Impact Analysis and Ranking System for Microservices

A Project Report

Submitted by

Rajeshwar R. Rathi 111603073

in partial fulfillment for the award of the degree

of

B.Tech Computer Engineering

Under the guidance of

Prof. S. N. Ghosh

College of Engineering, Pune

Mr. Hrishikesh Date

Veritas LLC

DEPARTMENT OF COMPUTER ENGINEERING

AND

INFORMATION TECHNOLOGY,

COLLEGE OF ENGINEERING, PUNE-5

May, 2020

DEPARTMENT OF COMPUTER ENGINEERING

AND

INFORMATION TECHNOLOGY,

COLLEGE OF ENGINEERING, PUNE

CERTIFICATE

Certified that this project, titled "IMPACT ANALYSIS AND RANKING SYSTEM FOR MICROSERVOCES" has been successfully completed by

Rajeshwar R. Rathi 111603073

and is approved for the partial fulfillment of the requirements for the degree of "B.Tech. Computer Engineering/Information Technology".

SIGNATURE
Prof. S. N. Ghosh
Project Guide
Department of Computer Engineering and Information Technology, College of Engineering Pune, Shivajinagar, Pune - 5.

SIGNATURE
Dr. Vahida Attar
Head
Department of Computer Engineering and Information Technology, College of Engineering Pune, Shivajinagar, Pune - 5.

Abstract

Abstract – Microservices are based on the principle of "single responsibility principle" which states "gather together those things that change for the same reason, and separate those things that change for different reasons." In Microservices Architecture(MSA), some types of applications are broken down into smaller, composable pieces which then become easier to build and maintain. Each of these services is responsible for discrete tasks and can communicate with other services through simple APIs to solve a larger complex business problem. But communication becomes complex when the services are in large numbers and they call each other for proper functioning i.e. some of these services depend on other services to run. So while developing microservices, this dependency relationship needs to be checked. So, in this paper we are proposing a way of solving the issue of complex dependencies by creating Microservices dependency graph which will enable developers to visually check these dependencies. Using dependency graph, the microservices can be analysed and validated if a code change happens in any of the microservice. This will help validating a service on which other services are dependent and rank the microservices according to their "confidence score". This will help organizations make business decisions for a product release or product update.

Keywords - Microservices, dependency graph, testing, confidence score, ranking system

Contents

List of Figures

Chapter 1

Introduction

1.1 Microservices

The Monolithic architecture consists of front-end (User Interface), logic component and database, whereas the Microservices Architecture (MSA) consists of the cloud based architecture broken down into components where each component can continuously developed and separately maintained and application is the sum of these components. Monolithic architecture is hard to maintain as there is a lack of modularization in the architecture. MSA provides many advantages over monolithic architecture such as developer independence, isolation and resilience, scalability, life cycle automation and relationship to business. The common definition of microservices generally relies upon each microservice providing an API endpoint, often but not always a stateless REST API which can be accessed over HTTP(S) just like a standard web page. The API endpoint can be exposed using Open API (Swagger). This process for accessing microservices make them easy for developers to consume as they only require tools and methods many developers are already familiar with. So, in MSA some services produce while other services consume and this whole makes a complete application. But, when the services are in large numbers and the inter communication between each

other for functioning, managing the services becomes complex.

1.2 Impact Analysis and Ranking System

Suppose a developer does some changes in a particular microservice, this can impact the functioning of other microservices dependent on this service and some services might fail. So, we are proposing a way of solving this issue by developing Impact Analysis and Ranking System (IARS) for microservices. The IARS will work by generating and visualizing dependency graph for complete system and particular microservice. This graph will help to validate the changes done in particular service using component automation. And using Ranking System for microservices, the business organization can make timely decisions for a particular microservice such as if a bug is encountered in a service, the "confidence score" of that microservice will help organization to resolve the bug before product release or resolve in upcoming patch.

