RWTH Aachen University

Master Thesis

University Degree in Software System Engineering

2018-2021

Master Thesis

“Design and evaluation of a subsystem (mircoservice) to create a deployment client for AI pipelines based on docker containers using gRPC”

Author

Sajid Naeem

Examiners

Prof. Dr. Jens Lehmann

Prof. Dr. Sven Behnke

Supervisor

Martin Welß

ABSTRACT

Before docker container technology, manual deployment of an application is complex, resource, and time-consuming. With the help of Kubernetes, it is possible to automatically deploy and manage the AI pipelines on a standard cluster. The thesis aims to provide the link between the AI4EU experiment platform catalog from the execution environment to make the system more scalable. In this thesis, we design a solution and implement the Kubernetes client, which takes the AI pipeline's topology as input from the catalog and constructs the deployment and service for all the nodes of the AI pipeline for the execution environment. Kubernetes client also generates a container specification based on the pipeline's topology, which the orchestrator uses to execute the pipeline. Different AI pipelines are deployed in separate namespaces with the help of a generic deployment script supporting standard Kubernetes cluster and minikube. The Kubernetes client tested on simple, advance, and hybrid AI pipelines and is also integrated with the production environment of the AI4EU experiment platform and gets feedback from the AI community of this platform.

Keywords: docker, Kubernetes, AI Pipelines, AI4EU experiment platform

DEDICATION

First, I want to thanks my colleague and my supervisor Martin Welß for his support throughout my thesis work. I am thankful to Martin Welß for supporting me when I was suffering from the Covid-19 infection and for providing me such a good infrastructure and other technical supports for experimentation.

Secondly, I would like to show my intense gratitude toward Prof. Dr. Jens Lehmann and Prof. Dr. Sven Behnke for supporting me. I am thankful to the AI4EU experiment platform community who are working with me. I am grateful to Fraunhofer IAIS and especially the NetMedia department for providing me everything for my thesis.

Lastly, I want to thanks my wife and my parents for motivating me to study and work hard and for every kind of emotional support. I am also thankful to all of my friends for supporting me.

Aachen, March 30, 2021

Sajid Naeem

Zentrales Prüfungsamt/Central Examination Office

Eidesstattliche Versicherung
Statutory Declaration in Lieu of an Oath

Name, Vorname/Last Name, First Name

Matrikelnummer (freiwillige Angabe)
Matriculation No. (optional)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Arbeit/Bachelorarbeit/ Masterarbeit* mit dem Titel
I hereby declare in lieu of an oath that I have completed the present paper/Bachelor thesis/Master thesis* entitled

selbstständig und ohne unzulässige fremde Hilfe (insbes. akademisches Ghostwriting) erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.
independently and without illegitimate assistance from third parties (such as academic ghostwriters). I have used no other than the specified sources and aids. In case that the thesis is additionally submitted in an electronic format, I declare that the written and electronic versions are fully identical. The thesis has not been submitted to any examination body in this, or similar, form.

Ort, Datum/City, Date

Unterschrift/Signature

*Nichtzutreffendes bitte streichen
*Please delete as appropriate

Belehrung:
Official Notification:

§ 156 StGB: Falsche Versicherung an Eides Statt
Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.
Para. 156 StGB (German Criminal Code): False Statutory Declarations
Whoever before a public authority competent to administer statutory declarations falsely makes such a declaration or falsely testifies while referring to such a declaration shall be liable to imprisonment not exceeding three years or a fine.
§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt
(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.
(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.
Para. 161 StGB (German Criminal Code): False Statutory Declarations Due to Negligence
(1) If a person commits one of the offences listed in sections 154 through 156 negligently the penalty shall be imprisonment not exceeding one year or a fine.
(2) The offender shall be exempt from liability if he or she corrects their false testimony in time. The provisions of section 158 (2) and (3) shall apply accordingly.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:
I have read and understood the above official notification:

Ort, Datum/City, Date

Unterschrift/Signature

CONTENTS

1. INTRODUCTION 1
1.1. Motivation 2
1.2. Problem Description 2
1.3. Objective 3
1.4. Goal 3
1.5. Research Problem 4
1.6. Layout of Thesis 4
2. BACKGROUND 5
2.1. Web Services 5
2.1.1. SOAP 7
2.1.2. REST 8
2.2. AI4EU Experiment Platform 9
2.2.1. Web Onboarding 10
2.2.2. Design Studio. 11
2.2.3. Market Place 11
2.3. Technology Utilization 12
2.3.1. Why Docker is used for virtualization in AI4EU experiment platform instead of Virtual Machine 12
2.3.2. Docker Execution Environment. 14
2.4. Kubernetes 15
2.4.1. Pods 16
2.4.2. Deployments 16
2.4.3. Service 16
2.4.4. Namespaces. 16
2.4.5. Kubernetes Persistent Volumes 17
2.5. Pykaldi 17
3. RELATED WORK 18
3.1. AI Platform Pipelines 18

3.2. Acumos 18
4. METHODOLOGY 20
4.1. AI4EU Experiment Platform Solution Deployment in Private Kubernetes Cluster 20
4.1.1. Kubernetes Client Architecture 21
4.2. Component Design of Kubernetes Client 23
4.2.1. Common Data Service 25
4.2.2. Automatic Generation of Services and Deployments 26
4.2.3. Docker image handling 29
4.3. Mapping of the Pipeline Topology to the Artifacts 30
4.3.1. Pipeline Topology 31
4.3.2. Container Specifications 31
4.4. Kubernetes Deployment Script 31
4.5. Generic Serial Orchestrator 33
4.6. Challenges Faced During Development 33
5. EXPERIMENTS. 35
5.1. AI Pipelines and Its Deployment 35
5.2. Example Pipelines 35
5.2.1. Deployment of Simple Solution 36
5.2.2. Deployment of Composite Solution 36
5.3. How to make a Pipeline more Robust and Secure 40
5.4. Challenges Faced During Experimentation 42
6. DISCUSSION 43
7. CONCLUSION AND FUTURE WORK 46
BIBLIOGRAPHY. 48

LIST OF FIGURES

2.1 Communication Between Client and Web Server via Internet 6
2.2 SOAP Nodes 7
2.3 Rest API Method 8
2.4 Modern Approach of Rest API 9
2.5 Audio Pipeline 11
2.6 A Comparison of Virtual Machine and Docker Container Design [14] . . 13
2.7 Docker Engine 13

4.1 Solution Folder 21
4.2 Kubernetes Client Architecture 22
4.3 Kubernetes Client Execution 24
4.4 Solution Package 25
4.5 Creation of Solution Package 26
4.6 NodePort Service 27
4.7 Deployment of Sentiment Analysis 30
4.8 Topology of Sentiment Analysis Pipeline 32
4.9 Container Specification of Sentiment Analysis Pipeline 33

5.1 Audio Segmentation Docker Image 37
5.2 Simple Solution of Sentiment Analysis 38
5.3 Audio Dialogue Creator Pipeline 39
5.4 Audio Segmentation Docker Image 41
5.5 Limiting the Pod Capabilities 41

LISTINGS

2.1 Sentiment Analysis Protobuf Definition [1] 9
4.1 Sentiment Analysis Service 27
4.2 Sentiment Analysis Deployment 28
5.1 Audio Data Broker [2] 37
5.2 Audio Segmentation [2] 38
5.3 Container User 40

1. INTRODUCTION

Nowadays, micro-services architecture is gaining very popularity in the software development industry. This architectural approach is used for building a distributed application to achieve faster delivery, high availability, independently deployable, and scalable system. In this approach, the software is broken into smaller methods, running as an independent micro-service that can be extended and deployed easily without affecting the other running micro-services. Usually, services are distributed on multiple hosts, which is challenging to track and deploy. A stand-alone micro-service can be deployed into different programming languages with different dependencies like it requires a different framework, libraries, and data sets. It makes deployment a complex and challenging task because it requires more resources to manage. These problems are solved using state-of-the-art Docker container technology.

Docker is a tool used for building and running a stand-alone containerized application that contains all of its dependencies. For each micro-service, one docker container is created, which is itself a stand-alone application. Communication and data-sharing between the running containers are done with the help of the gRPC framework. In a production environment, the container is an excellent way to pack all its requirements in one place, but its management is a challenging task. For example, if one container goes down, then the new container needs to start immediately. This problem is solved by Kubernetes, which ensures that there is no downtime. Kubernetes is an open-source orchestration tool which used for the management of containerized application and their automatic deployment. With the combination of these toolsets, we get a loosely coupled design, independently deployable components, communication between the service is managed by the Kubernetes.

For the implementation of our designed solution, we are using AI4EU Experiment platform, and we are practicing a microservice-based design approach. AI4EU platform is based on ACUMOS instances with lots of customization. In the customization of the ACUMOS instance, our approach was to build a language-agnostic instance that is solved using gRPC. The AI4EU experiment platform provides the facilities for modular AI solutions that support the different required languages. The first step is the AI resource onboarding which needs three things:

- A service definition implemented as a protobuf file (.proto)
- A license as a JSON file (license.json)
- A docker image URI, which is a reference to a container image.

A container contains a core programming logic, and it may be a data broker or generic data broker, a pre-trained model or training model, etc. The next step is to design a

pipeline by using a design studio. The design studio is a visual editor of the AI4EU experiment used to make an AI pipeline. This visual editor provides essential tools and perceptions for creating an AI solution, e.g., the matching point of nodes highlighted while constructing a pipeline. After making the pipeline, the design studio checks the connections between the nodes based on the protobuf interface definitions and then saves and validates it. In the AI4EU experiment platform, every node of a pipeline is considered an AI model. Design Studio provides the topology of an AI pipeline. Kubernetes-client uses the topology for the generation of micro-services and is also used to create deployment and services for each node of a pipeline and later download all the generated artifacts of the pipeline and run the deployment script to deploy the solution locally. After deployment, an orchestration is used for the execution of the whole pipeline. The later section of the first chapter explains the problem statement and my motivation toward the thesis.

1.1. Motivation

This section highlights the urge behind this thesis work. The motivation comes from the AI4EU experiment platform, which brings scientists, researchers, experts, and companies on one platform to design and solve AI-based problems. The other motivation is the learning aspect of the project as it provides the opportunity to work with state-of-art technologies such Kubernetes, docker, java, gRPC, and google Protobuf, etc. This project allows me to deploy my solution in the production environment and then received feedback from the AI community.

1.2. Problem Description

In the project AI4EU, a format has been specified for re-usable building blocks of AI pipelines. The typical granularity of such building blocks is a model (e.g., a model for speaker recognition and not single deep learning layers). The format mainly specifies the model to be a docker container that exposes its public services by gRPC/protobuf. There is already a visual editor that allows the composition of a pipeline by Drag and Drop. The result of the visual editor is the descriptions of the topology. On the way to the execution of the pipeline, two more steps are needed. First, the deployment of the pipeline to an execution environment (Kubernetes), and second, a runtime-orchestrator has to control the actual execution of the pipeline.

Design and evaluate a subsystem (microservice) to create a deployment client for AI pipelines based on docker containers using gRPC. An AI pipeline in this context consists of nodes (docker containers) and edges (information flow), thus forming a graph. The deployment client will create the necessary Kubernetes artifacts like deployment.yaml, service.yaml and container specifications (docker-info.json) based on the pipeline topology specification (blueprint.json). The following tasks require to be completed in the course of this master thesis:

1. Design and implement deployment client, which will create the deployments and services for the AI pipeline. It should work for a simple and advanced pipeline. The deployment client should provide the script which deploys the generated artifacts in the local environment.

2. Analyse the format for the topology specification file (blueprint. json) and create node parameters (docker-info.json), taking into account the scenarios from step 1 and describe the transformation rules.

3. Design and implement the subsystem as well as the necessary algorithms and test it with scenarios from step 1.

4. Verify the solution in a Kubernetes cluster by running the orchestrator to execute the pipeline.

1.3. Objective

The main objective is to design a deployment client which will generate artifacts from the topology of a pipeline and the topology generated from the design studio. The resultant artifacts will include deployment and services for each node with correct node mapping independent of the local execution environment. The deployment script is responsible for running all the services and deployments. All the running services should be accessible from the orchestrator. All the DNS name resolution should be resolved and stored in a docker-info file, and the deployment script will return the IP address and node port of the orchestrator server. In the end, the result should be tested on a single node pipeline and also on a composite solution pipeline.

The designed deployment client will run on different AI pipelines. Simple pipelines include house price prediction and sentiment analysis examples. The deployment client will also be working fine with an advanced AI pipeline, for example, an audio mining pipeline. This audio mining pipeline will be implemented using open-source software pyKaldi by using its pre-trained models. The client should also work fine with the hybrid pipeline, for example, sudoku.

1.4. Goal

The goal of the thesis, design and implement a scientific solution for deployment clients, which will work for all kinds of the pipeline. It will consider a simple, advance, and hybrid pipeline. The designed solution is integrated into the AI4EU experiment platform. The deployment client will provide the connection between the AI4EU experiment platform catalog and the local execution environment. The deployment client should create the artifacts for all kinds of models. The model should be implemented in any programming language, so the deployment client is independent of any programming language.

Multiple pipelines should be deployed in a separate way to ensure the separation of concerns.

1.5. Research Problem

Following are the research questions that will be answered in this master thesis.

- **RQ1:** How can we connect the AI4EU experiment platform catalog and its execution environment?
- **RQ2:** How can we transform the topology of the AI pipeline to deployment artifacts by using scientific research approach? As a topology of the pipeline is generated from the design studio, and artifact includes the deployment.yaml, service.yaml and docker-info.json files.
- **RQ3:** How can we deploy all kinds of deployments and services which the deployment client generates? Deployments need to be independent and adopt the local execution environment.
- **RQ4:** How will the deployment script deploy different kinds of AI pipelines without affecting the other pipelines on a cluster?

