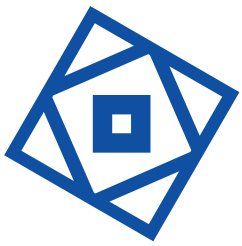


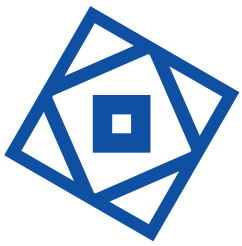
Transilvania
University
of Brasov

FACULTY OF MATHEMATICS
AND COMPUTER SCIENCE

DISSERTATION PAPER

Author: Kedves Szilard, engineer
Supervisor: Associate professor Sasu Lucian Mircea, Ph.D.

Brașov
July 2019



Transilvania
University
of Brasov

FACULTY OF MATHEMATICS
AND COMPUTER SCIENCE

DISSERTATION PAPER

Cloud framework for data streaming

Absolvent: Kedves Szilard, engineer
Coordonator: Associate professor Sasu Lucian Mircea, Ph.D.

Brașov
July 2019

Contents

Chapter 1

Introduction

1.1 Problem statement

In the world of Big Data, data visualization tools and technologies are essential to analyze massive amounts of information and make data-driven decisions. Hence it is not a surprise that applications that tackle this need are in great demand. These applications use graphical representation of information and data by using visual elements like charts and graphs.

1.2 Motivation

The idea started when I first came into contact with a major cloud service provider: Amazon Web Services also known as AWS.

AWS is a subsidiary of Amazon that offers reliable, scalable, and inexpensive cloud computing services. It provides on-demand cloud computing platforms to individuals, companies and governments, on a metered pay-as-you-go basis. In fact, these cloud computing web services provide a set of primitive, abstract technical infrastructure and distributed computing building blocks and tools. One of these services is Amazon Elastic Compute Cloud, which allows users to have at their disposal a virtual cluster of computers, available all the time, through the Internet. AWS's virtual machines emulate most of the attributes of a real computer including hardware (CPU(s) and GPU(s) for processing, local/RAM memory, hard-disk/SSD storage); a choice of operating systems; networking; and pre-loaded application software such as web servers, databases, etc. [1]

At the beginning its API seemed hard to comprehend, but with time I came to realize how easy it was in fact. Hence my determination to work with as many of its services as possible. At first I started with simple services like S3[TODO legend

for S3], but then I got to understand and use services like ECS[TODO legend for ECS]. The more services I used the more Stream Anlaysis began to expand.

1.3 Thesis structure

Chapter 2

Requirements

In this chapter the functional and nonfunctional requirements for the application are presented.

2.1 Functional requirements

2.1.1 Application overview

Stream Analysis is a web application that provides a hosting environment and defines a workflow for streaming applications, leveraged by cloud.

The hosting environment is responsible for user related sensitive information storage, while it paves the ground for an infrastructure that can handle incoming data. Once information pours in, a dashboard is shown at the user's disposal. This dashboard is updated each time new data arrives, which basically makes it a tool for analytical purposes. The user can visually check if a certain threshold was reached, exceeded, etc.

The workflow is predefined which makes it organized and coherent. Data is fetched or produced and packed by a custom application provided by an external user and funneled in through well defined channels. The stream of data arrives in a secure place where it waits to be queried. This place is located in the cloud so even if one can create the data on-premise, one still needs constant access to the internet. Based on the user's choice the packed information is then retrieved in a real-time fashion or stored. Both ways the data is then accessed and charted. 2D plots are shown with the values at the Y axis, where the X axis holds the timestamps.

The web application offers two services:

- Push data into cloud - users have to create their own packed software application that streams any type of data into a broker. The broker supports

5 types of protocols ([TODO]more information in chapter ...). Data is organized into topics and queues:

  - Topics - are used to handle real time messages
  - Queues - are used for historical data. Messages are stored and retrieved later

  Once the image is ready, the user can proceed with the step by step image upload and use it to create containers.

