Research Project - DataCloud

Christian Bauer

March 24, 2023

Contents

Listings

1 Introduction

In this introductory section, I will elaborate on the different sections that follow in project semester work. This project has been undertaken as a requirement for the completion of my master's degree. The topic I have chosen to focus on is creating a REST server (see 3) for the DataCloud (see 2) research project and more specifically the ADA-PIPE component (see 2.1). The topic has relevance to current trends and issues such as using RESTful services (see 3.1) to interact with numerous DataCloud components in a distributed infrastructure. These RESTful services are built based on a predefined API (see 3.2). The communication between those DataCloud components is required to be handled in a secure manner, for which the tool KeyCloak (see 3.4) is used. KeyCloak enables the DataCloud components to interact only with authenticated resources or users. Testing the ADA-PIPE REST server is done once by ensuring the availability of the service (see 3.6) and also the security of the server (see 3.6). To ensure that the REST server is extendable and maintainable, multiple code quality measures (see 6) are being considered. Finally, future work is mentioned that is not included in the current state of the project (see 7).

2 DataCloud

Nowadays, the amount of data that is generated and collected is bigger than ever before. Big Data leads to new challenges involving processing and managing this massive amount of information and turning it into valuable insights. The DataCloud toolbox offers a comprehensible solution for discovering, simulating, deploying and adapting Big Data pipelines. The toolbox is designed to also be executed on infrastructures with heterogeneous and untrusted resources.

> DataCloud delivers a toolbox of new languages, methods, infrastructures, and prototypes for discovering, simulating, deploying, and adapting Big Data pipelines on heterogeneous and untrusted resources. DataCloud separates the design from the run-time aspects of Big Data pipeline deployment, empowering domain experts to take an active part in their definitions [1].

The separation of the design from the run-time aspects of deployment, DataCloud empowers its users to create efficient and effective Big Data solutions. The architecture of DataCloud is shown in figure 1.

2.1 DataCloud Pipeline Components

The following sections describe the different DataCloud components that are part of the toolbox. Each of the components is also seen in figure 1.

DIS-PIPE

DIS-PIPE (Discover pipeline) [3] uses a set of process mining techniques and AI algorithms to analyse and learn the structure of Big Data pipelines. This is done by extracting, processing and interpreting the vast amount of event data that is collected by multiple heterogeneous data sources. Additionally, the DIS-PIPE module includes a variety of analytics techniques that help to visualise the discovered pipelines together with diagnostics information regarding their execution.

