HAROKOPIO UNIVERSITY

SCHOOL OF DIGITAL TECHNOLOGY

DEPARTMENT OF INFORMATICS AND TELEMATICS

POSTGRADUATE PROGRAMME STUDIES “INFORMATICS AND TELEMATICS”

COURSE ”COMPUTATIONAL AND INTERNET TECHNOLOGIES AND APPLICATIONS - WEB ENGINEERING”

Development of microservices based toolkit (platform) for automated analytic applications perceived as directed acyclic graphs that are deployes to k8s

Master Thesis

Georgia Chatzimarkaki

Athens, September 2021

HAROKOPIO UNIVERSITY

SCHOOL OF DIGITAL TECHNOLOGY

DEPARTMENT OF INFORMATICS AND TELEMATICS

POSTGRADUATE PROGRAMME STUDIES "INFORMATICS AND TELEMATICS"

COURSE "COMPUTATIONAL AND INTERNET TECHNOLOGIES AND APPLICATIONS - WEB ENGINEERING"

Ethics and Copyright Statement

Dedication page

Having completed my Master thesis, I complete my study cycle at the Department of Informatics and Telematics of Harokopio University. On this occasion, I would like to express my sincere gratitude to all those who played an important role during my studies at the Foundation and to all those who helped me complete this thesis.

I would especially like to thank my supervising professor, Mr. Iraklis Varlamis, who was a helper of my effort with his scientific knowledge and the moral support he provided. Without his valuable guidance I could not have completed this task successfully.

I would also like to thank the members of the examination committee, Mr. Konstantinos Tserpe and Mr. Georgios Kousiouris, for the help, support and advice they offered me during the writing, as well as for their understanding.

I would also like to thank all my teachers at all levels of education, whose teaching has opened up horizons in my life, and to whom I am grateful.

Finally, I could not omit to thank my family for the supplies they have provided and their support throughout my studies.

Quote page

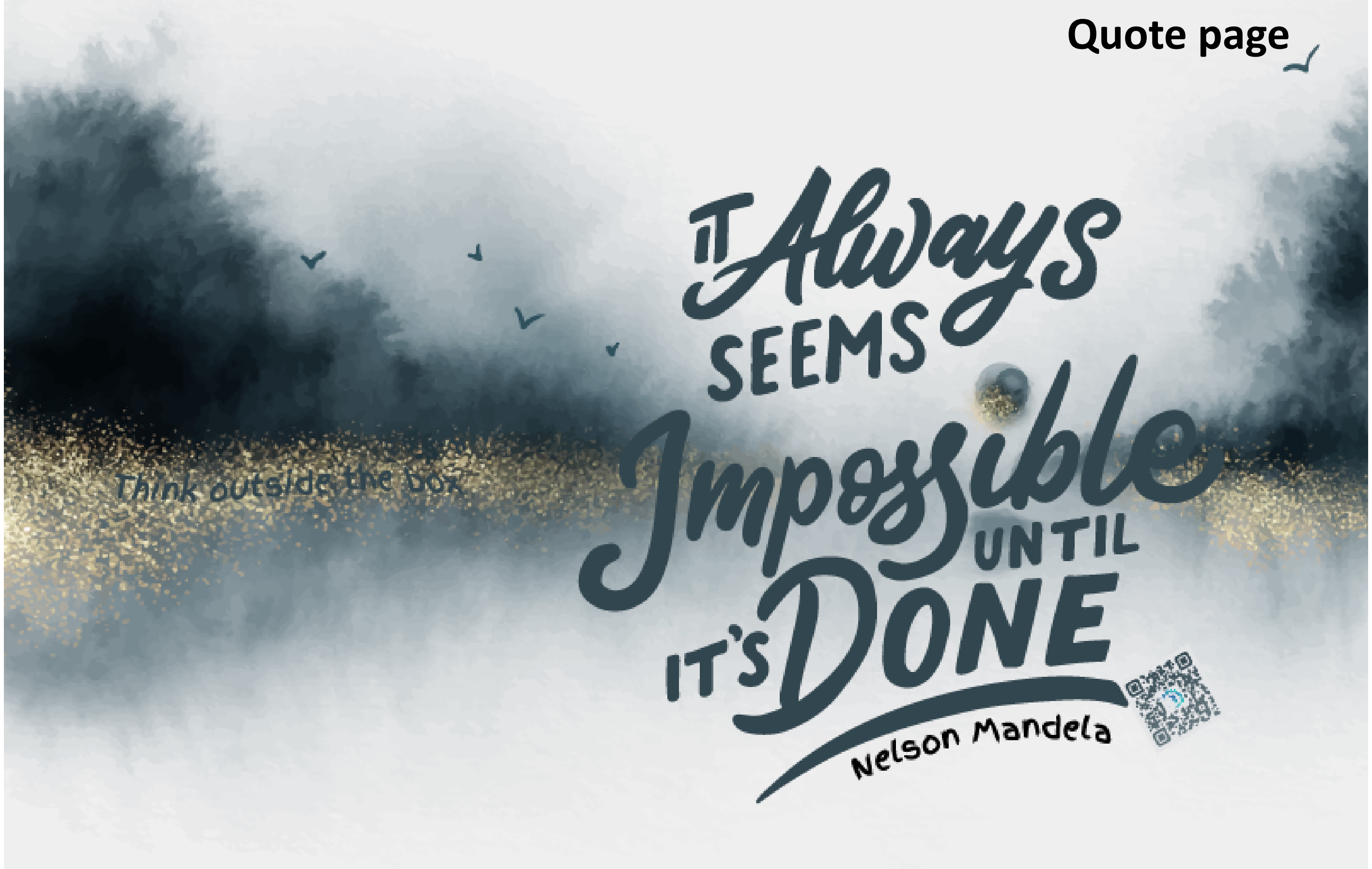

Think outside the box
It Always seems Impossible until it's Done
Nelson Mandela

TABLE OF CONTENTS

Abstract in Greek

Με την έλευση του cloud computing, το αρχιτεκτονικό στυλ των μικροϋπηρεσιών έχει προσελκύσει μεγάλο ενδιαφέρον στην κοινότητα του Software Enginneering. Σε αντίθεση με τη μονολιθική αρχιτεκτονική, η αρχιτεκτονική που βασίζεται σε μικροϋπηρεσίες αντιμετωπίζει τις προκλήσεις της δημιουργίας εφαρμογών εγγενών στο σύννεφο που αξιοποιούν τις ευκαιρίες που δίνει η υποδομή του νέφους για να διευκολύνουν τη δημιουργία εφαρμογών μεγάλης κλίμακας. Εδώ to Docker και Kubernetes αυτοματοποιούν την ανάπτυξη και τη διαχείριση αυτών των εφαρμογών.
Ο στόχος αυτής της Μεταπτυχιακής Εργασίας είναι να αναπτύχθεί μια συνδυασμένη εργαλειοθήκη (πλατφόρμα) βασισμένη σε microservices μέσω κατευθυνόμενων άκυκλων γραφημάτων που αναπτύσσονται στο Kubernetes. Βασισμένη στην υλοποίηση του Spring Cloud Dataflow (SCDF) με το ενσωματωμένο επίπεδο ενορχήστρωσης, παρέχουμε μια εξαιρετικά παραγωγική εμπειρία για την ανάπτυξη και τη διαχείριση εξελιγμένων data-pipilines που αποτελούνται από αυτόνομα microservices. Το περιβάλλον επιτρέπει στους χρήστες να ενορχηστρώσουν τον κύκλο ζωής της ανάπτυξης stream-ing, batch data pipelines για να δημιουργήσουν μια μικροϋπηρεσία. Διευκολύνουμε τη συνεργασία των χρηστών μέσω της εξαγωγής και τον διαμοιρασμό των ανεπτυγμένων pipeline σε JSON ή YAML αρχεία. Το διαμορφωμένο περιβάλλον επιτρέπει στους χρήστες να βελτιστοποιούν την

ανάπτυξη artificial intelligence and deep learning pipelines επιτρέποντας τη χρήση της GPU. Καθώς Big data analytic pipelines γίνονται όλο και πιο δημοφιλείς για την επεξεργασία δεδομένων μεγάλου όγκου, επεκτείνουμε τη λειτουργία του SCDF με την υποστήριξη δημιουργίας εγγράφων και note-books γραμμένων σε διαφορετικές γλώσσες προγραμματισμού και επιτρέπουμε την εξαγωγή του σε διαφορετικές μορφές ανάλογα με τις ανάγκες. Ενσωματώσαμε με τη δημιουργία ενός widget που συνδέει τον virtual assistant που ονομάζεται Mycroft με τους χρήστες, επιτρέποντάς τους να ορίσουν υπενθυμίσεις για τις εργασίες τους και να αναζητήσουν πληροφορίες που μπορούν να τους βοηθήσουν κατά τη διάρκεια της ανάπτυξης. Για την διαχείριση της ταυτότητας και της πρόσβασης των χρηστών σε όλη την εφορμογή και τις υπηρεσίες αξιοποιήσαμε το μηχανισμό ασφαλείας που ονομάζεται Keycloak για την σύδεση του χρήστη και την παροχή ενός token που θα διαμοιράζετε με τις άλλες υπηρεσίες για να επιβαιώσει την αυθεντικότητα του χρήστη.

Keywords: εργαλειοθήκη, streams, tasks, keycloak, notebooks, μικρουπηρεσίες, ροή πληροφορίας

Abstract in English

With the advent of cloud computing, the microservices architectural style has drawn a substantial amount of attention in the software engineering community. As opposed to monolithic architecture, the microservice based architecture tackles the challenges of building cloud-native applications that leverage the opportunities given by the cloud infrastructure to facilitate the creation of large-scale applications. Here Docker and Kubernetes, are automating the deployment and management of these applications. The goal of this Master Thesis is to develop a combined micro-services based toolkit (platform) for automated analytic applications perceived as directed acyclic graphs that are deployed to Kubernetes. By using the Spring Cloud dataflow (SCDF) with the integrated orchestration layer, we provide a highly productive experience for deploying and managing sophisticated data pipelines consisting of standalone microservices. The working environment allowing users to orchestrate the deployment lifecycle of streaming, batch data pipelines to create a microservice. We enable users collaboration through exporting and sharing the created pipelines manifests as JSON or YAML files. We configure the environment to allow users to optimize the deployment of artificial intelligence and deep learning pipelines by enabling the use of GPU for the deployment of the pipelines. As Big data analytic pipelines become popular for large volume data processing we extend the SCDF functionality to support document and notebook creation written in different programming languages and exporting them to different formats based on the need. An Angular widget that connects the virtual assistant called Mycroft with users by enabling them to set reminders for their tasks and search for information that can assist them during the development has been created for this project. Finally, to manage the identity and access of users throughout the application and services, we used a security mechanism called Keycloak to connect the user and provide a token that you will be shared with other services to verify the authenticity of the user.

Keywords: toolkit, streams, tasks, keycloak, notebooks, microservices, pipelines

LIST OF FIGURES

LIST OF TABLES

List of source Codes

Abbreviations

SPI Serial Peripheral Interface. 25, 58

ANSI American National Standards Institute. 24

API Application Programming Interface. 22

CLI command-line interface. 42

CORS Cross-Origin Resource Sharing. 21

DB database. 23

HMAC Hash-based message authentication code. 19

HOTP HMAC-based one-time password (HOTP). 58

HTTP Hypertext Transfer Protocol. 22

ID Identity Document. 34, 35

IDP Identity Provide. 62, 63

JSON	JavaScript Object Notation. 31–34, 38–40, 105, 109, 111
LDAP	Lightweight Directory Access Protocol. 56, 57, 62
OIDC	OpenID Connect. 55–57, 61, 62
OS	Operating System. 24
PAS	Platformm as a Service. 56
REST	Representational state transfer. 57, 59, 62
SAML	Security Assertion Markup Language. 56, 57, 61, 62
SAML 2.0	Security Assertion Markup Language 2.0. 62
TOTP	Time-based One-time Password. 58
UI	User Interface. 21
VM	virtual machine. 21
YAML	Yet Another UML front end. 33, 34, 37, 39

Glossary

(VM) An emulation of a computer system. Virtual machines are based on computer architectures and attempt to provide the same functionality as a physical computer. Their implementations may involve specialized hardware, software, or a combination of both.. 23, 39, 40

User Interface (UI) The space where interactions between humans and machines occur. The goal of this interaction is to allow effective operation and control of the machine from the human end, whilst the machine simultaneously feeds back information that aids the operators' decision-making process. Examples of this broad concept of user interfaces include the interactive aspects of computer operating systems, hand tools, heavy machinery operator controls, and process controls. The design considerations applicable when creating user interfaces are related to or involve such disciplines as ergonomics and psychology.. 56, 102, 103

CORS Cross-origin resource sharing is a mechanism that allows restricted resources on a web page to be requested from another domain outside the domain from which the first resource was served. A web page may freely embed cross-origin images, stylesheets, scripts, iframes, and videos.. 58

HTTP The Hypertext Transfer Protocol (HTTP) is a stateless application-level protocol for data exchange. It is in use since 1990 and is the foundation of the World-Wide Web.. 61, 72, 102

access the ability, right, or permission to approach, enter, speak with, or use; admittance: They have access to the files.. 55

API API is a set of subroutine definitions, communication protocols, and tools for building software. In general terms, it is a set of clearly defined methods of communication among various components. A good API makes it easier to develop a computer program by providing all the building blocks, which are then put together by the programmer.. 56, 57, 70–72, 94

authentication Authentication (from Greek: αὐθεντικός authentikos, ”real, genuine”, from αὐθέντης authentes, ”author”) is the act of proving an assertion, such as the identity of a computer system user. In contrast with identification, the act of indicating a person or thing’s identity, authentication is the process of verifying that identity.. 55, 56

browser A Web browser or browser is a program that retrieves and displays pages from the Web, and lets users access further pages through hyperlinks. A browser is the most familiar type of user agent.. 57

client A piece of computer hardware or software that accesses a service made available by a server. The server is often (but not always) on another computer system, in which case the client accesses the service by way of a network.[44] The term applies to the role that programs or devices play in the client–server model.. 58

cloud computing Shared pools of configurable computer system resources and higher-level services that can be rapidly provisioned with minimal management effort, often over the Internet. Cloud computing relies on sharing of resources to achieve coherence and economies of scale, similar to a public utility.. 55

data ndividual facts, statistics, or items of information. Data are measured, collected, reported, and analyzed, and used to create data visualizations such as graphs, tables or images. In computing, data is information that has been translated into a form that is efficient for movement or processing.. 29

database An organized collection of data, generally stored and accessed electronically from a computer system. Where databases are more complex, they are often developed using formal design and modeling techniques.. 56, 73

execution In computer and software engineering is the process by which a computer or (VM) executes the instructions of a computer program. Each instruction of a program is a description of a particular action which to be carried out in order for a specific problem to be solved; as instructions of a program and therefore the actions they describe are being carried out by an executing machine, specific effects are produced in accordance to the semantics of the instructions being executed.. 56

file (1) (ISO) A set of related records treated as a unit; e.g., in stock control, a file could consists of a set of invoices. (2) The largest unit of storage structure that consists of a named collection of all occurrences in a DB of records of a particular record type.. 105

gateway An intermediate system (interface, relay) that attaches to two (or more) computer networks that have similar functions but dissimilar implementations and that enables either one-way or two-way communication between the networks.. 55–57

interpreter A computer program that directly executes instructions written in a programming or scripting language, without requiring them to have been previously compiled into a machine language program.. 72, 73

operating system (OS) System software that manages computer hardware, software resources, and provides common services for computer programs.. 39

reCAPTCHA is a CAPTCHA system that enables web hosts to distinguish between human and automated access to websites. The original version asked users to decipher hard to read text or match images.. 24, 63

SAML 2.0 Security Assertion Markup Language 2.0 is a version of the SAML standard for exchanging authentication and authorization identities between security.SAML is an open standard that enables single sign-on (SSO). 58, 59

server A server is a computer or system that provides resources, data, services, or programs to other computers, known as clients, over a network. In theory, whenever computers share resources with client machines they are considered servers.. 56

software (ANSI) Programs, procedures, rules, and any associated documentation pertaining to the operation of a system. Contrast with hardware. See: application software, operating system, system software, utility software.. 22, 44

SPI The Serial Peripheral Interfaceis a synchronous serial communication interface specification used for short-distance communication, primarily in embedded systems. The interface was developed by Motorola in the mid-1980s and has become a de facto standard.. 19, 58

storage Computer data storage is a technology consisting of computer components and recording media that are used to retain digital data. It is a core function and fundamental component of computers.[1]. 23, 56

token Tokens are created through an initial coin offering, which represents the cryptocurrency version of an initial public offering (IPO).. 58–63, 94

user Is a person who utilizes a computer or network service. Users of computer systems and software products generally lack the technical expertise required to fully understand how they work.[2] Power users use advanced features of programs, though they are not necessarily capable of computer programming and system administration.. 55, 56

CHAPTER 1

Introduction

1.1 Research Inspiration

Inspiration and Motivation for this Master thesis were adapted by a European Commission project called **H2020 CYBELE**, where I was a member of the team that was working on this project. Almost a year ago at some phase where future steps of the development were discussed, an idea to include an editor to enable the user to create new nodes for the pipeline were presented. Later in the same week as I was studying for a Big-Data project in University, where I have to present Apache Zeppelin (Web-based notebook), I saw some features that remind me of this discussion. This arouse my curiosity and inspired on how to include this functionality. The different approach of the usage of the Apache Zeppelin notebooks feature is one of the key factors that differentiates this Master thesis from H2020 CYBELE project. In Ml Lab the goal will be to provide extra functionality for data scientist and students to have a combined environment that can create stream/task pipelines or notebooks for deep learning application's. The H2020 CYBELE project final goal is to include notebooks Workflows to provide a convenient way to prepare data-driven, interactive data analytics and collaborative scripts concerning spark execution properties.

1.2 Contribution

The aim of this Master thesis is to create, develop and design an application, which can be used by students and data scientists by extending the Spring Cloud Dataflow functionality to provide more features. I wanted to provide a combined solution that a user can:

(1) export the YAML manifest of the deployed pipeline that can be re-used for deployment to another environment, that can enhance collaboration between users.

(2) modify the configuration for the generic deployer properties to define the amount of GPU cores to be used to deploy/launch artificial intelligent or big data pipelines.

(3) create and share collaborative notebooks written in different programming languages that can be exported to different formats based on the need.

(4) prepare new stream and task application(nodes) like an editor by using Java interpreter by following the providing guide.

(5) use the chatbot (widget) (personal assistant) to get information about the toolkit by asking a question like ”what is a stream;” or even ”how can I write my first stream;”, where in the second the answer will include links on basic node creation steps.

(6) use the chatbot to set custom reminders to finish a task or even to find information. For example ask about weather conditions to validate the results of a weather prediction model.

(7) be authenticated with the application and other services though Keycloak (the identity and access management solution).

1.3 Organization of this Thesis

A brief introduction to the topic is given in the first chapter to describe where the inspiration of this project come and describe the organization of the following chapters.

The second chapter focuses on the foundations needed for this master’s thesis, including theoretical and technical background regarding JSON and YAML, Docker, Kubernetes, Authentication mechanism.

The third chapter introduces the prototype that I will be used to develop and provide the ML Lab toolkit secured with Keycloak, notebooks integration and a chatbot to set up reminders and ask questions. The implementation of this prototype will be presented in the fourth chapter. Finally, in the fifth and sixth chapter, a conclusion on this master’s thesis is drawn and future work is proposed.

CHAPTER 2

Background

2.1 Serialization

Serialization is a process for converting a data structure or object into a format that can be transmitted through a wire, or stored somewhere for later use. In terms of serialization there are a legion of different ways and formats that can be used. Which method and format to choose depends on the requirements set up on the object or data, and the use for the serialization (sending or storing). The choice may also affect the size of the serialized data as well as serialization/deserialization performance in terms of processing time and memory usage.

All serialization methods reading data as a series, once started the whole object will usually be serialized/deserialized. This enables the use of simple I/O interfaces to hold and pass on the state of an object, although difficulties arise in applications which require higher performance by having a nonlinear storage organization, or when the object contains large amounts of data.

Serialization is supported by many of the popular object-oriented programming languages like Java, PHP, Ruby, Smalltalk and Python along with the .NET Framework. All of these languages provide serialization methods either as implementable interface or as syntactic sugar.

A serialization strategy can be defined in cases when you want to restrict the serialization process (all instance variables are serialized by default) or handle data in specific ways. Most of the standard serialization implementations converts the data into a binary string, which means that the data will not easily be inspected by a human in its serialized form.
Serialization is preferable to use when transmitting data, as has been mentioned above. Some example of such cases are when storing user preferences in an object or for maintaining security information across pages and applications. In general,when transferring objects in applications, domains, or through firewalls, serialization can be very helpful.

2.1.1 Parsing

The term parsing in *computer science* means in general to analyze written text, determining its grammatical structure from a known formal grammar. In *linguistic* terms, parse means analyzing and describe the grammar of a sentence. The parser splits up an expression into tokens which are then inserted into some kind of data structure. This data is the evaluated to interpret the meaning of each expression by the rules from given grammar, followed by execution of the appropriate action.
Serialization is mainly a method to maintain easy ways of storing, in the sense of converting data and then restore it into a semantically equivalent clone. Unless the serialization method used serializes the data in a coherent order, where the data never changing, and expects the data to be read in the same order when deserializing, parsing will have to be done when the data is to be deserialized. When deserializing, parsing is done to identify the data identifiers like attribute

names or the like and their corresponding values, while at the same time often having to discern the type of data.