1.6. Layout of Thesis

The second chapter is about the background of the work, and it explains the web services and other tools and techniques used in this project and explains the AI4EU experiment platform. Related work about pipeline deployment is described in chapter three. Chapter four is about the methodology of the thesis, where the design of the Kubernetes client is explained. Next, the fifth chapter is about experimentation, showing how the different AI pipelines are deployed. Discussion about the thesis can be found in chapter six. Finally, the last chapter concludes the thesis with some indication about the future work.

2. BACKGROUND

The first section of this chapter starts with the introduction of web services. It gives the reader a brief overview of web services and then provides a general idea of CORBA, SOAP, and REST services. Afterward, section 2.2 talks about the AI4EU experiment platform where it provides the reader an overview of Platform architecture. Next, section 2.3 explains the technology used to develop the project, and section 2.4 explains the Kubernetes. The last section is describing the pykaldi, which is used for speech recognition.

2.1. Web Services

Most things are connected with the internet and exchanging data without physical hardware connections in the current digital age. It is an excellent development in the world of the website, that software starts communicating with each other and smoothly transfer the data over the network or the internet. This communication between software is possible with the help of web services, which is originating from the Remote Procedure Call (RPC). RPC was developed around the 1990s. It is a framework for Distributed Computing environments to make a distributed application on a single system. Later on, further, development was made like Microsoft RPC for inter-process communication across different systems. This evaluation process was going on from RPC to XML-RPC and then towards SOAP [3].

RPC solves the problem of two-tier application communication. It provides a solution for the development of a two-tier application which becomes very famous in distributed computing. As distributed computing becomes more popular in the software development industry, then it is realized that there is a need to develop an N-tier application. RPC does not provide a solution for this problem because it is not very flexible [4]. For such kind of application, data is getting more concerns, a researcher in the field of distributed computing purposed two solutions. The first one is typical object request broker architecture (CORBA) and the second one is distributed common object model (DCOM), and later, Java remote method invocation (RMI) was designed. CORBA was designed in the 1990s, and its architecture was designed to specify interoperability between distributed systems on a network. CORBA designed was very flexible, on a network, distributed objects can communicate with each other independent of the working environment. For example, if one object is running on windows, another object can communicate, running on the Linux operating system. CORBA has another primary feature: interoperability between different programming languages such as Java, C and C++, Ada, etc. [5]. DCOM is a protocol that Microsoft developed in 1996. It allows the two different applications to communicate, running on the different distributed computers, securely and reliably. It is an extension of the Component Object Model (COM), which defines how the object can interact. COM

allows the development of software components in different languages and environments which can be easily integrated and deployed. Using the DCOM model, the object can be accessed on a network by using proxies and stubs. These proxies and stubs allow the requested object to be accessed on the same addressed space [5]. CORBA and DCOM achieve great success in his area. Still, they showed certain limitations and shortcomings when they came into web development, such as creating a tightly coupled distributed web environment where the specific format of data and messages are defined [4]. After some time and growth of web development, it has been seen that its complement web services replace it.

Web services are a modern generation of web application development. It is a framework for developing self-contained application-to-application communications, modular applications that can be located and invoked across the web. Web services have different methods to complete the business logic, e.g., the request should be to update or access the record. Other applications on the internet can access web services, but they need to be deployed once [6]. Web services can create a loosely coupled design to archive high performance, scalability, easy modification, and flexibility in their design. It behaves like a server-oriented architecture to provide a communication channel between services over the internet that can invokable over the web [4]. It provides support in multiple languages, e.g., service is developed in one language and can be accessed and used in different languages [7]. Nowadays, web services are used in many different domains regarding data handling, such as virtual reality and machine learning, cloud computing, and high-performance computing [8] [9].

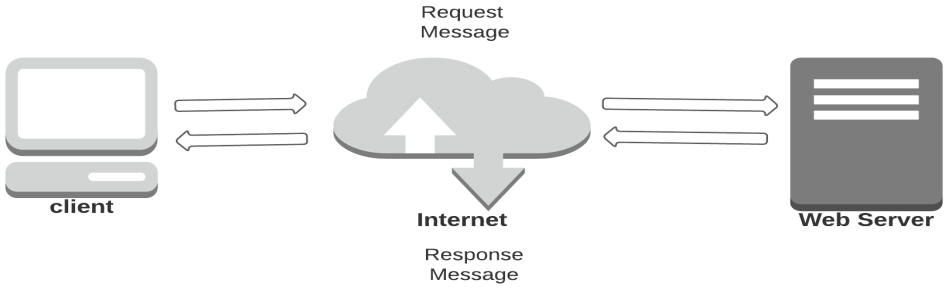

Fig. 2.1. Communication Between Client and Web Server via Internet

The communication between Web services and client done via the internet, the client sent HyperText Transfer Protocol (HTTP) request to the web server, and then the server sends a response message; this is shown in Figure 2.1. There are different kinds of communication protocols. The first one is Simple Object Access Protocol (SOAP), Representational State Transfer (REST), gRPC, etc. The next section describes different kinds of services.

2.1.1. SOAP

The team of Microsoft developed SOAP in 1998. It is a simple, lightweight, and stateless protocol for exchanging structured data in XML (Extensible Markup Language) format or triggering a call to a remote method among the different services over the web. It supports the various protocols for the transportation of messages like HTTP, FTP (File Transport Protocol), and SMTP (Simple Mail Transfer Protocol). HTTP protocol plays a vital role in bridging interactions between the different computers because HTTP protocol is firewall-friendly [10]. SOAP provides a one-way message exchange paradigm between the nodes. Combining this strategy with other features of the under-laying transport protocols, or with the help of application-specific information, creates the desired paradigm for the communication of nodes such as a request-to-response request-to-multiple-response, etc. SOAP consist of three main nodes: SOAP Sender, SOAP Receiver, and SOAP Intermediary node. SOAP nodes are shown in Figure 2.2.

- SOAP Sender node -It is responsible for the generation and transmission of a SOAP message.
- SOAP Receiver node -It is responsible for receiving and processing SOAP messages and sometimes it generates a SOAP response message or fault as well.
- SOAP Intermediary node - It acts as a SOAP receiver and a SOAP sender. It is responsible for receiving and processing the incoming SOAP header blocks and send back the SOAP message to the SOAP receiver [11].

Fig. 2.2. SOAP Nodes

However, the main side effect of the SOAP is to overinflate the data. When data is passed and processed by SOAP protocol, this is due to standards and rules set by the SOAP. Due to XML, SOAP is supporting the different programming languages, but it painful to use it in web technologies because the verb represents all the actions. Because of these and some other performance-related issues, REST is getting more favor. Because it supports different data representation like XML, JSON (JavaScript Object Notation), HTML (HyperText Markup Language) and YAML etc.

2.1.2. REST

Roy Fielding developed rest at the University of California in 2000 for his Ph.D. thesis. It is based on client-server style architecture, where requests and responses are constructed based on the transferring process. All the resources are represented by a distinctive URI (Uniform Resource Identifier), each URI map a document that is used to capture the state of the resource. Usually, the REST architecture is faster and lighter as compared to SOAP. This architecture is straightforward because it does not require a format like a header is part of the message, which is needed in SOAP [11]. On the other hand, parsing of JSON is easier and much faster as compared to the XML. In rest, resources are present in a specific environment on the web, and a way to access these resources is defined in REST. For a given instance, a server that contains some important documents of a business company can be accessed through a machine connected to this server of a company. These documents are known as resources, and a process to access these documents is restful in style. Now there is a question, how to access these resources.? this is done by exposing your API within your application for a client that is trying to access your website. The appropriate request is required to query these RESTFUL services. Usually, this is done by interfacing with HTTP and then invoke the desired functionality. REST supports multiple formats for data transfer, such as JSON, XML, and many more, but JSON is the most regularly used file format for data transfer. Each REST request is unique, and it utilizes one method. Here the method comes from a verb that explains an action that needs to be done on a given resource. The first method is a GET used for data retrieval; the second one is POST; it is used for data upload, whereas the PUT method is used for data updating of the existing records. The delete method is used for the deletion or removal of existing data.

Fig. 2.3. Rest API Method

REST gives a response on every request that includes the status code of your request, whether it is successful or not, whereas more detailed feedback is also attached in plain text. These features make a restful application very successful and even used in today's industry as standard. REST API is lightweight in size as compared to SOAP. It has a message smaller in size, so it is easy and fast to parse these messages. REST is more suitable for mobile application development because of its small message size.

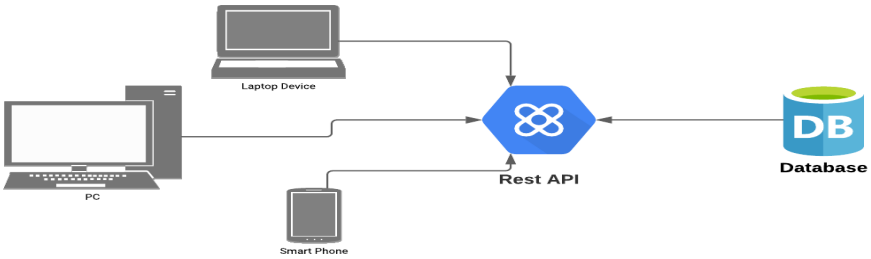

Fig. 2.4. Modern Approach of Rest API

2.2. AI4EU Experiment Platform

AI4EU experiment platform is an open-source project which Fraunhofer IAIS develops. It is based on Acumos, with some significant differences in its design to make it more operable and language agnostic. This platform is used to develop, train, share and deploy, and orchestration of AI models. It empowers data scientists, machine learning researchers, and computer scientists to use and understand the AI-based process for the development of software. AI-based collaboration environment is the main strength of this platform, and here models are trained and ranked according to their ability to understand and analyze the data sets that they are fed and then select the best fitting model for that task. This platform is not only strict to any specific programming language, tool-kits, or any other cloud services. It provides a mechanism for creating, packaging, distributing the AI models in a portable way by making the micro-services and then publish them in the marketplace. AI4EU experiment platform is different from the acumos project by providing more freedom to the model provider. In acumos, the model bundle approach is used where some subset of protobuf is used, protobuf was generated from the reflection of the model. Our approach provides more freedom to the model provider as the user has to define its protobuf. The bundle format has severe limitations, which put limits on the features of protobuf. AI4EU experiment platform works on a standardized approach by using the gRPC for micro-service communication, which is not unique to any programming language and provides interoperability. Using gRPC gives the opportunity to create the stubs and skeleton dynamically with the help of protobuf. It will work for any new unseen pipeline. Protobuf is simple and well structured as compared to JSON, and also it is more efficient and faster. Here is the example protobuf of the sentiment analysis pipeline.

CÓDIGO 2.1. Sentiment Analysis Protobuf Definition [1]

```
1 //Define the used version of proto
2 syntax = "proto3";
3
4 package fraunhofer.sentimentanalysis;
5
6 //Define a message to hold the features input by the client
7 message Text {
```

```
  string query = 1;
}

//Define a message to hold the classification result
message Review_Classify {
  float review = 1 ;
}

//Define the service
service sentiment_analysis_model {
  rpc classify_review(Text) returns (Review_Classify);
}
```

AI4EU Experiment platform is designed to bring all AI stakeholders and researchers together in one dedicated place to speed up AI-based innovation. This platform is a one-stop-shop to get all AI resources, technology, services, and software in one place. The following designs are considered while designing our desired solution. Its design is service-oriented and accessible via web browser only. It supports multiple disciplines like a wide range of symbolic AI algorithms, a hybrid approach, and machine learning problems. It raises the achievable AI innovation potential by providing access to different AI technologies in various fields. AI4EU platform offers a way to combine the diverse isolated communities to get other solutions to the same problems, enabling the discovery of a novel way of the same problem. AI4EU platform is scalable and interoperable in terms of toolkits, data, programming languages, third-party tools, and IT infrastructure. It provides an effective and efficient way to construct the solution by combining state-of-art technologies and significant data sources across the various AI fields. It allows the users of the multiple fields to build the virtual and interdisciplinary team to works on the same problems by sharing their algorithms, data, and experiment results.

2.2.1. Web Onboarding

The Acumos is using different onboarding techniques. The first technique is onboarding by the web, which is very specific to programming languages. It works for a particular pipeline but not for the generic pipeline. For this technique, create a bundle of your model and then make the bundle and onboard it using the web interface. The second technique is dockerized onboarding model, in which one docker container is pushed. The problem with this approach is the size of the container because it stores docker images inside the acumos DB, which raises a storage problem. For example, in the audio mining pipeline size of one segmentation container is 10GB. If somebody onboard 100 containers of size 10GB, then it reaches the standard hard disk size. So here, scalability is the biggest challenge of this technique. Sometimes commercial users don't want to store the docker container outside their repository.

In our experiment platform, the onboarding dockerized model URI is used. In this

approach, an image is created and stored in another repository while doing onboarding utilizing this approach, just provide the image's reference. For the AI4EU experimentation platform, nexuses repository is used to store the docker image where the docker hub is not used due to port binding problems. This onboarding approach is very scalable.

2.2.2. Design Studio

Design Studio is a submodule of the open-source AI4EU experiment platform, running on top of a customized Acumos instance. The AI4EU experiment has a visual editor for making AI pipelines called design studio. In the design studio, AI resources expose an interface description in a protobuf format which allows the user to create a pipeline rapidly and intuitively. It provides an essential tool for AI developers to collect the building blocks for your problem. The studio offers visual feedback to connect the node with the correct block by highlighting the target connection node. This experiment platform encourages the construction of AI solutions in a more scalable, portable, and containerized microservice. Users of this platform are free to choose any programming language, toolkit, and cloud infrastructure to deploy the designed solution. Example pipeline which is created in designed studio shown in this Figure (2.5).

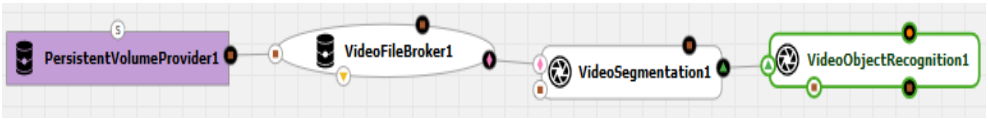

Fig. 2.5. Audio Pipeline

Design studio user interface invokes the composition engine API for the following task: It creates the machine pipelines called composite solution with the help of individual machine learning model uploaded by the open-source user community. It validates the composite solution and then generates the blueprint of the pipeline or composite solution, which the Kubernetes client uses to deploy the machine learning pipeline. Kubernetes client takes the blueprint, which is the pipeline's topology, and mapped them to artifacts such as deployment and services and container specifications. The topology of the pipeline is represented in the form of a graph. The node of the graph represents the containers, whereas the edges represent the connections of the graph.