  1. Define topics and queues used in the container
  2. Create a repository for the image
  3. Push image to repository
  4. Add configuration for the image
  5. Run image immediately or create a scheduling rule to run the image multiple times over time

  Once containers are being created the web application subscribes to the topics defined by the user at step 1. The same happens with queues but in this case the stream of data is dequeued into files so that they can later be retrieved. Stream Analysis also offers the user a dashboard to keep track of his created containers and scheduling rules.

- Visualize data from the cloud - since data is split into topics and queues, the web application takes advantage of this and shows real time plots for topic streams and large scale based plots for queue data.

2.1.2 Access

The Stream Analysis web application is accessible from all around the world as long as the user has internet connection, however it is not open-source and its usage is not free.

The application is targeting two types of users:

- Advanced software engineers - they push data into the application. They need to register with an account and they are charged based on time.
- Normal users - they can read and interpret that data from charts in a dashboard. They also need to register with an account and they are charged based on the network traffic they generate during data visualization.

2.2 Nonfunctional requirements

2.2.1 Diversity

One of its main advantages consists in the ability to feed on any type of data. This asset makes it valuable in any kind of business or domain. For example in the field of meteorology users can plot the weather related data. Having Stream Analysis at their disposal they can track real-time the wind changes. In the field of economics, users can map economical growth of a country next to another one and be able compare, draw conclusions, or forecast.

2.2.2 Performance

The application itself was intended to be very fast and responsive to user interactions. However it has no scaling policies set up so there is a limit of users that can use the web application simultaneously. The limit is around 500 reqs/second.

2.2.3 Security

In my views security is pivotal for any well designed and successful application. Users are guaranteed to not have exposed credentials or any kind of sensitive data. If there is a concern of information leakage, this matter is more deeply discussed in [TO DO Chapter ...].

2.2.4 User's Guide

This paper will softly guide users through its features in further chapters. At this moment it has no stand alone public documentation that users can access.

Chapter 3

Framework architecture

At the highest macro level the framework can be put into perspective as shown in figure 3.1.

Figure 3.1: Highest level of architecture view

The roles and components are:

- Data acquisition users - these are external users that get data from any source and have the responsibility to wrap the fetching into an application and use the framework to upload it
- Packed application - the application made on-premise by the data acquisition user
- Framework - a web application hosted in cloud, used for uploading the packed application and also to visualize its output
- Data visualization users - the beneficiaries of the wrapped application, they can visualize information with the help of plots

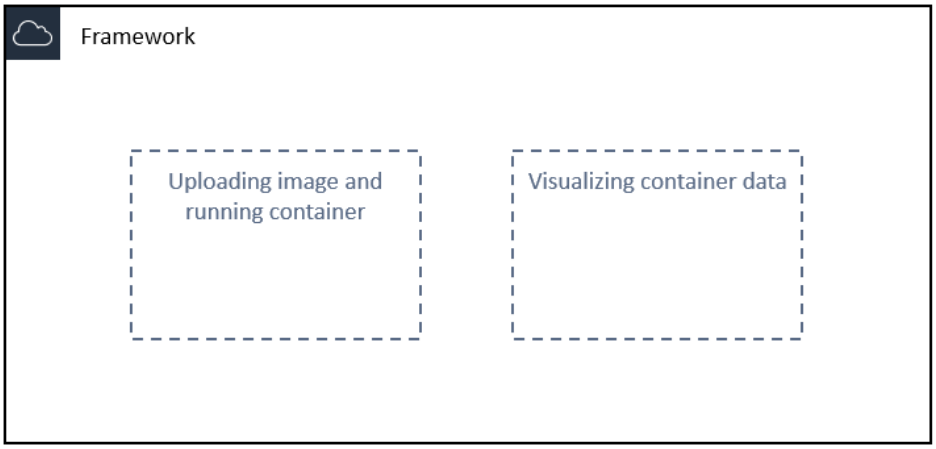


Figure 3.2: Framework main components

Figure 3.2 zooms in the Framework component. The two blocks are as follows:

- Uploading image and running containers - the data acquisition user after wrapping his fetching data application can use this side of the web application to upload and run it. The steps of achieving this goal is further discussed in chapter 4.
- Visualizing container data - this side of the framework is a tool which can be used to select plots filled with data coming from containers. Selection and usage of these plots is discussed in depth at chapter 4.