2.1.2 JSON

The (JavaScript Object Notation (JSON)) is a serialization format for data. JSON was originally specified and introduced by Douglas Crockford in 2001[3], who used it within his company State Software, but he was not the first person to invent the object notation, but he was the first one to give it a complete specification, based on parts of the JavaScript standard. Following that he launched the JSON.org website in 2002, which still exists and currently provides a listing of JSON libraries for different programming languages. It was initially developed for ECMAScript standartd(third edition, 1999) but was later derived and standardized as an independent standard in RFC4627 as a way to parse human-readable (in plain text format) representations of data into valid ECMAScript objects[4, 5]. It is completely language independent and uses notations similar to common programming languages such as C, C++, Java, etc.
It quickly grew in popularity partly due to its simplicity, which made it much more light weight -faster load times over the Internet- compared to XML, a format frequently used on the web. Also the growth in usage is a result of the increased use of JavaScript on the web.
As per RFC8259, "*JSON can represent four primitive types (strings, numbers, booleans, and null) and two structured types (objects and arrays)*". A JSON object consists of key-value pairs. Among these zero or more pairs, the key is a string and the value is either "*string, number, boolean, null, object, or array*" [6]. General syntax must meet the following rules:

- ***Comments*** are not allowed.
- ***Objects*** (unordered collection of name/value pairs) are denoted with braces ().
- ***Identifiers*** must be enclosed in quotes (as a string) and are followed by a colon and value.
- ***Objects*** (associative arrays / objects with key-value pairs)
- Multiple key-value pairs are separated with a comma
- ***Arrays*** (ordered set of values) are placed within brackets ([]) and separated by commas.
- The root node of a JSON document must be an object or an array.

A non-normative example of a JSON object including possible value types is shown below:

```
{
    ”a” :”b”,
    ”c” : 1,
    ”d” : null ,
    ”e” : {
        ”f” : ”g”
    },
    ”h” : [1, 2, 3],
    ”i” : true
}
```

Code 2.1: JSON Example

2.1.3 YAML

Yet Another UML front end (YAML) is a digestible data serialization language that is often utilized to create configuration files and works in concurrence with any programming language and designed for human interaction. YAML was first proposed by Clark Evans in 2001, who then designed it together with Ingy döt Net and Oren Ben Kiki. The format was developed from experience and discussions among sml-dev members on the Internet, and is still updated based on user input from the YAML-core mailing list. This answers the concern for it to be easy to understand and use, which is one of the primary goals for the format.

It's a strict superset of JSON, another (light-weight) data serialization language. But because it's a strict superset, it can do everything that JSON can and more. One major difference is that newlines and indentation actually mean something in YAML, as opposed to JSON, which uses brackets and braces. [7] By adding a simple typing system and aliasing mechanism upon the three most common data structures used when serializing (hashes, arrays and strings) it forms a language which is comparably very easy to use, while still including more complex features.

As it is mainly integrated and built upon concepts described by C, Java, Perl, Python, so to be easily extensible and readable by humans it is the mainly goal. JSON files are often valid YAML files because of the fact that JSON files are often valid YAML files because of the fact that JSON's semantic structure is equivalent to YAML's in line writing style (which was added in the new v1.2 specification of YAML). This means that YAML parsers adhering to the new specification also should be able to parse most JSON files.'s semantic structure is equivalent to YAML's in line writing style (which was added in the v1.2 specification of YAML). This means that YAML parsers adhering to the new specification also should be able to parse most JSON files. In addition to the basic data types available in JSON, YAML also supports relational trees. Relational trees are a language construct with which references to other nodes in the YAML document can be made[5]. A node in the YAML document tree can be defined as an anchor, and later references to that anchor will then include the data of the anchored node into the node. Smart use of this feature can lead to increased readability, compactness and clarity along with less chance of data entry errors.

The YAML specification also allows user-defined data types to be declared, as well as explicit data typing. This is especially useful for serialization purposes, allowing a parser to automatically construct an object of the correct class when deserializing, instead of an generic collection. YAML structures includes nodes and tags. A *node* represents a single native data structure, which can be a scalar, sequence or mapping. Each node can be marked with a *tag*, which restricts the set of possible values upon that node ID for data structures. YAML implements globally unique (means that the tag is unique through the whole process) local and URI tags. Local tags always starts with an exclamation mark and are specific for the current application. They are primary used to associate meta data to each node, but can also be used to specify additional information, such as allowed content or resolution.

The value is either "*string, number, boolean, null, date, map, sequence*" [6]. General syntax must meet the following rules: [7, 8]

- ***Comments*** ts begin with a hash/number sign (#) and continues to the end of the current line.

- ***Document*** data hierarchy is determined by indentation using double space characters (tab characters are not allowed as indentation).

- ***Comments*** are not allowed.

Sequences (arrays) syntax must meet the following rules:

- One item per line, marked with a dash and space.
- An alternative inline syntax exists, where the list is enclosed in brackets and items are separated by a comma followed by space.

Structures syntax must meet the following rules:

- Three repeated dashes denote the start of a document, and is also used to separate multiple documents in a single transmission.
- The root node of a document can be any valid data type
- Ending a transmission along with the current document is done with three repeated dots.
- Repeating nodes are defined with an ampersand and later referenced with an asterisk, where character is followed by an ID.
- A question mark and space in the beginning of a line denotes sets which are un-ordered.
- Values of user-defined data types can be denoted by prefixing them with a exclamation mark followed by the data type name, a space and finally the value.
- Explicit data type casting is done by prefixing the value in the same way as with user defined types but with an additional exclamation mark.

Strings syntax must meet the following rules:

- Quoting is often not required but can be, using either single or double quotes.
- The single quoted style is useful when no escaping is needed, while the double quoted style allows for escape sequences. It can span multiple lines and newlines are folded and included by a newline escape character (\n)
- Strings can be written using either the standard inline style (with or without quotes) or with block notation where a initial symbol determines how newlines in the document should be handled.
- Strings can be written using either the standard inline style (with or without quotes) or with block notation where a initial symbol determines how newlines in the document should be handled.
- Strings in block notation denominated with a pipe (|) will have their newlines preserved, while the greater than sign (>) will tell the YAML parser to convert newlines to spaces.

An example of a arbitrary YAML document including possible value types is shown below:

```yaml
--- # The Smiths
- {name: John Smith, age: 33}
- name: Mary Smith
  age: 27
- [name, age]: [Rae Smith, 4]   # sequences as keys are supported
```

```
---   # People, by gender
men: [John Smith,   Bill   Jones]
women:
  - Mary Smith
  - Susan Williams
```

Code 2.2: YAML Example

The YAML specification outlines four stages of data when loading and dumping to and from the format. Native data is seen as the first stage. The serialized YAML document (string) is the last stage of data. The two stages in between can be seen as working stages, where the data has been transformed into a node graph or event tree to be further processed. Serialization, or dumping as it is referred to, is done in three distinct stages which converts the data from a native data structure into series of bytes (strings). First, a directed graph is generated containing the structure - with nodes, sequences, mappings and scalars. The graph is then serialized, where sequential access mediums must be represented as ordered trees. In YAML they are created by ordered mappings, also called serialization trees. General mapping keys are unordered. Finally, the serialized tree is converted into a Unicode string. The load (deserialization) process is also compromised of three stages, which together does the opposite. The input (a string) is parsed to create a serialization tree in which the node hierarchy, keys, values and ordering is defined. This tree is then traversed node-to-node, where the data types of values are determined and converted to, as well as constructing relations and sequences. The final step converts the representation graph in to native data structures.

Type	JSON	YAML
Comments	Not allowed in the current specification, previously possible	Denoted with a hash/number sign, continues for the rest of the line.
Hierarchy	Objects and arrays can be nested, and are denoted by braces and brackets, respectively.	Mappings and sequences can be nested. Hierarchy is determined by indentation level.
Arrays	[”first”, ”second”, 3]	- first - second - 3 Alt. [first, second, 3]
Objects	”object”: ”a”: one, ”b”: 2	mapping: a: one b: 2 Alt. a: one, b: 2
Documents	Root node must be an array or object. Does not support multiple documents within a transmission.	Root note can be any valid data type. New documents in a transmission is denoted by three dashes. Repeated nodes are defined with ampersand, then referenced to with an asterisk.
Strings	Must be double quoted. Allows character (tabs, newlines, etc.) escaping with backslash as the escape character.	Does not require quoting but supports both single and double quotes (same functionality as JSON). Also provides two different block notations.
Numbers	Floating point numbers in scientific notation. Infinity is not permitted.	Built-in support for integers, floating-point, octal and hexadecimal numbers.

Table 2.1: Syntax comparison between JSON and YAML

In table 2.1 a summarization for the syntax comparison between JSON and YAML is provided.

2.2 Docker

2.2.1 Containers

The container abstraction has become a popular technique for running multiple application services on a single host. Containers are a way to bundle software in a format that can be run in an isolated manner on a shared operating system (OS). Containers should not be confused with virtual machines (VMs). However, in contrast to system virtualization, containerized applications do not contain an Operating system (OS) installation, but they share the host operating system kernel and services on he same host. So, they only wrap system libraries, files, and code that are required to make the target application run as expected – regardless of where the application is deployed. As stated in [9] the number of package changes varies across different types of applications and often the changing packages are utility packages. Container images can be supported across different OS platforms as they are available for both Linux and Windows-based apps and "**containerized**" software will always run the same, regardless of the environment. One feature Containers have, is that isolate software from its surroundings, for example, differences between development and staging environments and help reduce conflicts between teams running different software on the same infrastructure. [10]

2.2.2 Docker Registry

The Docker Registry is a platform for storing and sharing container images. It stores images in repositories, each containing different versions of the same image. Image layers are stored as compressed archival files and image manifests as JSON based files.

2.2.3 Docker Hub

Among all container solutions, **Docker containers** a complete packaging and software delivery tool, have recently become a prominent solution to provision multiple applications over shared physical hosts in a more lightweight fashion than traditional (VM) and provide near-native performance. This popularity has led to the creation of the Docker Hub public registry supporting both public and private repositories, which distributes at least half a million public images. [11, 12] It is important to state that **millions** of images have been downloaded from Docker Hub **billions** of times. In Docker Hub, the user repositories are name-spaced by user name, i.e., ***<username>/<repository_name>***, while the official repositories which are directly provided by Docker Inc. and partners, are called ***<repository_name>***.

2.2.4 Docker Images

At the center of Docker is the concept of container images for packaging, distributing and running applications. An Image is the "**template**" which contains the application and any associated binaries or libraries that are needed to build a Container.

An image is a lightweight, stand-alone, executable package that includes everything needed to run a piece of software, including the code, a run-time, libraries, environment variables, and config files.

Every package that gets added to the Image gets added as a new layer and the container is basically the running instance of the Image. In simply words, Docker is a series of individual layers.

2.2.5 Docker Layers

A layer contains a subset of files in thee image and often represents a specific component/dependency of the image e.g. a library. This design allows layer to be shared between two images depend on the same component.

Image layers are read-only. When a user start a container, Docker instantly creates a new *writable* layer on top of the underlying read-only layers as shown in Figure 2.1. As you can see the writable layer are discarded when the container is deleted.

An image is represented by a manifest file, which contains a list of layer identifiers (digests) for all layers required by the image. Also it describes the various parameters of a Docker image, such as the target hardware platform and environment settings.

2.2.6 Dockerfile file

Docker can build images automatically by reading the instructions from a Dockerfile. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Using docker build users can create an automated build that executes several command-

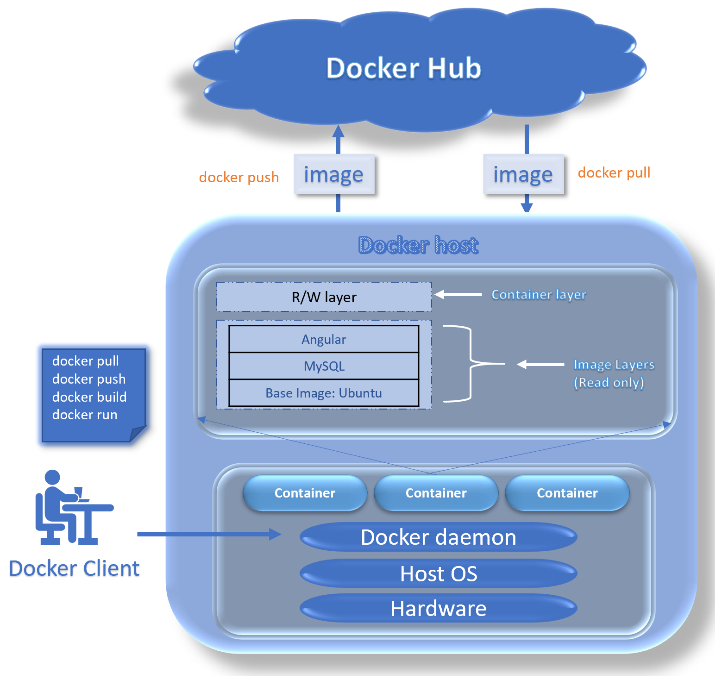


Figure 2.1: Docker Ecosystem

line instructions in succession. The advantage of a Dockerfile over just storing the binary image (or a snapshot/template in other virtualization systems) is that the automatic builds will ensure you have the latest version available. This is a good thing from a security perspective, as you want to ensure you're not installing any vulnerable software.

2.2.7 .dockerignore file

Before the docker CLI sends the context to the docker daemon, it looks for a file named .dockerignore in the root directory of the context. If this file exists, the CLI modifies the context to exclude files and directories that match patterns in it. This helps to avoid unnecessarily sending large or sensitive files and directories to the daemon and potentially adding them to images using ADD or COPY.

The cli interprets the .dockerignore file as a newline-separated list of patterns similar to the file globs of Unix shells. For the purposes of matching, the root of the context is considered to be both the working and the root directory.

2.2.8 Docker engine

A docker engine acts as a heart of any docker system. A host system that is installed with a docker application is technically termed as a docker engine.

- A long type of running process in docker termed is defined as a daemon process.
- The actual meaning of a client (CLI) is a command-line interface.
- For a virtual communication between CLI client and Docker daemon, a REST API is used.

2.2.9 Commands

Now that we have a basic understanding of what Docker is about and some of the key components and concepts are. It ‘s time to discuss Docker commands like RUN, CMD and ENTRYPOINT. [13]

- ***FROM*** command, which tells us what image to base this off of. This is the multi-layered approach that makes Docker so efficient and powerful. This is the multi-layered approach that makes Docker so efficient and powerful. In this instance, it’s using the *base* Docker image, which again references a Dockerfile to automate the build process.

- ***RUN*** executes command(s) in a new layer and creates a new image. E.g., it is often used for installing software packages. With this, we can execute commands. We need to specify an image to derive the container from. This command is mainly used for installing a new package.

- ***VOLUME*** command allows it to be externally mounted via the host itself or a Docker data container. If it is not defined, then it's not possible to access outside of the container.

- ***ENV*** command sets the environment variables, which can be used in the Dockerfile and any scripts that it calls. These are persistent with the container too, so they can be referenced at any time.

- ***COPY*** command is simply as it sounds. It can copy a file (in the same directory as the Dockerfile) to the container. You can do this for things like custom configuration files or like in this instance, a file to run commands after the container has been set up.

- ***CMD*** sets default command and/or parameters, which can be overwritten from command line when docker container runs. The CMD command specifies the instruction that is to be executed when a Docker container starts. This CMD command is not really necessary for the container to work, as the echo command can be called in a RUN statement as well. The main purpose of the CMD command is to launch the software required in a container. For example, the user may need to run an executable ".exe" file or a bash terminal as soon as the container starts – the CMD command can be used to handle such requests.

  The syntax of the CMD command have to follow the Shell syntax:

```
            CMD executable parameter1 parameter2 (shell form)
```

However, it is better practice to use the JSON array format:

```
CMD ["executable", "parameter1", "parameter2"] (exec form)
```

In principle, there should only be one CMD command in our Dockerfile. When CMD is used multiple times, only the last instance is executed. Many developers confuse CMD with ENTRYPOINT. However, ENTRYPOINT cannot be overridden by docker run. Instead, whatever is specified in docker run will be appended to ENTRYPOINT – this is not the case with CMD.

- ***ENTRYPOINT*** configures a container that will run as an executable. This command ultimately allows us to identify which executable should be run when a container is started from your image. It looks similar to CMD, because it also allows us to specify a command with parameters. The difference is ENTRYPOINT command and parameters are not ignored when Docker container runs with command line parameters. ENTRYPOINT has two forms The ENTRYPOINT command have to follow the Shell syntax:

```
ENTRYPOINT command param1 param2 (shell form)
```

However, it is better practice to use the JSON array format:

```
ENTRYPOINT ["executable", "param1", "param2"] (exec form)
```

Note that ENTRYPOINT and CMD cannot both be string values. They can both be array values, and ENTRYPOINT can be an array value and CMD can be a string value; but if ENTRYPOINT is a string value, CMD will be ignored. This is an unfortunate but unavoidable consequence of the way argument strings are converted to arrays. This is among the reasons I always recommend specifying arrays whenever possible.

- ***Other Usefull Commands*** By pushing the application image to a registry like Docker Hub, we make it available for subsequent use as you build and scale the containers. We will use this commands to push our custom containers to Docker Hub. The first step to pushing the image is to login to the Docker Hub account :

  ```
  docker login -u "dockerhub_username"
  ```

  We can then push the application images to Docker Hub using the tag we created:

  ```
  docker push "dockerhub_username/image-demo"
  ```

  To view the running containers the following command is the most appropriate:

  ```
  docker ps
  ```

  To stop a running application container the only requirement is to know its **ID**

  ```
  docker stop CONTAINER_ID
  ```

  To list all the images with the name of the image, dockerhub_username/image-demo, along with the node image and the other images from the build:

  ```
  docker images -a
  ```

  To remove the stopped container and all of the images, including unused or dangling images, with the following command:

2.3 Kubernetes

Kubernetes (also known as k8s or "kube") is an open source container orchestration platform that automates many of the manual processes involved in deploying, managing, and scaling containerized applications. You can cluster together groups of hosts running Linux containers, and Kubernetes helps to easily and efficiently manage those clusters. [14]

The platform was first developed by a team at Google, and later donated to the Cloud Native Computing Foundation (CNCF). It is not the only option for container management, but has gain great popularity over the years. As Opensource.com notes, ”Today, Kubernetes is a true open source community, with engineers' works from Google, Red Hat, and many other companies actively contributing to the project.”

Kubernetes clusters can span hosts across on-premise, public, private, or hybrid clouds. For this reason, Kubernetes is an ideal platform for hosting cloud-native applications that require rapid scaling, like real-time data streaming through Apache Kafka.[**?**]

The primary advantage of using Kubernetes in your environment, especially if you are optimizing development for the cloud, is that it gives you the platform to schedule and run containers on clusters of physical or virtual machines. [15]

More broadly, it helps you fully implement and rely on a container-based infrastructure in production environments. And because Kubernetes is all about automation of operational tasks, you can do many of the same things other application platforms or management systems let you do—but for your containers.

Kubernetes eases the burden of configuring, deploying, managing, and monitoring even the largest-scale containerized applications. It also helps application lifecycles, and issues including high availability and load balancing.

Kubernetes is not a traditional, all-inclusive PaaS (Platform as a Service) system. Since Kubernetes operates at the container level rather than at the hardware level, it provides some generally applicable features common to PaaS offerings, such as deployment, scaling, load balancing, and lets users integrate their logging, monitoring, and alerting solutions. However, Kubernetes is not monolithic, and these default solutions are optional and pluggable. Kubernetes provides the building blocks for building developer platforms, but preserves user choice and flexibility where it is important.

2.3.1 Kubernetes Features

Kubernetes provides you with: [16]

- Kubernetes allows to automatically **mount a storage system** of your choice, such as local storages, public cloud providers, and more.
- Kubernetes can expose a container using the DNS name or using their own IP address. If traffic to a container is high, Kubernetes is able to **load balance** and distribute the network traffic so that the deployment is stable.
- Allow to describe the desired state for the deployed containers using Kubernetes, and it can change the actual state to the desired state at a controlled rate. For example, you can automate Kubernetes to create new containers for the deployment, remove existing containers and adopt all their resources to the new container.
- Kubernetes can fit containers onto nodes to make the best use of your resources.
- If a container fails, Kubernetes restarts it. Kubernetes also replaces containers, kills containers that don't respond to the user-defined **health check**, and doesn't advertise them to clients until they are ready to serve.

2.4 User Authentication

Authentication is about validating your credentials like username and password to verify your identity. The system determines whether you are who you say you are using your credentials. In public and private networks, the system authenticates the user identity via login passwords. Authentication is usually done by a username and password, and sometimes in conjunction with factors of authentication, which refers to the various ways to be authenticated.

2.4.1 Authorization

Authorization is a function of the Identity Provider. This function is not based on a protocol. Authorization happens in the same way: the Identity Provider will first identify the user, then ask the user to authorize what the Identity Provider shares with the Application. The Identity Provider can ask for authorization to control what identity information and content is shared with the Application. For example, the IDP can ask if the user wants to share his email address with the Application. An Identity Provider can also ask for authorization to control the scope of an Access Token that is handed to an Application for API protection use cases. For example, the IDP can ask if the user wants to allow an Application access to some other service on their behalf. Access Token scope only exists for OAuth / OIDC.

2.4.2 Authentication

An application that wants to authenticate a user needs to receive identity information, i.e. claims identifying the user. Originally, claims were only defined for OIDC and SAML. However, with the introduction of the OAuth introspection service, an application can use OAuth only to receive identity claims implementing authentication use cases.

2.4.3 OAuth 2.0

OAuth 2.0 is an authorization framework that delegates user authentication to the service provider that hosts the user account, and authorizes third-party applications to access the user account. OAuth 2.0 provides authorization flows for web applications, desktop applications and mobile devices.

By introducing an authorization layer, OAuth 2.0 separates the role of the client from the resource owner, or end user. If the client requests access to resources controlled by the end user and hosted by the resource server, instead of using the end user's credentials to access protected resources, the client gets an access token. With the approval of the end user, the authorization server will issue access tokens to the requesting client.