2.2.3. Market Place

AI4EU experiment platform has its app store called Marketplace. It is used to store datasets, toolkits, and machine learning models. It provides a way to securely distribute AI microservices along with basic information on how they work. It provides the connection between model developer and application by automating user feedback, software updates.

2.3. Technology Utilization

Developing an AI4EU platform based on Micro-service oriented architecture was quite challenging because the designed platform should be language agnostic and highly interoperable. To provide more freedom to the model provider and to make it language-agnostic, we have been using gRPC API instead of REST API. Google developed gRPC API, which is used for data communication between different microservices. gRPC is built on HTTP 2 rather than HTTP 1.1 and using protobuf instead of JSON.

2.3.1. Why Docker is used for virtualization in AI4EU experiment platform instead of Virtual Machine

Virtualization is a technique of making a software-based representation of something, for example, some virtual applications, servers, sometimes virtual storage and network, etc. Cloud computing is based on virtualization, which is used to achieve elasticity of a shared resource. There are many advantages of using virtualization, such as dynamic allocation of available physical resources and multi-tenancy in which multiple resources share the same physical resource. Virtualization optimally reduces the cost of operation and also provides the scalability of applications [12].

Virtual Machines have been used in cloud computing as their core and central part. It is used to provide different cloud services to the end-user, such as infrastructure, platform, and software. Virtual machines provide a complete operating system to the user where a user can install different software to work on it [13]. Virtual machines are controlled by a manager known as a virtual machine manager (VMM) or hypervisor, and it is responsible for providing abstractions to the underlying hardware. The machine on which the hypervisor is installed is known as the host. All the other machines running independently inside the host machine are known as guest machines [14].

There are many advantages of using the virtual machine as compared to the traditional system. Many virtual machines are running on a single host, but all of them have their security as none of them is accessed from other virtual machines. In the virtual machine, all the operating systems are isolated and their application as well. It provides a way for better resource consumption and also improves the performance of the machine. Virtualization decreases the various server requirements and provides a fault-tolerant system as compared to the traditional systems. Similarly, there is some limitation of virtualization, which also needs to be considered when using virtual machine-based solutions. Generally, virtual machines communicate with each other by exchanging data, and if communication is not secure, it leads to a security threat. In a virtual machine, a single point failure will lead to the stop of the whole system. If the hypervisor is stopped working, then the entire system will stop working [14].

Similarly, the docker container is also used in the same way as the Virtual machines, but docker containers are lightweight because less time and resources are required to start

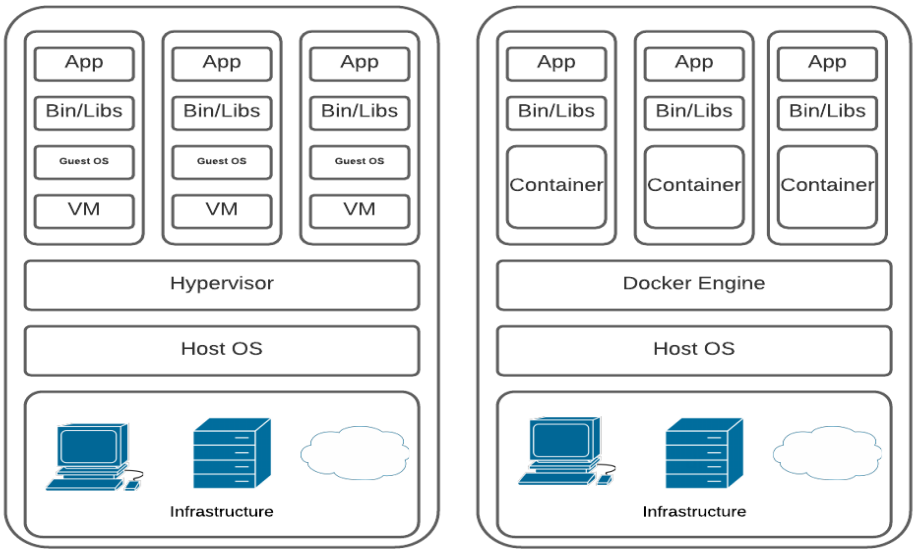

Fig. 2.6. A Comparison of Virtual Machine and Docker Container Design [14]

it, and container and host share the same kernel [15]. Docker is an open-source project used for OS-level virtualization and automation of software for the rapid construction of applications running under a container. A docker engine is used to run a docker container similar to the hypervisor of the virtual machine. Docker engine is a client-server application which consists of the following major components. The first one is a server known as a demon, and the second one is REST API, the third one command-line interface (CLI) client, etc.

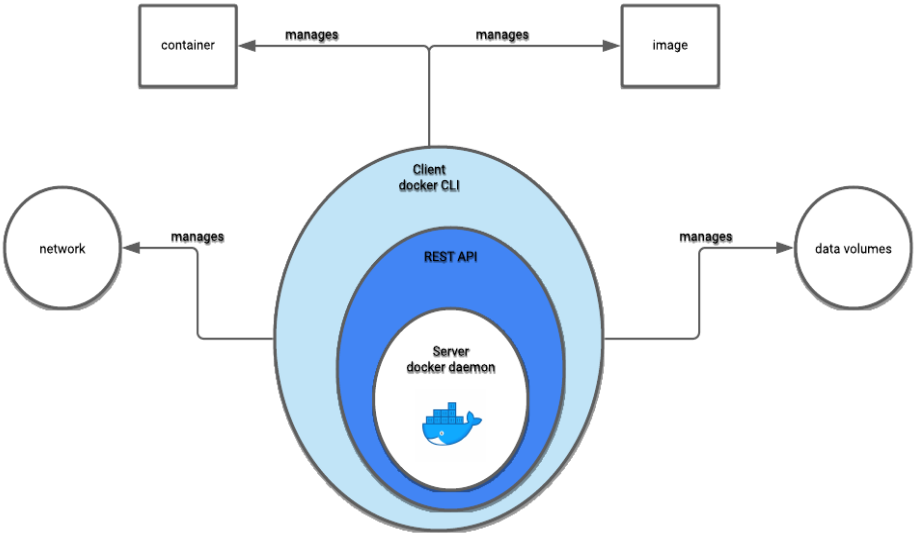

Fig. 2.7. Docker Engine

Docker is considered as a platform that is designed and builds once and used to run different applications. It provides batter portability, and interoperability [16]. Docker

container acts as a director with all the required packages used to run an application. Nowadays, operating system-level virtualization is getting very popular in the software industry. By using containers, all the isolated applications can be run on a single host operating system. Containers are present on the top layer of the host operating system: Windows, Linux, or any other. All the containers share the host operating system and other required documents used to run the application. Due to these shared read-only resources, containers are very light in weight that why a container takes few seconds to start [15].

After the invention of virtual machines, different problems related to cloud computing, for example, scheduling and resource management, have been resolved. Security of the applications is also improved by making isolated applications with the help of virtual machines. Problem related to the application management and packaging, which is not effectively addressed in virtual machines. Docker provides the solution for these problems effectively, and it provides a responsive deployment and scaling of a project. Docker container is highly scalable and safe to use, and it is a package that has ready to deploy parts of the application. Docker container is highly portable, so it does not contain its operating system. In this way, the application can be run on various platforms. Docker container is highly scalable, and it can deploy in several environments like a physical server, data server, and cloud platform. It can be easily moved from cloud environment to local environment and vice versa. It can be started and shut down within a second, and it is much faster and cost-efficient and wastes fewer resources than the virtual machines. Docker containers utilize the resources more efficiently that why more containers are deployed on a single server. Due to these reasons, we are using docker contain in the AI4EU experiment platform.

2.3.2. Docker Execution Environment

Docker provides a set of tools and a platform to manage the life-cycle of a running container. First, the user needs to develop an application and support the application's components with the help of docker containers. All the code of the application and its dependencies required to run the application are needed to be transferred into a container so that a container can work independently. After this container becomes a unit for distributing an application that can be uploaded to the docker repository, other users can download this independent project and use it for their purpose. In the AI4EU project, each composite solution pipeline consists of at least two nodes, and a container represents each node. It is challenging to deploy each node individually. Docker composed is a tool used for defining and automatically running multi-containers docker applications. In docker-compose, the YAML file is used to configure your application's services. All services which are related to an application are mentioned in YAML. The single command is to run the YAML, creating and running all the services from your configuration file. Docker-compose is used to manage the multi-container application on a single node but is not working for cluster

management.

Docker swarm was developed and maintained by Docker inc. Docker swarm is an in-build docker container orchestration tool, and it is used for cluster management. A docker swarm consists of multiple Docker hosts, and it runs in a swarm mode. Swarm acts as a manager, which is used to manage the membership and delegation, and it also acts as workers who are responsible for running the swarm services. It is usually preferred to use it for simple architecture where ten to twenty containers are present in the production environment. However, it is very easy to set up the cluster compared to Kubernetes, but cluster strength is not very strong. In the docker swarm, scaling of the application is done manually. Docker swarm allows the autoload balancing, whereas, in Kubernetes, manual configuration is required for load balancing the traffic.

Google develops Kubernetes, and now it is an open-source project with a huge developer community compared to docker swarm. It is preferably used for complex architecture where hundreds of containers are present in a production environment. It is challenging and complicated to set up the cluster, but the cluster strength is stronger than the docker swarm. Kubernetes also provides the dashboard to manage the cluster, with the help of the GUI app, which can be easily deployed and scaled.

In the AI4EU experiment platform, Kubernetes is used instead of docker swarm because it provides automatic scaling of the application based on server traffic and also a dashboard for the visual interface. In Kubernetes, automatic rollbacks are done in case of failure of the pod where the docker swarm does not provide the automatic rollbacks. In-built logging and monitoring tools are present in Kubernetes, but docker swarm uses third-party logging and monitoring.

2.4. Kubernetes

Kubernetes [17] is containerized application management system developed by Cloud Native Computing Foundation and hosted by Google open source. It is a portable, open-source container management tool that can manage the docker container and deploy it on the different servers and schedule these containers on the cluster. It is a platform for automating the deployment process and scaling the containerized application beyond the cluster. The Kubernetes cluster has a master-slave architecture, and it consists of a set of different nodes that may be physical or virtual. The master node is responsible for storing information about the various nodes, managing the cluster, monitoring the nodes, and planning which container goes. The slave node is managed by the master node and is responsible for running the containers.

2.4.1. Pods

Kubernetes has the smallest deployable unit called a pod, with one or more containers with shared storage and network resources and specifications for running the containers. Pod represents a process running on a cluster with a unique IP address, persistent volumes storage if required, and information on how to run a container. Kubernetes manages the pods but not the container which is running inside the pod. Kubernetes can also automatically generate a new replica of a pod if it fails during its execution [18].

2.4.2. Deployments

A Kubernetes deployment [19] describes how to create the instance of the pods that have the docker container application. It can scale the number of replica of the pods and roll back the updated code to the earlier development version if needed. It automates the launching method of the pod and ensures that all the instances of the pods are running across all the nodes in a cluster. The deployment controller ensures the continuity of the application by monitoring the pod's health and replacing the failed pod or bypassing the down nodes. There are different deployment strategies of the Kubernetes, such as recreate and rolling updates, etc. In the recreate deployment, the running pods are terminated, and it recreates the new version, and this strategy is commonly used in the development environment. The rolling update strategy is mapping one version of an application to the more recent version. In this version replica of the new version is launched, and it terminates the old version pods.

2.4.3. Service

Kubernetes services [20] provide a logical abstraction for the deployed pods in a cluster. Services are responsible for exposing an interface to the pods, and it enables network access within the cluster or between the external process. It also defines the policies on how to access. Service uses the selectors and labels for matching the pods which are running in the different deployments. There are different types of services.

- ClusterIP: It exposes the service which is accessible within the cluster.
- NodePort: It exposes the service is accessible with the static port of node and cluster IP.
- LoadBalancer: It exposes the service with the help of a cloud provider load balance.

2.4.4. Namespaces

Namespaces [21] provide a way to organize a cluster in a more manageable form by dividing it into virtual sub-clusters. Namespaces are very helpful when the different teams

are working on projects in a shared Kubernetes cluster. An arbitrary number of namespaces are supported in a cluster, but they are logically separated from the other and can communicate. Only a low level of resources such as node and persistent volume exist outside the namespace, but all other resources exist within the default namespace or in a user-created namespace. It provides a way to separate the development, testing, and deployment. It helps to manage the project in a separate virtual cluster without affecting the other projects.

2.4.5. Kubernetes Persistent Volumes

Persistent Volumes [22] is the part of the storage which the administrator provides as part of a cluster. It is independent of the lifecycle of the pods, meaning data inside the volume not erased even if the pod shuts down. Persistent volume claim is the request for storage by the Kubernetes node. The claim includes the specific storage parameters such as size required by the application. Pods can send a request for a specific level of the resources, e.g., CPU and memory, etc.

2.5. Pykaldi

Kaldi [23] is an open-source project for speech recognition written in C++ and available under the Apache license. Kaldi development based on finite-state transducer (FST) framework with the extensive support of linear algebra [24]. PyKaldi is a python layer over the Kaldi speech recognition system. It wraps the Kaldi code and OpenFST library so that it is easy to use and low-overhead. Pykaldi uses the low-level Kaldi functionality by writing the simple python code. Pykaldi automatic speech recognition module contains easy-to-use high-level classes and ignores the boilerplate code. In the AI4EU experiment platform, Pykaldi is used for constructing the audio pipeline. Different features are used for making the pipeline, such as segmentation of the audio file and decoding the audio segments.