Chapter 2

Literature Review

2.1 Microservices Architecture

The Microservices Architecture(MSA) corresponds to the approach of Y-axis of Scale Cube. Y-axis of Scale cube tackles scaling of the product, by functional decomposition that is scale by splitting an application into different multiple services based on their functionalities. The entire application is not deployed as one rather, each component or function is treated as a service which can be developed independently. Analogically, Microservices are much similar to Human Organ System, where each organ has different function and each organ create a product which consumed by another organ directly or indirectly. Every organ works independent of irrespective of knowing what other organs' functionalities, all an organ takes care is the product created by other organ. In similar manner, the one developer developing a service didn't need to know the working of other service. In MSA, services produce and consume through exposing the API endpoints to each other. The microservices provide flexibility of developing a service without knowing technology stack of other services. Microservices provides better scalabilty over monolithic application, as it gives freedom to scale the only services rather than the whole application. The one of the major problem, the MSA suffers is

from intercommunication becomes complex over time as more services are added to application.

2.2 Advantages of Microservices

Following I have discussed the advantages of Microservices:

2.2.1 Isolation

If any of the service in the microservice fails, then developer can use any other service which is backup to failed service. So, the application will continue to work. The larger part of the application remains unaffected even if one of the service fails. Even if a feature is added in any service, all other services doesn't need to be changed. In this way, MSA provides better fault isolation and resilience than monolithic architecture.

2.2.2 Scalability

If some services of application are needed to scale at a time rather than the whole application, MSA provides such freedom so that, only needed services can be scale instead of entire application. This leads to cost saving for scaling up the application.

2.2.3 Flexibility

The MSA approach led developer to create new services in any technology stack and can use any framework to implement it. The working of one service implemented using any language does not impact others. So, this eliminates the need of learning the complete technology stack for all the services, the developers can develop a new service irrespective of what framework other

services are using for implementation. In this way, microservices provide flexibility.

2.2.4 Productivity

As when it comes to understand the microservices, the one only have to take care of the response generated from one service and response needed from another service, this makes understanding the working of application easy, which leads to increase in the productivity. To expand the application that is to add more services, the developer doesn't need to understand the working of all services, hence he can directly start the development of service without going through all other services working. This enhances productivity which is not the case in monolithic architecture.

2.2.5 Faster Deployment and Maintainability

Microservices provide developers the benefit of Continuous deployment, as the entire application is broken down into smaller components which leads to smaller code bases, hence resulting in faster deployment and easy to maintain. The developers can independently add any feature and test the service, in this way microservices are easy to maintain.

2.3 Drawbacks of Microservices

- **Intercommunication between service becomes complex**: Over the time as more services are added into application, the services will request responses with each other and this becomes complex in nature. These requests are needed to handle carefully.

- **Debugging turns difficult**: As there are many number of services, so to maintain logs for all services becomes difficult over time and going through all the logs is cumbersome process.

- **Need of More resources**: As services increase in the application, handling multiple databases and transactions increases, which requires lot of work.

- **Global Testing**: As in the monolithic architecture application can be tested directly, but before testing an application developed using microservices globally, all the services are needed to be checked, if the response generated by each service is appropriate.

Chapter 3

Problem Statement

In Microservices Architecture, each component can be separately developed and maintained and can be easily scaled. But to run the whole application, these microservices consume the output generated by a microservice as an input, hence it requires a intercommunication. This intercommunication remains simple when number of services are less,but over a time when more services are being added to system, this intercommunication becomes complex and if a particular service fails to run due to any of the reasons whether a bug is encountered or a new feature is getting added, then it directly or indirectly "impacts" other services dependent on that service. At the time of testing, all the microservices are needed to be tested, which can be time consuming. And also, at the time of product release, the organization needs to make a decision whether to fix the service depending on the level of severity of bug or to fix the service by giving patch in next update. So, to tackle this issue, we have developed the graph-based Impact Analysis and Ranking System (IARS) for Microservices which will generate the dependency graph for particular services and rank the services based on the confidence score generated for each service.

Chapter 4

System Requirement Specification

4.1 Overall Description

4.1.1 Product Perspective

The application will be used for component automation (testing and validation) as well for ranking the microservices depending on their success and failure rate of build.

4.1.2 User Class

The IARS for microservices will be used by software (microservices) developers, quality assurance team and testing team at Veritas LLC.