OAuth 2.0 is explicitly designed to support a variety of different client types that access REST APIs. This includes applications running on enterprise web servers conversing with the cloud as well as applications running on employee or customer mobile devices. The OAuth framework supports a variety of client types by defining multiple mechanisms for getting a token where the different mechanisms acknowledge the client type constraints.

2.4.4 OpenID Connect

OpenID Connect (OIDC) is an identity layer built on top of the OAuth 2.0 framework. It allows third-party applications to verify the identity of the end-user and to obtain basic user profile information. OIDC uses JSON web tokens (JWTs), which you can obtain using flows conforming to the OAuth 2.0 specifications. OpenID Connect (OIDC) verifies a user's identity when they try to access a protected HTTPS endpoint. It uses straightforward REST/JSON message flows with a design goal of "making simple things simple and complicated things possible". It's uniquely easy for developers to integrate, compared to any preceding Identity protocol. OIDC was developed to work together with open authorization (OAuth) by providing an authentication layer to support the authorisation layer provided by OAuth. OpenID Connect lets developers authenticate their users across websites and apps without having to own and manage password files. For the app builder, it provides a secure verifiable answer to the question: "What is the identity of the person currently using the browser or native app that is connected to me?

2.4.5 SAML

One of the most important proposals in this area is the Security Assertion Markup Language (SAML) [17, 18]. SAML is a very extensible, open standard, which makes it attractive as a basis for further development. SAML is used for exchanging authentication and authorization data between parties, in particular, between an identity provider and a service provider. SAML is an

XML-based markup language for security assertions (statements that service providers use to make access-control decisions).[19] SAML allows so-called protocol bindings that embed SAML constructs in other structures for transport. SAML, for instance, builds on the Simple Object Access Protocol (SOAP) with its SOAP over HTTP binding. In addition, the SAML standard includes descriptions of the use of SAML assertions in communication protocols and frameworks. These so-called profiles contain protocol flows and security constraints for applications of SAML. An important use case that SAML addresses is web-browser single sign-on (SSO). Single sign-on is relatively easy to accomplish within a security domain (using cookies, for example) but extending SSO across security domains is more difficult and resulted in the proliferation of non-interoperable proprietary technologies. The SAML Web Browser SSO profile was specified and standardized to promote interoperability.

2.4.6 Differences between OAuth 2 and OpenID Connect

The main difference between OpenID and OAuth 2 is that OpenID is an authentication protocol while OAuth 2 is an authorization framework. OpenID and OAuth are both open standards that complement each other, but OpenID allows users to be authenticated by relying parties. An OIDC relying party is an OAuth 2.0 Client application that requires user authentication and claims from an OIDC provider. OAuth 2 allows access tokens to be issued to third-party clients by an authorization server. OpenID Connect is built on a profile of OAuth 2 and provides additional capabilities in conveying the identity of the user using the application. Clients use OAuth 2 to request access to an API on a user's behalf, but nothing in the OAuth 2 protocol tells the client user information. OpenID Connect enables a client to access additional information about a user, such as the user's real name, email address, birthdate or other profile information.

2.4.6.1 Protocols

The protocols being compared in this article are listed in the Table 2.2 below.

SAML 2.0	Security Assertion Markup Language
OAuth 2.0	Web Authorization Protocol
OpenID Connect 1.0 (OIDC)	Simple identity layer on top of OAuth 2.0

Table 2.2: Protocols for OAuth, SAML and OIDC

2.4.6.2 Actors and Roles

Role	**OAuth**	**OIDC**	**SAML**
End User	Resource Owner	End User	End User
Application	Client	Relying Party (RP)	Service Provider (SP)
Identity Provider	Authorization Server (AS)	OpenID Provider (OP)	Identity Provider (IDP)
Web Browser	User-Agent	User-Agent	User Agent

Table 2.3: Actors and roles for OAuth, SAML and OIDC

For the sake of simplicity I’m using a common terminology for various actors and roles in Table 2.3.

2.4.6.3 Comparison of Tokens

The underlying format for most tokens is either SAML Assertion or JWT. Both token formats are used to represent claims to be transferred between two parties.

JWT is a generic token format where claims are represented as JSON. JWT integrity is implemented by JSON Web Signature (JWS). The JWT specification by itself does not define any specific use cases. The JWT token format is, however, widely used. OpenID Connect specifies the ID Token as JWT – many vendors have selected JWT as a format for Access Token and there exist numerous API protection use cases, un-related to OAuth / OIDC, where JWT formatted tokens are used.

The format for Access Tokens is surprisingly not defined by the OAuth specification. Instead, an interoperable application is expected to treat Access Tokens as opaque strings and use the OAuth Introspection service for validating tokens. Because representing Access Tokens as JWT is quite popular, there is, however, work going on to define an interoperability profile for JWT formatted Access Tokens.

SAML Assertion is based on XML. SAML Assertion integrity is implemented by XML Digital Signature. A SAML Identity Provider uses SAML Assertion token to transfer claims to an Application. The Web Services Security profile defines how to implement protection of SOAP APIs with SAML Assertion tokens.

The table 2.4 lists tokens present in OAuth / OIDC and SAML protocols.

Token	**Format**	**API protection use case**	**Validation**
JWT	JSON	REST and Bearer token	JSON Web Signature
Access Token	Not defined, often JWT	REST and Bearer token	OAuth Introspection
ID Token	JWT	REST and Bearer token	Json Web Signature
SAML Assertion	XML	SOAP and Web Services Security	XML Signature

Table 2.4: Tokes of OAuth, SAML and OIDC

2.4.7 Keycloak

Managing user identity is a critical feature for science gateways [20, 21], which must provide secure and auditable access to restricted resources such as supercomputers, data sets, licensed scientific applications, and for-fee cloud computing. Science gateways must authenticate users, decide if they are authorized to access specific resources, manage expired accounts, and disable compromised accounts. The basic approach is for a gateway to provide its own user management and authentication system that is an integral part of the gatewayś implementation.

Gateway developers today have several additional options, building on larger trends. First is the emergence of well-supported authentication services, such as the In Common Federation [22] that is used by many academic institutions. Facebook, Google, GitHub and other Web-based companies also provide free authentication services that can be integrated into online applications. OpenID Connect (OIDC) has become a popular protocol for Authentication; it builds over the OAuth2 authorization protocol.[23]

CILogon [24] provides a unifying authentication layer over these different providers where the gateways may outsource user authentication to various services. Although, the gateway may still choose to manage its users internally through a user store (such as an attached database or an LDAP server), or it might outsource this as well. The second important trend has been the emergence of science gateway platform-as-a-service (PAS) offerings. These are hosted services that can serve multiple gateway tenants simultaneously. Science gateway platforms provide general purpose services such as user management, data management, and job execution, while the gateway tenant provides specific access to computational and storage re-sources, data and applications and user interfaces geared towards a specific user community. Gateway tenants access the gateway platform middleware through secure, network accessible APIs.

I will use Keycloak [25], an open source identity management system, to implement the Identity and Access Management system. Keycloak can authenticate users through a number of different mechanisms. With KeyCloak, the user management and authentication functions may be integrated with an externally managed system, such as LDAP or active directory. KeyCloak provides single sign-on infrastructure for authentication and session management.

Keycloak offers a feature set very similar to WSO2 IS. For example, Keycloak provides a user store and administrative functions for administering users, including user roles. Keycloak also provides some interesting new features. For example, Keycloak also supports identity federation for identity providers that support OpenID Connect or SAML.

Once the user is authenticated during the signin process, Keycloak redirects back to the web portal with an authorization code. The web portal can then use the authorization code to get an access token and retrieve the user's profile from Keycloak. This user profile (first name, last name, email address) will match the details that Keycloak itself retrieves from ML Lab.

Users are assigned to one or more roles to grant them access to different subsets of ML LAb. Keycloak exposes a REST that allows a gateway administrator to manage a user's roles. This functionality is exposed in the web portal to users with the admin role. Admins are able to manage the roles assigned to a user. Typically new users are assigned to a role which has no API access and a decision must be made by an admin user as to which role(s) to assign the user.

2.4.7.1 Basic Features

Some of the basic features coming from Keycloak build-in are:

- ***Single-Sign On and Single-Sign Out*** for browser applications.
- ***OpenID Connect*** support.
- ***OAuth 2.0*** support.
- ***SAML*** support.
- ***Identity Brokering*** authenticate with external OpenID Connect or SAML Identity Providers.
- ***Social Login*** Enable login with Google, GitHub, Facebook, Twitter, and other social networks.
- ***User Federation*** that sync users from LDAP and Active Directory servers.
- ***Kerberos bridge*** which automatically authenticate users that are logged-in to a Kerberos server.

- ***Admin Console*** for central management of users, roles, role mappings, clients and configuration.
- ***Account Management Console*** that allows users to centrally manage their account.
- ***Theme support*** Customize all user facing pages to integrate with your applications and branding.
- ***Two-factor Authentication*** Support for TOTP/HOTP via Google Authenticator or FreeOTP.
- ***Login flows*** optional user self-registration, recover password, verify email, require password update, etc.
- ***Session management*** Admins and users themselves can view and manage user sessions.
- ***Token mappers*** Map user attributes, roles, etc. how you want into tokens and statements.
- ***Not-before*** revocation policies per realm, application and user.
- ***CORS support*** Client adapters have built-in support for CORS.
- ***Service Provider Interfaces (SPI*** A number of SPIs to enable customizing various aspects of the server. Authentication flows, user federation providers, protocol mappers and many more.
- ***Client adapters*** for JavaScript applications, WildFly, JBoss EAP, Fuse, Tomcat, Jetty, Spring, etc.
- ***Supports any platform/language*** that has an OpenID Connect Resource Provider library or Service Provider library.

2.4.7.2 Security Flow

Applications are configured to point to and be secured by a Keycloak separate manageable server. Keycloak uses open protocol standards like Open ID Connect or SAML 2.0 to secure your applications. Browser or Mobile applications redirect a user's browser from the application to the Keycloak authentication server where they are prompt to enter their credentials. Keep in mind that users are completely isolated from applications and applications never see users credentials. Applications instead are given an identity token or assertion that is cryptographically signed. These tokens can have identity information like username, address, email, and other profile data. These tokens can also hold permission data so that applications can make authorization decisions and can also be used to make secure invocations on REST-based services.

2.4.7.3 Core Components and Terms

There are some key Components and Terms important when attempting to secure your web applications and REST services.

- ***authentication*** The process of identifying and validating a user.
- ***authorization*** The process of granting access to a user.
- ***credentials*** Credentials are pieces of data that Keycloak uses to verify the identity of a user, like passwords, one-time-passwords, digital certificates, or even fingerprints.
- ***roles*** Roles identify a type or category of user. Applications often assign access and per-

missions to specific roles rather than individual users as dealing with users can be too fine grained and hard to manage.

- ***user role mapping*** A user role mapping defines a mapping between a role and a user. A user can be associated with zero or more roles. This role mapping information can be encapsulated into tokens and assertions so that applications can decide access permissions on various resources they manage.

- ***composite roles*** A composite role is a role that can be associated with other roles.

- ***groups*** Groups manage groups of users. Attributes can be defined for a group. You can map roles to a group as well. Users that become members of a group inherit the attributes and role mappings that group defines.

- ***realms*** A realm manages a set of users, credentials, roles, and groups. A user belongs to and logs into a realm. Realms are isolated from one another and can only manage and authenticate the users that they control.

- ***clients*** Clients are entities that can request Keycloak to authenticate a user. Most often, clients are applications and services that want to use Keycloak to secure themselves and provide a single sign-on solution. Clients can also be entities that just want to request identity information or an access token so that they can securely invoke other services on the network that are secured by Keycloak.

- ***client adapters*** Client adapters are plugins that can be install into the application environment to be able to communicate and be secured by Keycloak. Keycloak has a number of adapters for different platforms that you can download. There are also third-party adapters you can get for environments that we don't cover.

- ***client templates*** When a client is registered you need to enter configuration information about that client. It is often useful to store a template to make create new clients easier. Keycloak provides the concept of a client template for this.
- ***client role*** Clients can define roles that are specific to them. This is basically a role namespace dedicated to the client.
- ***users*** Users are entities that are able to log into the system. They can have attributes associated with themselves like email, username, address, phone number, and birth day. They can be assigned group membership and have specific roles assigned to them.
- ***consent*** Consent is when you as an admin want a user to give permission to a client before that client can participate in the authentication process. After a user provides their credentials, Keycloak will pop up a screen identifying the client requesting a login and what identity information is requested of the user. User can decide whether or not to grant the request.
- ***identity token*** A token that provides identity information about the user. Part of the OIDC specification.
- ***access token*** A token that can be provided as part of an request that grants access to the service being invoked on. This is part of the OIDC and OAuth 2.0 specification.
- ***assertion*** Information about a user. This usually pertains to an XML blob that is included in a SAML authentication response that provided identity metadata about an authenticated user.

- ***service account*** Each client has a built-in service account which allows it to obtain an access token.
- ***direct grant*** A way for a client to obtain an access token on behalf of a user via a REST invocation.
- ***protocol mappers*** For each client you can tailor what claims and assertions are stored in the OIDC token or SAML assertion. You do this per client by creating and configuring protocol mappers.
- ***session*** When a user logs in, a session is created to manage the login session. A session contains information like when the user logged in and what applications have participated within single-sign on during that session. Both admins and users can view session information.
- ***user federation provider*** Keycloak can store and manage users. Often, companies already have LDAP or Active Directory services that store user and credential information. Just point Keycloak to validate credentials from those external stores and pull in identity information.
- ***identity provider*** An Identity Provide (IDP) is a service that can authenticate a user. Keycloak is an IDP.
- ***identity provider federation*** Keycloak can be configured to delegate authentication to one or more IDPs. Social login via Facebook or Google+ is an example of identity provider federation. You can also hook Keycloak to delegate authentication to any other Open ID Connect or SAML 2.0 IDP.

- ***identity provider mappers*** When doing IDP federation you can map incoming tokens and assertions to user and session attributes. This helps you propagate identity information from the external IDP to your client requesting authentication.
- ***required actions*** Required actions are actions a user must perform during the authentication process. A user will not be able to complete the authentication process until these actions are complete. For example, an admin may schedule users to reset their passwords every month. An update password required action would be set for all these users.
- ***authentication flows*** Authentication flows are work flows a user must perform when interacting with certain aspects of the system. A login flow can define what credential types are required. A registration flow defines what profile information a user must enter and whether something like reCAPTCHA must be used to filter out bots. Credential reset flow defines what actions a user must do before they can reset their password.
- ***events*** Events are audit streams that admins can view and hook into.
- ***themes*** Each screen of Keycloak is backed by a theme. Themes define HTML templates and stylesheets which you can override as needed.

2.5 Spring cloud dataflow

Spring Cloud Data Flow is the cloud native redesign of Spring XD - a project that aimed to simplify Big Data application development. This redesign allows running stream and batch applications as data microservices and they can independently evolve in isolation. Spring Cloud Data Flow (SCDF) is an open-source Java-based cloud-native toolkit developed by Pivotal (VMWare) to orchestrate data integration, real-time data streaming, and batch data processing pipelines by stitching together spring boot microservices that can be deployed on top of different modern runtimes like Cloud Foundry, Kubenetes, YARN, Mesos, etc. in addition to a local runtime. Data pipelines deployed using Spring Cloud Data Flow consist of Spring Boot apps built using Spring Cloud Stream or Spring Cloud Task microservice frameworks. e. It's a single toolkit that developers can employ to create, orchestrate, and refactor data pipelines through one programming model to address common use cases such as data ingestion, real-time analytics, and data export/import across popular source and destination systems. SCDF allows developers to interact to define and deploy data pipelines through multiple endpoints like:

- Dashboard GUI (that allows defining pipeline through a fluid drag and drop palette)
- Command-Line Shell
- Stream Java DSL
- RESTful APIs

2.5.1 Features of Spring Cloud Data Flow

- **Orchestrate applications** across a variety of distributed runtime platforms, including Cloud Foundry, Apache YARN, Apache Mesos, and Kubernetes.
- Design, deploy, and manage data pipelines using: Java DSL, Shell, REST-APIs, and Admin-UI.
- The programming model offers runtime and message broker abstractions.

- Building streaming and batch applications using popular configuration driven Spring Boot backed Spring Cloud Stream and Spring Cloud Task projects.
- Pluggable messaging broker binders let developers use the same application code and bind to any popular messaging services.
- Take advantage of metrics, health checks, and remote management of data-microservices.
- Standard security semantics in the form of OAuth2 and OpenID Connect backed authentication and authorization.
- Scale stream and data pipelines with almost zero downtime without interrupting the data flow.
- UI dashboards to design, deploy, and manage large-scale and compute-intensive batch data pipeline through Spring Batch jobs.

2.5.2 Components

- ***Data Flow Server*** The core of the SCDF ecosystem is the Data Flow Server, a Spring Boot based microservice application that provides the main entry point to define data pipelines in SCDF through RESTful APIs and a web dashboard. This server is responsible for parsing the stream and batch job definitions based on a Domain Specific Language (DSL). The server requires a relational database to persist the metadata related to stream, task, or job definitions and register artifacts such as additional library jar files or docker images used in the pipeline definitions. The data flow server can deploy the Batch Jobs to one or more supported runtime platforms.

- ***Skipper Server*** The skipper server in the SCDF ecosystem is responsible for deploying the streaming data pipeline definitions to one or more supported runtime platforms using the Spring Cloud Deployer family of libraries. It is a Spring Boot based microservice that behaves as a package manager that installs, upgrades, and rolls back applications to one or more runtime platforms using a blue-green deployment strategy. Just like the Data Flow server, it also exposes RESTful APIs to access the functionalities it offers for stream deployment and application management.

2.5.3 Applications

There are two types of applications can be packaged as Docker App Images or as spring boot jar hosted in maven repository, file or httpthat are supported by Spring Cloud Data Flow.

- **Short-Lived Applications:** that finite period of time (minutes or hours) and then terminate. The executions are triggered on a recurring schedule (such as every day at midnight) or as a response to some external event (such as a file being copied into a landing zone). These applications are developed using the **Spring Cloud Task framework** sthat records lifecycle events (such as the start time, end time, and the exit code) of the application into the relational database attached to the Data Flow Server. These applications can also be developed as Spring Batch jobs since Spring Cloud Task is well-integrated with it. A short-lived application is registered with Data Flow using the category name task to describe the type of application.

- **Long-Lived Applications:** These run continuously as part of the data-streaming pipeline. A typical data-streaming pipeline operation involves consuming events from external systems, processing or transforming the data from the events, and writing to persistent storage. In SCDF, these event-streaming pipelines are generally composed of Spring Cloud Stream applications which are broadly categorized as *Source*, *Processor* and *Sink* applications:

    1. **A source** represents the first step in the data pipeline. It is a producer that consumes data from external systems like databases, filesystem, FTP servers, IoT devices, etc.
    2. **A processor** represents an application that can consume from an upstream producer (a source or another processor), perform the business operation on the consumed data and emit the processed data for downstream consumption.
    3. **A sink** represents the final stage in the data pipeline, which can persist the consumed data to external systems like HDFS,Cassandra, PostgreSQL, Amazon S3, etc.

Sources, sinks, and processors all have a single output, a single input, or both. This is what makes it possible for Data Flow to set application properties that pair an output destination to an input destination. However, a message processing application could have more than one input or output destination. Spring Cloud Stream supports this by letting you define a custom binding interface.

2.5.4 Message Broker

A messaging middleware service is required to facilitate communication between applications in a SCDF pipeline. The framework provides a programming model that allows support for pluggable message binder libraries. The currently available binders support following messaging broker services are:

- RabbitMQ
- Kafka
- Amazon Kinesis
- Google Pub/Sub
- Azure Event Hubs

2.5.5 Storage

The ML Lab Server and Skipper Server need to have an RDBMS installed. By default, the servers use an embedded H2 database.The supported databases are H2, HSQLDB, MySQL, Oracle, Postgresql, DB2, and SqlServer and the schemas are automatically created when each server starts.

2.6 Differences

In this section, I am going to state the most used alternatives of the Spring Cloud dataflow and describe the cons and pros of each. Spring Cloud dataflow's architectural style is different than other Stream and Batch processing platforms.

As we discovered during the evolution of Spring XD, the rise of multiple container frameworks in 2015 made creating our own runtime a duplication of effort. There is no reason to build your own resource management mechanics when there are multiple runtime platforms that offer this functionality already. Taking these considerations into account is what made to think about what extensions can I provide to provide more feature like the export of the manifest as YAML file and the GPU deployer property configuration. Let's discuss about the other platforms.

(1) **Cloud Dataflow** is a Google-powered processing platform designed to execute data processing pipelines. The platform, allow us to develop simple streaming data pipelines with lower data latency. Google Cloud Dataflow has a serverless approach that allow developers' focus to programming instead of managing countless server clusters. It offers an infinite capacity to manage your workloads. And with that, you don't have to worry about high ownership costs. By using Apache Beam SDK for MapReduce operations and accuracy control for batch and streaming data, it reduces complexities and makes stream analytics very accessible to both data analysts and data engineers. The framework can be used to develop anomaly detection applications or a real-time website analytics dashboard or a pipeline that processes log entries from various sources. Some of the advantages that Google cloud Dataflow offers is that it is fully managed, by removing operational complexities, minimize pipeline latency and providing access native integrations with AI Platform enable to develop Unified stream and data processing analysis. But It is restricted to only Cloud Datastore service and streaming mode can be expensive. One other thing is that Google Content Delivery Network does not work with custom sources, something that I wanted for my toolkit to provide.