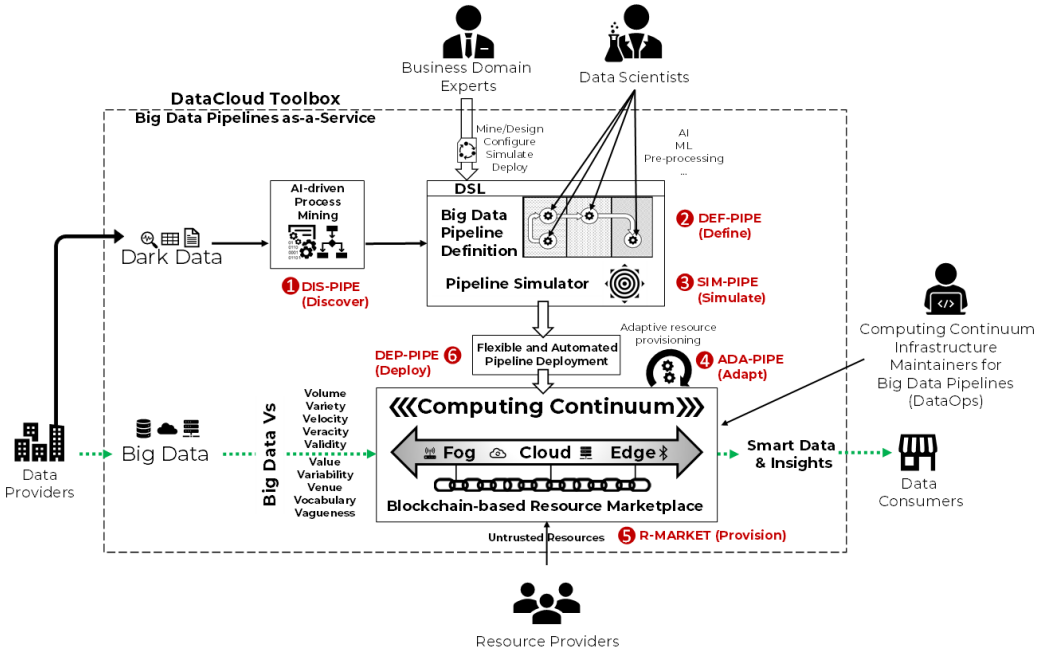

Figure 1: DataCloud Toolbox Overview [2]

DEF-PIPE

DEF-PIPE (Definition pipeline) [4] uses domain-specific language (DSL) [5] to provide a visual design for the implementation of Big Data pipelines. This includes support for storing and loading the pipeline definitions and displaying them in a UI. For this, the pipeline structure is declared by domain experts. Furthermore, it is designed to allow Data Scientists to define the content by providing the configuration of each pipeline step.

SIM-PIPE

The SIM-PIPE (Simulation pipeline) [6] module generates and simulates a deployment configuration that takes hardware requirements into account and also includes additional middleware information that is required. The generated result is then used for the final deployment. The module also includes a sandbox that enables the evaluation of individual pipeline steps to analyse the performance as well as a simulator for determining the performance of the Big Data pipeline.

ADA-PIPE

ADA-PIPE (Adaptation pipeline) [7] is the module that we are working on. ADA-PIPE provides a data-aware algorithm for smart and adaptable provisioning of resources and services. The tool allows resource reconfiguration for improved computational performance and interoperability by monitoring and analysis of diverse resources.

The monitoring of resources and the mapped pipeline tasks allow analysing of the gathered data to be able to make more accurate predictions in the future based on machine learning algorithms that are specifically made for the prediction of time-series or sequential data.

R-MARKET

R-MARKET (Resource-MARKET) [8] deploys a decentralized resource network that is based on a hybrid of permissioned and permissionless blockchain. This blockchain federates a set of heterogeneous resources from multiple providers spread across the Computing Continuum. R-MARKET provides a marketplace for resources at every service layer that is managed in a trustful democratic manner. The marketplace enables the provisioning of resources over multiple domains for external use.

DEP-PIPE

DEP-PIPE (Deploy-PIPE) [9] provides elastic and scalable deployment and orchestration of Big Data pipelines while addressing run-time aspects. The module also features real-time event detection and automated decision-making for automated deployment and orchestration.

2.2 Required REST Interfaces for Collaboration

2.2.1 Pipelines

Endpoint `/pipelines`

REST Methods `GET, POST`

Description The pipelines endpoint is used to get a JSON file that contains a list of all pipelines that are currently running in the DataCloud system with the `GET` method. To upload a new pipeline and its tasks, the `POST` method can be used. The process after posting the pipeline is described in section 3.5.2. Once a pipeline is successfully uploaded using the `POST` method, it can also be retrieved with the `GET` method as long as the pipeline is running inside the system.

2.2.2 Requirements

Endpoint `/requirements/<pipeline_id>`

REST Methods `GET`

Description This endpoint returns the requirements of each job of the pipeline that is specified with its `pipeline_id`. The requirements include parameters such as the maximum resource limits of a task that are not to be exceeded or the docker image and its location. Also included in the requirements are the dependencies of each task to tasks that have to be executed beforehand.

2.2.3 Resources

Endpoint `/resources`

REST Methods `GET`

Description This endpoint returns a list of all resource providers and their worker pools to the requesting client. The worker pools contain metadata, such as their name, the name of each worker, the number of workers and their resource capabilities. The resource capabilities include values such as the number of virtual CPU cores, and the memory and storage capacity.

