

COMP4651 Fall 2019, HKUST | Project Report

Real-time Serverless File Processing: A Markdown to HTML parser

CHEUNG Tsz Yan (20431308, katrincheung), CHUN Hiu Sang (20421860, bryanchun), WONG Jia Yeung (20420684, exnight)

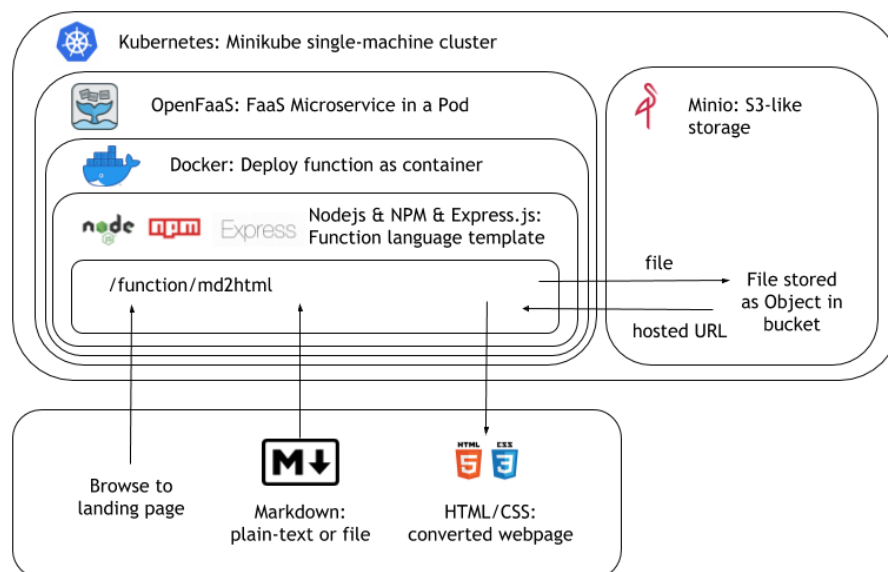
1 Introduction

The markup language Markdown (.md) has gained popularity among developers as a documentation and data description format, and many web apps (e.g. GitHub) have been consuming markdown files (READMEs) to render an HTML for enhancing readability. We aim to further enhance the conversion approach by parsing user-uploaded or text-input markdown notes to static web pages with various templates (e.g. blog post, user guide, documentation).

We identify our project as a serverless application, which takes markdown text or files as input and returns a set of generated, styled HTML files. This allows the application to be scalable and event-driven, as opposed to traditional hosted web apps, so that it can potentially handle more users and requests. Starting from reimplementing a similar AWS serverless application¹, we aim to adjust it to the open-source stack and add customised features.

The system includes Kubernetes, Docker, Minio, OpenFaaS, node.js and other open-source libraries. It is delivered as a docker image. The project source is available on GitHub².

2 System Architecture and Deployment



¹ <https://github.com/aws-samples/lambda-refarch-fileprocessing>

² <https://github.com/bryanchun/COMP4651>

As shown in the system architecture diagram, an encompassing local cloud service is hosted by Kubernetes, a container orchestration tool - in our case, as a single-machine cluster on Minikube (a macOS software for running Kubernetes). OpenFaaS is installed in Kubernetes via its Helm package (chart), so that it exists in a dedicated pod (unit of deployment). These steps are automated using a custom shell script³ to ease deployment.

Our markdown-to-html converter is implemented as a node.js-based OpenFaaS function (/function/md2html). Since express.js is needed for server routing, we selected an OpenFaaS language template with built-in express support ('node10-express-service'⁴) as the starting point. In the frontend, the user will submit an HTML form containing the markdown plain-text source or a markdown file; this triggers a HTTP POST request listened by express.js, which receives the input and starts converting it using a number of NPM modules. In particular, the parser part of the project focuses on providing themes and additional features for the rendering, and is currently using ejs (embedded javascript templates) for injecting content. As a result, the user will be shown a rendered HTML using the provided markdown. Different from the AWS sample, the output HTML file is directly returned to the browser for display, instead of being stored in a bucket.

Then, the entire function is built, which the docker image will be generated; pushed to the docker repository; and deployed onto Kubernetes as a running container. At this point, the function is served at the local machine, and the user can start to interact with it on their browser - which would issue HTTP GET calls.

Minio⁵, an AWS S3-like storage service, is added to support general file uploads and access. To reimplement the file processing feature in the AWS sample, we replaced the AWS S3 buckets with an open-source alternative: Minio, to store input files and allow the user to directly pass as parameter the path of the file to the AWS Lambda function / OpenFaaS function. Note that upon receiving a file URL, the function will download the file and pass the content to the converter, so it actually works on any hosted markdown URL (e.g. raw README.md files hosted in public GitHub repositories); Minio is an option to support user-provided markdown files not hosted anywhere. After uploading a file, the user can reuse the hosted URL for that markdown file to invoke the function multiple times. Minio is hosted inside Kubernetes and the function communicates with it using an NPM library. The steps to deploy Minio is also automated using another custom shell script⁶.