(2) **Apache Pulsar** is a cloud-native, distributed messaging and streaming platform. Originally deployed inside Yahoo, Pulsar that a high-performance solution for server-to-server messaging and geo-replication of messages across clusters. Additionally, it can scale to over a million topics and expand to hundreds of nodes. It's lightweight, easy to deploy, and doesn't need an external stream

processing engine. The processing platform has a multi-layer architecture. Each of these layers is scalable and can be distributed and decoupled from the other. Not only that, it has granular resource management that prevents producers, consumers, and topics from overwhelming the cluster. It is also easily managed by user and supports high-level ”APIs” for Java, Go, Python, C++, and C# and data replication across data centers in different geographical locations with end-to-end encryption from the client to the storage nodes. But since it is a newer solution the community is still small and online instructions and guides are hard to find. It also not allow consumers to acknowledge message from a different thread.

(3) **Apache Spark** is an engine for data processing, it is being highly used for data intensive processing and data science. It has libraries such as ML (Machine Learning), Graph (graph processing), integration with Apache Kafka (Spark Streaming), among others.

(4) **Apache Flink** is an open source system for fast and versatile data analytics in clusters. Flink supports batch and streaming analytics, in one system. Analytical programs can be written in concise and elegant ”APIs” in Java and Scala. Apache Flink is a tool in the Big Data Tools category of a tech stack. Apache Flink is an open source tool with 17.2K GitHub stars and 9.6K GitHub forks.

Some interesting conclutions is that in Apache Spark, Apache Flink, and Google Cloud Dataflow, applications run on a dedicated compute engine cluster. The nature of the compute engine gives these platforms a richer environment for performing complex calculations on the data as compared to Spring Cloud Data Flow, but it introduces the complexity of another execution environment that is often not needed when creating data-centric applications. That does not mean you cannot do real-time data computations when using Spring Cloud Data Flow. Refer to the section Analytics, which describes the integration of Redis to handle common counting-based use cases. Spring Cloud Stream also supports using Reactive ”APIs” such as Project Reactor and RxJava which can be useful for creating functional style applications that contain time-sliding-window and moving-average functionality. Similarly, Spring Cloud Stream also supports the development of applications in that use the Kafka Streams API.

2.7 Apache Zeppelin

Data scientists use different applications like R, Python or Scala (with notebook tool like Apache Zeppelin) to develop data science models. For example, some prefer R to create their models, others like to write code for their models in languages like Python or Scala using notebook tools like Apache Zeppelin and so on. ML Lab, a real-time streaming analytics platform, allows users to build and deploy data models by using different tools like Scala, pyspark. This streaming analytics platform supports multiple languages and formats, enabling users to create the code in their preferred technology. Once the model is prepared, it can be deployed on ML Lab to run and perform scoring over the data in a distributed fashion.

Apache Zeppelin is a web-based notebook that enables interactive data analytics. Zeppelin supports many interpreters such as Scala, Python, Spark SQL, JDBC, Markdown and Shell.

Computer science that facilitates extraction of knowledge or insights from large amounts of structured or unstructured data. This process of data discovery can be divided into multiple steps. Thus, this is an iterative process of data collection, cleaning, analysis, visualization and decision making. Apache Zeppelin is an open source project for simplifying big data analytics with web-based notebooks that enable interactive data analytics. It has multiple language backends and Apache Spark [26] integration, thus allowing to address various analytic tasks inside notebooks. Moreover it allows interactive visualization [27] and collaboration with ready insights into data.

Thus, Apache Zeppelin can cover whole data discovery flow inside a single Zeppelin notebook. Apache Zeppelin is currently an incubating project under Apache Software Foundation (ASF) [28] with all the following copyright, development process, and community decision making implications. It has more than 70 contributors and its first release of 0.10.0 version happened in August of 2021. The evolution of the different version of Apache Zeppelin is shown in Figure 2.2.

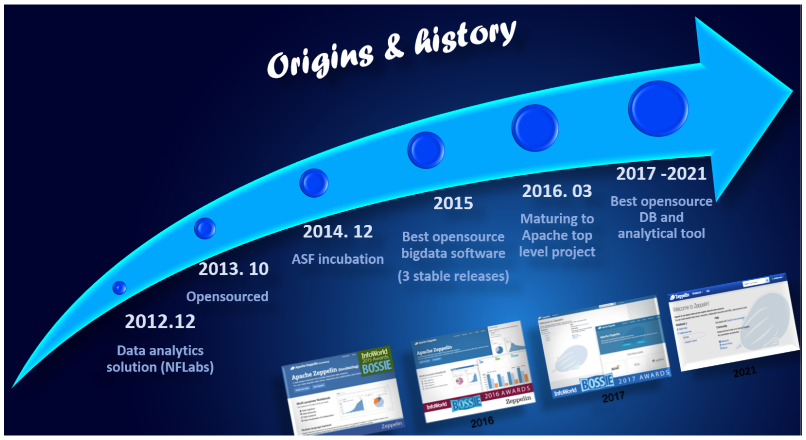

Origins & history
2012.12
Data analytics solution (NFLabs)
2013. 10
Opensourced
2014. 12
ASF incubation
2015
Best opensource bigdata software
(3 stable releases)
2016. 03
Maturing to Apache top level project
2017 -2021
Best opensource DB and analytical tool
2016
2017
2021

Figure 2.2: Apache Zeppelin Evolution

Apache Zeppelin has pluggable backend architecture in terms of its interpreters. New interpreters can be implemented and plugged in via its Interpreter interface. Figure 2.3 showcases some of the existing interpreters supported by Zeppelin. Thus all of the backend frameworks implemented by interpreters can be used inside Zeppelin. Moreover, one Zeppelin notebook may contain different paragraphs each using its own language/interpreter. Since Zeppelin notebooks are web-based, Zeppelin has its web-app front-end (client) and back-end server. Zeppelin client communicates with back-end server using HTTP REST and websocket ”APIs”. Further, server communicates with the interpreter processes. For example, Spark interpreter is just one of the interpreter processes running. According to clients query, server calls corresponding interpreter with a query to interpret the code. Note that interpreter initialization is lazy and is not started unless client submitted query for it.

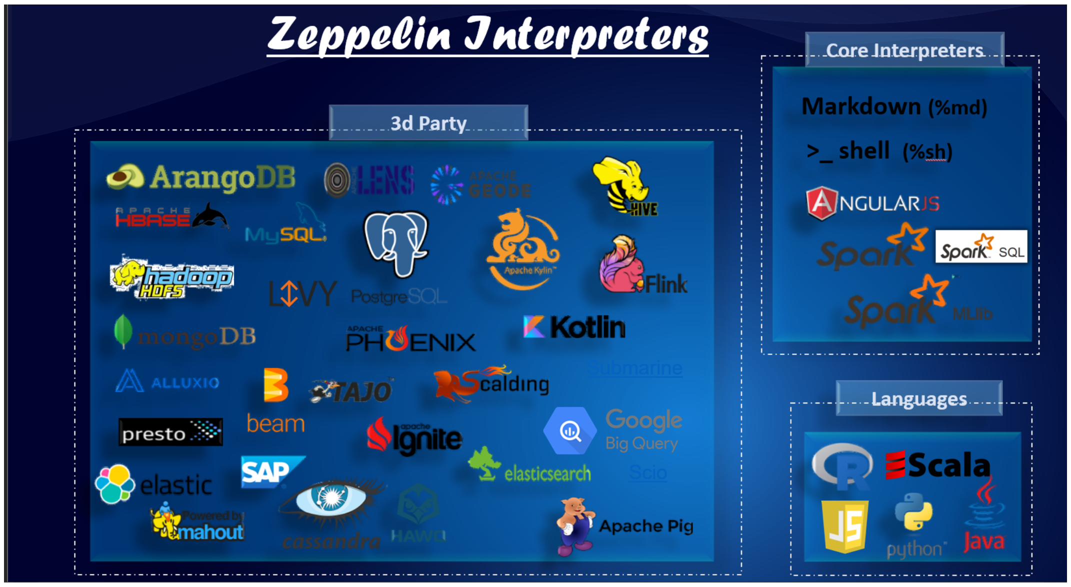


Figure 2.3: Apache Zeppelin Interpreters

In Zeppelin we can download data using simple shell script as if typing in terminal. This is realized by the means of shell interpreter. Or even load data from different databases. Some basic charts are already included in Apache Zeppelin. Any output from any language backend can be recognized and visualized. Apache Zeppelin aggregates values and displays them in pivot chart with simple drag and drop. You can easily create chart with multiple aggregated values including sum, count, average, min, max. But can also create some input forms in the notebooks, which can be very useful when want to visualize result or get user input.

2.8 Virtual Assistant Chatbot

Smart Personal Assistants (SPAs; such as Amazon’s Alexa or Google’s Assistant) let users interact with computers in a more natural and sophisticated way that was not possible before. Although there exists an increasing amount of research of SPA technology in education, empirical evidence of its ability to offer dynamic scaffolding to enhance students problem-solving skills is still scarce. A virtual assistant chatbot is a mash up of two separate programs – a chatbot, and a virtual assistant. The only similarity between these two programs is that they are both built to make the lives of humans easier through conversations. Let’s understand the basic differences between chatbots and virtual assistants. Chatbots are programs that are designed with the purpose of engaging with customers in human-like conversations. Thus, chatbots are deployed by businesses to interact with customers (or prospects) and offer assistance around the clock. Chatbots are intelligent enough to sense the context of the conversation and execute the right bot flow. However, chatbots cannot find answers or perform a set of activities on their own. On the other hand, a virtual assistant can crawl through existing resources and offer assistance for a wide range of requests. There are two main types of chatbots:

- ***Rule-based chatbots*** These are decision-tree chatbots coded with specific rules aimed at addressing particular inquiries. The system maps out potential conversations like a flowchart and provides answers to all possible questions. It uses simple or highly complex rules which don’t answer random or unrelated questions. They only answer already programmed questions.

- ***AI-powered chatbots*** Unlike their rule-based counterparts, AI-powered virtual assistants can understand context and intent through machine learning. With time, AI-powered bots can learn from mistakes and feedback to provide more accurate responses.

Virtual assistants (also popularly known as intelligent virtual assistants or intelligent personal assistants) are online personal assistants that help people with their day-to-day activities such as managing their email, scheduling meetings, etc. Popular virtual assistants include Amazon Alexa, Apple's Siri, Google Assistant, and Microsoft's Cortana. While these virtual assistants can assist you with many everyday tasks. Now that we have a basic understanding of what chatbots and virtual assistance are, let's dive deeper into the key differences.

CHAPTER 3

Prerequisites

3.1 Prototype

As shown in Figure 3.1 I am going to provide a combine solution that can not be found in the literature at the moment.

ML Lab comes with collection of best practices and best collection of microservices based distributed streaming, task and batch applications. The key parts of the ML Lab are Ml Lab-server and Skipper, which comprise the actual DataFlow experience and app deployment. Grafana and Prometheus, which are used for metric gathering and visualization, RabbitMQ/Kafka and Zookeeper which is our messaging platform and the coordination server.

I customise and extend the functionality of the Spring Cloud dataflow to enable more features. I have also take advantage of the Apache Zeppelins notebooks functionality by enable the notebooks functionality inside the ML Lab using the Apache Zeppelin Server and replicating all the Apache Zeppelin Web Client functionality to meet with the Spring Cloud Dataflow dependencies and coding style requirements. I also connected the Mycroft Core (server) to the ML Lab by creating a web widget written in Angular that enables the exchange of the intent and the answers

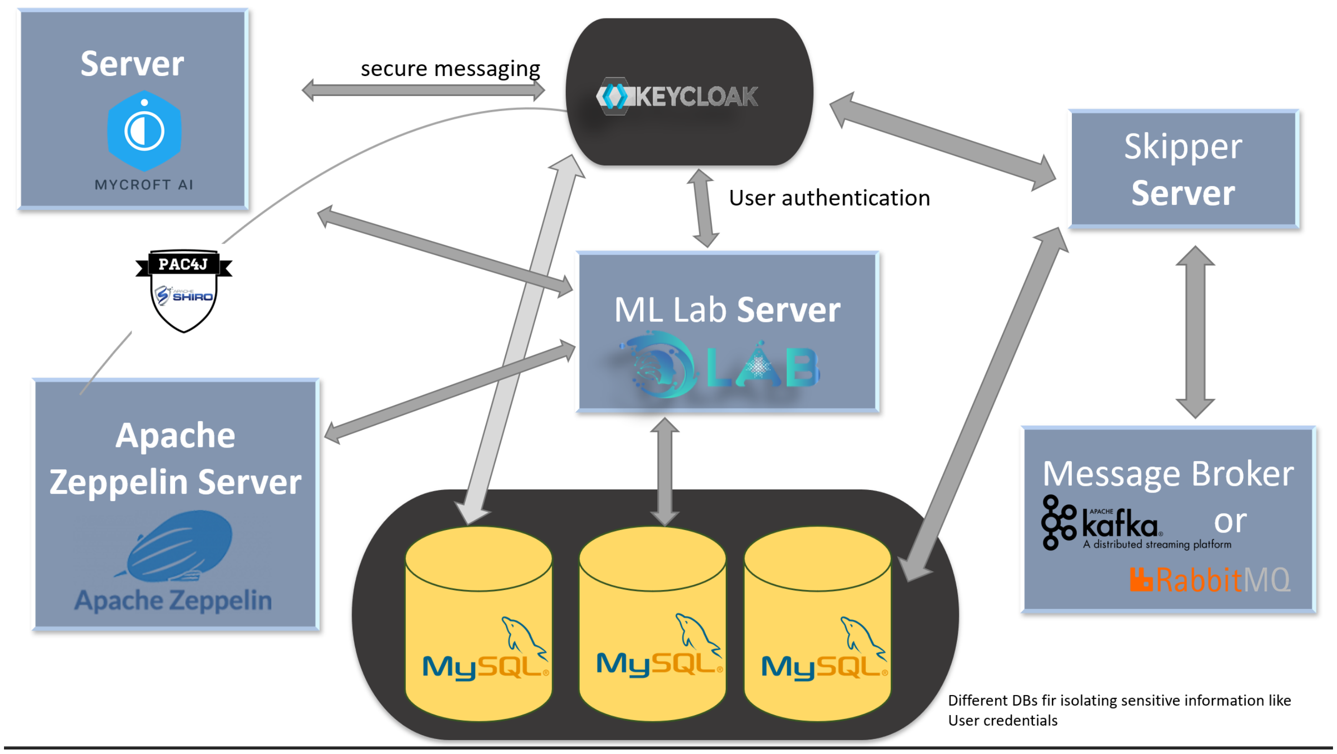


Figure 3.1: ML Lab Prototype

between the user and the Mycroft Core to provide user with capabilities to ask for information and receive an answer. But also to provide them with them with the answers that might have about how to use the environment or set reminders or timers to complete their tasks. For the authentication mechanism Spring Cloud dataflow Server use the Oauth2, Apache Zeppelin uses Apache Shiro and Mycroft comes without any security mechanism. A selected to use Keycloak provide user authentication for the ML Lab and secure the REST API and the websockets to provide a secure environment. It is important to state that during the development of this prototype this mechanism has not been used before to secure this applications. ML Lab Server, Skipper Server and Keyclock need to have an RDBMS installed. I configured a database for each inside container where a different database created for each.

3.2 Technologies and Languages

To implement this Master thesis I have to use the Docker and Kubernetes to containerised the combined application to make sure that all provided service could scale based on the user need. I take advantage of the Spring clouf Dataflow to provide the strea/batch processing of data and extend its functionality to enable user to create and write notebooks. That they can in future version of this Toolkit, insert the prepared code as an application to enable future to support deep learning project preparation with Spark. The main programming languages that was required for this deployment are:

- **Angular** is a JavaScript framework for building web applications and apps in JavaScript, HTML, and TypeScript, which is a superset of JavaScript. Angular provides built-in features for animation, HTTP service, and materials which in turn has features. The code is written in TypeScript, which compiles to JavaScript and displays the same in the browser. Angular, Google's TypeScript based web application framework is currently at version 12.1.4 (stable).
- **Python** Python is an interpreted high-level general-purpose programming language. Its design philosophy emphasizes code readability with its use of significant indentation. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. Python is currently at version 3.9.7 (stable).
- **Java** is a high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. Java is is currently at version JDK 17

(stable).

3.3 Kubernetes Configuration

Containers are a good way to bundle and run your applications. In a production environment, you need to manage the containers that run the applications and ensure that there is no downtime. For example, if one of the containers in our prototype goes down, another container needs to start. Kubernetes provides a framework to run distributed systems resiliently. It takes care of scaling and failover for your application, provides deployment patterns, and more. For example, Kubernetes can easily manage a canary deployment for your system. Our prototype consiting of many elements that provide services to the user, if one goes down we want Kubernetes to start a new one. Also as when deploying a stream or a task pipeline each node of the pipeline triggers the creation of a container.

The Components described in Figure 3.1 are deployed at the implementation phase using Minikube is local Kubernetes, focusing on making it easy to develop for Kubernetes. We configured our Kubernetes cluster to have enough recourses to develop our vision. Afterwards to start the cluster by using the minikube start command. We use the version 1.21.2 for be compatible with the Spring Cloud Dataflow Kubernetes Integration.

```
minikube config set driver hyperv
minikube config set cpus 2
minikube config set memory 12G
minikube config set disk-size 10G
minikube config set EmbedCerts true
```

```
    minikube config set insecure-registry true
    minikube config set kubernetes-version 1.21.2
    minikube config set profile minikube-testing
```

I configure the deployments of the Spring Cloud dataflow template files. And change the images for the server to use our custom image with the functionality I created. I start by deploying the message brokers, the RabitMQ and Kafka.

```
apiVersion: v1
kind: Service
metadata:
  name: rabbitmq
  labels:
    app: rabbitmq
spec:
  type: ClusterIP
  ports:
  - port: 5672
    targetPort: 5672
  selector:
    app: rabbitmq

---
apiVersion: apps/v1
```

```
kind:  Deployment
metadata:
  name: rabbitmq
   labels :
     app: rabbitmq
spec:
   replicas :  1
   selector :
     matchLabels:
        app: rabbitmq
  template:
     metadata:
        labels :
          app: rabbitmq
     spec:
        containers :
        -  image: rabbitmq:3.6.10
          name: rabbitmq
          resources:
             limits :
               cpu:  "500m"
               memory: "800Mi"
        requests:
          cpu: "100m"
          memory: "100Mi"
      ports:
      - containerPort: 5672
```

Code 3.1: RabbitMQ deployment files

Then I have to set-up the MySql container based on the mysql:5.7.25 image, and including a persistent volume claim and a secret that holds the sensitive admin credentials. I enable monitoring using the Prometheus and the Grafana. Deploy the keycloak container with my custom docker image that provides the custom UI and the pre-created realm. and then deploy the the ML Lab server. Before deploying Skipper we need to create the RoleBindings and ServiceAccount used by the Spring Cloud Dataflow Server. To enable the stream management features we must deploy Skipper. First apply the appropriate Skipper ConfigMap, according to the message broker (RabbitMQ or Kafka) I wanted to use in my pipelines. After all the previous pods deployed Deploy the ML Lab server (that I change the UI on the building of the Docker image to include the custom UI as a maven dependency. To make requests between ML lab and other components like Mycroft Server, Zeppelin server and Keycloak I had

```
apiVersion: v1
kind: Service
metadata:
  name: keycloak
```

```
    labels:
      app: keycloak
spec:
  ports:
  - name: http
    port: 8083
    targetPort: 8080
  selector:
    app: keycloak
  type: ClusterIP
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: keycloak
  namespace: default
  labels:
    app: keycloak
spec:
  replicas: 1
  selector:
    matchLabels:
      app: keycloak
  template:
    metadata:
      labels :
        app: keycloak
    spec:
      containers :
      - name: keycloak
        image: ginadock/mllab−keycloak:1.0.0
resources:
  limits :
    cpu: '1000m'
    memory: '1.5Gi'
    ephemeral−storage: '1Gi'
  requests:
    cpu: '1000m'
    memory: '1.5Gi'
    ephemeral−storage: '1Gi'
env:
- name: KEYCLOAK_USER
  valueFrom:
    secretKeyRef:
```

```
      name: keycloak
      key: keycloak−user
- name: KEYCLOAK_PASSWORD
  valueFrom:
    secretKeyRef:
      name: keycloak
      key: keycloak−password
- name: KEYCLOAK_LOGLEVEL
  value: ”INFO”
- name: ROOT_LOGLEVEL
  value: ”INFO”
- name: JDBC_PARAMS
  value: ’useSSL=false’
- name: PROXY_ADDRESS_FORWARDING
  value: ”true”
- name: DB_VENDOR
  value: ’mysql’
- name: DB_ADDR
  value: ’mysql’
- name: DB_PORT
  value: ’3306’
- name: DB_USER
```

```yaml
          valueFrom:
            secretKeyRef:
              name: mysql
              key: mysql-root-username
        - name: DB_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql
              key: mysql-root-password
        ports:
        - name: http
          containerPort: 8080
        - name: https
          containerPort: 8443
        readinessProbe:
          initialDelaySeconds : 120
          failureThreshold : 3
          periodSeconds: 15
          httpGet:
            path: /auth/realms/mllab
            port: 8080
---
```