2.2.4 Schedules

Endpoint `/schedules/<pipeline_id>`

REST Methods `GET`

Description This endpoint returns a schedule of tasks of a pipeline that corresponds to the identifier `pipeline_id`. The tasks are returned in a list that is ordered based on the order they should be deployed on the resources. The tasks contain metadata such as the resource they need to be deployed on, the resource limits that are not to be exceeded for the task as well as the adapted resource utilisation that was predicted via machine learning.

3 Server

This section describes the used third-party tools and concepts used to implement the server of ADA-PIPE. It was agreed upon by all developers to use REST for the communication between the different pipeline components of DataCloud with specific interfaces (see 3.2) to send messages.

3.1 REST

REST [10] (short for Representational State Transfer) was created for building web services and provides a set of constraints and properties that are used to make web services *RESTful*. RESTful web services are built to be scalable, flexible and maintainable. REST is based on the HTTP protocol and relies on client-server communication, where a client first sends a request and the server sends back a response. The response contains a representation of the requested resource, which is JSON for the ADA-PIPE module. The communication between ADA-PIPE and other modules uses Application Programming Interfaces in other to forward or retrieve data.

3.2 Application Programming Interface

An Application Programming Interface (API) [10] is a set of protocols and tools for building software and applications. APIs are used to provide endpoints for different software systems (i.e. applications, services, etc.) to communicate and exchange data with each other. The communication and interaction of different software components are defined via an API and they also allow these interactions to take place over the internet. The required API endpoints to cooperate with other DataCloud pipelines are defined in section Required REST Interfaces for Collaboration.

3.3 Flask

Flask [11] is a Python framework for creating web applications. The main benefit of Flask compared to other web application frameworks for Python like Django is that it is easy and quick to get started, yet scaling up to complex web applications is easily possible and most important a lightweight framework. Flask is often used for small to medium-sized projects and is known for its simplicity and ease of use.

In ADA-PIPE, Flask is used to implement and expose the REST server to other components (see 1) of DataCloud via an Application Programming Interface.

3.4 KeyCloak Authentication

KeyCloak [12] is an open-source identity and access management tool. DataCloud has decided to use this tool to ensure authenticated communication between all services as well as its users. In DataCloud, the possibly sensible data requires to be handled securely, making KeyCloak a feasible tool for user federation and access control. KeyCloak is based on strong standard protocols and provides support for many common communication protocols such as OpenID Connect, OAuth 2.0 and SAML. Another major feature of KeyCloak is that it can be used for many popular programming languages such as Java, JavaScript and in our case Python. The KeyCloak library implementation of each language can be used with every other KeyCloak implementation via REST calls and given the distributed working environment and usage of different programming languages depending on the expertise of each team, KeyCloak is a suitable choice for providing authentication and authorization of users and services.

In DataCloud, the responsibility of providing authentication tokens is done by a KeyCloak server instance. In order to be able to communicate with other DataCloud microservices, ADA-PIPE first has to retrieve a KeyCloak authentication token from the server instance. This authentication token is similar to a session key often found for web services and is only valid for a predefined time frame. This authentication token must be sent with every request to other microservices and every microservice inside of DataCloud is required to verify the validity of the sent token.