3. RELATED WORK

This chapter discusses the technologies which are related to the AI4EU experiment platform. Section 3.1 explains the AI platform pipeline, which was developed by the google cloud. Later, Section 3.2 describes the Acumos and shows how the approach of the AI4EU experiment platform is better.

3.1. AI Platform Pipelines

AI platform pipelines [25] developed by google cloud to simplify the process of continuous deployment. MLOps applies the DevOps practices on the machine learning model to automate, manage, and audit the ML workflow. AI platform pipelines support the user in implementing the MLOps by providing the platform where the user can orchestrate the pipelines. In this platform, the Kubeflow pipelines are used with TensorFlow Extended to deploy and orchestrate the pipelines. Kubeflow pipeline is an open-source platform for deploying, running, managing, auditing, and monitoring the ML pipelines on the Kubernetes. TensorFlow Extended is an open-source project for orchestrating the end-to-end ML pipelines. AI platform pipelines provide you google Kubernetes cluster and cloud storage to set up a Kubeflow pipeline cluster in 15 minutes. ML pipeline is portable, scalable where each task is packed inside containers. However, this platform is not free, and users have to pay for training the models and getting the predictions.

AI4UE experiment platform is free so that scientists, researchers, and commercial users can use this platform for their purpose. Both platforms support the standard Kubernetes cluster, but the AI4EU experiment platform also supports the minikube. AI4EU experiment platform is language agnostic and supports different kinds of orchestrators. AI4EU experiment platform provides the facility the visual pipeline creation, and users can also publish their models for the AI community. In contrast, the AI platform pipeline has no design studio for the visual pipeline composition. Still, the AI4EU experiment has a design studio that provides the pipeline's topology to map the artifacts. AI4EU experiment platform defines the standard public interface in protobuf for the pipeline communication, but the AI platform pipeline has no such stander interface.

3.2. Acumos

Acumos [26] is an open-source project and part of the LF AI Foundation [27], an umbrella organization inside Linux Foundation to support open-source AI and machine learning-based projects. Acumos is a platform used to enhance the development, sharing, training, and deployment of AI models. Acumos provides a mechanism for packing, sharing, and deploying AI models in a portable form that can be published and shared in a secure

catalog. Acumos supports the different programming languages, toolkits, and run-time infrastructure for deploying AI models. In Acumos model can be onboarded and packed as containerized microservices that are interoperable with other components. Acumos consist of different components such as data broker, design studio and marketplace, etc. Data broker is used to getting the data from external sources, and this data is used for model training or tuning. Acumos design studio is used to create visual pipelines from the onboarded models, and it returns the topology of the pipelines. Acumos marketplace is an app store where onboarded models and pipelines are published, and users get feedback and rating on different models.

Acumos support the web and CLI onboarding, where the client libraries generate the model package for onboarding. It generates the model-based microservice images inside the platform, and it contains the image for the orchestrator. This bundled approach is not language-agnostic, where the model bundle is generated in a specific language. In this approach, images are created inside the platform, which is a less scalable approach. Sometimes the size of the image is quite big, and the user has to deploy the multiple pipelines of which consist of multiple nodes, then the system exceeds the size of the standard hard disk. Docker image creation inside the platform is not interactive for commercial users because sometimes they have very sensitive docker images. Acumos providing less freedom to the model providers because it is protobuf is generated from the model reflection. In AI4EU experiment platform allow the users to utilize all the feature of the protobuf. Users have to provide the protobuf definition during onboarding. Docker images are created before model onboarding and stored in some external public or private repositories.

4. METHODOLOGY

The last chapters discuss many technical details about the different components, technologies, and infrastructure that we have chosen to form our generic infrastructure for AI pipeline deployments in the local Kubernetes environment. Our goals were to design a solution that will work for all kinds of AI pipelines and developed a language-agnostic and more operable solution. To achieve our goals, we design and implement the Kubernetes client that has two different parts.

- It generates a solution.zip inside the AI4EU experiment platform.
- It adjusts the deployment templates to the local environment on the user server and applies the service and deployment definitions.

Section 4.1 describes the architecture of the Kubernetes client. Section 4.2 shows the generation of the solution.zip, whereas the Kubernetes client took the pipeline's topology as input and mapped them into deployable artifacts is explained in section 4.3. Section 4.4 shows the deployment script, which deploys the generated artifacts in the local execution environment. Orchestrator, which is integrated into the Kubernetes client, is explained in section 4.5. The last section presents the challenges that we faced during the development process.

4.1. AI4EU Experiment Platform Solution Deployment in Private Kubernetes Cluster

This section explains the support of the AI4EU experiment platform in creating deployment from the blueprint of a pipeline and then deploying the machine learning models in the private Kubernetes cluster as a simple pipeline consisting of one model and composite pipeline, which consists of multiple models. Here private Kubernetes cluster means that a virtual machine or physical machine on which the Kubernetes cluster is deployed so that the model user can use the kubectl CLI tool on the Kubernetes cluster master node to manage running applications of the cluster. Minikube builds a local Kubernetes cluster with small resources to run the small Kubernetes deployment with one single node on a local machine. In contrast, kubectl is a command-line tool used for interaction with clusters to create pods, services, and other components.

Our designed solution is currently tested on a private Kubernetes cluster but not on public or other cloud environments. AI4EU experiment platform provides a local Kubernetes cluster as a playground for experimentation. The deployment process on the local Kubernetes cluster begins after downloading the solution package from the web interface of the AI4EU experiment platform.

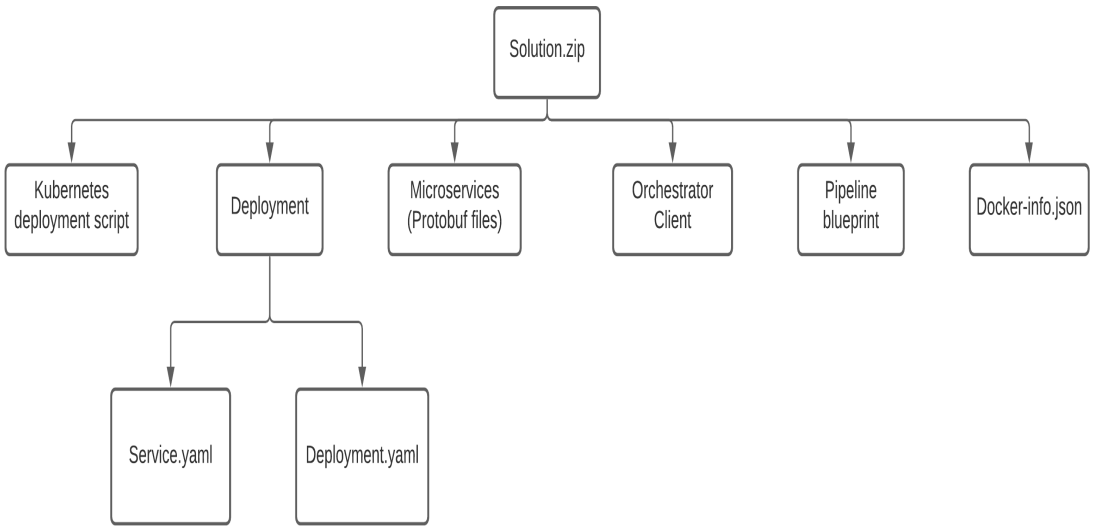

Fig. 4.1. Solution Folder

The solution folder consists of

- The deployment directory contains all the deployment and services for all the nodes of a pipeline and orchestrator.
- The microservice folder includes all protobuf specifications for all the pipeline nodes, whereas the model user provides protobuf files during model onboarding.
- The orchestration client folder contains scripts that are used for the orchestration of the AI pipeline.
- Pipeline blueprint file, produced by design studio, which includes the topology of the pipeline.
- The container specifications of the pipeline are stored in the docker-info file.
- Kubernetes client script is used for the deployment of pipelines in a local Kubernetes environment.

4.1.1. Kubernetes Client Architecture

In this section, the architectural overview of the Kubernetes client is present in different segments. In the first segment, a high-level overview of the design is shown with the help of a design diagram. It provides a generic overview of the AI4EU experiment platform and how the Kubernetes client interacts with other components. The remaining segment will give a deeper look at the internal working of the Kubernetes client. Kubernetes client establishes the connection between platform catalog and execution environment. Figure 4.2 shows its connection with the catalog.

The reader can understand the architecture of the AI4EU experiment platform from the step-by-step summary of processes. First, a user of the platform has to open the platform's web interface and then select the desired pipeline and choose "deploy to local." Here the assumption is the user already creates that pipeline and validates it.

- The user has to click on the "Download Solution Package" button for downloading the solution.zip file.
- The portal-marketplace triggers a rest API call the /getSolutionZip of the Kubernetes client service.
- Kubernetes client sends a request to common-data-service to retrieve the URL of the artifacts of the pipeline.
- Then Kubernetes client calls the maven artifact API of the nexus repository to retrieve the artifacts like protobuf files.
- Prepare a solution package containing microservices and deployments and blueprint of AI pipeline and return this package to the portal-marketplace, which downloads the solution package to the user machine.

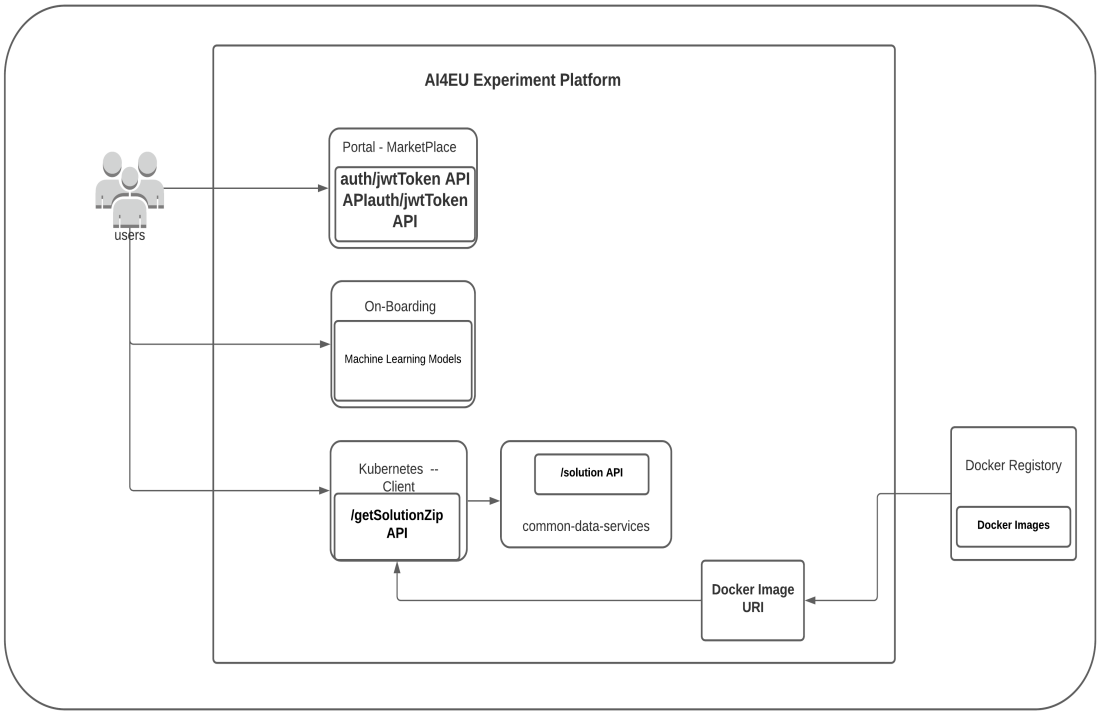

Fig. 4.2. Kubernetes Client Architecture

In figure 4.2, the docker registry is shown outside the AI4EU experiment platform, which means that docker images are stored outside. During onboarding, the URI of docker images is provided, retrieved from the docker registry during the deployment for

the pipeline. Docker registry can be public or private Kubernetes client is supporting them both.

Kubernetes client also establishes the connection with the local execution environment by generating the artifacts. These artifacts are deployed in the local execution environment with the help of the deployment script. Figure 4.3 shows the execution environment of the Kubernetes client. The user has to download the solution package and needs to start deployment in the local Kubernetes environment. For the deployment process, the user needs to unpack the solution package in their local system.

- Run the Kubernetes-client-script.py, and this script deploys the services and deployments for all the models of the AI pipeline. It also runs the service and deployment for a serial orchestrator. The user needs to provide the name of the separate namespace for each pipeline to ensure the separation of concerns and make our design more scalable. The role of namespaces to separate or isolate the pipelines from each other, increase security and exclude side effects. This script is used to deploy the solution in a local Kubernetes cluster or on mini-kube.
- After running the services and deployment for the orchestrator server, the user needs to run the Orchestrator client. The cluster IP address and port number are input parameters for the execution of the orchestrator.

4.2. Component Design of Kubernetes Client

In this section, the components of the Kubernetes client are explained here. On the call to /getSolutionZip API, the Kubernetes client performs different actions to compose the downloadable solution package. The first step is to retrieve the artifacts like blueprint.json from the Nexus repository by querying the Common Data Service (CDS) and providing the solution and revision id of the AI pipeline as a parameter. The AI pipelines are classified based on the presence of the blueprint.json file which the design studio creates. Suppose the buleprint.json is present in the repository. In that case, the solution is considered a composite solution, and the solution package contains deployments, protobuf definition, orchestrator-client, Kubernetes-client-script, and artifacts like dockerinfo.json and buleprint.json. For simple solutions, if the blueprint is not present on the nexus repository, then the solution does not contain the orchestrator-client and artifacts like blueprint and docker-info, as shown in figure 4.4. Deployment and services are generated based on information that is present inside the database. URI of the docker image and model name is required, and the model user provides this information during model onboarding.