4.1.3 Operating Environment

The application will be running on cloud in a container with Linux installed on headless server having enough storage and processing power to clone all the repositories (microservices) and run the application.

4.1.4 Design and Implementation Constraints

As the application will be running on cloud, every person will need a access to Virtual Machine installed on cloud with the python packages installed.

This is done by CI/CD pipeline. As the Virtual Machine is headless server, the graph cannot be visualized on the server. For visualizing purpose, a python script is needed to run on local machine with dot file generated by server. The application cannot be generalized as it is specific to company. Microservices need to be developed either in Spring Boot framework or logs should be generated for each process triggered by User Interface.

4.2 Software Requirements

- Access to Virtual Machine on cloud.
- Jenkins and Apache Groovy Script for CI/CD pipeline
- Python 3.6+
- Python Modules installed on cloud as well on local machine:
  - PyYAML==5.3
  - graphviz==0.13.2
  - pygraphviz==1.3.1
  - plotly==4.5.0
  - networkx==2.3
  - psutil==5.6.3
  - argparse==1.4.0
  - requests==2.22.0
  - numpy==1.17.2
  - pydot==1.4.1
  - UpSetPlot==0.4.1

- YAML and log files of particular services or of whole system
- npm and orca installed on local machine
- Internet Browser installed on local machine

4.3 Hardware Requirements

- A headless server with minimum 32 GB storage and 8 GB RAM
- Local computer with minimum 32 GB storage and 8 GB RAM

4.4 Functional Requirements

- Microservices should be implemented in Spring Boot Framework where a configuration file should be maintained.
- Even if microservices are implemented using different platform, but a configuration file such as YAML file should be maintained which consists of all the API endpoints.
- For dependency graphs to be generated from the logs, proper logging should be embedded in the service, so that when it runs, appropriate logs are generated.

4.5 Non-Functional Requirements

- High speed and reliable internet connection will be required for setting up the environment for running the project.
- Incorrect data should not be entered into the configuration file, as the results yielded will be inappropriate.

Chapter 5

System Design and Architecture

5.1 Modules

Following are the modules of the application:

- Validation of repositories list
- Extracting the dependency relationship between services using YAML file
- Extracting the dependency relationship between services using logs
- Validating the dependencies in the code
- create data for generating confidence scores
- Generating confidence scores

5.2 Components

There are three major components of the project:

- Dependency Graphs Generation
- Component Automation
- Generating Confidence Scores

5.3 Interfaces

- The input that is the list of repositories to the application will be given through Jenkins server through CI/CD pipeline and output that is dot file will also be accessed as Jenkins artifact.
- The dot file will be given input to a python script and interactive graph will be generated on the local machine.
- Python script for generating the dependency graph from logs will access the logs on local machine while result will also be generated on local machine