Chapter 4

The Stream Analysis Web Application

In this chapter the making of the web application is presented in details. The level of description should encourage other developers in using the technologies that were used in this application.

4.1 Front-end

The Stream Analysis Web Application actually consists of two web applications, since Front-end is decoupled from the Back-end and both of them run on different ports. The Front-end/Back-end pattern is very popular nowadays, because the role of each application is well defined. By loose coupling the two parts means that the Front-end can be reused with another Back-end. Moreover this Back-end can be written in any programming language and any web framework.

4.1.1 Angular

The Front-end of Stream Analysis Web Application was written with Angular. Angular is a TypeScript-based open-source web application framework led by the Angular Team at Google and by a community of individuals and corporations. Angular is a complete rewrite from the same team that built AngularJS.[2]

Angular supports Single Page Applications which is used by Stream Analysis Web Application as well. Single Page Applications are a type of web applications that load a single HTML page, and the page is updated dynamically according to the interaction of the user with the web app. Single Page Applications, also known as SPAs, can communicate with the back-end servers without refreshing the full webpage, for loading data in the application. SPAs provide better user experience as no one likes to wait too long for reloading of the full webpage.[3]

Angular takes advantage of modularity. One can think of modularity in Angular as if the code is organized into “buckets”. These buckets are known as “modules” in Angular. The application’s code is divided into several reusable modules. A module has related components, directives, pipes, and services grouped together. These modules can be combined with one another to create an application. Modules also offer several benefits. One of them is lazy-loading, that is, one or more application features can be loaded on demand. If properly used, lazy-loading can increase the efficiency of an application a lot.[3]

To run Angular the developer needs to install NodeJs and Angular CLI. Once these two components are installed the developer is three steps away from running a brand new Angular application.

npm new ProjectName

It creates a folder named ProjectName and downloads in it from official Angular repository the web application boilerplate files. From this point on the developer just needs to add additional to build its web site.

npm install

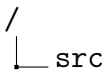
In Angular the dependencies are located in a .json file: package.json. Based on this file the command will download and install all the dependencies in the newly created root folder. The newly installed dependencies can be found in the folder ”nodemodules”.

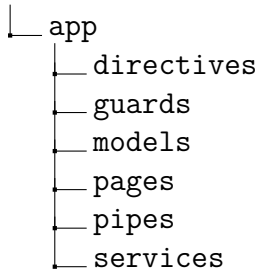
ng serve -o

Once the dependencies are successfully installed the developer can run the server and see web site in the browser. The command will build web application and open the default browser automatically to the default Angular url: http://localhost:4200.

4.1.2 File structure

In case of Stream Analysis Web Application the file structure is pretty simple. Every Angular entity type has its own folder.





As seen in the tree folder structure above there are six different types of Angular entities used. Thanks to the Angular CLI these can be easily created as shown below.

ng g c fileName

Directives represent an extended HTML syntax. Developers using the native HTML tags may want to create a customized tag that englobes the application's menu. For example in Stream Analysis Web Application's case the menu directive is used with the following HTML syntax: <app-menu></app-menu>. Other examples are: <app-live-chart></app-live-chart>, <app-zoom-chart></app-zoom-chart>, <app-login-form></app-login-form> and <app-register-form></app-register-form>. Once declared and defined, directives can be reused. Moreover they can also be parameterized. This feature is used to make charts plot different data types.

ng g g fileName

Guards can be used as an internal security for routes. In the application there is a file called auth.guard.ts, which guards the menu routes: it checks of the user was marked as logged in. In case the user did not log in he is redirected to the login page. If the user checks the 'Remember me' checkbox in the login page, then he can refresh the page and he is will still be marked as logged in. The application saves the current user to the browser's local storage and retrieves it with the guard once the user tries to access the page.