³ <https://github.com/bryanchun/COMP4651/blob/master/host.sh>

⁴ <https://github.com/openfaas-incubator/node10-express-service/>

⁵ <https://min.io/>

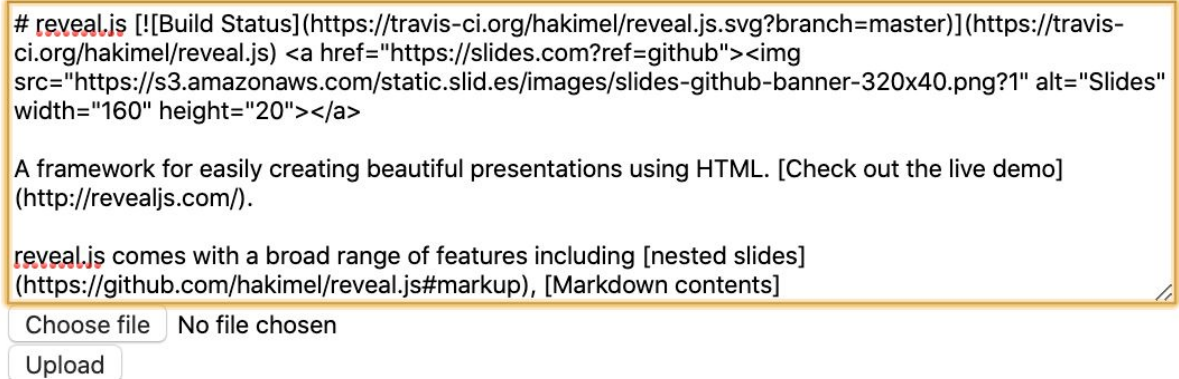
⁶ <https://github.com/bryanchun/COMP4651/blob/master/minio.sh>

3 Results and Delivery

Markdown files can be converted into HTML with an intuitive user interface.

1. Landing page: http://YOUR_MINIKUBE_IP:31112/function/md2html

- With plain-text input



The screenshot shows a web form for converting Markdown to HTML. The text area contains the following content:

```
# reveal.js [![Build Status](https://travis-ci.org/hakimel/reveal.js.svg?branch=master)](https://travis-ci.org/hakimel/reveal.js) <a href="https://slides.com?ref=github"></a>
```

A framework for easily creating beautiful presentations using HTML. [Check out the live demo] (http://revealjs.com/).

reveal.js comes with a broad range of features including [nested slides] (https://github.com/hakimel/reveal.js#markup), [Markdown contents]

Below the text area are two buttons: "Choose file" and "Upload". The "Choose file" button is highlighted with a yellow border.

- With file upload

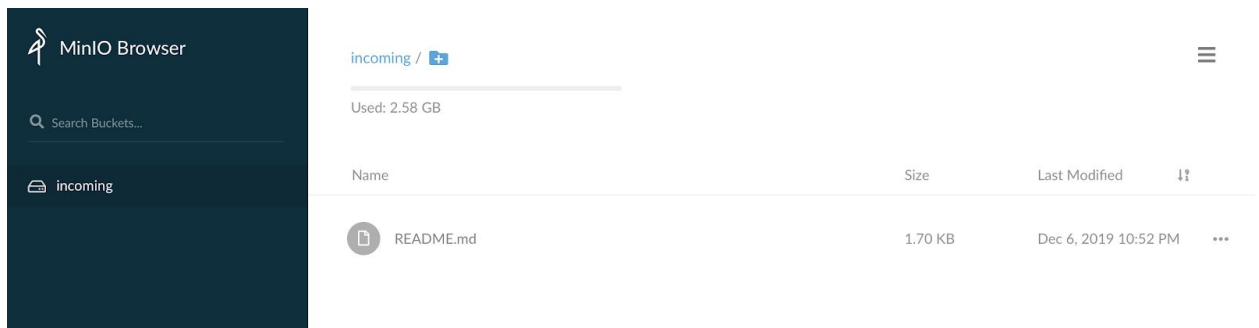


The screenshot shows the same web form as above, but with a file selected. The text area contains the following content:

```
# Hello
```

