ISSN: 0976-3104
ISSUE: COMPUTER SCIENCE

ARTICLE

DEBUGGING MICROSERVICES WITH PYTHON

Sameer Shukla*
System Architect, HCL America, Fort Worth Texas, USA

ABSTRACT

The modern world applications are expected to be highly available, resilient and the applications are supposed to deal with many concurrent requests and a huge data set (Terabytes) now is a reality. To deal with such demanding requests a new architecture style has been introduced known as Microservices architecture where the services are broken into many small services, where each service is designed on the Single Responsibility Principle. There are situations where there are more than 100 microservices are running in parallel the major challenge comes in front of developers is how to test or debug all these microservices before moving to production and analyzing data during their testing and discovering important information, conclusions and taking corrective measures. In this paper I am trying to showcase an approach of testing, analyzing and debugging data in the Microservices architecture using Python and some of its fantastic packages from Developers Standpoint. I will highlight some of the important issues Developers and Testers face on daily basis in their testing because of the complexities Microservices architecture brings with itself and how we can use these Python packages to solve those issues, irrespective of the language in which the services are developed it can be Java, Scala, .NET or any other language.

KEY WORDS
Microservices, Python, Kafka-Python, Pandas, SQLAlchemy

INTRODUCTION

Microservice is SOA in new cloth, in Microservices architecture complex application is divided into various small services which can be developed and Scaled independently, and they can communicate with each other which indicates that we are developing a distributed system. Microservices are Fault Tolerant in nature meaning if one service is down others will keep running fine and the failed service has no impact on others. Another very useful advantage of Microservices are they can be Polyglot in nature consider a scenario where few Microservices needs to deal with lots of records (in Petabytes) here we can use NoSQL say Cassandra as our database and few microservices needs to deal with only a few millions of records here we can very well use traditional RDBMS as our data store. Working in Microservices architecture brings Clarify of Domain for the entire team since the application is composed of several services it's easier to understand the functionality of every service in the Microservice Architecture. Microservices have few disadvantages as well like Developing Distributed systems can be very complex and Testing of Microservices is a very cumbersome process, in Monolithic architecture performing Testing was very easy as there is one huge service which was doing everything but in Microservices architecture there can be hundreds of tiny services you can imagine Testing of such hundreds of tiny services is very challenging. Similarly debugging of these tiny services is a very cumbersome process for developers, this article focuses on a unique approach of debugging Microservices using Python [1] and its lightweight packages. No matter in which technologies you have developed your services, but we can very well use Python [1] and packages for debugging.

Received: 6 Mar 2019
Accepted: 7 May 2019
Published: 17 May 2019

Challenges in microservices architecture

No Technology or Architecture styles are perfect, similar is the case with Microservices it brings certain major challenges with itself. Few of them are Testing Microservices, Monitoring Microservices, Debugging Microservices, Learning Curve of the Technologies because of Polyglot Architecture, Integration Pain Points

Let's investigate more detail what exactly the challenges are in Testing and Debugging in Microservices. First, we will understand the Use Case and the challenges within that Use Case, and we will look at the approach to overcome the Challenge.

Use case

Let's discuss a real-world scenario which can be divided from monolith to actual functional services, Reliant Energy or Salt River Project (SRP) Energy would be an ideal microservice architecture use case.

Fig-1 shows the typical microservice environment where each service is responsible for one use case based on the Single Responsibility Principle. For ex: Customer Profile Service is responsible for managing customer data. Billing Service is responsible for handling customer billing. Consider a scenario where you are developing a Moving Microservice which is dependent on Consumption Microservice for its data, now something is wrong with Consumption Microservice data which is developed by some other developer or team then the most challenging part is to identify the problem in Consumption service. Let's look into it in detail [Fig-1].