5.4 Architecture

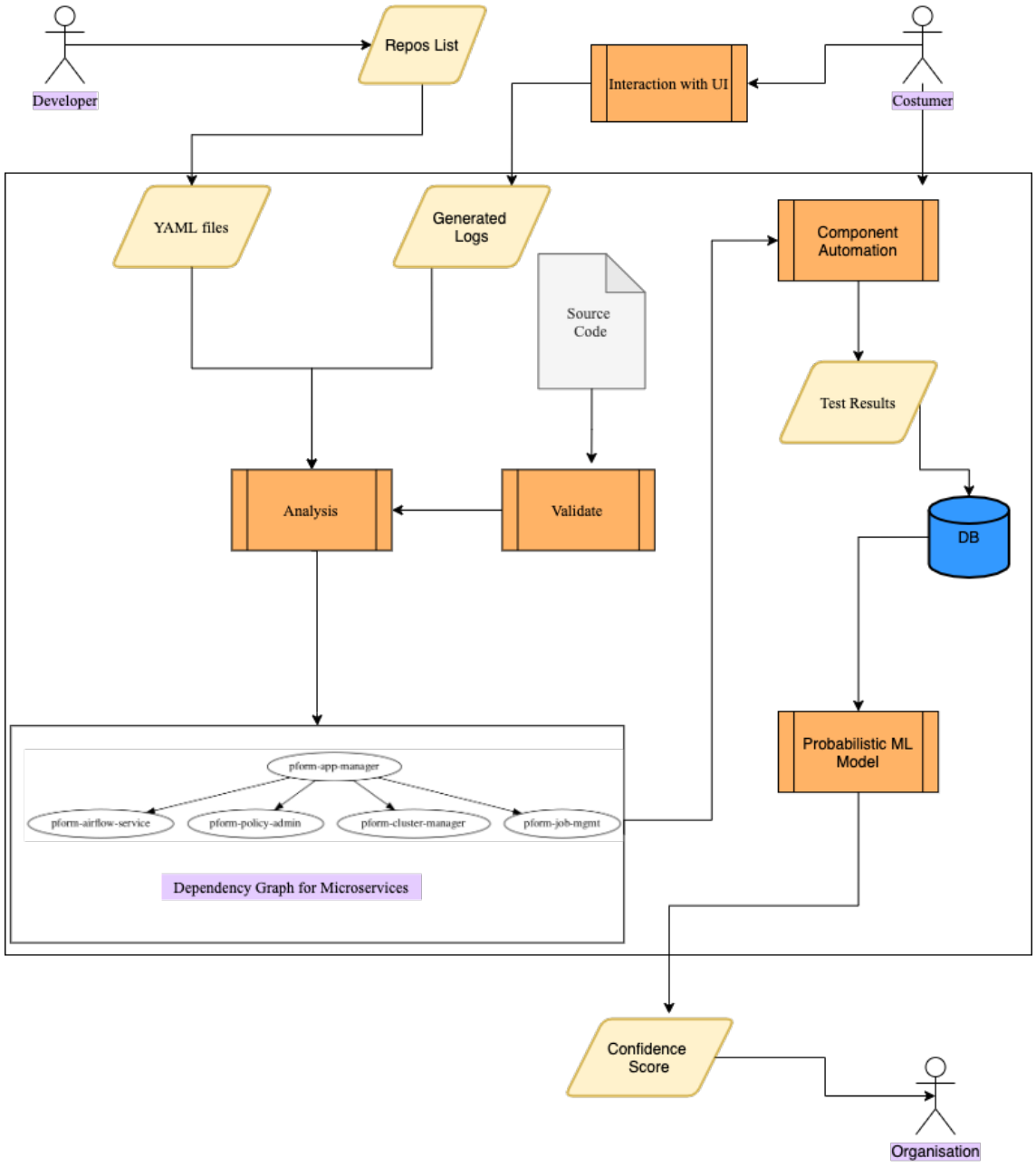

Fig: System Architecture Diagram

Figure 5.1: System Architecture Diagram

Chapter 6

Implementation

For Implementation of the complete project, Python3.6 is used for data analysis, and Jenkins and Apache Groovy are used for CI/CD pipeline. The Python scripts are deployed on a headless server having Linux operating system.
Following is the detailed description about the implementation of the proposed solution Impact Analysis and Ranking System (IARS) for Microservices in different phases:

6.1 Phase 1: Generating Dependency Graphs

There are two approaches for creating dependency graphs. The approaches are discussed as follows:

6.1.1 Using YAML files

So in each microservice, a configuration file such as YAML is maintained where the endpoints of the other microservices are exposed. The python script takes the list of microservices name stored in a text file as argument. We can change the list of microservices as per our needs. The os module helps to locate the respective repository from where the information is extracted

to create graphs. These microservices mentioned in the YAML file are being called by respective microservice. So, by using YAML file we extract the services name and the URL of the corresponding microservices by parsing the YAML files. For extracting information from YAML files the PyYAML module used. By using recursive way, only the repository name is extracted from the endpoints using urllib module of python. This data is stored into a dictionary. Using these dictionaries each for a service, we mapped all the services and create a graph using networkx module of python. This graph is stored into a dot file. These graphs can be created for a specific microservice, combination of some microservices and also for the complete system. For each microservice we can generate a specific dot file. For visualising graphs in this approach, we are using two modules one is graphviz which creates diagrams into dot format while another one is Plotly which is interactive diagram, which can be viewed into browser. For visualising the interactive graphs, one should installed npm and orca on his/her local machine. In interactive graphs, the color bar is used to show the number of services that depends on one particular service.