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: keycloak
type: Opaque
stringData:
  keycloak-user: <USER_NAME>
  keycloak-password: <USER_PASSWORD>
```

Code 3.2: Keycloak deployment file

CHAPTER 4

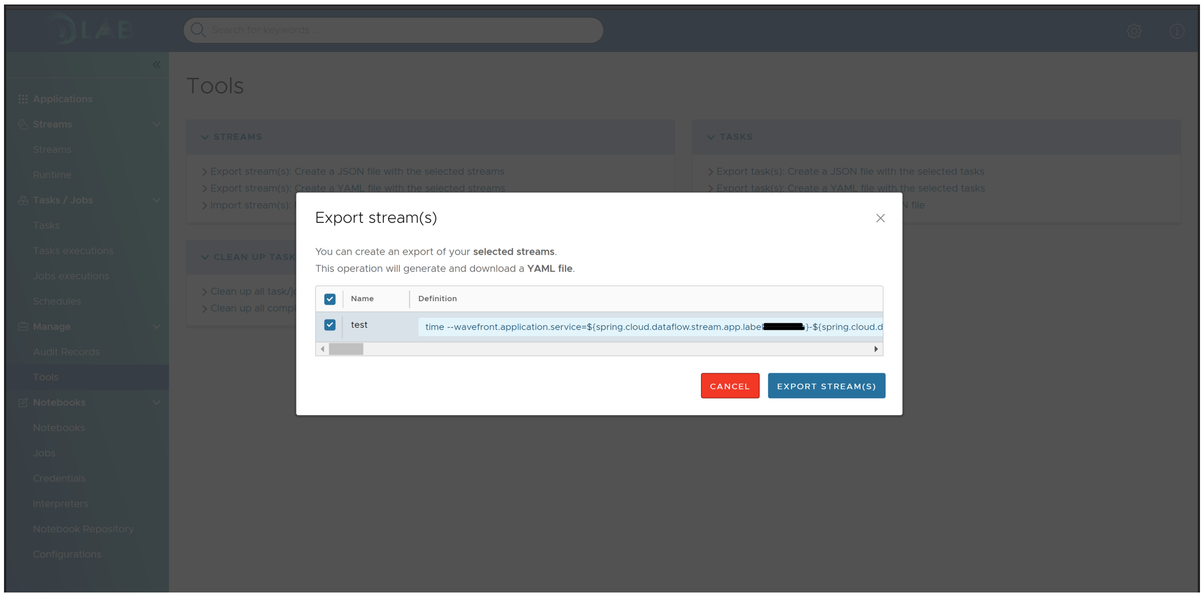
Implementation

4.1 ML LAb - Spring Cloud Dataflow extension

Many source, processor, and sink applications for common use-cases (e.g. s3, jdbc, hdfs, http, router) are already developed and provided as publicly consumable pre-built applications by the Spring Cloud Data Flow. A user of the ML Lab can directly use or extend any out-of-the-box utility applications to cover common use cases or write a custom applications. The aim is to simplify the writing of message-driven microservice applications connected to a common messaging system. This enables the developers to develop model with the use of specific messaging middleware.

4.1.1 Export YAML file

My integration on the Spring Cloud Dataflow base code was to provide two extra functionalities. One was to include the export as YAML file that that takes the streams/tasks pipeline definition. This functionality enable users to deploy a task or a stream and get the deployed manifest of the pipeline they created that can be deployed. This file can be later used to share a created pipeline. As for the manifest to be deployed the pipeline must be deployed successfully, If the stream/-task has not been deployed before, then first an **deploy** action is triggered before the **exportYaml** action. This steps makes sure that the manifest has been generated successfully and can be exported without errors. This feature is located on the streams/tasks list pages and redirects users to the Export-YAML configuration page to configure the pipeline arguments and commands before exports action takes place. To enable this feature I have to create 2 new REST endpoints and configure the Spring Cloud Dataflow Server to provide me with the full details of the deployed pipeline, at the bottom of the deploy(streams)/launch(tasks) configuration page and at the tools menu item. as shown in Figures 4.1 and 4.2.



Tools

STREAMS

> Export stream(s): Create a JSON file with the selected streams
> Export stream(s): Create a YAML file with the selected streams
> Import stream(s):

TASKS

> Export task(s): Create a JSON file with the selected tasks
> Export task(s): Create a YAML file with the selected tasks
N file

CLEAN UP TASK

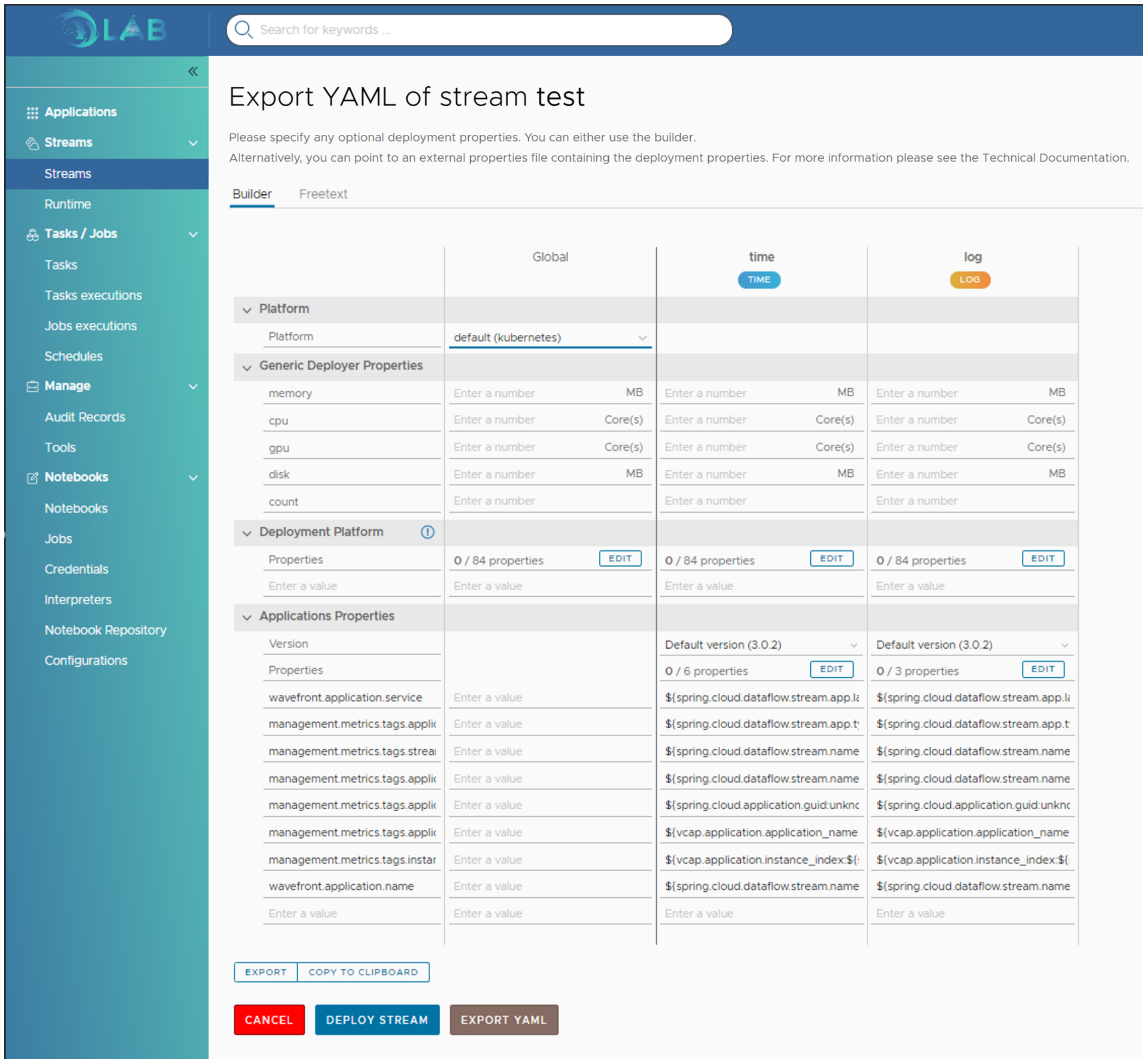
> Clean up all task/j
> Clean up all comp

Export stream(s)

You can create an export of your **selected streams**.
This operation will generate and download a **YAML file**.

✓	Name	Definition
✓	test	time --wavefront.application.service=${spring.cloud.dataflow.stream.app.label[redacted]}-${spring.cloud.d

Figure 4.1: Export YAML under Tools Menu Item



Export YAML of stream test

Please specify any optional deployment properties. You can either use the builder.
Alternatively, you can point to an external properties file containing the deployment properties. For more information please see the Technical Documentation.

Builder | Freetext

	Global	time TIME	log LOG
Platform			
Platform	default (kubernetes)		
Generic Deployer Properties			
memory	Enter a number MB	Enter a number MB	Enter a number MB
cpu	Enter a number Core(s)	Enter a number Core(s)	Enter a number Core(s)
gpu	Enter a number Core(s)	Enter a number Core(s)	Enter a number Core(s)
disk	Enter a number MB	Enter a number MB	Enter a number MB
count	Enter a number	Enter a number	Enter a number
Deployment Platform			
Properties	0 / 84 properties EDIT	0 / 84 properties EDIT	0 / 84 properties EDIT
Enter a value	Enter a value	Enter a value	Enter a value
Applications Properties			
Version		Default version (3.0.2)	Default version (3.0.2)
Properties		0 / 6 properties EDIT	0 / 3 properties EDIT
wavefront.application.service	Enter a value	${spring.cloud.dataflow.stream.app.la	${spring.cloud.dataflow.stream.app.la
management.metrics.tags.applic	Enter a value	${spring.cloud.dataflow.stream.app.ty	${spring.cloud.dataflow.stream.app.t
management.metrics.tags.strea	Enter a value	${spring.cloud.dataflow.stream.name	${spring.cloud.dataflow.stream.name
management.metrics.tags.applic	Enter a value	${spring.cloud.dataflow.stream.name	${spring.cloud.dataflow.stream.name
management.metrics.tags.applic	Enter a value	${spring.cloud.application.guid:unknc	${spring.cloud.application.guid:unknc
management.metrics.tags.applic	Enter a value	${vcap.application.application_name	${vcap.application.application_name
management.metrics.tags.instar	Enter a value	${vcap.application.instance_index:${	${vcap.application.instance_index:${
wavefront.application.name	Enter a value	${spring.cloud.dataflow.stream.name	${spring.cloud.dataflow.stream.name
Enter a value	Enter a value	Enter a value	Enter a value

Figure 4.2: Export YAML of Stream Configuration Page

an example output of a batch exported YAML is:

```
apiVersion: skipper . spring . io /v1
kind: SpringCloudDeployerApplication
metadata:
  name: log
```

```
spec:
  resource: maven://org.springframework.cloud.stream.app:log-sink-kafka:jar
  resourceMetadata: maven://org.springframework.cloud.stream.app:log−sink−kafka:jar
      :jar :metadata:2.1.3.RELEASE
  version: 2.1.3. RELEASE
  applicationProperties :
    spring.metrics.export. triggers . application . includes: integration **
    spring.cloud.dataflow.stream.app.label: log
    spring.cloud.stream.metrics.key: test−gina−12121.log.${spring.cloud. application
        .guid}
    spring.cloud.stream.bindings.input.group: test−gina−12121
    spring.cloud.stream.kafka.streams.binder.zkNodes: zookeeper:2181
    spring.cloud.stream.metrics. properties: spring. application .name,spring.
        application .index, spring.cloud. application .*,spring.cloud.dataflow.*
    spring.cloud.dataflow.stream.name: test−gina−12121
    spring.cloud.stream.kafka.binder.zkNodes: zookeeper:2181
    spring.cloud.dataflow.stream.app.type: sink
    spring.cloud.stream.bindings.input. destination: test−gina−12121.time
    spring.cloud.stream.kafka.streams.binder.brokers: PLAINTEXT://kafka−broker:9092
    spring.cloud.stream.kafka.binder.brokers: PLAINTEXT://kafka−broker:9092
  deploymentProperties:
    spring.cloud.deployer.group: test−gina−12121
```

```
---
apiVersion: skipper . spring . io /v1
kind: SpringCloudDeployerApplication
metadata:
  name: time
spec:
  resource: maven://org.springframework.cloud.stream.app:time−source−kafka:jar
  resourceMetadata: maven://org.springframework.cloud.stream.app:time−source−kafka
      :jar: jar :metadata:2.1.2.RELEASE
  version: 2.1.2. RELEASE
  applicationProperties :
    spring . metrics . export . triggers . application . includes: integration **
    spring . cloud . dataflow.stream.app. label: time
    spring . cloud . stream.metrics . key: test −gina−12121.time.${ spring . cloud .
        application . guid }
    spring . cloud . stream. kafka . streams. binder . zkNodes: zookeeper:2181
    spring . cloud . stream.metrics . properties: spring . application . name,spring.
        application . index , spring . cloud . application . *, spring . cloud . dataflow.*
    spring . cloud . stream. bindings . output.producer.requiredGroups: test −gina−12121
    spring . cloud . dataflow.stream.name: test−gina−12121
    spring . cloud . stream. bindings . output. destination : test −gina−12121.time
    spring . cloud . stream. kafka . binder . zkNodes: zookeeper:2181
```

```
      spring.cloud.dataflow.stream.app.type: source
      spring.cloud.stream.kafka.streams.binder.brokers: PLAINTEXT://kafka-broker:9092
      spring.cloud.stream.kafka.binder.brokers: PLAINTEXT://kafka-broker:9092
    deploymentProperties:
      spring.cloud.deployer.group: test-gina-12121
```

Code 4.1: exported YAML Example

4.1.2 GPU property configuration

I also enabled user to select to deploy the pipeline with GPU as shown in Figure 4.1. This will assist users that want in the future to run complex deep learning pipelines. This enables the Configuration of a pipeline at deployment/launch to take the GPU usage that user wants to have. The inspiration of this feature was that any data scientist or machine learning enthusiast who has been trying to elicit performance of training models at scale will at some point hit a cap and start to experience various degrees of processing lag. Deep Learning requires a lot of hardware. Tasks that take minutes with smaller training sets may now take more hours — in some cases weeks — when datasets get larger. GPUs are optimized for training artificial intelligence and deep learning models as they can process multiple computations simultaneously. They have a large number of cores, which allows for better computation of multiple parallel processes. Additionally, computations in deep learning need to handle huge amounts of data — this makes a GPU's memory bandwidth most suitable.

4.1.3 Securing ML Lab with Keyclaok

User authorization occurs in the ML Lab server, which securely brokers requests to other services. When the gateway tenant calls an API server method, it passes the user’s access token. The server first makes a call to Keycloak to verify the access token is valid. If the access token is valid a second call is made to Keycloak to get a list of roles that are assigned to the user. The server has a list of methods that are accessible to each role. The request is authorized if the user has a role that can access the given method.

When a user is doing a standard username-password login, this is accomplished by using a Resource Owner Password Credentials grant flow by which the web portal directly submits the username and password to Keycloak and gets a code that can be exchanged for an access token. When a user is logging in, the web portal redirects to Keycloak with a special query parameter indicating to Keycloak which identity provider to redirect to for the userś authentication.

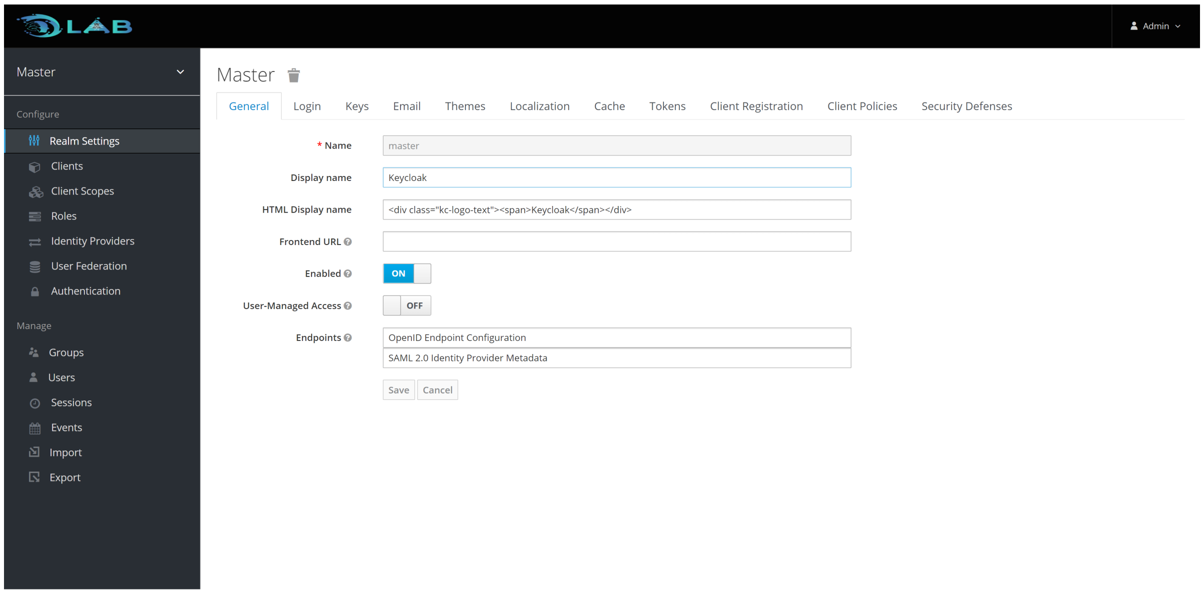
4.1.3.1 Keycloak Configuration for ML Lab

I began by creating a simple Dockerfile for the Keycloak server [29] as shown here:

```
FROM quay.io/keycloak/keycloak:15.0.2
ENV KEYCLOAK_USER <username>
ENV KEYCLOAK_PASSWORD <password>
EXPOSE 8080
```

Code 4.2: Keycloak base Dockerfile

After the image is build and run, the keycloak home page is shown, see Figure 4.3 where the Admin page can be found. When booting Keycloak for the first time a pre-defined realm will be created called **master** realm and is the king of all realms. Admins in this realm have permissions to view and manage any other realm created on the server instance. It is recommended that we do not use the master realm to manage the users and applications, so I kept the master realm as a place for super admins to create and manage the realms in the system. This will keep things clean and organized. It is also possible to disable the master realm and define admin accounts at each individual new realm we create.



* Name	master
Display name	Keycloak
HTML Display name	<div class="kc-logo-text"><span>Keycloak</span></div>
Frontend URL	
Enabled	ON
User-Managed Access	OFF
Endpoints	OpenID Endpoint Configuration
	SAML 2.0 Identity Provider Metadata

Figure 4.3: Keycloak Admin Page

The creation of a different realm 4.4 for security reasons as stated before is required. This realm is a key aspect for connecting the Keycloak authentication mechanism with the applications. As stated before, A realm manages a set of users, credentials, roles, and groups. A user belongs to and logs into a specific realm.

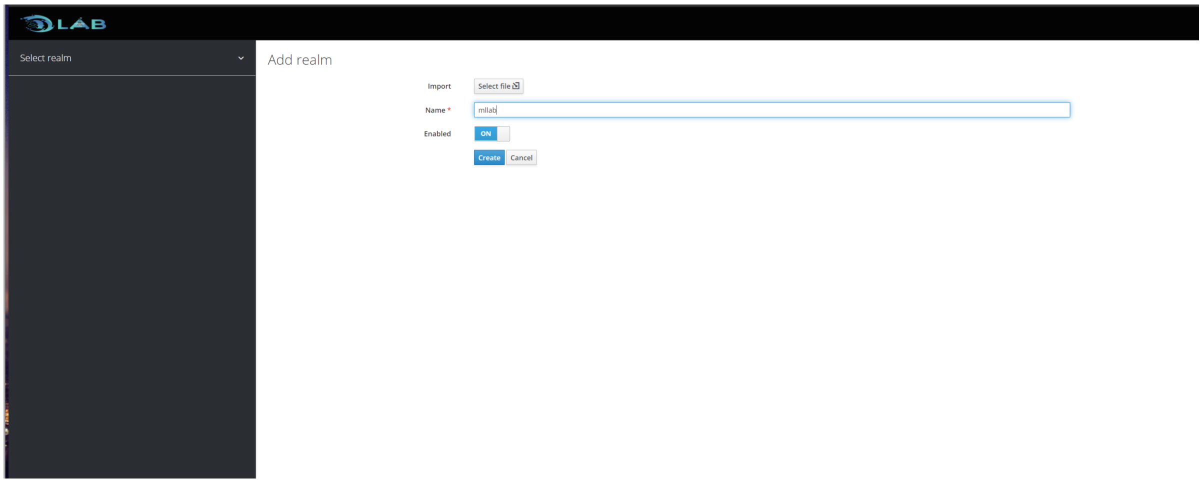


Figure 4.4: Create Realm mllab

Realms are isolated from one another and can only manage and authenticate the users that they control. Creating the new **mllab** realm is very simple. Mouse over the top left corner drop down menu that is titled with Master and select from this drop down menu the created realm. The last entry of this drop down menu is always **Add Realm**. Figure 4.5

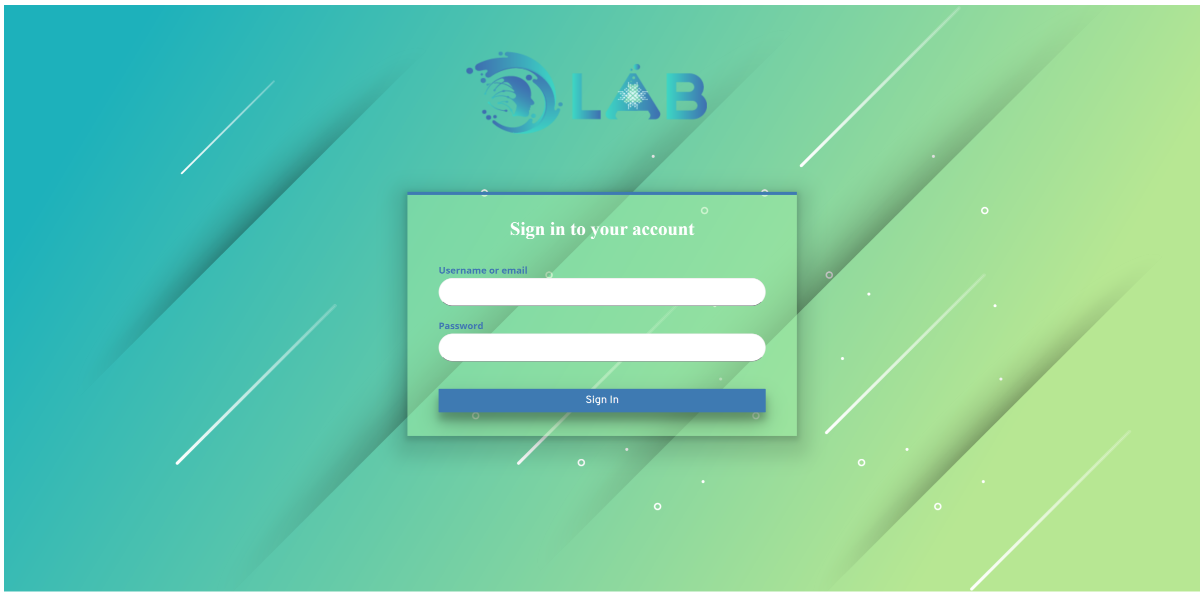
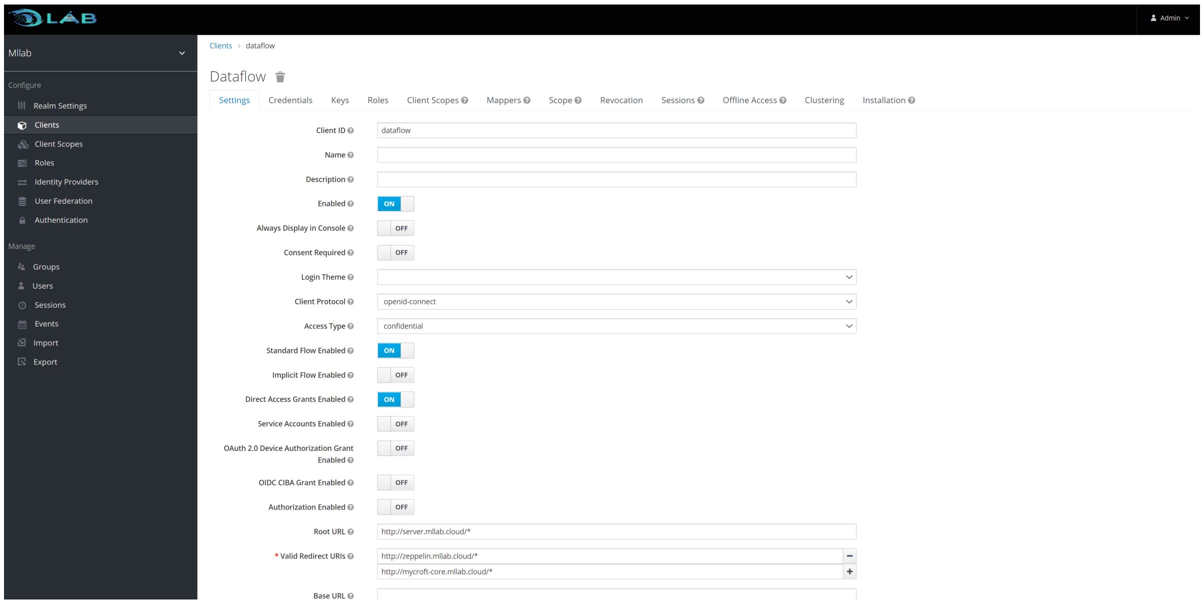


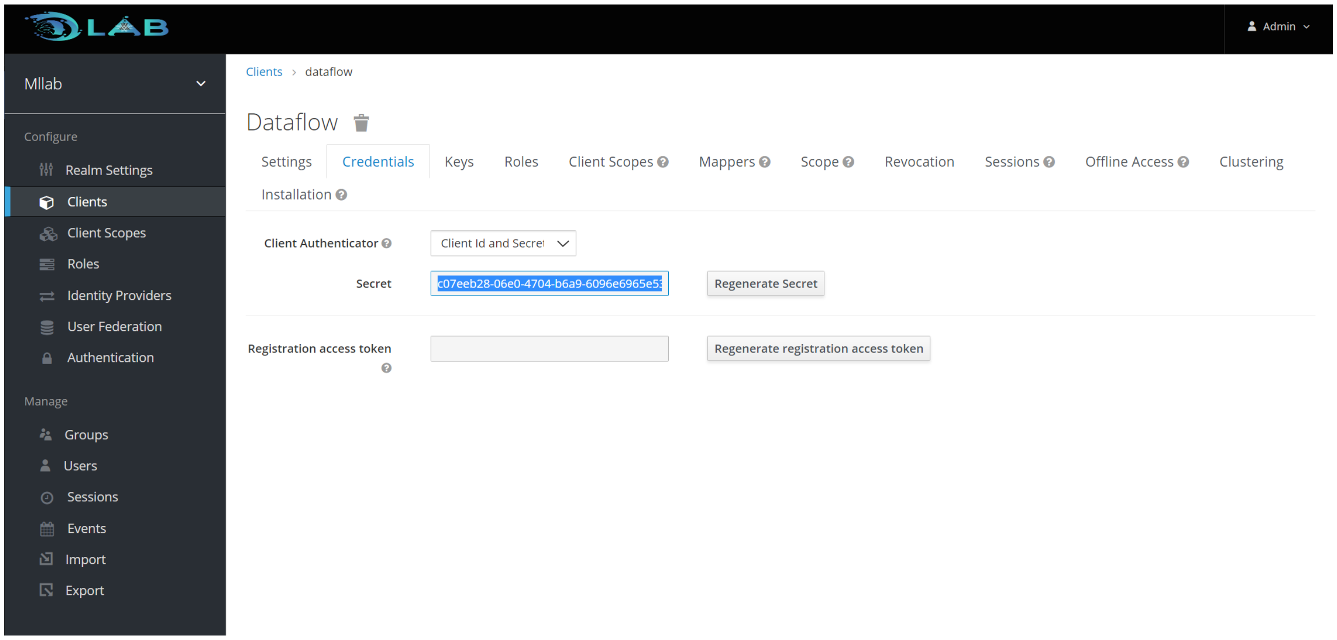
Figure 4.5: Keycloak master realm (default) Login Page

(Step 1) Create client dataflow and Configure to Use Access Type confidential and enable Service Accounts. I have to provide the Valid Redirect URIs of ML Lab, Apache Zepelin and Mycroft-core. And then generate a secret as shown in Figures 4.6 and 4.7.



Client ID	dataflow
Name	
Description	
Enabled	ON
Always Display in Console	OFF
Consent Required	OFF
Login Theme	
Client Protocol	openid-connect
Access Type	confidential
Standard Flow Enabled	ON
Implicit Flow Enabled	OFF
Direct Access Grants Enabled	ON
Service Accounts Enabled	OFF
OAuth 2.0 Device Authorization Grant Enabled	OFF
OIDC CIBA Grant Enabled	OFF
Authorization Enabled	OFF
Root URL	http://server.mllab.cloud/*
* Valid Redirect URIs	http://zeppelin.mllab.cloud/*
	http://mycroft-core.mllab.cloud/*
Base URL	

Figure 4.6: Configure Valid Redirect URLs of Keyclock



Client Authenticator	Client Id and Secret
Secret	c07eeb28-06e0-4704-b6a9-6096e6965e53
Registration access token	

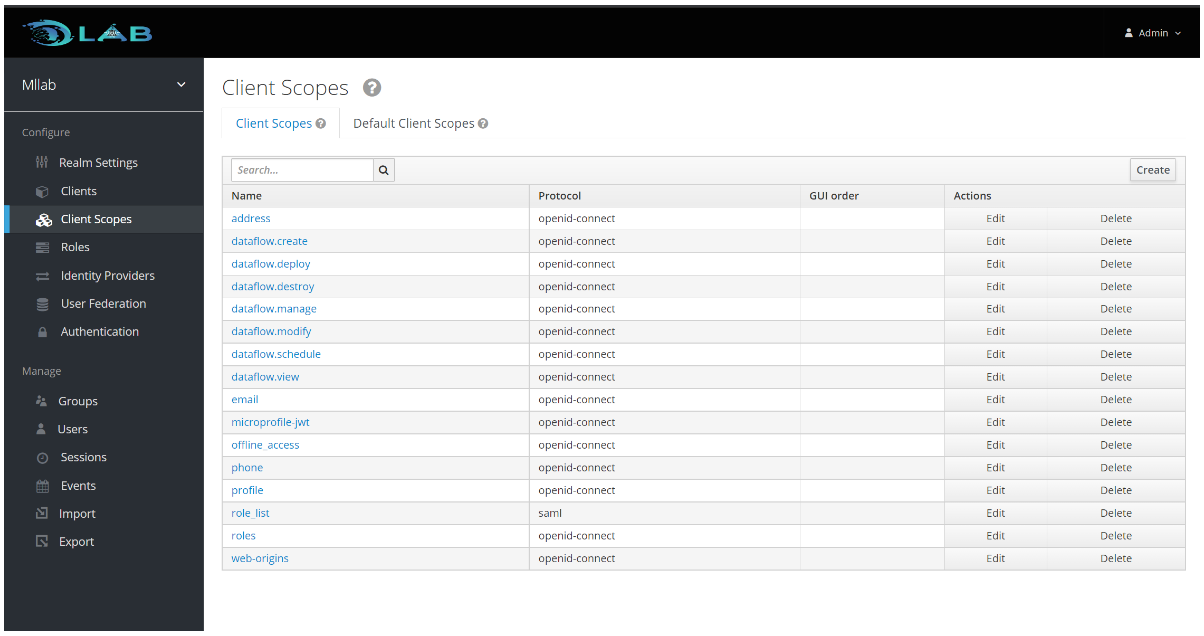
Figure 4.7: Generate Secret for dataflow Client

The rest of the steps I have followed to enable the Keycloak functionality was:

(Step 2) Create Role **ADMIN** which is basically a namespace dedicated to the **dataflow** client. Each client gets its own namespace. Client roles are managed under the *Roles tab* under each individual client.

(Step 3) Create Protocol Mapper to map expected *user_name* create new mapper with Name Username, Mapper Type User Property, Property username, Token Claim NAME *user_name* and Claim JSON Type String. Create user admin, and set its password and disable requirement to change it by setting Temporary to OFF.

(Step 4) Spring Cloud Dataflow comes with some roles (auth guard) for creating, viewing, deploy, destroy, mange, modification and schedule of pipelines. To integrate the keycloak mechanism I had to create the scopes for each auth guard, dataflow.view, dataflow.create, dataflow.manage, dataflow.deploy, dataflow.destroy, dataflow.modify and dataflow.schedule. 4.8



Name	Protocol	GUI order	Actions	
address	openid-connect		Edit	Delete
dataflow.create	openid-connect		Edit	Delete
dataflow.deploy	openid-connect		Edit	Delete
dataflow.destroy	openid-connect		Edit	Delete
dataflow.manage	openid-connect		Edit	Delete
dataflow.modify	openid-connect		Edit	Delete
dataflow.schedule	openid-connect		Edit	Delete
dataflow.view	openid-connect		Edit	Delete
email	openid-connect		Edit	Delete
microprofile-jwt	openid-connect		Edit	Delete
offline_access	openid-connect		Edit	Delete
phone	openid-connect		Edit	Delete
profile	openid-connect		Edit	Delete
role_list	saml		Edit	Delete
roles	openid-connect		Edit	Delete
web-origins	openid-connect		Edit	Delete

Figure 4.8: Create scopes for dataflow Client

(Step 5) User role mappings that can be assigned individually to each user through the Role Mappings tab for that single user. I map Role to User admin. and then assign all scopes as Optional Client Scopes. 4.8 After configuring all this steps about keycloak realm, we move on with the configuration of ML Lab and Skipper to support keycloak authentication mechanism. The ML Lab and Skipper Server executable jars use OAuth 2.0 authentication to secure the relevant REST endpoints. They will be accessible the Keycloak access tokens to provide user authentication to user by modifying the configuration of dataflow-keycloak.yml and skipper-keycloak.yml files to provide the url, realm, client_id, scopes and client_secret. Inside the Dockerfiles of the MLLab and Skipper we have to configure the command that runs them and provide the additional locations for the skipper and the mllab keycloak deployment files.

```
java -jar spring-cloud-skipper-server -2.6.2. BUILD-SNAPSHOT.jar --spring.config.additional-
    location=skipper-keycloak.yml
```

Code 4.3: Keycloak Skipper command (Dockerfile)

Dataflow:

```
java -jar mllab-2.9.0-SNAPSHOT.jar --spring.config.additional-location=dataflow-keycloak.yml
```

Code 4.4: Keycloak Skipper command (Dockerfile)

```
spring:
  cloud:
    dataflow:
      security:
```

```
      authorization :
        provider−role−mappings:
          keycloak:
            map−oauth−scopes: true
            role−mappings:
              ROLE_VIEW: dataflow.view
              ROLE_CREATE: dataflow.create
              ROLE_MANAGE: dataflow.manage
              ROLE_DEPLOY: dataflow.deploy
              ROLE_DESTROY: dataflow.destroy
              ROLE_MODIFY: dataflow.mocdify
              ROLE_SCHEDULE: dataflow.schedule
  security :
    oauth2:
      client :
        registration :
          keycloak:
            redirect −uri: '{baseUrl}/login/oauth2/code/{registrationId}'
            authorization −grant−type: authorization_code
            client −id: dataflow
            client −secret: <keycloak_secret>
            scope:
```

```yaml
              - openid
              - dataflow.view
              - dataflow.deploy
              - dataflow.destroy
              - dataflow.manage
              - dataflow.modify
              - dataflow.schedule
              - dataflow.create
        provider:
          keycloak:
            jwk-set-uri: https://keycloak-mllab.cloud/auth/realms/dataflow/protocol
                /openid-connect/certs
            token-uri: https://keycloak-mllab.cloud/auth/realms/dataflow/protocol/
                openid-connect/token
            user-info-uri: https://keycloak-mllab.cloud/auth/realms/dataflow/
                protocol/openid-connect/userinfo
            user-name-attribute: user_name
            authorization-uri: https://keycloak-mllab.cloud/auth/realms/dataflow/
                protocol/openid-connect/auth
      resourceserver:
        opaquetoken:
          introspection-uri: https://keycloak-mllab.cloud/auth/realms/dataflow/
```

```
              protocol/openid−connect/token/introspect
          client −id: dataflow
          client −secret: <keycloak_secret>
      authorization:
        check−token−access: isAuthenticated ()
```

Code 4.5: Keycloak Skipper Dockerfile

When running Keycloak in localhost, it does not enforce the s protocol. But when we deployed to Kubernetes (remote server), s is required for running Keycloak on remote servers as shown in Figure **??**. The problem is shown when domain name is not yet registered. To resolve it, we create an self-signed certificate. This allowed us to perform request to Keycloak from the ML Lab and the Mycroft Bot.

4.1.3.2 Keycloak Custom Theme

As mentioned before Keycloak provides theme support. This allows customizing the look and feel of end-user facing pages so they can be integrated with the ML Lab applications.[30] I wanted to provide the Login Page customized for ML Lab. There where two approached that could be used to achieve this. The first and the one we preferred was to customize the user interfaces of the Keycloak [31, 32, 33]. The other approach was to use Keycloak as a backend service so that the user is never exposed to the Keycloak user interfaces. The reason for not to expose the user to the Keycloak User Interface (UI) is simply to avoid needing to build user trust in this additional authentication service and thus avoid user confusion.

A theme can provide one or more types to customize different aspects of Keycloak. The types available are:

- ***Account*** - Account management
- ***Admin*** - Admin console
- ***Email*** - Emails
- ***Login*** - Login forms
- ***Welcome*** - Welcome page

Keycloak comes bundled with default themes in the server's root themes directory. To avoid editing the bundled themes directly, I create a new custom theme that extends one of the bundle ones as I was not planning to replace every single page. As I wanted to change significantly the look and feel of the pages, I preferred to extend the base theme by overriding the individual resources.

All theme types, except the **welcome** one, can be configured through the Admin Console. To change the theme used for a realm the user must open the Admin Console, select the realm from the drop-down box in the top left corner and Under Realm Settings click Themes and then define the the theme for each page.

Every User Interface (UI) screen is internationalized in Keycloak. The default language is English, but by turning on the Internationalization switch on the Theme tab, I was prompted to select the locales I wanted to support and what the default locale will be. By enabling this option, users are able to choose the language on the login page for the selected language to be presented see in

Figures 4.9 and 4.10, User Account Management UI, and Admin Console.

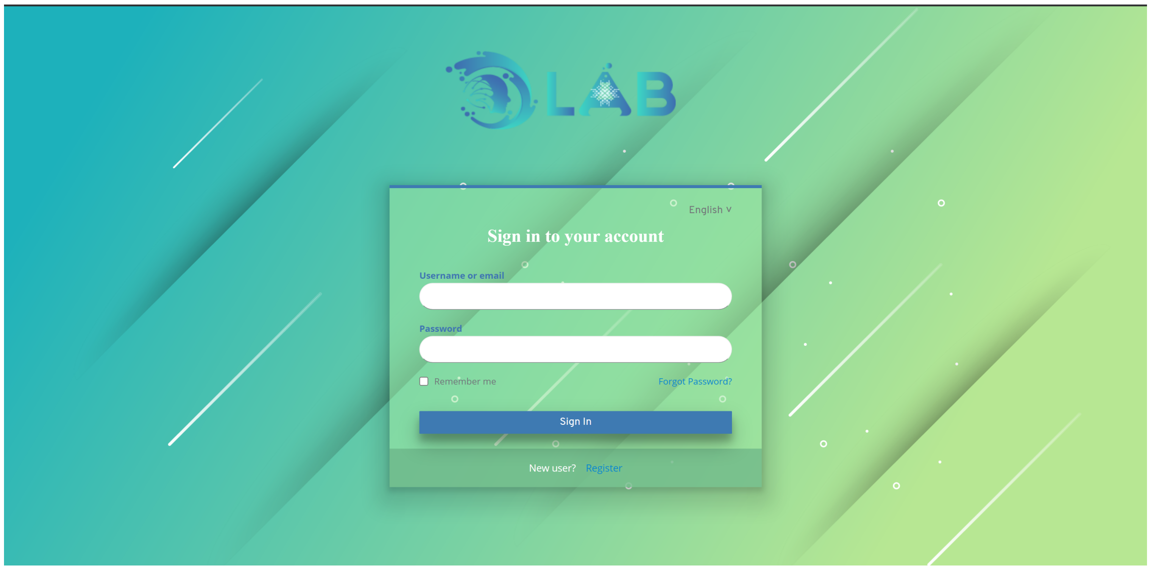


Figure 4.9: Keycloak mllab realm custom Login Page

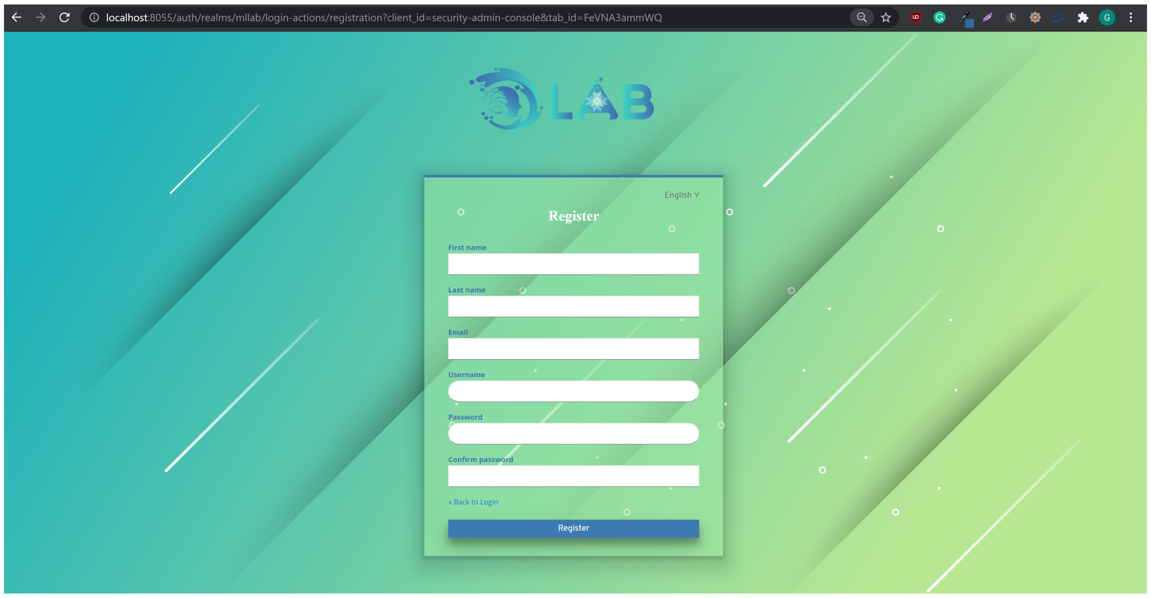
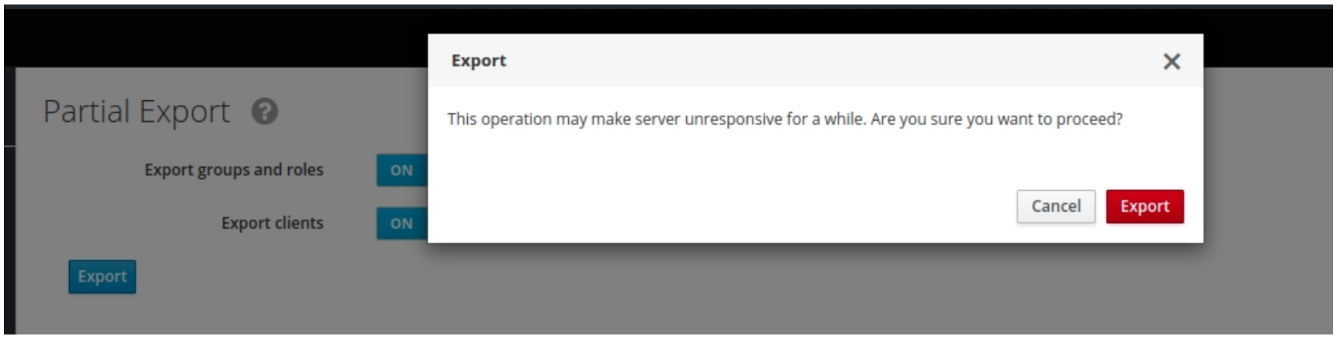

localhost:8055/auth/realms/mllab/login-actions/registration?client_id=security-admin-console&tab_id=FeVNA3ammWQ
LAB
English
Register
First name
Last name
Email
Username
Password
Confirm password
« Back to Login
Register

Figure 4.10: Keycloak mllab realm custom Registration Page

After configuring all the changes for our custom theme and the realm settings required for the ML-Lab Keycloak authorization integration, the exported realm as a JSON file will be used in the final Dockerfile to provide the changes on the run of the image, see Figure 4.11. As a result, the **realm-export.json** file will be saved on our machine. We have to keep in mind that all the users information are not provided in this exported JSON file and that must be inserted by hand on top of the file inside the global object as stated in [34].



This operation may make server unresponsive for a while. Are you sure you want to proceed?

Figure 4.11: Export Keycloak Realm

```
FROM quay.io/keycloak/keycloak:15.0.2
COPY ./mllab/ /opt/jboss/keycloak/themes/mllab

ENV KEYCLOAK_DEFAULT_THEME mllab
ENV KEYCLOAK_WELCOME_THEME mllab
ENV KEYCLOAK_LOGIN_THEME mllab
ENV KEYCLOAK_ADMIN_THEME mllab
ENV KEYCLOAK_USER "user"
ENV KEYCLOAK_PASSWORD "password"

...