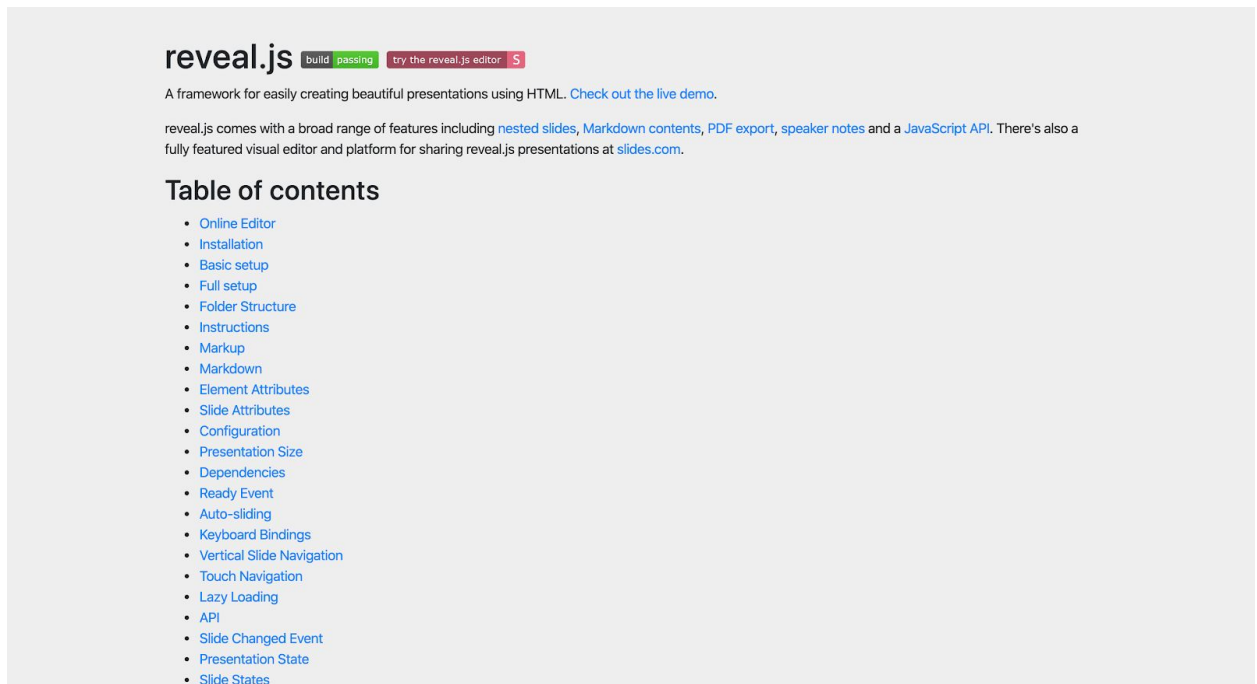
Below the text area are two buttons: "Choose file" and "Upload". The "Choose file" button is highlighted with a yellow border, and the text "README.md" is displayed next to it.

where the file will be uploaded to the Minio server, accessible from the Minio browser.



2. Resulting page after conversion (automatically redirected)

- With file URL: e.g.
http://YOUR_MINIKUBE_IP:31112/function/md2html/with?mdUrl=https://raw.githubusercontent.com/hakimel/reveal.js/master/README.md



or a Minio file URL: e.g.

http://192.168.99.143:31112/function/md2html/with?mdUrl=http://192.168.99.143:30941/incoming/README.md?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=minio%2F20191206%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20191206T144935Z&X-Amz-Expires=86400&X-Amz-SignedHeaders=host&X-Amz-Signature=0112904bdcd30adfff452d24f60b859063df3cfbe1e77fa185f50c435a4c5588

- With plain-text input: the URL is unchanged and the page temporarily shows the result

For delivery, the Docker image (Dockerfile) is available at the project GitHub repository⁷; or you may build from scratch following instructions in the project README.md.

4 Progress and Workload

We focused on setting up the environment in the early stage, e.g. designing the system architecture and end-user action flows, researching about installation and initialization of multiple softwares and dependencies. During the development stage, we split our work among all three members. Bryan was responsible for setting up the system and integrating components and functionalities, e.g. linking OpenFaaS with Minio. Katrin completed the user interface and tested the system. Leo set up the express.js web serving and implements the markdown parser.

⁷ <https://github.com/bryanchun/COMP4651/tree/master/build/md2html>

5 Technical Challenges

As we were going to try out these new platforms, learning new programming APIs was needed in order to finish the project, which would include but was not limited to Kubernetes, Docker and OpenFaaS.

For deployment, since we were new to the serverless architecture ecosystems, we would have to research the proper tools and steps to build the image and launch it on a hosting platform.

At the beginning of the project, we had to try deploying a function on OpenFaaS first. Due to the complicated installation process, we failed quite a lot of times. Finally we figured out a simpler setup (as shown in the README.md⁸ of the project repository).

Some of us were not familiar with node.js. We had to explore the functions in node.js and express.js, in order to build the landing page and link the resources between client side and server side. Because of some unknown reasons, we had inconsistent results when deploying the same function. Although we alleviated it error-by-error, our experience shows that deployment can still be inconsistent across different local machine environments even with docker.

⁸ <https://github.com/bryanchun/COMP4651>