Corresponding Author
Email:
sameer.shukla@gmail.com
Tel.: +1-480-754-9793

IIOAB JOURNAL
ISSN: 0976-3104

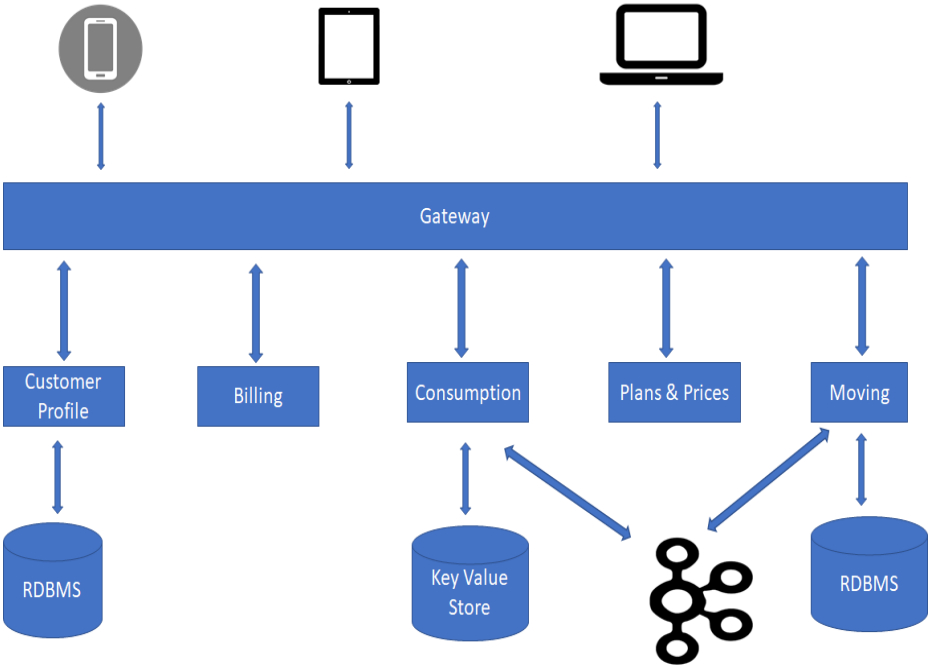

Fig. 1: Microservices environment

ISSUES IN TESTING/DEBUGGING MICROSERVICES

Composite nature

It is recommended that Microservices should be of Atomic Nature, meaning each service should have its own database otherwise if services share a common database than they are the candidate of Single Point of Failure if the common database is down all our services are down. Instead, if each service shares a separate database and then if one of the databases is down it has no impact on other services and they continue to work fine. For example, in our use case if "Customer Profile" service database is down it has no impact on "Billing" Microservice.

So far so good, services will work fine independently as they are loosely coupled but imagine a scenario where there are 100 microservices are running in our system and we want to test the entire pipeline, it will not be easy to test the pipeline and bigger issue is the data verification. Sometimes services transform the data based on the event received and we want to test our data at every step.

Another scenario, if something went wrong in Production, say service is expected to return some data but it's not as per the expectation how we will verify which service is the culprit because each service is atomic.

Logging

Logging and Monitoring are not easy in Microservices, many microservices each one of them logging and reporting issues together it will be very challenging to go and checks logs in individual services and finding the service and the root cause of the error. The solution to this problem is to externalize logging by pushing log messages to Kafka Topic and later analyzing them again with the help of a few tools like Kibana via Elastic Searches.

Testing dependent services

Again, the issue is same if services are dependent on each other say one asynchronous service needs to invoke a RESTful service to validate the data before saving, update or deletion testing of such dependent services and verifying the data the dependent service is returning is little difficult.

Testing Actor models

If we are developing reactive microservices, Akka [3] is the default choice for building reactive microservices. But how can we test actors which are dependent on other services and actors?

Verifying data in key-value store

As discussed in the Use-Case we are using Polyglot architecture in our system and one of the services is using a Key-Value store for persisting and retrieving data say, Cassandra [12], now Cassandra works on Partitioning key and Clustering key mechanism but during our development if we want to verify whether some other column exists in Cassandra database or not but we don't have a Partitioning key, how we should handle such problems during our development phase

IIOAB JOURNAL
ISSN: 0976-3104

Tools for debugging and testing

We are going to use the following Python [1] Packages for our debugging and analysis

- Pandas [9]
- SQLAlchemy [2]
- Requests module [10]
- Kafka-Faust [5] / Kafka-Python [7, 8]
- PySolr [11]
- PyAkka [6]

The beauty in these packages is they are lightweight, easy to learn & use, tools can be developed using these packages in very less time. A lot of online help on these packages are available and their documentation is also very crisp

METHODS

Debugging and analyzing RDBMS

Scenario: I am a developer who is working on the development of some Microservice but for some reason, I need to check the "Customer Profile" service and its data for debugging purposes and analysis reasons. Two things a developer can do here either Invoke a Customer Profile service or directly dip into a database and do the verification. I have decided to directly go into a database, Firing manual queries one by one will be a tedious and time-consuming job as I need to check many things like whether the customer is an active customer or not, what are the details of the customer based on Customer Id for billing related purposes, since how long the customer is an active customer to recommend him some better plans etc. Here I would like to use a package called SQLAlchemy [2] and combine it with Pandas [9].

SQLAlchemy

It's a lightweight database toolkit written in Python, the advantage of using SQLAlchemy [2] toolkit here is we can very easily use it in Jupyter notebook, and we don't have to create the entire framework or write hundreds of lines of code just to communicate to a database, hardly in 7 lines we can query any RDBMS and get result. Another powerful advantage of using SQLAlchemy [2] is that we can define the database schema in the application code itself, but we are more interested in only reading the data from the database. The code is very readable because it is written completely in Python. Our scope here is limited that's why I am not going into the details of ORM here].