In ADA-PIPE, before acquiring the authentication token for our service, an instance of the *KeycloakOpenID* has to be created with the proper credentials, such as *server URL, client id, realm name* and *client secret key*. Since especially the client's secret key has to be concealed, a function is used to load the key into memory so that it can not be easily read. This KeycloakOpenID instance is created as is shown in listing 1.

```
def _get_keycloak_open_id() -> KeycloakOpenID:
    """
    Creates a KeyCloak ID instance that is required to authenticate
    our own service to other services and also to authenticate
    other services.

    Returns:
        KeycloakOpenID: Instance of a KeyCloak ID
    """
    return KeycloakOpenID(
        server_url='https://datacloud-auth.euprojects.net/auth/',
        client_id=__get_client_id(),
        realm_name='user-authentication',
        client_secret_key=__get_secret_key()
    )
```

Listing 1: Retrieve KeyCloak OpenID Instance

After successfully creating the KeycloakOpenID instance, the authentication token for ADA-PIPE can be retrieved from the KeyCloak server. This is done by providing the username and password that is used for ADA-PIPE to the method `_get_keycloak_token(...)` that can be seen in listing 2. This method uses the KeycloakOpenID to send the provided credentials to the Keycloak server, and if those are valid, then the server sends back the keycloak token.

```
def _get_keycloak_token(username: str, password: str) -> dict:
    """Get the assigned KeyCloak token for the ADA-PIPE backend to be able to send
     requests to other DataCloud services.

    Returns:
        dict: the keycloak token for ADA-PIPE
```

```
    """
    if username is None or len(username) == 0:
        raise KeycloakAuthenticationError('The provided username is either None or
     empty')
    if password is None or len(password) == 0:
        raise KeycloakAuthenticationError('The provided password is either None or
     empty')

    token = __keycloak_open_id.token(
        username,
        password)
    return token
```

Listing 2: Get ADA-PIPE KeyCloak Token

After the successful request of acquiring a keycloak token, we can send this keycloak token with each of our requests to other DataCloud services. This is done to authenticate the messages that are sent to other microservices and since each microservice has a unique token that is linked to it, the token is verifiable by a method such as the code in listing 3, that is used in ADA-PIPE to authenticate incoming messages. This is done by first checking the HTTP status code. If the HTTP status code is `201 OK`, then the token is valid. Additionally, the response body is checked for the fields `email_verified` and `preferred_username`. Either of these must be included in the response for the message to be verified by our service. Otherwise we either get a `4xx` HTTP error status code or reject the message if the required fields are missing in the response.

```
def verify_keycloak_token(keycloak_token: dict) -> bool:
    """Verify a given KeyCloak token

    Args:
        keycloak_token (dict): a KeyCloak token to be verified

    Returns:
        bool:  returns True if the token could be verified by KeyCloak, else
    returns False
    """
    if keycloak_token is None or len(keycloak_token) == 0:
        return False
    if 'access_token' not in keycloak_token:
        return False
    try:
        keycloak_response = __keycloak_open_id.userinfo(
            keycloak_token['access_token'])

        if 'email_verified' not in keycloak_response or 'preferred_username' not
    in keycloak_response:
            return False

        return True

    except KeycloakAuthenticationError as err:
        # Token could not be verified
        print('Invalid Access Token', err)
        return False
```

Listing 3: KeyCloak Token Verification

3.5 Processing of Incoming Data

In this section, the processing of the incoming data will be discussed. The incoming data in this scenario is a Big Data task pipeline that should be mapped and then deployed to the resources

in the Computing Continuum. The Big Data task pipeline is defined in a custom DSL [13] that is sent from DEF-PIPE to ADA-PIPE and is then preprocessed before analysing and scheduling the tasks.

3.5.1 Requirement File Structure

As mentioned, the incoming file structure is a custom DSL. It is structured similarly to a JSON file with a few distinctions, but the key-value pattern and the nested value sections are similar. At the root level, the keyword `Pipeline` is always followed by the name of the pipeline. At the next nesting level, the keywords are as follows:

- `communicationMedium`, which denotes what type of service the pipeline tasks are, for example, that they consist of a set of web service tasks.
- `environmentParameters`, which is a set of parameters that are required to interact with the environment such as Kubernetes and other services.
- `steps`, which is the task pipeline itself that consists of all the tasks. This also includes the dependencies of tasks to other tasks and is usually shown by the order in the tasks mentioned in the pipeline as well as a nested keyword `previous` that denotes all services that have to be running before the task this keyword resides in. Each step also describes the hardware requirements and how scalable a task has to be.