6.1.2 Using Logs

Another way of generating the graph is by using logs generated while customers triggered different processes on User Interface. These processes' logs carry Trace Id and Span Id. Each process has a unique Trace Id which flows through the whole system and each call between two microservices in that flow generates a different Span Id. For extracting the traceID and spanID from logs, we parse the logs using regular expression since traceID is alphanumeric unique digit of fixed length. To work with regular expression we use python regex module "re". Now since we have traceIDs, we match the com-

mon traceIDs among all the services using pandas dataframe and put them into a set. Now, we create different such sets for different traceIDs. After this, process we get many redundant sets, thus we discard them keeping their count. Now we have a set of sets which consist of certain services sharing same traceIDs. Now using UpsetPlot module, we plot the graph, which consist of different sets and number of time those services were called together for a process.

6.2 Phase 2: Validation using Component Automation

Now, the dependency graph generated in phase 1 is used by component automation to validate the working of different microservices. The component automation uses Mock server and Open API to check whether a service is working correctly or not. The response JSON is feed to Mock Server by Swagger (Open API) and the generated response is compared with expected response of the service. By using dependency graph, the workload of the component automation reduces as it now knows which are the dependent microservices for a particular microservice. So, now it will only validate the services which are being called by the respective service. While validating the build of the services whenever a new feature is added or a bug is encountered, whether the build is successful or it gets failed, the corresponding data will get stored into a database in a format of csv.

6.3 Phase 3: Generating Ranking System

Now the csv generated will be analyzed by using probabilistic model where the considered features for the model will be ID of microservice, number of times microservice failed or succeed, severity level of bug, time required to fix the

bug each time, etc. These features will help predict the probability of "service should be fixed or not" which we termed confidence score. Confidence score will be ranging from [0, 1]. Higher the confidence score, lower the chances of service getting failed after fixing the service. Based on the confidence score, the microservices will be ranked and this will help organization to make business decisions during a product release whether they should fix the bugs before release or fix the bugs in upcoming patch. The dependency graph will help developers to look directly where the possible error could have occurred by checking the call relationship between these services.

Chapter 7

Results

The results are generated in the format of dependency graphs which can be visualized by the generated images or stored in the dot files, which can later be used to generate the dependency graphs. There are 3 types of dependency graphs one can generate using different techniques.
The results after running the python script using different configurations for different microservices is discussed as below:

- Here, the dependency graph is generated for repository called "pform-app-manager" of the product Information Studio, which is dependent on the services shown.

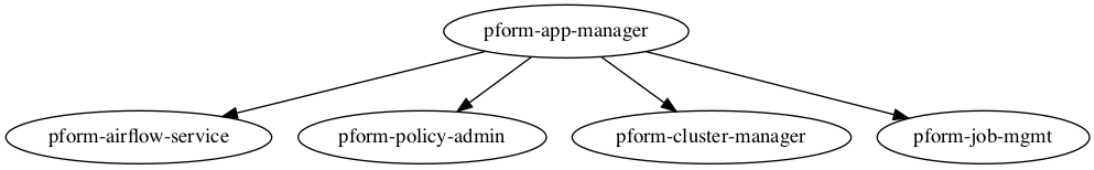

Figure 7.1: pform-app-manager repository dependencies

- Here, the dependency graph is generated for repository called "pform-job-mgmt" of the product Information Studio, which is dependent on the services shown.

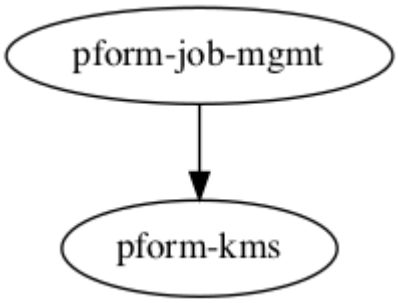

Figure 7.2: pform-job-mgmt repository dependencies

- Here, the dependency graph is generated for repository called "pform-kms" of the product Information Studio, which is dependent on no other service as it is the key manager service, so it does not rely on any other microservice.