For the composite solution, the blueprint.json containing topology of a pipeline, models represent the nodes whereas edges explain the connection of the pipeline, thus forming a graph. Metadata for each node retrieved after parsing the file once and stored inside the object. Kubernetes client only considers the information about nodes for the generation

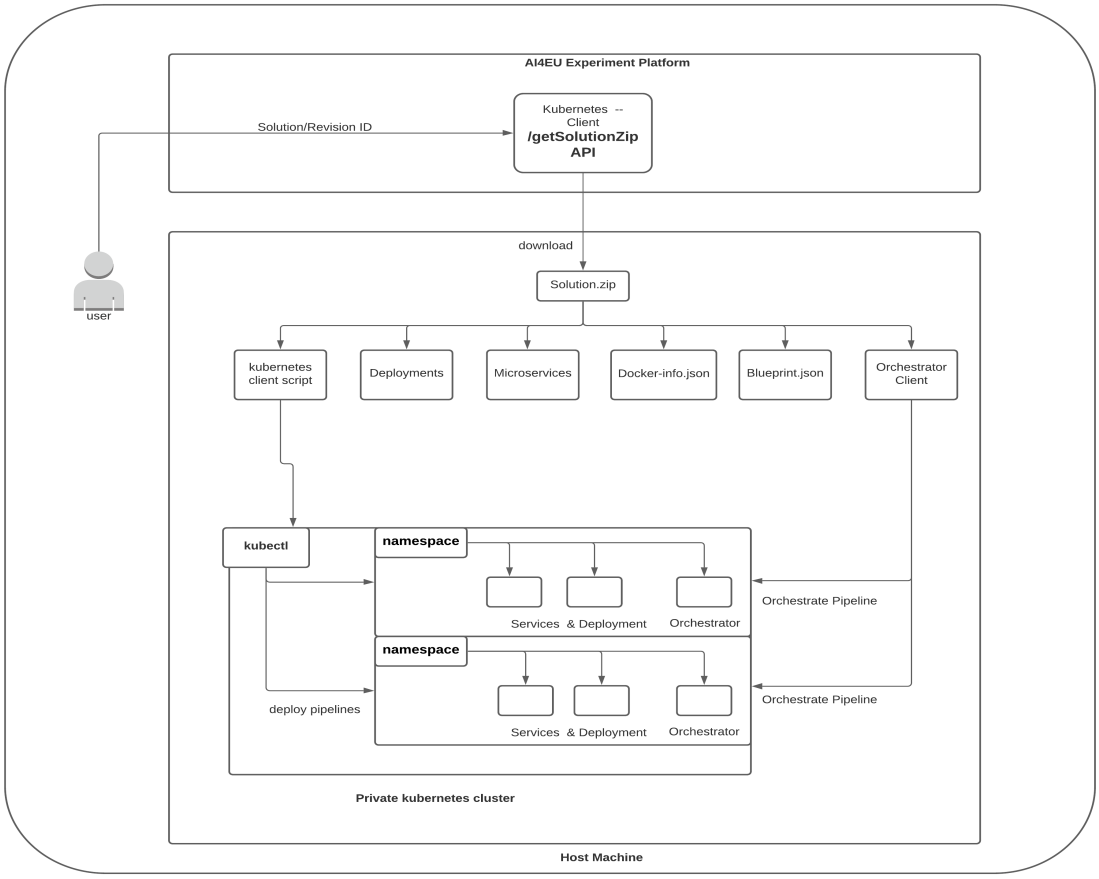

Fig. 4.3. Kubernetes Client Execution

of deployments and services. Two YAML files are generated first one is service, and the second is deployment, stored in a hashmap. The name of a file is a key of the hashmap, and its value is the content of the file. The pipeline's container specification contains the information about the nodes stored inside the docker-info.json file. These specifications are generated based on the metadata of the pipeline and contain information like node name and IP address. The orchestrator uses these specifications for the execution of the pipeline.

Kubernetes client service exposes the API of the portal marketplace to get a downloadable solution package. Kubernetes client service exposes the API of the portal marketplace to get a downloadable solution package. The URL for this API is http://<kubernetes-client-host>:<port> whereas URL to get resources is /getSolutionZip/solutionId/revisionId. SolutionId is a unique identification of a solution present in CDS, whereas revisionId presents the version of the solution present in CDS. On the successful creation solution package, the http response is 200, which means the request is successful, and the body of the response contains the solution package. The unsuccessful request response is 404, which confers the invalid solutionId and revisionId, as shown in figure 4.5.

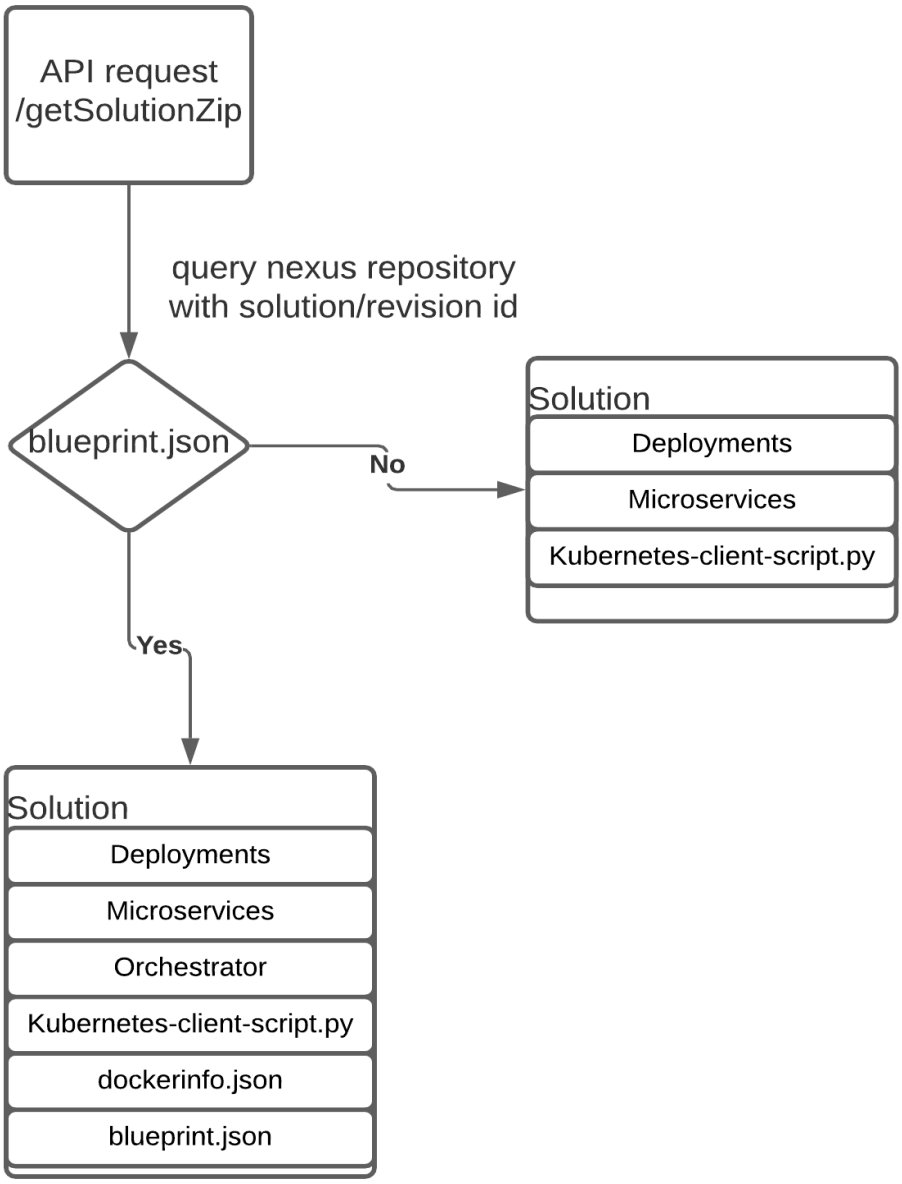

Fig. 4.4. Solution Package

4.2.1. Common Data Service

The AI4EU experiment platform Common Data Service (CDS) provides the storage and query layer between the AI4EU experiment platform component and relational database. Kubernetes client developed in Java Spring-Boot framework, the server components in Spring-Boot application provides the rest services, and it uses Hibernate to manage the persistent storage. Here Java library is a client component that includes business models and functions to utilize the REST services.

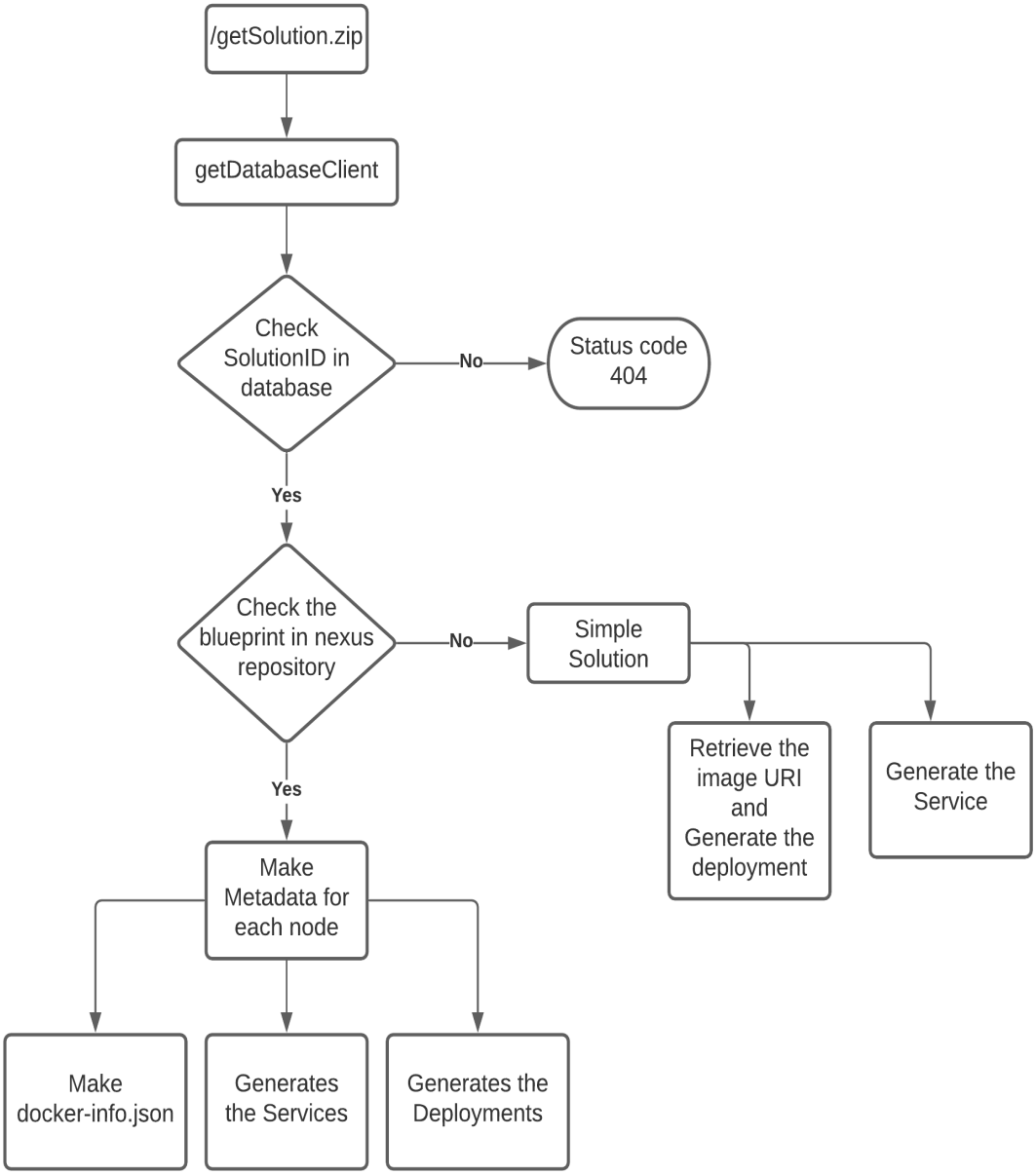

Fig. 4.5. Creation of Solution Package

4.2.2. Automatic Generation of Services and Deployments

Kubernetes client automatically generates deployments and services for each model of an AI pipeline from the logical definition of a pipeline given by the design studio. The primary role of deployment is to keep pods running, and services are used to access these set pods by enabling network access. When the network request is sent to the service in a cluster, and the service can access similar pods based on their labels, select one pod and forward it to the network request. Kubernetes has different kinds of services, and it allows you to specify the type of service you want to expose in your service definition. ClusterIP is the default type of service that opens the service on the internal cluster IP.

Only applications inside a cluster can access this service, so there is no external access. This kind of service is used for debugging your services or allowing internal traffic like displaying an internal dashboard. NodePort is a type that exposes the service on each node IP with a static freely available port on the host machine. NodePort is the most natural and cost-efficient way to get traffic to your service as it opens a specific port on all the nodes of the VMs, and any traffic sent to this port is forwarded to this service, as shown in Figure 4.6. Some disadvantages of service are that only users can bind one port to service, and users can use the port from 30000 - 32767 [28].

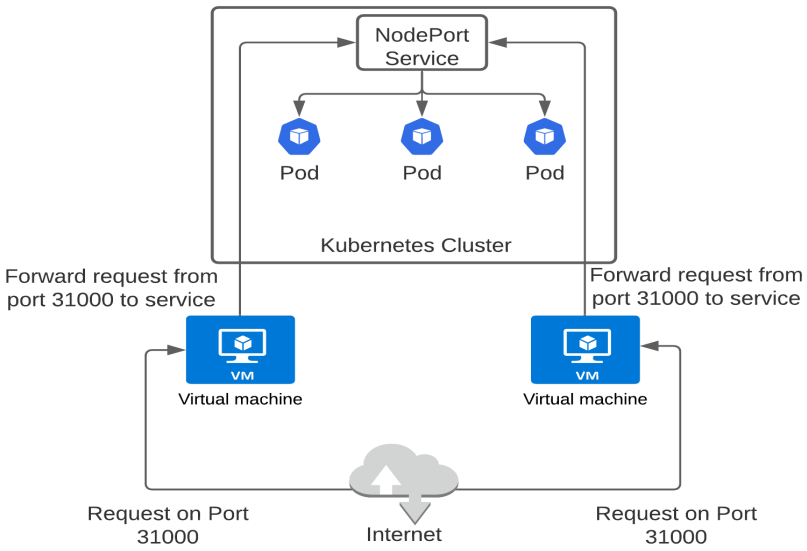

Fig. 4.6. NodePort Service

LoadBalancer service is a standard and default way to expose a service on the internet as a network load balance. It will provide you a single IP address to forward all the traffic on your service via a specific port. The disadvantage is that each service that the user exposes with a load balancer will get its IP address, and the user has to pay for each exposed service with the load balancer [28].