3.5.2 Requirement File Preprocessing

Since the incoming data is provided in a DSL scheme, and the next DataCloud module DEP-PIPE is responsible for the deployment of the tasks onto resources expects the file format in a JSON scheme, we parse the incoming data and convert the file as a JSON file. For this, we receive the data via REST and extract the message body and parse it. After parsing the JSON message body, the extracted data is stored in Python objects of the classes `JobDataContainer` and `PipelineDataContainer`. The `PipelineDataContainer` class is used to store the data of the entire incoming pipeline, its structure, order and its metadata. It holds the jobs/tasks as `JobDataContainer` objects that each holds the metadata and properties of the jobs. The jobs in the pipeline are stored in a list, that is sorted ascending based on the dependencies upon other tasks.

3.6 Testing the Server

In this section, two approaches for testing the REST server are described. First, the REST functionality of the server is tested with the tool Insomnia. Next, the overall network functionality and security are tested with the tool Nmap.

Insomnia

For testing the server REST responses, we use the tool Insomnia [14]. Insomnia is a popular cross-platform REST (Representational State Transfer) client for testing and debugging REST APIs. It allows developers to easily send HTTP requests and view responses, making it an indispensable tool for working with RESTful APIs. Some of the key features of Insomnia include support for multiple request methods (such as GET, POST, PUT, etc.). Insomnia also provides real-time request and response previews, as well as the ability to validate and format JSON data.

As is already stated, Insomnia stores the requests and is also capable of validating JSON data, and we used this functionality to test the responses of the server for correctness.

Insomnia additionally provides functionality to unit test [15] a REST application. These unit tests are used to test if the REST server responds with the expected message and values.

```
1 const response = await insomnia.send();
2 const body = JSON.parse(response.data);
3 const item = body[0];
4
5 expect(body).to.be.an('array');
6 expect(item).to.be.an('object');
7 expect(item).to.have.property('id');
```

Listing 4: Insomnia Unit Test Example

In listing 4, an example unit test can be seen, that waits for a response from a REST server (line 1). Next, the JSON payload body is parsed (line 2) and an item value is extracted (line 3). These values are then tested for the expected data types in lines 5-7.

Nmap

Network Mapper (Nmap) is a popular open-source tool for network discovery and security auditing. It is often used for tasks such as network inventory, monitoring hosts or service uptime as well as looking for vulnerabilities in network components.

> Nmap uses raw IP packets in novel ways to determine what hosts are available on the network, what services (application name and version) those hosts are offering, what operating systems (and OS versions) they are running, what type of packet filters/firewalls are in use, ... [16]

In ADA-PIPE we use Nmap to test if the network component that provides our REST services (see section 3.1) is reachable and also only exposes the necessary ports to do so. Nmap also proves a service to run scripts that look for common vulnerabilities. If a possible vulnerability is found by a script, it gets listed with the corresponding CVE number. A found vulnerability in the system was found (see listing 5) by Nmap using the command: `$ sudo nmap -sV -script=vuln <ip_number>`. To enable the scripts that look for vulnerabilities, the flag `-script=vuln` needs to be included. As stated in the Nmap report, the port number `8011` is vulnerable to a *directory traversal* attack in the *phpMyAdmin* [17] service. The details of the vulnerability can be found with its identifier `CVE-2005-3299` and also by the references provided by the script.

```
1 8011/tcp  open     http           Gunicorn 20.0.4
2 | http-phpmyadmin-dir-traversal:
3 |   VULNERABLE:
4 |   phpMyAdmin grab_globals.lib.php subform Parameter Traversal Local File
      Inclusion
5 |     State: LIKELY VULNERABLE
6 |     IDs:  CVE:CVE-2005-3299
7 |       PHP file inclusion vulnerability in grab_globals.lib.php in phpMyAdmin
      2.6.4 and 2.6.4-pl1 allows remote attackers to include local files via the
      $__redirect parameter, possibly involving the subform array.
8 |
9 |     Disclosure date: 2005-10-nil
10|     Extra information:
11|       ../../../../../etc/passwd not found.
12|
13|     References:
14|       https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3299
```

```
15 |_      http://www.exploit-db.com/exploits/1244/
```