Pandas

Pandas [9] is an excellent data analysis library, it can load data from any data source such as CSV, JSON, database and creates a Python object called DataFrame that looks very similar to the table. Once DataFrame is created we can perform a variety of operations like Grouping, Filtering, sorting by keys and values, Creating Series from DataFrame, etc.

CREATE TABLE db. customer_profile (customer_id INT, first_name VARCHAR (20), last_name VARCHAR (20), email VARCHAR (20), is_active boolean, created_date DATE) [Fig-2].

```
In [2]: import pandas as pd
        import pymysql
        from sqlalchemy import create_engine
        pymysql.install_as_MySQLdb()
        engine = create_engine("mysql://root:root@localhost/db")
        conn = engine.connect()
        result = conn.execute("SELECT * from customer_profile").fetchall()
        result[:2]

Out[2]: [(100, 'Sameer', 'Shukla', 'sshukla@gmail.com', 1, datetime.date(2019, 2, 28)),
         (101, 'John', 'Doe', 'jd@gmail.com', 1, datetime.date(2019, 2, 28))]

In [3]: df = pd.DataFrame(result)
        df.columns = result[0].keys()

In [16]: # Find all inActive Customers
         df[df['is_active'] == 0]

Out[16]:
```

	customer_id	first_name	last_name	email	is_active	created_date
3	103	Dav	P	dp@gmail.com	0	2019-02-28

0 indicates Inactive Users and 1 Indicates Active Users.

Fig. 2: Example of Pandas with SqlAlchemy

IIOAB JOURNAL
ISSN: 0976-3104

We are simply fetching the result from the Database using SQLAlchemy [2] and giving the result set to Pandas [9] which then created the DataFrame object on which we are executing our analysis. Here we are looking into all the Inactive Users in the Customer Profile Database

Debugging and analyzing rest service

I am extending the scenario mentioned above, now I have decided to invoke the RESTful service and then validate the JSON Response. The package I will like to prefer here is "requests" combining it with Pandas.

Requests package

Requests [10] is a simple, lightweight HTTP library or we can say a simple HTTP Client library. Advantage of using this library is the same we don't have to create the entire framework just for invoking a RESTful service, plus combining this library with Pandas [9] makes it more powerful because sometimes a JSON response can be huge thousands of line, try invoking https://api.github.com/events [4] service you can see response is of approximate 2000 lines, identifying or locating particular field, in this case, would be a difficult task. The approach here is simple, invoke the service and give the JSON response to Pandas and perform operations on DataFrame. For the above URL counting how many Push Events or Create Events the response has is impossible to achieve, but with Pandas, it's just a line of code, have a look [Fig-3].

```
In [2]: import requests

In [3]: # Invoke Service
        request = requests.get('https://api.github.com/events')

In [26]: # Pandas
         import pandas as pd
         import json
         df = pd.DataFrame.from_records(request.json())
         # Counting Push events
         df['type'].value_counts()

Out[26]: PushEvent           16
         CreateEvent          4
         WatchEvent           4
         ForkEvent            2
         PullRequestEvent     2
         GollumEvent          1
         IssuesEvent          1
         Name: type, dtype: int64
```

Fig. 3: Example of Pandas with requests package

After executing the Rest service, we are taking the JSON response and handing it over to Pandas, the DataFrame is returned by Pandas on which we have executed our analysis which is count the Push Events

Debugging and analyzing Kafka

There are two very important packages are available to deal with Kafka, first is Kafka-Python and another one is Kafka-Faust. Faust is a Stream Processing Library; we can process both Streams and Events using this library. Another one is Kafka-Python, it's a Python Client for Apache Kafka, using Kafka-Python library we can consume messages directly from Kafka by subscribing to Kafka Topic and we can also produce messages to Kafka Topic. To test the Moving service, which is consuming messages from Kafka Topic, we can write a lightweight Kafka Producer using Kafka-Python library and post a message on the topic from there Moving service will pick up the message and further process it. We enhance our Test client using SQLAlchemy + Pandas and analyze the data in the RDBMS [Fig-4].

Other important packages

PySolr: It's a lightweight wrapper for Apache Solr, you can query Solr server and analyze the data returned based on the query executed. Use Case for PySolr can be, redirect message from Logging Kafka Topic and index the logs using Apache Solr and query the Solr Server using PySolr.

PyAkka: It's a Python Implementation for Actor Model [3].