Kubernetes client creates services of type NodePort but without specifying the node-Port because services generated by AI4EU experiment platform without knowing any information about the host machine. NodePort is assigned directly during the deployment process on the local Kubernetes environment. Service definition is shown in CODIGO 4.1 contains one target port responsible for application communication. The application is listening to the gRPC request on this port for the service to work. In the AI4EU experiment platform, there are two standard ports, and the first port is 8061 is called protobuf-api, which Kubernetes client uses for port mapping like gRPC server communication and the second port is 8062 for Web-UI, which is used for human interaction, whereas Web-UI is optional for pipeline execution.

CÓDIGO 4.1. Sentiment Analysis Service

```yaml
---
apiVersion: v1
kind: Service
metadata:
  name: sentimentanalysis
spec:
  selector:
    app: sentimentanalysis
  type: NodePort
  ports:
  - name: protobuf-api
    port: 8556
    targetPort: 8061
---
```

Kubernetes deployments present the user a declarative update to your application because it is a resource object in Kubernetes. A deployment describes an application life cycle, such as which docker image is used by the running pod, how pods are present, and how they are updated. Kubernetes deployment is also responsible for automatically updating the application, which takes a couple of steps if operated manually. The Kubernetes backend manages deployment, and the whole update process is performed on the server-side without client interaction. In the AI4EU experiment platform, the Kubernetes client generates a deployment with one replica-set and docker image URI present in a public or private docker repository. If docker images are present in a private repository, the user needs to provide the docker registry under the specification section of the deployment definition file. Sentiment Analysis deployment is generated by Kubernetes client as shown in CODIGO 4.2, where platform standard ports are open to send and receive traffic.

CÓDIGO 4.2. Sentiment Analysis Deployment

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sentimentanalysis
  labels:
    app: sentimentanalysis
spec:
  replicas: 1
  selector:
    matchLabels:
```

```
12          app: sentimentanalysis
13      template:
14        metadata:
15          labels:
16            app: sentimentanalysis
17        spec:
18          imagePullSecrets:
19          - name: acumos-registry
20          containers:
21          - name: sentimentanalysis
22            image: cicd.ai4eu-dev.eu:7444/sentiment_analysis
                 :latest
23            ports:
24            - name: protobuf-api
25              containerPort: 8061
26            - name: webui
27              containerPort: 8062
28
29  ---
```

In the AI4EU experiment platform, everything the user onboard is considered a model, such as a data broker and other pipeline modules. A deployment is automatically created for each model in an AI pipeline, which has the same label as the service has, as shown in figure 4.7. After generation of deployment and service, Kubernetes deployment script deploys it in a target Kubernetes environment. The service and deployment are generated from the information that the user only provides during model onboarding. URI of the docker image is provided during model onboarding is concatenated with the host's address and port number. The complete address of the docker image is present in the deployment definition.

4.2.3. Docker image handling

The design of the AI4EU experiment platform is very scalable to support the hundreds of running AI pipelines. All the docker images are stored in a public or private docker repository like the docker hub. During the onboarding of the machine learning models, the model users provide the URI of the docker images. Sometimes the size of docker images is in gigabytes. If docker images are stored inside the AI4EU experiment platform, they will overflow the size of the standard hard disk. Kubernetes client gets the references of the docker images from the blueprint of the pipeline supplied by the design studio. During the deployment of the pipeline, pods are created based on these docker images.

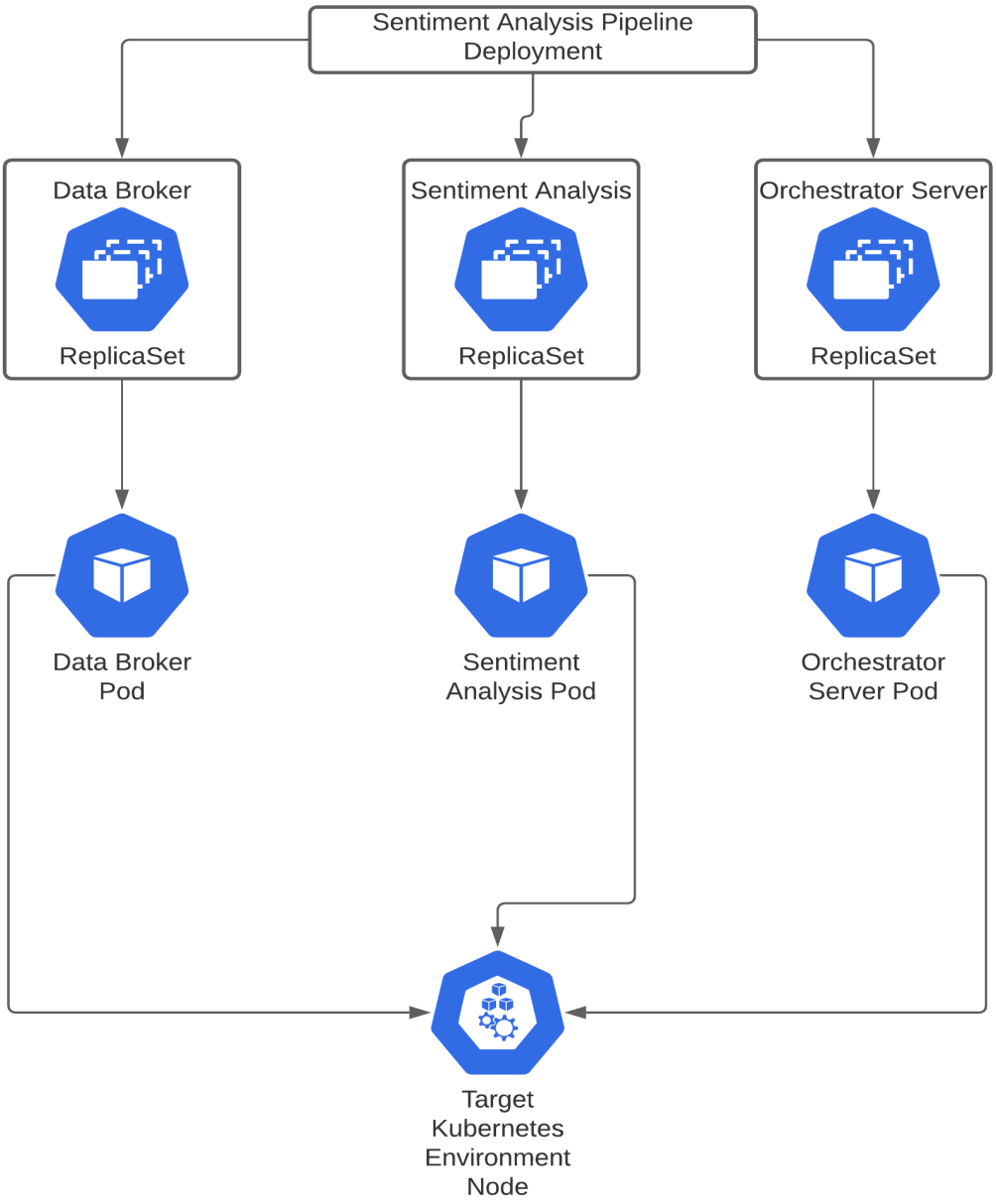

Fig. 4.7. Deployment of Sentiment Analysis

4.3. Mapping of the Pipeline Topology to the Artifacts

The main goal of the Kubernetes client is to generate the artifact from the topology of the pipeline. Topology of the pipeline consist of node and edges. Kubernetes client only considers the information about the node parameters, whereas information about edges is considered during the execution of the pipeline.

4.3.1. Pipeline Topology

The topology of the pipeline is very important because it stores the information about the input and output parameters of the nodes in the form of a graph. A design studio for composite solutions generates it. Kubernetes client uses the information of the node parameters and ignores edges. Figure 4.8 shows the topology of the sentiment analysis pipeline. In this topology, all the nodes are present under the node section, whereas each node contains the name, URI of a docker image, and URI protobuf. The operation signature list has information about the task that a specific model has to perform. Base on this information, it generated all the artifacts for the composite solution.

4.3.2. Container Specifications

It contains all the information about the docker container, and in the solution archive, the docker-info file contains this information. Kubernetes client generates it based on the topology of a pipeline, and the orchestrator uses it for the execution of a pipeline. It includes the information about the container IP address and node port during its creation. Default values are assigned, but all information is updated in this file during deployment. Figure 4.9 shows the container specification of the sentiment analysis pipeline. In this example name of the container is similar to its IP address because the DNS name is used, and it automatically resolves the cluster IP address. Here node is identical to the service name because it is easy for humans to understand.

4.4. Kubernetes Deployment Script

Manual deployment of all services and deployment is very time-consuming. It takes a lot of effort to search free port, deploy a service on it, and then update it in the docker-info file. The deployment script resolves this problem and working perfectly with different kinds of AI pipelines. Kubernetes deployment script is responsible for automatically deploy all the services and deployments of the AI pipeline, including the deployments for the orchestrator. The deployment script is developed in python programming language because it is easy to maintain and extend its functionality compared to shell script. Kubernetes client creates the deployments and services inside the AI4EU experiment platform without the information of the host Kubernetes cluster. Kubernetes client generates services of type nodePort, but the client cannot assign any port to the services because services are deployed in the host Kubernetes cluster. The deployment script is used to deploy the pipeline in a local cluster, so its first responsibility of the deployment script is to fetch free ports from 30000 to 32767 and select one port and assign it to service as nodePort. NodePort opens the service externally to the cluster as it is a port on the node where the external traffic will come on. The second responsibility of the deployment script is to save the nodePort in the docker info file. A generic serial orchestrator uses a docker info

```
{
    "name": "Sentiment_Analys",
    "version": "v1",
    "input_ports": [],
    "nodes": [
        {
            "container_name": "SentimentAnalysisDataBroker1",
            "node_type": "MLModel",
            "image": "cicd.ai4eu-dev.eu::7444/csv_databroker_sentiment_analysis:v2",
            "proto_uri": "org/acumos/6bb89915-de23-4c6c-8474-4a11c58e8ce5/sentimentanalysisdatabroker/1.0.0/sentimentanalysisdatabroker-1.0.0.proto",
            "operation_signature_list": [
                {
                    "operation_signature": {
                        "operation_name": "get_next_row",
                        "input_message_name": "Empty",
                        "output_message_name": "Features"
                    },
                    "connected_to": [
                        {
                            "container_name": "SentimentAnalysisModel1",
                            "operation_signature": {
                                "operation_name": "predict_sentiment_analysis"
                            }
                        }
                    ]
                }
            ]
        },
        {
            "container_name": "SentimentAnalysisModel1",
            "node_type": "MLModel",
            "image": "cicd.ai4eu-dev.eu:7444/sentiment_analysis:v2",
            "proto_uri": "org/acumos/efe2eeee-6038-4a06-b2d9-01f19225aad9/sentimentanalysismodel/1.0.0/sentimentanalysismodel-1.0.0.proto",
            "operation_signature_list": [
                {
                    "operation_signature": {
                        "operation_name": "predict_sentiment_analysis",
                        "input_message_name": "Features",
                        "output_message_name": "Prediction"
                    },
                    "connected_to": []
                }
            ]
        }
    ],
    "probeIndicator": [
        {
            "value": "false"
        }
    ]
}
```


Fig. 4.8. Topology of Sentiment Analysis Pipeline

file for the iteration of a pipeline. As each running node doesn't know anything about the other nodes of a pipeline, the orchestrator is responsible for transferring the information between all the pipeline nodes. This deployment script also deploys the orchestrator server for the execution of an AI pipeline. Later, the user runs an orchestrator client to trigger the execution of a pipeline.

The deployment script deploys all the AI pipelines in separate namespaces to ensure the separation of concerns. The different users on the same Kubernetes cluster can deploy different or same AI pipelines simultaneously in his namespace without modifying the