Listing 5: Nmap Vulnerability Output Example

4 Monitoring

In this section, the approach and tools used to monitor the system are described. First, the monitoring tool Netdata is described that is used to monitor the computing resources of the infrastructure. Next, the tool Prometheus is mentioned, which is used to scrape monitoring data of all resources that have Netdata installed.

4.1 Netdata

Netdata [18] is an open-source tool that collects real-time metrics, including CPU usage, disk activity, bandwidth usage and furthermore. The reason we chose Netdata for monitoring is that it is a lightweight tool mostly written in C, Python and Javascript and it requires minimal resources, which is necessary when monitoring edge devices. One of its major features is that it runs without interrupting any of the applications running on the same device. This is achieved by only using idle CPU cycles while running.

Netdata provides an in-browser dashboard to analyse each metric in real-time with help of visual representation. As an example, a screenshot was taken of a cloud resource in our system as can be seen in 2.

Figure 2: Netdata Dashboard Example

Netdata has a vast amount of support for other tools in order to gather data. One of the supported tools is Prometheus, which is used to scrape the monitored data from all resources that have Netdata installed.

4.2 Prometheus

Prometheus [19] is an open-source application used for monitoring and alerting events and is designed to run across various platforms in a scalable and easily deployable manner. Same as Netdata it records in real-time, and stores the gathered metrics in a time series database by using a *HTTP pull model*. It also allows real-time alerting via a rule-defining configuration and also has a flexible query language called *PromQL*, that enables the retrieval and processing of the gathered data. Prometheus has great integration with other tools such as Netdata. In the project, it is used inside a monitoring master node that is continuously scraping all resources that have Netdata installed for monitoring data. In case a predefined rule is broken at a monitored resource, such as CPU over-allocation, Prometheus triggers an event that notifies ADA-PIPE in order to be able to take measurements.

5 Project Planning

The internal project planning was done with Skype Business [20] and Trello [21]. Skype is used as a video communication tool to discuss the current progress of the project and the future steps to take. Trello is a project management tool that uses boards and cards to organize tasks among all members. We organized our Trello board similar to a Kanban board and assigned the tasks according to the previous discussions on Skype.

For project steps that are important for other stakeholders of DataCloud, the ADA-PIPE GitHub project board is used similarly to the Trello board. This is done to ensure others see the current tasks of the team members as well as the ones that are already done or are in the backlog. Also, they can add tasks to the project board, such as bugs to be fixed or missing API functionality.

6 Code Quality

6.1 Code Review

Code reviews are an important part of software development and have several key tasks:

- **Improving code quality:** Code reviews help ensure that the code is well-written, follows best practices and standards, and is free of bugs or potential security issues.
- **Sharing knowledge:** Code reviews are a way for team members to learn from each other and share their expertise and knowledge.
- **Enhancing team collaboration:** Code reviews encourage collaboration and teamwork among team members, helping to foster a positive and inclusive work environment.
- **Maintaining consistency:** Code reviews help ensure that code is consistent and follows the project's style guide and coding standards.
- **Enhancing documentation:** Code reviews can help identify areas where documentation needs to be improved and encourage developers to add more detail and clarity to their code.
- **Finding and fixing bugs:** Code reviews can help identify and fix bugs before they make it into production, reducing the need for bug fixing in the future.

- **Improving performance:** Code reviews can help identify areas where code performance can be improved, helping to ensure that the application is running optimally.
- **Enhancing maintainability:** Code reviews can help ensure that the code is maintainable and easy to understand, making it easier to make changes and upgrades in the future.