ng g i fileName

Models are various class declarations with the properties. These are used throughout the application. This is possible because TypeScript is strongly typed, opposite to plain JavaScript.

ng g c fileName

Pages in the terms of Angular entities are in fact components. In this applica-

tion they are the end result of routes. Once the user is routed to a certain page, he sees the HTML declared in these components.

ng g p fileName

Pipes change the variable values used in components. They receive the current value of the variables and mutate them as the developer wishes on page rendering. Stream Analysis takes advantage of this to trim down chart dashcard card header names, since they contain as a prefix the user id. Hence on page rendering only the topic/queue names are shown, but in fact the variables used inside the charts are a concatenation of user id and topic/queue name.

ng g s fileName

Services have a certain responsibility assigned to them. Usually they are used to communicate with the Back-end. However they can also persist variable values inside Angular. This feature is needed in case the user makes modifications in a certain page, then routes to another page and then routes back to the previous page. In this case the altered information on the first page will be lost. To persist the state across routings services are a great candidate. Services use the so called dependency injection. Developers need to inject them into components and make use of their declared methods.

4.2 Back-end

4.2.1 ASP .NET Core

The Back-end of Stream Analysis Web Application is written in ASP. NET Core, hence the language used is C-Sharp.

ASP.NET Core is a cross-platform, high-performance, open-source framework for building modern, cloud-based, Internet-connected applications. Its benefits are:[4]:

- A unified story for building web UI and web APIs.
- Architected for testability.
- Ability to develop and run on Windows, macOS, and Linux.
- Open-source and community-focused.

- Integration of modern, client-side frameworks and development workflows.
- A cloud-ready, environment-based configuration system.
- Built-in dependency injection.
- A lightweight, high-performance, and modular HTTP request pipeline.
- Ability to host on IIS, Nginx, Apache, Docker, or self-host in your own process.
- Tooling that simplifies modern web development.

4.2.2 File structure

The application is structured as follows:

```
/
├── Controllers
├── Exceptions
├── Hubs
├── Interfaces
├── Logs
├── Models
└── Services
```

Controllers are directly responsible for handling incoming requests from the Front-end. They are declared as simple C-Sharp classes, but are decorated with attributes. These attributes indicate the routing paths they are responsible for. In ASP .NET Core the 'Route' attribute can be parameterized:

- controller - At class level it interpolates the path string with the controller class name. In case the class it is sufixed with the word 'Controller' then this trimmed.
- action - At method level it interpolates the path string with the method name.

Stream Analysis Web Application's Back-end has the following routes:

```
import numpy as np

def incmatrix(genl1,genl2):
m = len(genl1)
n = len(genl2)
M = None #to become the incidence matrix
VT = np.zeros((n*m,1), int)  #dummy variable
```

```
8
9  #compute the bitwise xor matrix
10 M1 = bitxormatrix(genl1)
11 M2 = np.triu(bitxormatrix(genl2),1)
12
13 for i in range(m-1):
14 for j in range(i+1, m):
15 [r,c] = np.where(M2 == M1[i,j])
16 for k in range(len(r)):
17 VT[(i)*n + r[k]] = 1;
18 VT[(i)*n + c[k]] = 1;
19 VT[(j)*n + r[k]] = 1;
20 VT[(j)*n + c[k]] = 1;
21
22 if M is None:
23 M = np.copy(VT)
24 else:
25 M = np.concatenate((M, VT), 1)
26
27 VT = np.zeros((n*m,1), int)
28
29 return M
```

4.3 Cloud environment

4.4 Containerized application

Bibliography

[1] *"Amazon Web Services"*. URL: https://en.wikipedia.org/wiki/Amazon_Web_Services.

[2] *"Angular"*. URL: https://en.wikipedia.org/wiki/Angular_(web_framework).

[3] *"Angular"*. URL: https://hackr.io/blog/why-should-you-learn-angular-in-2019.

[4] *"ASPDotNetCore"*. URL: https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-2.2.