```
{
    "docker_info_list": [
        {
            "container_name": "SentimentAnalysisDataBroker1",
            "ip_address": "SentimentAnalysisDataBroker1",
            "port": "8556"
        },
        {
            "container_name": "SentimentAnalysisModel1",
            "ip_address": "SentimentAnalysisModel1",
            "port": "8556"
        },
        {
            "container_name": "orchestrator",
            "ip_address": "orchestrator",
            "port": "8061"
        }
    ]
}
```


Fig. 4.9. Container Specification of Sentiment Analysis Pipeline

other users running services and deployments. The deployment script requires the name of a namespace as an argument to deploy an AI pipeline. The deployment script creates one extra service called Web-UI to make the designed solution more interactive. Web-UI allows the user to play with the model on target port 8062.

4.5. Generic Serial Orchestrator

In the AI4EU Experiment platform, the orchestrator is responsible for the communication between all the nodes of the AI pipeline. Each node is a stand-alone model with no information about the other nodes of a pipeline, so the orchestrator is responsible for all kinds of message transfer and management of the running pipeline. Orchestrator consists of two parts the client part and the server part. The server side of the orchestrator is responsible for executing the pipeline, whereas the client-side is accountable for initiating the execution of a pipeline. Orchestrator client script takes two arguments as a parameter the node IP address and port number of running orchestrator service.

4.6. Challenges Faced During Development

We faced different kinds of challenges during the development of a Kubernetes client. Some problems are easily solved, but others take more than a week. For development, we use acumos, but the Kubernetes client is not functional there. The first challenge was to resolve all the problems and make them work in our local system. The problem was solved in more than one week after setting all the application properties according to our local environment, and then it's running. The second main challenge that I face

was the generation of the corrupted zip folder. Swagger api integrated inside Kubernetes client to see output locally, but the user interface of the swagger returns the corrupted zip file. Problem solved after generating a link to download the solution zip instead of the user interface of the api. But downloaded zip files give you all the files and directories correctly if the user unzips with the command line. The deployment script works well with the standard Kubernetes cluster, but it's causing a problem with minikube that running services are not accessible cluster IP. After a lot of research, this problem needs a node IP address to access the service. Integration of the orchestrator with Kubernetes client was also very challenging. Communication between the services was also causing problems, but the problem was solved using the orchestrator service with cluster IP and correction using target port and node port. The naming conflict was also faced during the integration of the orchestrator, and that problem was solved by setting naming rules for Kubernetes client and orchestrator. The generation of the simple solution was also consuming a lot of time.

5. EXPERIMENTS

This chapter explains the experimentation of the thesis by implementing the different AI pipelines. Section 5.1 describes a pipeline and how a user can deploy it in a local Kubernetes environment. In contrast, section 5.2 explains the construction and deployment of example pipelines such as simple AI, advanced AI, and hybrid AI pipelines.

5.1. AI Pipelines and Its Deployment

AI pipelines consist of different components it includes a data broker, trained model, and orchestrator. Deploying and managing these pipelines in an ad-hoc manner is very difficult and time-consuming. AI4EU experiment platform provides you a mechanism to deploy an AI pipeline automatically, and then the user can orchestrate the entire pipeline with the help of a generic orchestrator.

A pipeline represents a machine learning workflow, which has information about all the components and how they are connected. A pipeline includes a logical definition of each component, whereas each component is independent of other components. Each component in a pipeline is self-containing a packed code along its dependencies to run the docker image that performs the one step in a pipeline. For example, a component can provide a video broker, image recognition, image classifier, etc. All the nodes of the AI pipeline must follow the container specifications [29]. The container specification contains the rules about writing the protocol buffer and information about a container of each node. Protobuf interface meant to specify the public interface of the node and not to expose the private internal methods. In the AI4EU experiment platform, the user can generate a pipeline from the visual editor just by dragging the components and connect the node in the correct position. Each component is a machine learning model and packed inside a docker image stored on the Docker registry. The user only provides the URI of the docker image while onboarding the model. Kubernetes client generates the deployment and services for each component of the model and then deployed with the help of a generic deployment script. Each pipeline deployed in a separate namespace to avoid conflicts and to ensure the separation of the concerns. User can clean their working space just by deleting the namespace. These pipelines are reusable, portable, and can be deployed anywhere.

5.2. Example Pipelines

In this section, we test our designed solution by implementing different AI pipelines. To create pipelines, we pick one deep-learning example, split it into blocks based on the task, and then write the protobuf for each block to specify the public interface and generate the

necessary stubs and skeleton for each section for the required programming language. For each block, create a gRPC based container and push these containers into the docker registry. Then onboard the block of pipeline and connect the in the design studio. In the first subsection, we execute the sentiment analysis pipeline, whereas, in the second section, we choose the audio mining pipeline.

5.2.1. Deployment of Simple Solution

In Kubernetes client, a simple solution consisting of a single node pipeline without an orchestrator. We choose a sentiment analysis example for experimentation. First, we write a protobuf definition for the sentiment analysis node and generate necessary stubs and skeletons for gRPC communication, as shown in CODIGO 2.1. Then we implement sentiment analysis in a client-server architecture where the client sends user review as a request to the sentiment analysis server. The server sends the sentiment of review as a response to the client. After implementing this application, we pack it inside docker images and push it to the Docker repository. Docker file is shown in figure 5.1, which contains all the tools as libraries required to run this application.

After packing the application inside the docker image, we did onboarding by providing the docker URI and protobuf file and then download the simple solution. The downloaded solution consists of deployment and services generated by the Kubernetes client and a deployment script responsible for automatic deployment. AI4EU Experiment Platform has its Kubernetes cluster for deployment, and with the help of deployment script, we deploy our solution, as shown in figure 5.2.

After the deployment, the user has to provide the node port number of the running service to the sentiment analysis client script and bind it with the localhost and run it. Sentiment analysis client script retrieves the data from the attached CSV file and sends the request to the server. The sentiment analysis server sends the response back to the client, whereas the whole communication between client and server is done with the help of the gRPC protocol.

5.2.2. Deployment of Composite Solution

In the context of deployment, a composite solution consists of multiple nodes and orchestrators as well. For the composite solution, we choose an advanced AI example, the audio mining pipeline, which consists of an audio-data-broker node, segmentation node, audio-to-text node, dialogue-creator node, and orchestrator node. This audio mining pipeline is responsible for generating the text dialogue from the given audio file. We use Pykalid [30], a python layer on the Kaldi speech recognition system. First, we write the protobuf definition for all the nodes and generate the necessary stubs and skeleton. The responsibility of the audio data broker is to receive an audio file request and send a response as an audio file job. In the audio file request, the name of the working directory is provided to

```
FROM ubuntu:18.04

MAINTAINER Sajid  "sajid.naeem@iais.fraunhofer.de"

RUN apt-get update -y
RUN apt-get install -y python3-pip python3-dev
RUN pip3 install --upgrade pip
RUN pip3 install numpy
RUN pip3 install pandas
RUN pip3 install tensorflow==1.13.1

#RUN pip3 install keras
RUN pip3 install keras==2.3.1
RUN pip3 install 'h5py<3.0.0'

COPY ./requirements.txt /requirements.txt

RUN pip3 install -r requirements.txt

RUN useradd app
USER app



COPY license-1.0.0.json model.proto model_pb2.py model_pb2_grpc.py sentiment_analysis_server.py classify_review.py model.h5 train.csv test.csv ./

WORKDIR /

ENTRYPOINT [ "python3","sentiment_analysis_server.py" ]
```

Fig. 5.1. Audio Segmentation Docker Image

the data broker, which contains all the audio files. Audio file job includes the name of the audio file, the path of the working directory, its priority, etc. The protobuf definition of the audio broker is shown in CODIGO 5.1.

CÓDIGO 5.1. Audio Data Broker [2]

```
1
2   syntax = "proto3";
3
4   message AudioFileJob {
5     string job_uuid = 1;
6     int64 priority = 2;
7     string file_name = 3;
8     string work_dir = 4;
9     int64 length = 5;
10  }
```

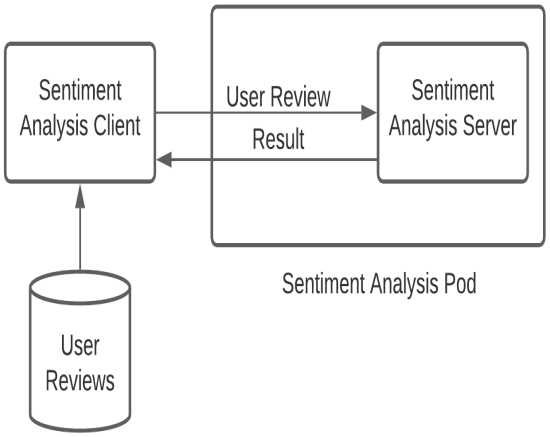

Fig. 5.2. Simple Solution of Sentiment Analysis

```
11
12  message AudioSegment {
13    string job_uuid = 1;
14    string segment_uuid = 2;
15    string segment_file = 3;
16    string work_dir = 4;
17    int64 index = 5;
18    int64 length = 6;
19    int64 start_time = 7;
20    int64 end_time = 8;
21  }
22
23
24  service AudioSegmentation {
25    rpc getNextAudioSegment(AudioFileJob) returns(AudioSegment);
26  }
```

The audio segment is the core part of this pipeline. In this part, the large-sized audio file is split into the smallest audio segment by using different approaches for making segmentation of the audio file, e.g., utterance to utterance, speaker to utterance, etc. For this pipeline, we used the utterance to utterance segmentation technique. For segmentation, we used already trained speech activity detection models (SAD) model [31]. Segmentation makes the segments of the audio file, and based on segmentation small audio file is generated. This segmentation node received an audio file job and returned the audio segment, which contains all the necessary details of the segment, and its protobuf definition is shown in CODIG 5.2.

CÓDIGO 5.2. Audio Segmentation [2]

```
1
2  syntax = "proto3";
```

```
3
4  message AudioFileJob {
5    string job_uuid = 1;
6    int64 priority = 2;
7    string file_name = 3;
8    string work_dir = 4;
9    int64 length = 5;
10 }
11
12 message AudioSegment {
13   string job_uuid = 1;
14   string segment_uuid = 2;
15   string segment_file = 3;
16   string work_dir = 4;
17   int64 index = 5;
18   int64 length = 6;
19   int64 start_time = 7;
20   int64 end_time = 8;
21 }
22
23
24 service AudioSegmentation {
25   rpc getNextAudioSegment(AudioFileJob) returns(AudioSegment);
26 }
```

Each segment is passed to the audio-to-text node, which is responsible for decoding the audio segment. We are using a pre-trained zamia model [32] for offline audio to text conversion, and based on this converted text, we generate a dialogue. After running each segment of the audio pipeline, we make a docker image of each node and pack the application and its dependencies inside the container. On-board all the pipeline nodes in the AI4EU experiment platform and create an audio pipeline in the design studio by connecting the correct nodes, as shown in figure 5.3.

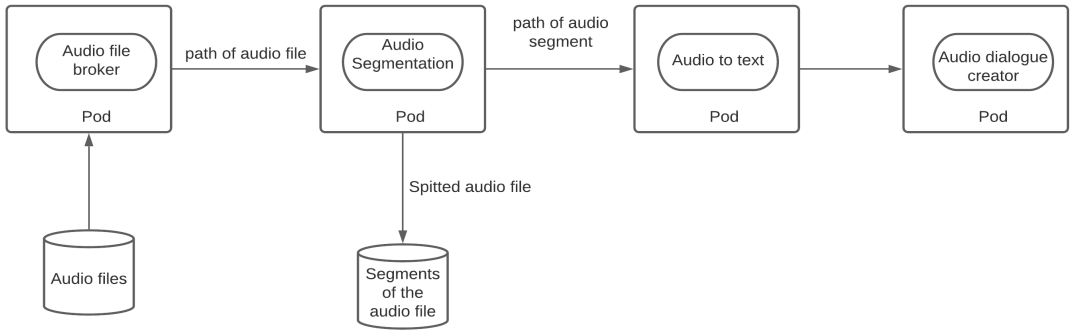

Fig. 5.3. Audio Dialogue Creator Pipeline

After saving the pipeline, we download a composite solution that contains deployments, microservices, orchestrator client, and deployment script. We run the deployment

script, which needs the namespace name as an argument for deploying all the services and running the pods. Each pod of a pipeline runs independently without knowing the internal details of other pods on the cluster. The deployment script returns the node IP address and orchestrator server port number. These values are passed to the orchestrator client to trigger the orchestration process.

In this experiment section, we test our solution on a simple and advanced AI pipeline. After successful testing on our development environment [33] now, our designed solution is integrated with our production environment [34]. We received positive feedback from the AI4EU experiment platform community. Our designed generic solution is working fine with a hybrid AI pipeline which is implemented by Dr. Peter Schüller [35].

5.3. How to make a Pipeline more Robust and Secure

AI Pipelines can be made robust by using different approaches like container optimizations which significantly reduces the container image size. It allows Kubernetes to retrieve that docker image faster and run the docker container more efficiently. An optimized container image should pack one application or perform only one task, e.g., data-broker or audio-segmentation, etc. It contains small-size docker images because it is difficult to handle the bulky docker image, and it is cumbersome for the port to manage. Users can reduce the size of the docker image by avoiding the extra layers, but it is not always helpful. Every RUN instruction in the docker file will add a new layer in the docker image that why different layers need to be merged into one docker instruction using operator. Docker caching is also very important for building docker images in a faster way. Docker builds use the previously build docker layers if there is no change in docker layers. There are some rules for caching algorithm, and if there is a change in one layer, then the cache is not loaded for all subsequent layers. These instructions are vital, especially when users are working to make large-size docker images. The audio-mining pipeline size of one container is nearly 10 GB which takes an hour to build again if caching is not used wisely. Docker file of audio segmentation is shown in figure 5.4.

AI pipelines can be made secure by reducing the kernel capabilities of the root user, only provide those capabilities required to run the application. Docker demon attack can be controlled by running the application as an app user instead of a root user. For our experimentation we are making our containers by making our app user as shown in CODIGO 5.1. Kubernetes pod can be less vulnerable to running as the non-root user, as shown in figure 5.2.

CÓDIGO 5.3. Container User

```
1 RUN useradd app
2 USER app
3 CMD ["java", "-jar", "/app.jar"]
```

```
# Dockerfile for building PyKaldi image from Ubuntu 18.04 image
FROM ubuntu:18.04

# Install necessary system packages
RUN apt-get update \
    && apt-get install -y \
    python3 \
    python3-pip \
    python2.7 \
    autoconf \
    automake \
    cmake \
    curl \
    g++ \
    git \
    graphviz \
    libatlas3-base \
    libtool \
    make \
    pkg-config \
    sox \
    subversion \
    unzip \
    wget \
    zlib1g-dev \
    vim

# Make python3 default
RUN ln -s /usr/bin/python3 /usr/bin/python \
    && ln -s /usr/bin/pip3 /usr/bin/pip

# Install necessary Python packages
RUN pip install --upgrade pip \
    numpy \
    setuptools \
    pyparsing \
    jupyter \
    ninja

# Copy pykaldi directory into the container
COPY . /pykaldi

# Install Protobuf, CLIF, Kaldi and PyKaldi
RUN cd /pykaldi/tools \
    && ./check_dependencies.sh \
    && ./install_protobuf.sh \
    && ./install_clif.sh \
    && ./install_kaldi.sh \
    && cd .. \
    && python setup.py install \
    && cd /pykaldi/examples/setups/zamia \
    && ./models.sh \
    && ./path.sh \

RUN python3 -m pip install grpcio
RUN python3 -m pip install grpcio-tools googleapis-common-protos
RUN cd /pykaldi/examples/setups/zamia && python3 -m grpc_tools.protoc -I. --python_out=. --grpc_python_out=. audio-to-text-pipeline.proto