Therefore, a code review of each commit by a collaborator is a healthy process to make sure that each member of the team is aware of the key components of a module. This is ensured by using *Pull Requests* on GitHub, where the project resides. In this, a (squashed) commit is only merged into the development branch after at least one other team member has reviewed the code and deemed it acceptable to be included in the project. This process is done to ensure that coding conventions are kept consistent[1] with the existing code base, to ensure an overview understanding of the code base of all members and to quickly find code sections that need improvement or could lead to an erroneous behaviour.

6.2 Code Refactoring

Code refactoring of pre-existing code was necessary in order to be able to use the components with other modules of the project. The reason for this is that many functionalities were written in Python scripts that were not written to be included with other modules at that stage.

> Code refactoring is defined as the process of restructuring computer code without changing or adding to its external behavior and functionality [22].

Therefore, the functionality of the scripts was analysed and then put in separate methods. Hard-coded sections of the scripts were put into variables or abstracted were possible. Since most of the hardcoded sections were specific file paths that were manually changed depending on which functionality was tested, those hardcoded sections were exported to configuration YAML files to be able to add further configuration files to the project.

6.3 Type Hinting and Linting

Python supports *type hinting*[2] since version 3.5. This enables type annotation for functions, variables and other components. Note that type hinting is not enforced and can only be used to declare a type of a code section, but even then, it is not checked nor analysed by the Python runtime. Yet, correctly using type hinting enhances the code readability and even makes the method documentation section of each parameter and its type unnecessary in trivial cases. Also, it enables other users to easily use the methods and classes since all its parameters and fields are annotated with the corresponding type.

As an example of the usage of type hinting in the project, the `PipelineState` class constructor is shown in the following listing:

```
class PipelineState():

    def __init__(
                self,
                initial_pipeline_state: Dict[str, PipelineDataContainer] = None,
                add_dummy_data: bool = False
                ) -> None:
        if initial_pipeline_state is None:
```

[1] For this project, the Python coding convention PEP8 is used and enforced. See `https://peps.python.org/pep-0008/`

[2] https://docs.python.org/3/library/typing.html

```
9               initial_pipeline_state = {}
10          self.__pipelines: Dict[str, PipelineDataContainer] =
        initial_pipeline_state
11
12          if add_dummy_data is True:
13              self.__init_with_dummy_data()
```

Listing 6: PipelineState Constructor

Type hints for variables and parameters are denoted with the syntax `<variable_name> ":" <type>`. There is also the variant of `<method_signature()> "->" <type>`, that denotes the return type of a method. The return type of the function in listing 6 is `None`, meaning it does not return a value.

A major benefit of using type hinting not mentioned above is that *Linters* use the provided type hints to enable type checks before runtime. In short, a linter is a tool to improve the written code. Linters diagnose the code and warn about technical issues, inconsistent coding style, security and performance issues. This helps to omit using the wrong data types even in an interpreted programming language such as Python in many cases as we get a warning if a type is used that was not hinted at by us. What makes linter tools a valuable addition to the used tool set, and in the project, the linter *PyLance*[3] is used.

7 Future Work

In this section, the future work of sections mentioned that are planned to be done in the ADAPIPE project but were not done thus far, or only to some degree.

7.1 Extensive Documentation

Documentation is available to some extent in the current version of the project, yet lacks documentation for many sections, such as an overview of the overall system, and the installation guide for new users. For this, we will use GitHub wiki functionality that enables us to provide documentation for the project next to the GitHub repository.

7.2 Swagger

Swagger is an open-source software framework used for designing, building, and documenting RESTful APIs. It provides functionality to describe the structure of APIs, including the inputs and outputs. Additionally, it generates interactive documentation and enables APIs to be more readable and accessible, making it easier to understand how to interact with an API. The use of Swagger has become a standard for documenting and describing RESTful APIs, making it easier for developers to build and consume these APIs. While Swagger is already in use, it only covers parts of the REST API and project. It is planned to further increase the usage of Swagger to document the API endpoints.