COPY mllab-realm.json ./mllab-realm.json
```

```
ENV KEYCLOAK_IMPORT ./mllab-realm.json
EXPOSE 8080
```

Code 4.6: Final Keycloak Dockerfile

The **KEYCLOAK_IMPORT** environment variable to specify the realm file mounted to the local directory. To view that our realm holds all the information we configured, we can check with the following the command :

```
docker run -e KEYCLOAK_USER=<USERNAME> -e KEYCLOAK_PASSWORD=<
    PASSWORD> -e KEYCLOAK_IMPORT=./example-realm.json
```

4.1.4 Streaming Object Detection Pipeline

Machine Learning (ML) and Deep Learning (DL) have brought unprecedented abilities to the software engineering field. ML/DL allows us to reason about and to solve otherwise **un-programmable** tasks, such as computer-vision and language-processing. But how can we leverage ML/DL to deliver richer business solutions to the end-user? My goal here is to show how Spring Cloud Stream and Spring Cloud Dataflow can make this much easier.

The ML/DL paradigm works by making observations, running experiments, and using statistics to analyze the results from the experiments. Usually the process is divided in two phases:

- model training on historical datasets
- using the pre-trained models for predictive analytics at run-time (called ML inference).

While most ML tools will help through the tasks of data exploration, preparation, and model training you are left on your own to figure out how to integrate and use those tools. Model inference for predictive analytics is the most common use of ML/DL in Java applications, where you take a pre-trained (out-of-the-box) model and use it in your applications to do real-time predictions (e.g. fraud detection or vehicle predictive maintenance). The ML inference requires a portable format for exchanging models between the training and the inference phases. Several initiatives such as PMML, PFA, MLeap and ONNX aim to a portable format.

Here we will focus on TensorFlow, an emerging DL framework that has gained a lot of momentum. This success is partly due to the model serialization capabilities it provides. TensorFlow's pre-trained models can be serialized and reused across multiple programming languages, CPU & GPU processors and platform architectures. Spring Cloud Stream and Spring Cloud Data Flow significantly simplify the tasks of deploying and operationalizing pre-trained Tensorflow models, to add ML/DL capabilities for your own business solutions.

An out-of-the-box TensorFlow Processor is available to perform real-time predictive analytics using pre-trained TensorFlow models. The new Object Detection processor provides out-of-the-box support for the Tensorflow Object Detection API. It allows for real-time, localization and identification of multiple objects in a single image or image stream. The object-detection processor uses one of the pre-trained object detection models and corresponding object labels. The object-detection processor has the following options:

- **tensorflow.expression:** How to obtain the input data from the input message. If empty it defaults to the input message payload. The headers[myHeaderName] expression to get input data from message's header using myHeaderName as a key. (Expression, default:

<none>)

- **tensorflow.mode:** The outbound message can store the inference result either in the payload or in a header with name outputName. The payload mode (default) stores the inference result in the outbound message payload. The inbound payload is discarded. The header mode stores the inference result in outbound message's header defined by the outputName property. The the inbound message payload is passed through to the outbound such. (OutputMode, default: <none>, possible values: payload,header)

- **tensorflow.model:** The location of the pre-trained TensorFlow model file. The file, http and classpath schemas are supported. For archive locations takes the first file with **.pb** extension. Use the URI fragment parameter to specify an exact model name
(e.g. https://foo/bar/model.tar.gz#frozen_inference_graph.pb) (Resource, default: <none>)

- **tensorflow.model-fetch:** The TensorFlow graph model outputs. Comma separate list of TensorFlow operation names to fetch the output Tensors from. (List<String>, default: <none>)

- **tensorflow.object.detection.color-agnostic:** If disabled (default) the bounding box colors are selected as a function of the object class id. If enabled all bounding boxes are visualized with a single color. (Boolean, default: false)

- **tensorflow.object.detection.confidence:** Probability threshold. Only objects detected with probability higher then the confidence threshold are accepted. Value is between 0 and 1. (Float, default: 0.4)

- **tensorflow.object.detection.draw-bounding-box:** When set to true, the output image will be annotated with the detected object boxes (Boolean, default: true)

- **tensorflow.object.detection.draw-mask:** For models with mask support enable drawing the mask of the detected objects (Boolean, default: true)
- **tensorflow.object.detection.labels:** The text file containing the category names (e.g. labels) of all categories that this model is trained to recognize. Every category is on a separate line. (Resource, default: <none>)
- **tensorflow.output-name:** The output data key used for the Header modes. (String, default: result)

If the pre-trained model is not set explicitly, then the following defaults are used:

- **tensorflow.modelFetch :** detection_scores, detection_classes, detection_boxes, num_detections
- **tensorflow.model:**
  dl.bintray.com/big-data/generic/faster_rcnn_resnet101_coco_2018_01_28_frozen_inference_graph.pb
- **tensorflow.object.detection.labels:**
  dl.bintray.com/big-data/generic/mscoco_label_map.pbtxt

Processor's input is an image byte array and the output is a JSON message in this format:

```
{
  "labels" : [
    {"name":"person", "confidence":0.9996774,"x1":0.0,"y1":0.3940161,"x2":0.9
        465165,"y2":0.5592592,"cid":1},
    {"name":"person", "confidence":0.9996604,"x1":0.047891676,"y1":0.03169123
        ,"x2":0.941098,"y2":0.2085562,"cid":1},
    {"name":"backpack", "confidence":0.96534747,"x1":0.15588468,"y1":0.859577
        95,"x2":0.5091308,"y2":0.9908878,"cid":23},
    {"name":"backpack", "confidence":0.963343,"x1":0.1273736,"y1":0.57658505,
        "x2":0.47765,"y2":0.6986431,"cid":23}
  ]
}
```

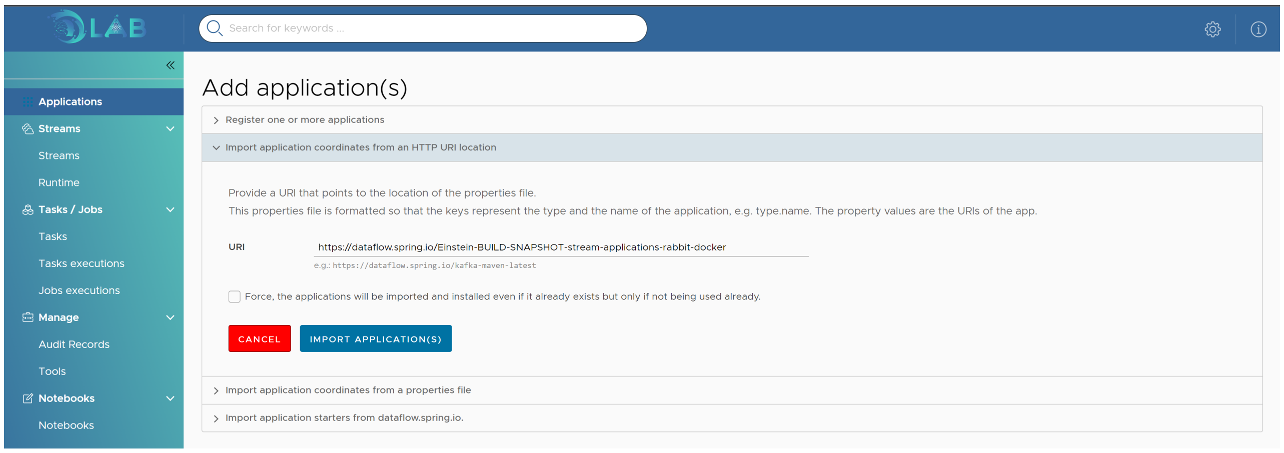
Code 4.7: Object Detection Output JSON Example

The output format is:

- **object-name:confidence** - human readable name of the detected object (e.g. label) with its confidence as a float between [0-1]
- **x1, y1, x2, y2** - Response also provides the bounding box of the detected objects represented as (x1, y1, x2, y2). The coordinates are relative to the size of the image size.
- **cid** - Classification identifier as defined in the provided labels configuration file.

Tensor Flow package, has a specific processor called object detection processor, where it can detect various objects available in an image. [35] So what I can do, I can provide any image as a input and it can detect what are all the objects that are available and the probable type of object what it is, it can predict and it can give the output. I can route the output into a log file and it can log the output as a JSON output or even can generate the output as a image file and preview the result, Or even store it to a database. I used skipper server and ML Lab's server for this particular demo. I register the new *apps* by bulk import application and the URI as shown in Figure 4.12. All the applications are imported. Within this I do have a processor of type object detection, which is the processor that I am going to use to detect what are all the objects are available within that specific image see Figure 4.13.

The following diagram illustrates a Spring Cloud Data Flow streaming pipeline that predicts object types from the images in real-time.



Add application(s)

> Register one or more applications

Import application coordinates from an HTTP URI location

Provide a URI that points to the location of the properties file.
This properties file is formatted so that the keys represent the type and the name of the application, e.g. type.name. The property values are the URIs of the app.

URI https://dataflow.spring.io/Einstein-BUILD-SNAPSHOT-stream-applications-rabbit-docker
e.g.: https://dataflow.spring.io/kafka-maven-latest

☐ Force, the applications will be imported and installed even if it already exists but only if not being used already.

CANCEL IMPORT APPLICATION(S)

> Import application coordinates from a properties file

> Import application starters from dataflow.spring.io.

Figure 4.12: Register Application from URI

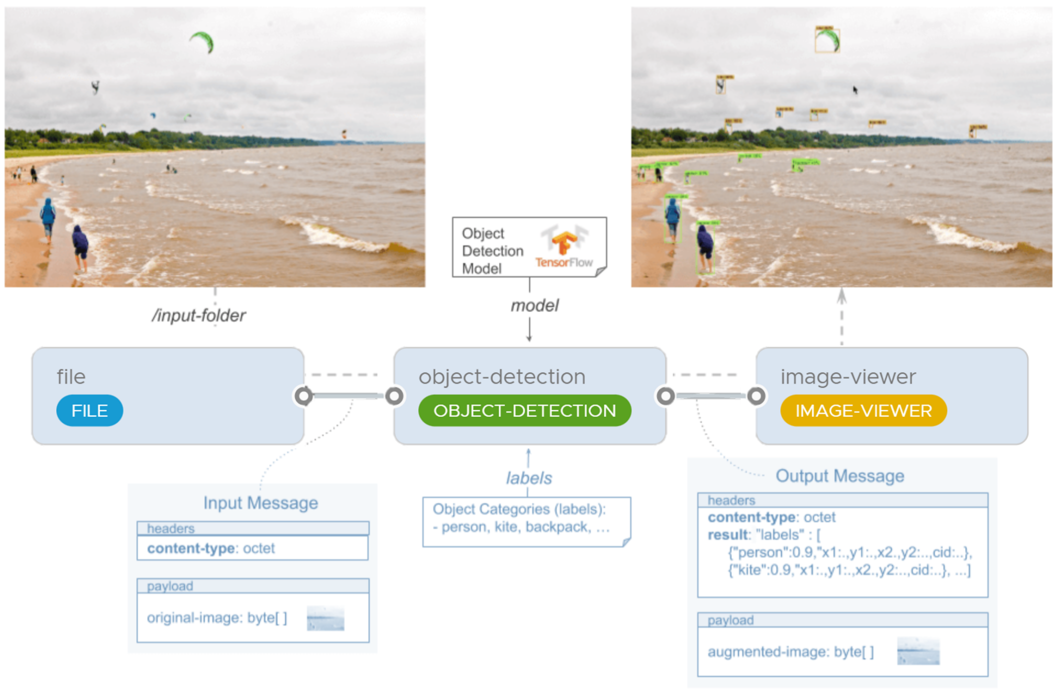


Figure 4.13: Object Detection Pipeline

Start with a file source node, which is going to use this specific directory as the source and that's going to pipe the output to object detection processor, which will use a tensor flow model, which is already built and it's available in this specific URL and the information that it is going to capture are the score what class it is detected and what is the box that is the boundary of that detected object. It will be made as a box or the coordinate of the box will also be identified and that will be made as a output an image-viewer to present the image based on the pipeline. So basically I am going to have a file source object detection processor an image-viewer sync. The test image can be in flickr.

4.2 Notebook Implementation

To integrate the notebook logic to the ML-Lab application two where the major issues that I face. The **first** is the out-dated code of the Apache Zeppelin Web Angular App that created incompatibility problems with the dependencies of the Spring Cloud Dataflow core Application. Both are develop using the same programming language, Angular, both have a websocket and a REST mechanism to communicate with their servers. But some of their dependencies were conflicted. Especially the ones that created the UI. The Spring cloud datflow was using the Angular NgModule library whereas the Apache Zeppelin was using the NG-ZORRO component library. So to integrate Notebooks into ML-Lab, I re-created the look and feel of the Apache Zeppelin using the one the Spring Cloud dataflow has deployed.

I modified the Menu items on ML Lab to insert the functionality of the Apache Zeppelin. I have to create REST endpoints based on the Apache Zeppelin Web Angular App to meet the code style and the http communication of the Spring Cloud dataflow. For example the REST call to get the Apache Zeppelin version transformed:

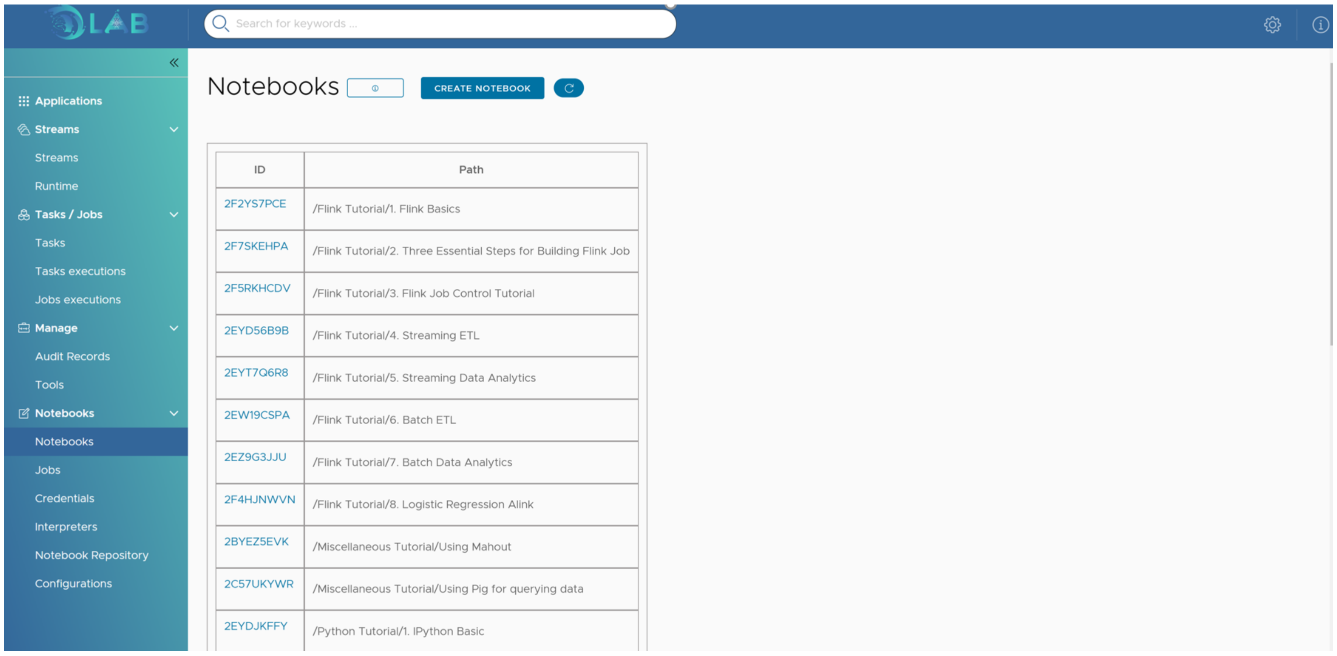
```
//Service
getZeppelinVersion(): Observable<string | unknown> {
    const headers = HttpUtils.getDefaultHttpHeaders();
    return this.httpClient
      .get<IZeppelinVersion>(`${this.baseUrlService.getRestApiBase()}/version`, {headers}).pipe(
        map(data => data.version),
        catchError(ErrorUtils.catchError)
    );
  }
// on AboutZeppelinComponent
getZepellinVersion(): void {
    this.ticketService.getZeppelinVersion().subscribe(data => {
      console.log("data.toString():",data)
      this.version = data.toString();
    })
}
```

Code 4.8: ML-Lab receiving Apache Zeppelin version

I also integrate the Apache Zeppelin web app inside the ML-Lab to enable users to create their notebooks in different programming languages and collaborate with colleagues to share results and use the environment as an build-in editor.

To have an overlook on the Notebooks menu item, the ML Labs have:

- a page to view the created notebooks and create or select an existing one to view, edit or run it as shown in Figure 4.14
- a page to view the credentials and configure new ones for the different interpeters
- a page to view the existed interpreters and enable new ones
- a page that zeppelin configuration is provided
- a page where user can monitor the status of the notebooks



ID	Path
2F2YS7PCE	/Flink Tutorial/1. Flink Basics
2F7SKEHPA	/Flink Tutorial/2. Three Essential Steps for Building Flink Job
2F5RKHCDV	/Flink Tutorial/3. Flink Job Control Tutorial
2EYD56B9B	/Flink Tutorial/4. Streaming ETL
2EYT7Q6R8	/Flink Tutorial/5. Streaming Data Analytics
2EW19CSPA	/Flink Tutorial/6. Batch ETL
2EZ9G3JJU	/Flink Tutorial/7. Batch Data Analytics
2F4HJNWVN	/Flink Tutorial/8. Logistic Regression Alink
2BYEZ5EVK	/Miscellaneous Tutorial/Using Mahout
2C57UKYWR	/Miscellaneous Tutorial/Using Pig for querying data
2EYDJKFFY	/Python Tutorial/1. IPython Basic

Figure 4.14: ML Lab Notebooks View

4.2.1 Securing Apache Zeppelin

I wanted to allow the users to do a single-sign-on from ML Lab but also secure REST calls (using the access token from keycloak) that configure the communication between ML Lab notebooks and the Apache Zeppelin Server. Users will be already logged in on ML Lab with Keycloak. Therefore, I needed Keycloak to share the existing user credentials with the Apache Zeppelin.

Zeppelin uses **Apache Shiro** [36] as its security framework, which, itself, is an open source software framework for authentication, authorisation, cryptography and session management. As from the time that this Master thesis is being written, there is no Keycloak OIDC client directly implemented on Shiro, but I found a solution to bypass this problem by using Pac4j [37], a "security engine for Java to authenticate users. Pac4j provide the user profiles and manage authorisations in order to secure web applications and web services. It provides a comprehensive set of concepts and components from which we will use: client, authenticator, user profile, security filter, callback and logout endpoints. This integration with Shiro, Pac4j will allow Zeppelin user credentials to be shared from ML Lab with the Keycloak without promted user to re-enter credentials to login to the notebooks.

The steps to Follow was:

(a) The modification of the **[$ZEPPELIN_HOME]/conf/shiro.ini** by adding the **OIDC Pac4j Config** and the **Pac4j client**.

```
# OIDC Pac4j Conf
oidcConfig = org.pac4j.oidc.config.OidcConfiguratio
oidcConfig.withState = false
oidcConfig.discoveryURI = https://<KEYCLOAK_URL>/auth/realms/mllab/.well-known/openid-
    configuration
oidcConfig.clientId  = dataflow
oidcConfig.secret  = ******************
oidcConfig.clientAuthenticationMethodAsString = client_secret_basic
oidcClient  = org.pac4j.oidc.client.OidcClient
oidcClient.configuration  = $oidcConfig
# Pac4j Client
 clients  = org.pac4j.core.client.Client
 clients.callbackUrl = https://<ZEPPELIN_URL>/api/callback
 clients.clients  = $oidcClient
```

Code 4.9: Pac4j COnfiguration for Apache Zeppelin authentication with Keycloak

(b) Afterward I configure the Pac4jRealm and SecurityFilter using the following line:

```
pac4jRealm = io.buji.pac4j.realm.Pac4jReal
pac4jRealm.principalNameAttribute = preferred_username
pac4jSubjectFactory = io.buji.pac4j.subject.Pac4jSubjectFactory
securityManager.subjectFactory = $pac4jSubjectFactory
oidcSecurityFilter = io.buji.pac4j.filter.SecurityFilter
```

Code 4.10: Pac4jRealm and SecurityFilter for Apache Zeppelin authentication with Keycloak

(c) Create a custom Pac4j CallbackLogic that will take the callback default URL instead of the Zeppelin standard URL.