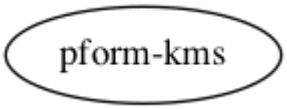

Figure 7.3: pform-kms repository dependencies

- This is an interactive dependency graph generated using plotly scatter plot, which shows the dependencies and also highlights the number of services depended on one particular service using the colorbar.

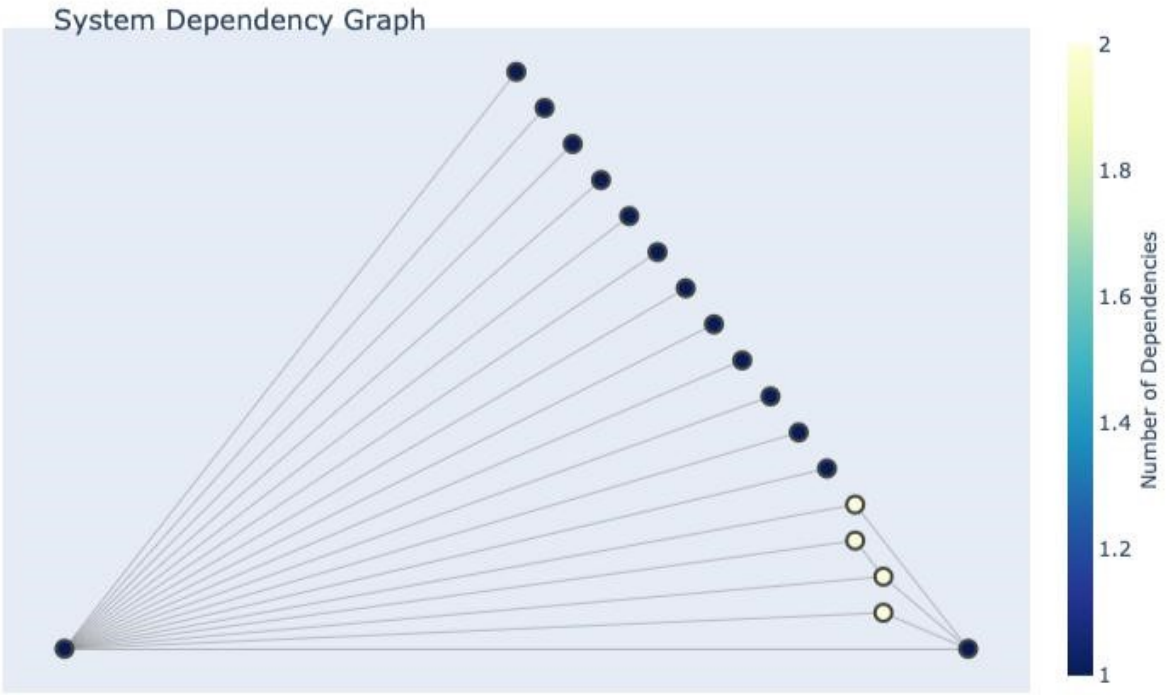

Figure 7.4: Interactive graph for few of microservices

- Here, the dependency graph is generated using logs for a task called "Discovery", on the left side we have services' name, while on the top, the count of how many times those particular services were used together for the process is shown.