7.3 Server Load Testing

Server load testing is the process of evaluating the performance of a server under a simulated heavy load. This simulation involves a high number of users or requests to the server so the performance under stress can be analysed. Server load testing has the purpose of identifying

[3] https://marketplace.visualstudio.com/items?itemName=ms-python.vscode-pylance

performance bottlenecks or limitations before deploying the server to a production environment. The results of server load testing can be used to optimize the server's configuration, identify and fix any scalability issues, and improve the overall performance of the server.

References

[1] R. Dumitru, "About the Project — Project — DataCloud Project." https://datacloudproject.eu/project/about-the-project.

[2] R. Dumitru, "DataCloud Toolbox." https://datacloud-project.github.io/toolbox/.

[3] S. Agostinelli, J. Rossi, A. Marrella, and D. Benvenuti, "DIS-PIPE." DataCloud, Feb. 2023. https://github.com/DataCloud-project/DIS-PIPE.

[4] V. Mitrovic, A. Layegh, B. Elvesæter, and S. Tahmasebi, "DEF-PIPE Frontend." DataCloud, Feb. 2022. https://github.com/DataCloud-project/DEF-PIPE-Frontend.

[5] JetBrains, "What are Domain-Specific Languages (DSL) — MPS by JetBrains." https://www.jetbrains.com/mps/concepts/domain-specific-languages/.

[6] N. Nikolov, A. Pultier, A. Thomas, and B. Elvesæter, "SIM-PIPE." https://github.com/DataCloud-project/SIM-PIPE.

[7] N. Mehran and C. Bauer, "ADA-PIPE." DataCloud, Feb. 2023. https://github.com/DataCloud-project/ADA-PIPE.

[8] S. Sengupta and A. Djari, "R-MARKET." https://github.com/DataCloud-project/R-MARKET.

[9] G. Ledakis, I. Plakas, B. Elvesæter, and K. Theodosiou, "DEP-PIPE." https://github.com/DataCloud-project/DEP-PIPE-Pipeline-Deployment-Controller.

[10] RedHat, "What is a REST API?." https://www.redhat.com/en/topics/api/what-is-a-rest-api.

[11] "Welcome to Flask — Flask Documentation (2.2.x)." https://flask.palletsprojects.com/en/2.2.x/.

[12] KeyCloak, "Documentation - Keycloak." https://www.keycloak.org/documentation.

[13] N. Mehran and C. Bauer, "DSL-DEF-PIPE Example." DataCloud, Feb. 2023. https://github.com/DataCloud-project/ADA-PIPE/blob/04ed7959883afbf5ba536a5688d7e61e30ff4be4/ImportFrom-DEF-PIPE/tellu.dsl.

[14] K. Inc., "Introduction to Insomnia." https://docs.insomnia.rest/insomnia/get-started.

[15] K. Inc., "Unit Testing — Insomnia Docs." https://docs.insomnia.rest/insomnia/unit-testing.

[16] "Nmap: The Network Mapper - Free Security Scanner." https://nmap.org/.

[17] phpMyAdmin, "Bringing MySQL to the web." https://www.phpmyadmin.net/.

[18] Netdata, "Getting started — Learn Netdata." https://learn.netdata.cloud/docs/getting-started/, Mar. 2023.

[19] Prometheus, "Overview — Prometheus." https://prometheus.io/docs/introduction/overview/.

[20] Microsoft, "Skype — Stay connected with free video calls worldwide." https://www.skype.com/en/.

[21] Atlassian, "Manage Your Team's Projects From Anywhere — Trello." https://trello.com/.

[22] S. Watts and C. Kidd, "What is Code Refactoring? How Refactoring Resolves Technical Debt." https://www.bmc.com/blogs/code-refactoring-explained/, Mar. 2018.