the components of the application respect the single responsibility principle¹¹. The main component has four sub-components - where each of them consists again of multiple sub-components - representing 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.

⁷<http://www.bpmn.org>

⁸<https://github.com/bpmn-io/diagram-js/issues/78>

⁹<https://coreui.io>

¹⁰<https://getbootstrap.com>

¹¹https://en.wikipedia.org/wiki/Single_responsibility_principle

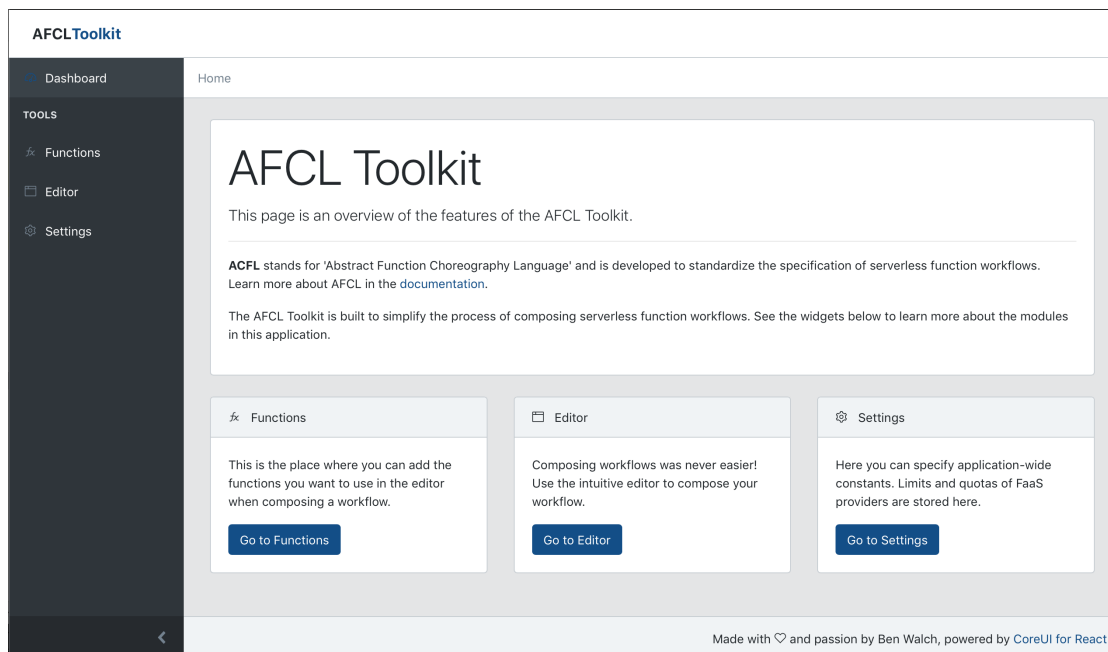


Figure 0.3: the dashboard

Functions

A function repository is needed to know about available functions when composing a workflow. An item in the function repository holds the following data: name, type and provider. For In the implementation of this thesis, this data is persisted to an internal file.¹²