```
customCallbackLogic = bio.ferlab.pac4j.ForceDefaultURLCallbackLogi
callbackFilter = io.buji.pac4j.filter.CallbackFilter
callbackFilter.defaultUrl = https://zeppelin.mllab.cloud
callbackFilter.callbackLogic = $customCallbackLogic
```

Code 4.11: Callback Filter for Apache Zeppelin authentication with Keycloak

(d) Lastly, I enabled the protected Urls.

```
[urls]
/api/version = anon
/api/callback = callbackFilter
/api/notebook/** = anon
/api/notebook/**/permissions = oidcSecurityFilter
/** = oidcSecurityFilter
```

Code 4.12: Callback Filter for Apache Zeppelin authentication with Keycloak

4.3 Chatbot Widget Creation

To create the chatbot widget I extended the functionality (skills) provided by an open source voice assistance called Mycroft which is the world's leading open source voice assistant. It is private by default and completely customizable, where you give him intents and he gives you answers. Mycroft AI, Inc. maintains a device and account management system known as Mycroft Home. As I wanted to take advantage of the Mycroft Home service that provides access to a range of APIs keys for specific services, I created an account on the Mycroft Home. The alternative was to use Mycroft without Home. The Mycroft backend provides access to a range of API keys for specific services. And without pairing with the Mycroft backend, to keep access to the functionality I have to add our own API keys, install a different Skill or Plugin to perform that function. But at this phase the extension of the functionality is the goal.

At this moment Mycroft **does not come with any security mechanism** for the websocket. I worked on securing the websocket (messaging service) with Keycloak, as was the authentication protocol I selected for ML Lab. Mycroft-core is written n Python using tornado framework. To include the security mechanism, I edit the test Dockerfile to provide the requirements to enable the keycloak mechanism using the Python Keycloak library, where I included the installation of tornado_http_auth, redis, pymongo, sqlalchemy, pyjwt, cryptography,python-keycloak before the mycroft core vm installation. On the mycroft core message bus message file i inserted the command to secure the websocket with the keycloak.

```
from keycloak import KeycloakOpenID
keycloak_openid = KeycloakOpenID(server_url=os.getenv('KEYCLOAK_URL', 'https:keycloak.mllab.
    cloud/auth/'), #TODO
```

```
                                  client_id=os.getenv('KEYCLOAK_CLIENT_ID', 'mllab'),
                                  realm_name=os.getenv('KEYCLOAK_REALM_NAME', 'mllab'))
...
```

Code 4.13: Keycloak Configuration in Mycroft Core

And then, I modify the deserialize message method to include the token, where the intent from the user includes the token of the user. If the token validated the response-answer will be send back.

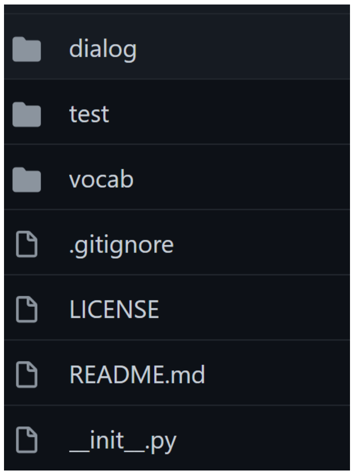
```
@staticmethod
    def deserialize(value):
        obj = json.loads(value)
        # if request send from mllab
        if obj.get('token'):
            # if contains the message and token from mllab
            if obj.get('type') == 'recognizer_loop:utterance':
                input_token = obj.get('token')[0].replace("\n", "")
                # verify user authentication
                userinfo = keycloak_openid.userinfo(input_token)
            ...
                return Message(obj.get('type') or '',
                               obj.get('data') or {},
                               obj.get('context') or {})
```

Code 4.14: Secure Messaging between Mycroft Core and ML Lab

This widget idea came from a recent H2020 project, I am working on creating a virtual personal assistant for manufacturing purposes, called COALA https://www.coala-h2020.eu/. When working on Machine Learning models I caught myself being in a position that I was trying to validate my results and see how well my models predict. For example, on a weather prediction model for the following week, that I wanted to validate my result. The only thing I have to do, was to check about the actual weather conditions that days. To found out these information I have to leave the project window or browser tab and search on different weather services and compare the results. On a recent project where I wanted to predict the rate of the deaths due to Covid-19 each day, I also have to search online for the answer each day. The time that I was spending on searching made my think that it will be a great addition when developing to having a service where all I have to do was to ask for something and get an instant response. To provide this functionality I use the some of the default skills and provide extra functionality with the addition of three new skills. For example the default skills I enabled was:

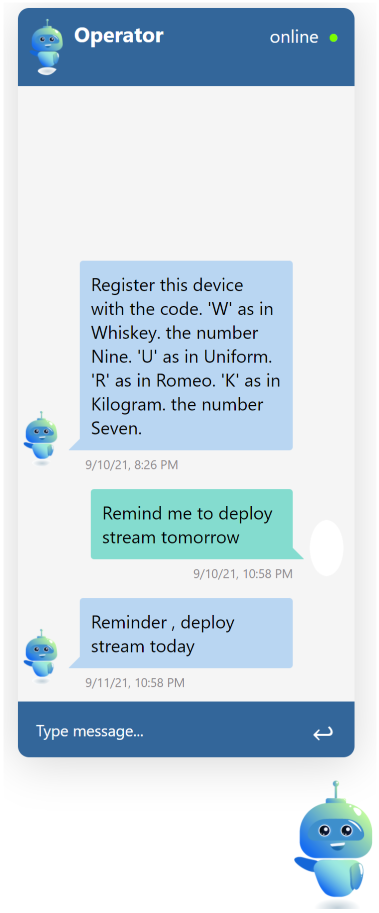
- **skill-stock** that provide current stock prices and can be used on a stock price prediction model, as it uses the Financial Modeling Prep API, to provide you the current price of the stocks.
- **skill-weather** that returns weather conditions, forecasts, expected precipitation and more.
- **skill-date-and-time** that returns the local time or time for major cities around the world.
- **skill-npr-news** that returns the latest news report from your favorite broadcast
- **skill-query** that negotiates the best answer to a question. This is done by sending a question:query message with the utterance to give the skills the posibility to report back if they can answer the question and at which confidence.

I also created a skill called **gina-reminder** based on Mycrofts **skill reminder** where I extended the functionality set reminders for a specific task a user want to remember. To use this skill I have to disable the default skill-reminder and install using mycroft bash command the new created skill. For example 4.15. Inside the folder **dialog** the responses of the Mycroft are existing based on user intent, for exmaple if the user writes just "Set reminder" then the response will be "About what?" or **Sure, What should I remind you about?**. Inside the **vocab folder** the user sentence to set the reminder must be placed. For example **(could you |)remind me (about|of|to) reminder**. Inside the __init__.py all thelogic of the skills are existing. From receiving the intent, setting the reminder and triggering the response based on the time frame user defined (same day, tomorrow).



```
dialog
test
vocab
.gitignore
LICENSE
README.md
__init__.py
```

Figure 4.15: Custom Reminder Skill File Structure



Operator online

Register this device with the code. 'W' as in Whiskey. the number Nine. 'U' as in Uniform. 'R' as in Romeo. 'K' as in Kilogram. the number Seven.
9/10/21, 8:26 PM

Remind me to deploy stream tomorrow
9/10/21, 10:58 PM

Reminder , deploy stream today
9/11/21, 10:58 PM

Figure 4.16: Mycroft Custom Reminder Skill

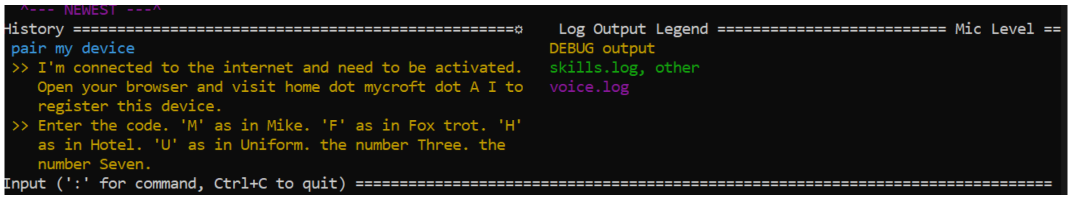
To serve all this functionality, I created an Angular widget using Angular Elements to serve the chatbot at the right bottom of the screen. Angular Elements is a new package in Angular that helps on publishing Angular components as custom elements. It does this by taking the Angular component and compiling it into a web component. The library that enabled my to configure the websocket communication between ML-Lab chatbot and Mycroft core was **ngx-socket-io**. Also angular animations were created to perform the fade in and fade out of the chatbot modal. To integrate the chatbot widget to ML-Lab Dashboard I have to modify the AppModule.

To deploy the Mycroft core after the configuration for the additional skills and the security mechanism(Keycloak) the Dockerfile created for custom mycroft-core must be build and run. To run the Docker image the following commands can b used:

```
docker build -t mllab-bot:1.1.0 .
docker run --name=mllab-bot -p 8181:8181 -it --rm mllab-bot:1.1.0
```

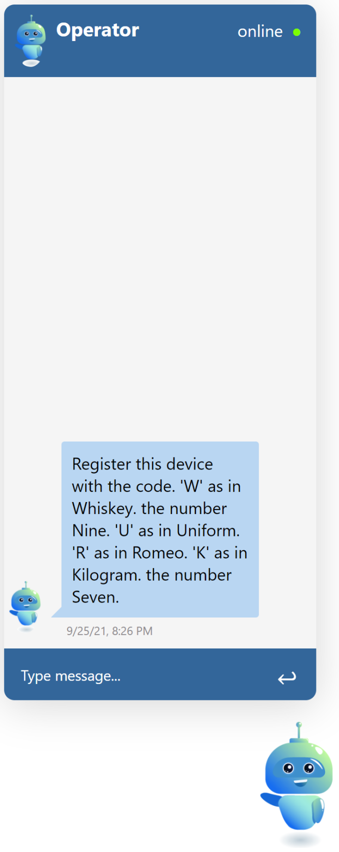
By running the **./start-mycroft.sh debug** command inside the bash we start the Mycroft core. The services and the skills take a little time to be configured. After the configuration is finish, the pairing code is presented.

By saying ”Hey Mycroft, pair my device”, user will be informed that the device needs to be paired. Mycroft will provide a 6-digit code as shown in Figure 4.17 which I enter into the pairing page within the Mycroft Home site that containing the core services (server) as shown in Figures 4.18. I created a device on the Mycroft Home based on the pairing key I received and that enabled me to start asking questions, Figure 4.19.



```
 ^--- NEWEST ---^
History ===============================================☼   Log Output Legend ========================= Mic Level ==
 pair my device                                           DEBUG output
 >> I'm connected to the internet and need to be activated.  skills.log, other
    Open your browser and visit home dot mycroft dot A I to  voice.log
    register this device.
 >> Enter the code. 'M' as in Mike. 'F' as in Fox trot. 'H'
    as in Hotel. 'U' as in Uniform. the number Three. the
    number Seven.
Input (':' for command, Ctrl+C to quit) ==========================================================================
```

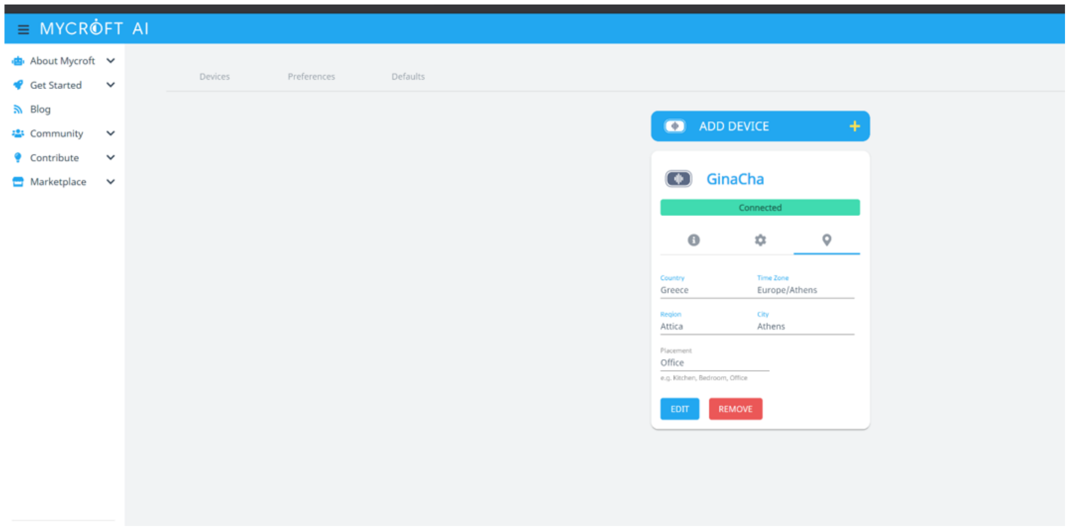
Figure 4.17: Mycroft Core (server) Pairing



Register this device with the code. 'W' as in Whiskey. the number Nine. 'U' as in Uniform. 'R' as in Romeo. 'K' as in Kilogram. the number Seven.

9/25/21, 8:26 PM

Figure 4.18: ML Lab integrated Bot Pairing



GinaCha

Connected

Country: Greece
Time Zone: Europe/Athens
Region: Attica
City: Athens
Placement: Office
e.g. Kitchen, Bedroom, Office

Figure 4.19: Bot Home Pairing

CHAPTER 5

Conclusions

The purpose of this Master thesis was to develop a micro-service application for automated analytic applications deployed to k8s. This thesis started with an introduction to the theoretical and technical background of the key technologies and components and took a dive into the different approach of platforms providing this functionality to develop micro-services as directed acyclic graphs. The rest of this thesis is divided into 2 parts.

In the first part, the key technologies and programming languages required for this implementation are presented and provide the prototype of the application. In the second part, the development of the presented prototype is shown. Followed by the description of the steps required to extend the functionality, to provide user with the exported YAML manifest of a deployed pipeline that can be used to share their data pipelines with others, where they can be run and out-side this toolkit. An explanation of the problem occurring with the dependencies incompatibility of the Spring Cloud Dataflow and the Apache Zeppelin that force me to create a new interface for the notebooks functionality inside the ML Lab toolkit and create new services to get the functionality from Apache Zeppelin Server using new REST endpoints. I also stated how the inclusion of a widget written in Angular that handles the communication between the Mycroft core and the user can assist the user to set reminders to finish tasks, or ask for guidance for creating a new stream or task, or simply ask for information, using the default skills of Mycroft and two custom skills developed for this purpose. To manage the identity and access of users throughout the application and services, a security mechanism called Keycloak is used to connect the user and provide a token that you will be shared with other services to verify the authenticity of the user. It is important to state that the combined solution presented can not be found in literature or the open source.

CHAPTER 6

Future Implementations

To answer this question I asked myself “What should be the next steps or features this toolkit you want to provide to end-user?”. Search is so easy for users to use, and they almost expect it in every application. On the other hand, it can be so difficult to build; that simple text box in your application can mean so much work if you have to do it all by yourself. At this moment, user can search for elements or find information about this toolkit using the search box, or the chatbot or inside the pop modal of the info icons by the side of key elements, like the different status of a stream pipeline, what is notebooks?, streams, task, and how to use or start create your first node, and inside the manual that is linked in the info section of the application. A feature that I have imagine this application to have related to that is a cognitive search mechanism by creating skill for the chatbot with AI capabilities that enrich all types of information to identify and explore relevant content inside ML-Lab. This can significantly increase productivity through personalization and by predicting the user’s actual search intends(question). User for example can ask “what is the fist pipeline app I used in stream **Name_OF_Stream**” and the chatbot will reply with the answer. Or the chatbot can inform the user a long running task finished its deployment. In this Master thesis the integration of the notebooks logic aimed to enable users to create and run different machine learning tasks, by using the notebooks as an editor. Although to support the docker container registration to the application as stated in the Spring Cloud Dataflow, Polyglot Data Storage is necessary to register Python Applications and deploy Dockerized apps inside this toolkit. As a future step I would like this tool to enable users to have their custom python applications inside their pipelines by using the notebooks environment.

Bibliography

[1] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware Software Interface ARM Edition*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2016.

[2] Jargon file entry for ”user”. Available at http://catb.org/jargon/html/U/user.html.

[3] Json. Available at https://en.wikipedia.org/wiki/JSON.

[4] R. Crockford. (2006, 7) The applicationjson media type for javascript object notation (json). Available at https://datatracker.ietf.org/doc/html/rfc4627.

[5] Introducing json. Available at https://www.json.org/json-en.html.

[6] R. Bray. (2017, 12) The javascript object notation (json) data interchange format. https://datatracker.ietf.org/doc/html/rfc8259.

[7] V. Sinha, F. Doucet, C. Siska, R. Gupta, S. Liao, and A. Ghosh, “Yaml: a tool for hardware design visualization and capture,” in *Proceedings 13th International Symposium on System Synthesis*, 2000, pp. 9–14.

[8] erik. (2018, 12) Yaml tutorial: Everything you need to get started in minutes. Available at https://www.cloudbees.com/blog/yaml-tutorial-everything-you-need-get-started.

[9] S. Gholami, H. Khazaei, and C.-P. Bezemer, "Should you upgrade official docker hub images in production environments?" in *2021 IEEE/ACM 43rd International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, 2021, pp. 101–105.

[10] K. Jangla, *Docker Basics*. Berkeley, CA: Apress, 2018, pp. 27–53. [Online]. Available: https://doi.org/10.1007/978-1-4842-3936-0_4

[11] T. Combe, A. Martin, and R. Di Pietro, "To docker or not to docker: A security perspective," *IEEE Cloud Computing*, vol. 3, no. 5, pp. 54–62, 2016.

[12] N. Zhao, V. Tarasov, H. Albahar, A. Anwar, L. Rupprecht, D. Skourtis, A. S. Warke, M. Mohamed, and A. R. Butt, "Large-scale analysis of the docker hub dataset," in *2019 IEEE International Conference on Cluster Computing (CLUSTER)*, 2019, pp. 1–10.

[13] T. Butler. (2015, 9) Jargon file entry for "user". Available at https://www.cloudbees.com/blog/what-is-a-dockerfile.

[14] R. Gatev, *Getting Up to Speed with Kubernetes*. Berkeley, CA: Apress, 2021, pp. 51–67. [Online]. Available: https://doi.org/10.1007/978-1-4842-6998-5_3

[15] A. Poniszewska-Marańda and E. Czechowska, "Kubernetes cluster for automating software production environment," *Sensors*, vol. 21, no. 5, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/5/1910

[16] J. Shah and D. Dubaria, “Building modern clouds: Using docker, kubernetes amp; google cloud platform,” in *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, 2019, pp. 0184–0189.

[17] P. H.-B, “Assertions and protocol for the oasis security assertion markup language (saml),” 2002.

[18] P. M., “Bindings and profiles for the oasis security assertion markup language (saml),” 2002.

[19] W. Staff. (2021, 9) Saml. Available at https://www.webopedia.com/definitions/saml/.

[20] M. Pierce, M. Miller, E. Brookes, M. Wong, E. Afgan, Y. Liu, S. Gesing, M. Dahan, T. Walker, and S. Marru, “Towards a science gateway reference architecture,” *CEUR Workshop Proceedings*, vol. 2357, Jan. 2019, 10th International Workshop on Science Gateways, IWSG 2018 ; Conference date: 13-06-2018 Through 15-06-2018.

[21] J. Basney and V. Welch, “Science gateway security recommendations,” in *2013 IEEE International Conference on Cluster Computing (CLUSTER)*, 2013, pp. 1–3.

[22] Incommon federation. Available at https://www.incommon.org/.

[23] D. Hardt, “The oauth 2.0 authorization framework,” *RFC*, vol. 6749, pp. 1–76, 2012.

[24] J. Basney, T. Fleury, and J. Gaynor, “Cilogon: A federated x.509 certification authority for cyberinfrastructure logon,” *Concurrency and Computation: Practice and Experience*, vol. 26, no. 13, pp. 2225–2239, 2014. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.3265

[25] Keycloak. Available at http://www.keycloak.org/.

[26] Apache spark. Available at http://spark.apache.org/.

[27] Y. Cheng, F. C. Liu, S. Jing, W. Xu, and D. H. Chau, "Building big data processing and visualization pipeline through apache zeppelin," in *Proceedings of the Practice and Experience on Advanced Research Computing*, ser. PEARC '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3219104.3229288

[28] Apache software foundation:. Available at http://www.apache.org/.

[29] jbosskeycloak 15.0.2. Available at https://hub.docker.com/layers/jboss/keycloak/15.0.2/images/sha256-d8ed1ee5df42a178c341f924377da75db49eab08ea9f058ff39a8ed7ee05ec93?context=explore.

[30] Baeldung. (2020, 11) Customizing themes for keycloak. Available at https://www.baeldung.com/keycloak-user-registration.

[31] Keycloak themes docs. Available at https://www.keycloak.org/docs/4.8/server_development/#_themes.

[32] S. Wagde. (2020, 10) Customizing the login page for keycloak. Available at https://www.baeldung.com/keycloak-custom-login-page.

[33] ——. (2020, 10) Customizing themes for keycloak. Available at https://www.baeldung.com/spring-keycloak-custom-themes.

[34] little_pinecone. (2021, 8) Keycloak in docker #2 – how to import a keycloak realm. Available at https://keepgrowing.in/tools/keycloak-in-docker-2-how-to-import-a-keycloak-realm/.

[35] Available at https://spring.io/projects/spring-cloud-stream-app-starters.

[36] Available at https://shiro.apache.org/integration.html.

[37] Available at https://github.com/bujiio/buji-pac4j.