WORKDIR /pykaldi/examples/setups/zamia/

#ENTRYPOINT [ "python3","server.py" ]
CMD [ "python3", "./audio_segmentation_server.py" ]
```

Fig. 5.4. Audio Segmentation Docker Image

```
kind: ...
apiVersion: ...
metadata:
  name: ...
spec:
  ...
  containers:
  - name: ...
    image: ....
    securityContext:
          ...
          runAsNonRoot: true
          ...
```

Fig. 5.5. Limiting the Pod Capabilities

5.4. Challenges Faced During Experimentation

Our goal was to construct AI pipelines that can be deployed in the execution environment successfully during experimentation. The construction of the simple AI pipelines is not very challenging, but the advanced AI pipelines' construction was complicated. The first challenge was the installation of pykaldi, which takes a week to work correctly. The second challenge was the building of a docker image, which takes nearly 5 to 6 hours. The size of one pykaldi image is near ten gigabytes, and we need at least three containers of that size. During the building of the docker image, it slows down the whole working environment. On the slight change, we need to build the complete docker image.

6. DISCUSSION

This section summarizes our reflection about outcomes and answers the research questions from our AI4EU experiment platform. To answer the first research question, we can establish the connection between the AI4EU experiment platform and its execution environment. For establishing the connection with the AI4EU experiment platform, we design and implement the Kubernetes client who takes the topology of the AI pipelines and maps them into the deployable artifacts. The design studio generates the topology of the pipeline by connecting the onboarded AI models. The topology of the pipeline contains the information which the user provides during the model onboarding. The user needs to give the model's name, docker image URI, and protobuf definition. In the topology of a pipeline, information about models is stored as nodes, whereas information about the connection of models is stored as edges of a graph. Kubernetes client only considers the information about nodes for the mapping of artifacts, whereas information about the edges is important for the execution of the pipeline. We mapped the pipeline's topology into different sections to address the second research question so that the pipeline was deployed and orchestrated easily. The artifacts like deployment and service can be deployed into the execution environment.

In contrast, information about the container specification is stored inside the docker-info file, which adopts the local execution environment after the deployment of the pipeline. We implement the deployment script to address the third and fourth questions, deploying all the services and deployment into the local execution environment. The deployment script solves the problem of the manual deployment of the AI pipelines. Each pipeline is deployed in a separate namespace to ensure the separation of the concerns and avoid the other pipelines' side effects. Users need to provide the name of the namespace as an argument to the deployment script. Kubernetes client has no information about local execution environment, so deployment script fetch the free ports and bind it with service and deploy it on standard Kubernetes cluster. The deployment script is generic enough that work for all kinds of AI pipelines. For experiment and evaluation, we implement a simple and advanced AI pipeline. We implement the sentiment analysis pipeline and audio mining pipelines which consist of different nodes such as audio data broker, audio segmentation, audio to text, and dialogue creation. Kubernetes client creates the artifacts for both pipelines, and the generic deployment script deploys all the artifacts on a standard Kubernetes cluster.

Another objective of the thesis is to integrate the orchestrator with Kubernetes client, and this is achieved. Kubernetes client creates the deployment artifact for the orchestrator server and provides an orchestrator client for triggering the execution of the pipeline. The deployment script deploys two kinds of services first one is the protobuf api for the pipeline interaction, and the second one is the web-ui for the human interaction, and this

service is optional.

AI pipelines consist of a different number of nodes that are connected. Each node, the model, is packed inside the docker container with no information about the other nodes of the pipelines. Protobuf interfaces describe the only public interface of the nodes, whereas all other internal methods are hidden. gRPC is used for communication between the nodes. It provides the mechanism for communication and serialization based on the protocol buffers. All the nodes in a pipeline are independently working, whereas communication is performed with the help of the orchestrator. Protocol buffer and gRPC make the platform language agnostic as it supports the different programming languages and toolkits. Users can write the AI model in any programming language and then generate gRPC stubs and skeletons based on the protocol specifications.

Existing acumos uses a bundle formate approach where docker image is created inside the platform, which means it is less scalable. In the AI4EU experiment, platform images are stored outside the platform in a public or private registry. Researchers and commercial users can also store their docker images in a private registry. In Acumos, the user creates a bundle, zip them in a folder, and then on-board. In the AI4EU experiment platform, the user has to write the protobuf specification and then generate necessary stubs and skeleton for the gRPC communications. Acumos is not using the full power of Protobuf because it is generated inside the platform, whereas in the AI4EU experiment platform, it is provided by the user.

We face different challenges during development, port mapping between the service to support the gRPC based communication. We face different challenges in building the advanced AI pipeline for experimentation and evaluating the designed solution. Installing the pykaldi for audio mining and building the docker images for the audio pipeline is complex. During the development of the Kubernetes client, we faced different problems, such as integrating the orchestrator inside the Kubernetes client.

We have designed a solution which works well for all kind of AI pipeline deployment. Design a generic Kubernetes client who is working on unseen problems. Our designed approach is not only limited to deep learning models. It supports any AI tool from any AI area like reasoning, semantic web, symbolic AI, etc. Kubernetes client integrated into the production environment of the AI4EU experiment platform and used by the AI4EU partners and associated projects like ELG, AI4Copernicus. AI4Copernicus a project about artificial intelligence for earth observation. Artificial intelligence is used to quickly and efficiently analyze the time series of the earth observation. Some scientists working on robotic software are also inspired by our product and want to use the AI4EU experiment platform to develop robotic software using Robot Operating System (ROS). AI4EU experiment platform is used for the testing of High-Performance Computing (HPC). For HPC, Singularity containers are required to run the containerized application on HPC. AI4EU experiment platform made the docker container a singularity container by making the app user inside the docker file. Singularity limits the abilities of the non-privileged

user to escalate from its permissions.

7. CONCLUSION AND FUTURE WORK

The thesis research aims to design and implement the deployment client who takes the topology of the AI pipeline and maps them into deployable artifacts. It provides an opportunity for the researcher, scientist, and commercial user to onboard the AI models and make deployment and execute in their local execution environments. The designed solution was implemented and tested using different technologies and frameworks such as spring boot for deployment client, Kubernetes to execute docker containers and gRPC and Protocol buffers for communication.

Kubernetes client was designed and implemented to address the research questions that establish the link between the catalog and its execution environment. It took the topology of the AI pipeline and mapped them into the deployable artifacts. Kubernetes client has the generic deployment script, which will deploy all the deployment artifacts by adopting the local execution environment. The deployment script is integrated inside the Kubernetes client and packed along the downloadable solution.zip. Kubernetes client is integrated into the production environment of the AI4EU experiment platform and is working fine on different kinds of simple, advance, and hybrid AI pipelines.

For experimentation and evaluation of the thesis, we implemented a simple and composite solution. The simple solution consists of one node model, whereas the composite solution consists of an AI pipeline generated in the design studio. We have implemented a sentiment analysis pipeline for a simple solution, whereas, for a composite solution, we have to implement an audio mining pipeline, which consists of different nodes like an audio broker, audio segmentation, audio to text, and audio dialogue creator node. The Kubernetes client successfully mapped the AI pipeline topology into deployable artifacts that were automatically deployed by the generic deployment script. Kubernetes client provides supports to different kinds of orchestrators by producing container specification artifacts. These specifications are updated by deployment script according to the local execution environment.

This thesis solves the problem of automatic deployment of all kinds of AI pipelines, but the process can be improved using persistent volume. In this approach, a shared space is attached to the pipeline, and each node can use the data from the shared space.

This thesis solves the problem of automatic deployment of all kinds of AI pipelines, but the process can be improved using persistent volume. In this approach, a shared space is attached to the pipeline, and each node can use the data from the shared space. In the future AI4EU experiment platform can be made more secure by increasing the security at an external and internal level. For the external level, the docker container can be made safe by trust signature verification. We need to develop a mechanism for a docker container verification where the container can only run the signed images. It can also be made secured by reducing the Linux kernel capabilities of the root user because the user does

not need these capabilities such as:

- It denies all mount operations and also prohibits the raw socket from preventing packet spoofing.
- It restricts access to the file system operations, for example, altering the owner of the file.

We also need to make our deployment cluster more secure at the internal level of the AI4EU experiment platform by limiting the container's resources. In contrast, CPU requests and limits are associated with the containers. By resource and limit mechanism, we can save our system from DOS attacks. We also need to develop a strategy to check how many resources are required to run an AI pipeline so that automatic deployment cannot be stoped in the middle.

BIBLIOGRAPHY

[1] *Sentiment Analysis Protobuf definition*, https://github.com/ai4eu/tutorials/tree/master/Sentiment_Analysis, Accessed: 2021-04-29.

[2] *Audio Mining Protobuf definition*, https://github.com/ai4eu/interfaces/tree/master/audio-pipeline, Accessed: 2021-04-29.

[3] D. Box. (). "A brief history of soap," [Online]. Available: https://www.xml.com/pub/a/ws/2001/04/04/soap.html. (accessed: 4.3.2021).

[4] C. Pennington, J. Cardoso, J. Miller, R. Patterson, and I. Vasquez, "Introduction to web services," in. Jan. 2007. DOI: 10.4018/978-1-59904-045-5.ch007.

[5] S. Kim and S.-Y. Han, "Performance comparison of dcom, corba and web service.," Jan. 2006, pp. 106–112.

[6] H. Wang, J. Z. Huang, Y. Qu, and J. Xie, "Web services: Problems and future directions," *Journal of Web Semantics*, vol. 1, no. 3, pp. 309–320, 2004. DOI: https://doi.org/10.1016/j.websem.2004.02.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1570826804000058.

[7] S. Islam, R. Kumar, and A. Dar, "A comprehensive study on web services basics," Jul. 2018.

[8] Haishan Tian, Yuanjun He, and Hongming Cai, "A vr web service for active scene using x-vrml," in *IEEE International Symposium on Communications and Information Technology, 2005. ISCIT 2005.*, vol. 1, 2005, pp. 405–408. DOI: 10.1109/ISCIT.2005.1566879.

[9] R. Wu *et al.*, "A new workflow to interact with and visualize big data for web applications," in *2016 International Conference on Collaboration Technologies and Systems (CTS)*, 2016, pp. 302–309. DOI: 10.1109/CTS.2016.0063.

[10] S. Islam, R. Kumar, and A. Dar, "A comprehensive study on web services basics," Jul. 2018.

[11] F. Halili and E. Ramadani, "Web services: A comparison of soap and rest services," *Modern Applied Science*, vol. 12, p. 175, Feb. 2018. DOI: 10.5539/mas.v12n3p175.

[12] R. Ranjan, "The cloud interoperability challenge," *IEEE Cloud Computing*, vol. 1, no. 2, pp. 20–24, 2014. DOI: 10.1109/MCC.2014.41.

[13] R. P. Goldberg, "Survey of virtual machine research," *Computer*, vol. 7, no. 6, pp. 34–45, 1974. DOI: 10.1109/MC.1974.6323581.

[14] A. K. Yadav, M. L. Garg, and Ritika, "Docker containers versus virtual machine-based virtualization," in *Emerging Technologies in Data Mining and Information Security*, A. Abraham, P. Dutta, J. K. Mandal, A. Bhattacharya, and S. Dutta, Eds., Singapore: Springer Singapore, 2019, pp. 141–150.

[15] M. J. Scheepers, "Virtualization and containerization of application infrastructure : A comparison," 2014.

[16] R. Ranjan, "The cloud interoperability challenge," *IEEE Cloud Computing*, vol. 1, no. 2, pp. 20–24, 2014. DOI: 10.1109/MCC.2014.41.

[17] *Kubernetes*, https://kubernetes.io/, Accessed: 2021-04-03.

[18] *Pods*, https://kubernetes.io/docs/concepts/workloads/pods/, Accessed: 2021-04-15.

[19] *Deployments*, https://kubernetes.io/docs/concepts/workloads/controllers/deployment/, Accessed: 2021-04-14.

[20] *Service*, https://kubernetes.io/docs/concepts/services-networking/service/, Accessed: 2021-04-13.

[21] *Namespace*, https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/, Accessed: 2021-05-8.

[22] *Persistent Volume*, https://kubernetes.io/docs/concepts/storage/persistent-volumes/, Accessed: 2021-02-20.

[23] *Kaldi*, https://kaldi-asr.org/doc/index.html, Accessed: 2021-03-6.

[24] D. Povey *et al.*, "The kaldi speech recognition toolkit," in *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*, IEEE Catalog No.: CFP11SRW-USB, Hilton Waikoloa Village, Big Island, Hawaii, US: IEEE Signal Processing Society, Dec. 2011.

[25] *AI Pipelines*, https://cloud.google.com/ai-platform/pipelines/docs, Accessed: 2021-04-10.

[26] *Acumos*, https://www.acumos.org, Accessed: 2021-05-6.

[27] *LF AI Foundation*, https://lfaidata.foundation/, Accessed: 2021-05-9.

[28] *Service*, https://kubernetes.io/docs/concepts/services-networking/service/, Accessed: 2021-03-21.

[29] *Container Specification*, https://github.com/ai4eu/tutorials/blob/master/Container_Specification/ai4eu_container_specification.pdf, Accessed: 2021-05-16.

[30] *Pykaldi*, https://github.com/pykaldi/pykaldi, Accessed: 2021-04-29.

[31] *Speech Activity Detection Models*, https://kaldi-asr.org/models/m4, Accessed: 2021-04-29.

[32] *Zamia Speech*, https://github.com/gooofy/zamia-speech#zamia-speech, Accessed: 2021-04-29.

[33] *Development environment of AI4EU experiment platform*, https://acumos-dev-fhg.ai4eu.eu/, Accessed: 2021-04-29.

[34] *Production environment of AI4EU experiment platform*, https://acumos-int-fhg.ai4eu.eu/, Accessed: 2021-04-29.

[35] *Hybride AI: Sudoku Example*, https://github.com/peschue/ai4eu-sudoku/, Accessed: 2021-04-29.