¹²see 6.4 for detailed information on how data is persisted

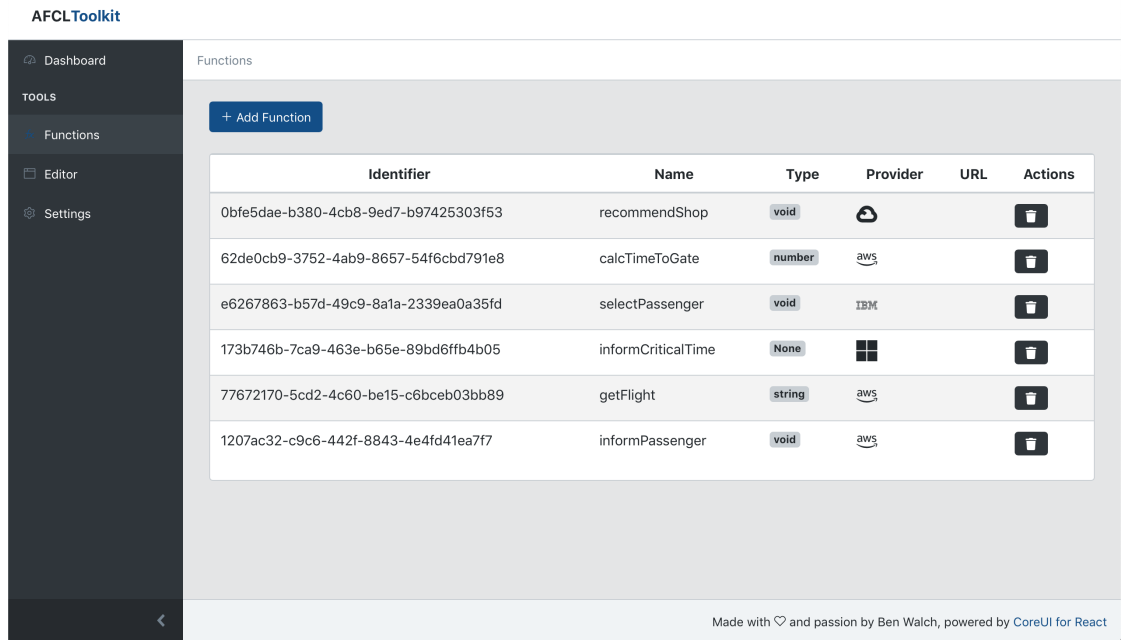


Figure 0.4: the function repository

Editor

The interface of the component responsible for composing workflows is divided into two parts. On the left side is the editor, consisting of a toolbar and a drawing area, on the right side is the property view. In the editor, the user can choose a function from the toolbar to place it in the drawing area below. For the sake of clarity, each function has a defined shape and style. In the drawing area, these shapes (functions) can then be connected to each other by drag and drop. This leads to a directed graph, which represent the execution flow. The possible connection points (ports) of a shape will be displayed when hovering over it. These ports are input and output for most of the functions. When hovering a port, the port name appears in the tooltip.

When selecting a shape in the canvas, the property view changes. It displays each specific property of the selected function, with the ability to change, add or remove properties.

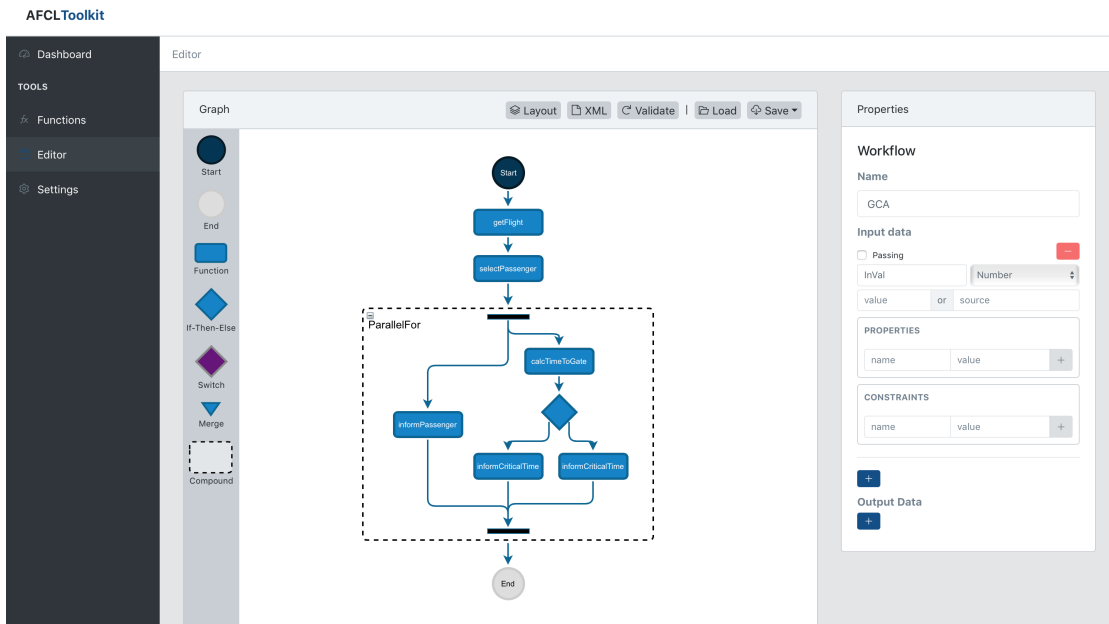


Figure 0.5: the editor

Settings

6 Backend

6.1 General

6.2 Maven

6.3 Servlets

6.4 Persistence

Composed workflows are be saved to the user's file system by offering a download. The user can then choose the target location. The storing of function data, on the other hand, is abstracted from the user and persisting of the data is handled internally. An interface¹³ for the repository implementation was created on the backend side to be capable of supporting any data source. The repository which implements the interface in this thesis, stores the serialized data into a file. This could be easily replaced with any other implementation, for example an implementation which stores the data in a DBMS.

¹³<https://docs.oracle.com/javase/specs/jls/se7/html/jls-9.html>

6.5 Api

7 Continuous Delivery

Improvements

Conclusion

Bibliography

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