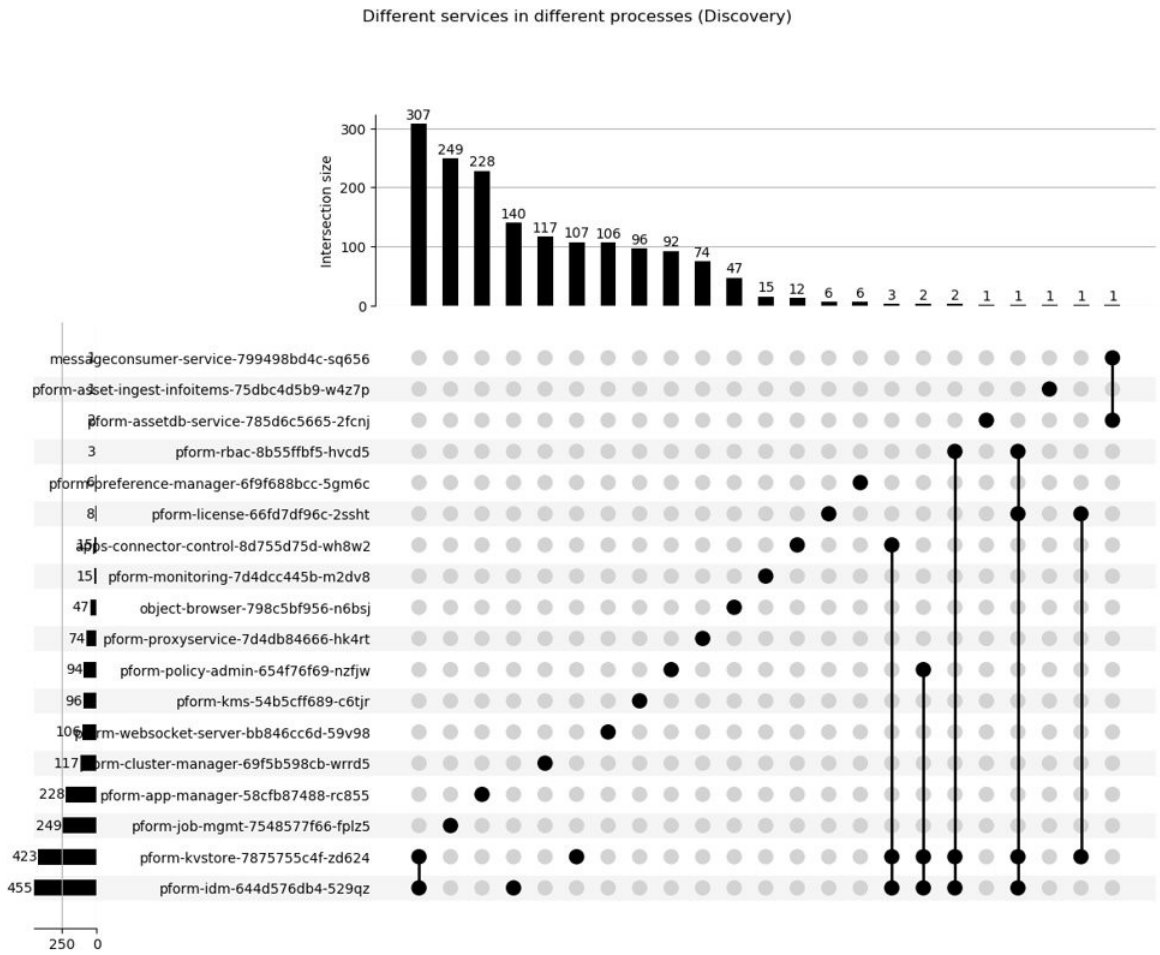

Figure 7.5: Dependency graph for task called Discovery

Chapter 8

Future Scope

Here I have discussed the work related to the project which can be done in the future:

- As currently the project run using command line, so this can be replaced it with Graphical User Interface and results shown in an application.
- Instead giving the list of repositories as input, GUI can include search bar and provide option for selection of multiple services graphically.
- The configuration file and logs for different services should be up-to-date and maintained.

Chapter 9

Conclusion

Impact Analysis and Ranking system for microservices is efficiently generating the different types of dependency graphs for different purposes using two approaches one by using YAML file and other by using logs of different services. By using these dependency graphs, developers can know which service is dependent on other services and include changes according to those services for better productivity. It helps in testing purposes which includes the component automation using Mock Server, while mocking the services, now testers know which services to target for testing, as they can look into dependency graphs and mock the responses of those particular services. Hence, this reduces the time required for testing. The dependency graphs also let know the weight of a service that is the services which are heavily relied upon, so now the changes added to services can be done more carefully as per the weight of service. Overall, the dependency graphs will enhance the product development and deployment process.

Appendix A

Abbreviations

MSA: Microservices Architecture

IARS: Impact Analysis And Ranking System

API: Application Programming Interface

JSON: JavaScript Object Notation

GUI: Graphical User Interface

YAML: YAML Ain't Markup Language

CI/CD: Continuous Integration, Continuous delivery, Continuous deployment