IIOAB JOURNAL
ISSN: 0976-3104

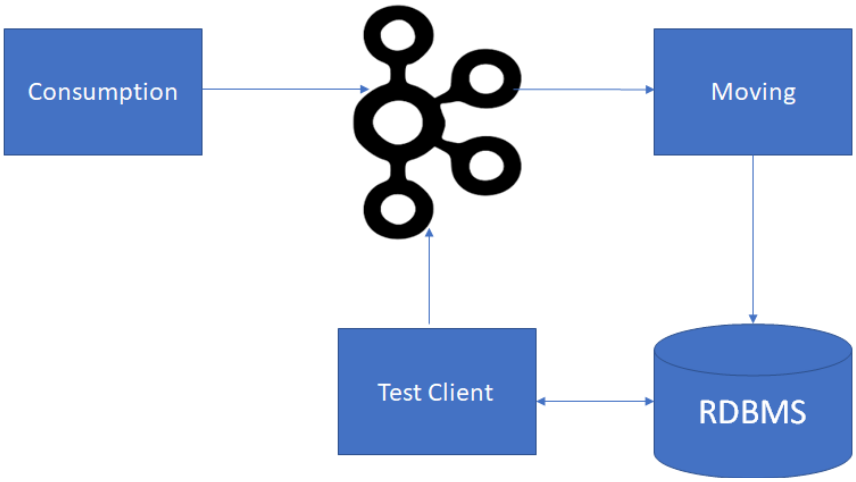

Fig. 4: Kafka in microservices

RESULTS AND DISCUSSION

In the modern era Gigabytes, Petabytes of data is the reality and distributed system and reactive systems are the need of the hour. Engineers must bring unique approaches to every aspect of SDLC for being efficient. In hundreds of Microservices environment it's difficult for an engineer to identify the issues in the system, Python is not purely functional but it's tools are amazing, like we don't have to deploy an application or we don't even need an IDE if we want to debug something a simple Jupyter Notebook is self-sufficient, on which we can do almost everything, all the packages I have mentioned in this article can very well run in Jupyter notebook, we can immediately verify what our microservices has done or we can quickly check the dependent services issues or data. I strongly recommend to the engineers to use Python [1] a lot in whatever environment they are working and whatever tools or technologies they are using.

Debugging and Testing Microservices with Python [1] and related packages in a unique approach. Each package I mentioned in the article is lightweight, for example using SQLAlchemy [2] for database operations has several benefits like we can define the database schema in the application code itself, we need to construct database queries in Python which brings FRM (Functional Relational Mapping) like flavor with itself. Requests package is very convenient to use we don't have to write hundreds of lines of code just to invoke a service the Requests package do it for us. The combination of these packages with Python Pandas [9] brings analytics also into the picture, Pandas is a package which is widely used in Data Analytics world and using Pandas with SQLAlchemy results can do wonders for us. Similarly, there are various other packages like Matplotlib for graphs which we can be very well used in the Microservices world, giving Pictures (Graphs) like Pie chart, Bar Graphs, Histograms to our data can be very useful for everybody as picture speaks a thousand words, say if we want to show the top 10 or last 10 kind of results from our Database how convenient and useful it is when we show them using bar graphs rather than seeing the plain data. Another Advantage of using Matplotlib + Pandas + SQLAlchemy or Requests or PySolr [11] can help us in identifying the problems in our system say we are using Cassandra as our data store and if we want to see whether partitioning balancing is proper or not in our system these graphs can help us in identifying such issues.

CONCLUSIONS

The Microservices architecture in today's world is a reality, there are many tools, languages, frameworks are available on which these Microservices are developed. We need to be smart and quick in figuring out the issues these services can have in Development or Production Environments, otherwise it will defeat the purpose and advantages this Architecture style brings with itself. We need to know all the Integration Points and we should be having some tools with us to figure out the problems in those Integration Points or in the Services. Python [1] and its packages provide us the best options in front of us to identify such problems. As an engineer in today's worlds, it is mandatory that we should not restrict ourselves to one technology or framework, this Python [1] approach has helped me big time in resolving the issues I have mentioned in this Paper. I strongly recommend engineers to follow this approach in quickly identifying and fixing issues in Microservices environment

CONFLICT OF INTEREST
None

ACKNOWLEDGEMENTS
None

FINANCIAL DISCLOSURE
None

IIOAB JOURNAL
ISSN: 0976-3104

REFERENCES

[1] http://docs.python-requests.org/en/master/
[2] https://www.sqlalchemy.org/
[3] https://doc.akka.io/docs/akka/2.5/testing.html
[4] https://api.github.com/events
[5] https://faust.readthedocs.io/en/latest/
[6] https://www.pykka.org/en/latest/
[7] https://kafka-python.readthedocs.io/en/master/usage.html
[8] https://www.confluent.io/
[9] https://pandas.pydata.org/
[10] https://pypi.org/project/requests/
[11] https://pypi.org/project/pysolr/
[12] http://cassandra.apache.org/