Platform for Discovery of Microservice Instances

Yordan Yankov*, Aleksandar Dimov**, Krasimir Baylov*
* Faculty of Mathematics and Informatics, Sofia University "St. Kliment Ohridski"
15 Tsar Osvoboditel Blvd., 1504 Sofia, Bulgaria
** Faculty of Mathematics and Informatics, Sofia University "St. Kliment Ohridski"
15 Tsar Osvoboditel Blvd., 1504 Sofia, Bulgaria
and Institute of Mathematics and Informatics, Bulgarian Academy of Sciences
Akad.G.Bonchev St, bl. 8, 1113 Sofia, Bulgaria
yankov.dev@gmail.com,{aldi, krasimirb}@fmi.uni-sofia.bg

***Abstract*—Service Oriented Architecture (SOA) is one of the most common styles used to design and develop contemporary software intensive systems. Microservices represent the latest trend to build SOA applications, offering interoperability, scalability, component sharing, improved fault isolation and etc. Although an emerging and still immature concept, it gains big impact in software development industry, as many companies try to migrate their projects to such architecture. Microservices lead to construction of highly flexible scalable and dynamic software systems. Such dynamism, however may provoke dynamic reallocation of running microservice instances which results into a change of their addresses. This raises some problems with discovery of microservices. This paper presents a platform for discovery of microservices, which tackles this problem.**

I. INTRODUCTION

Service Oriented Architecture (SOA) is an architectural style that provides tools and techniques for development and integration of heterogeneous distributed systems, using software components, called services that are reusable, loosely coupled and accessible via standard protocols. SOA is widely adopted today and there is an extremely large number of applications that are based on it. In recent years, major problem with service oriented systems is that they are usually built as big monolith applications. Such applications have all of sub-modules packaged into a single component. This has a number of drawbacks [9]. For example, they are hard to understand and modify, the team productivity is decreased because of the large code base, continuous delivery and scaling are very difficult, they are tied to a specified technology stack, etc. All those limitations in conjunction with the increasing need of quick response to the constantly changing requirements, have forced software engineering community to develop new approaches for development of service oriented systems. One such very appealing and widely used approach is microservice-based development [8, 9].

Microservices is an architectural style that is based on SOA and a set of principles devised around the idea of building small and easy to deploy services. Microservices are heavily impacted by Domain Driven Design [2], as they are all designed around domain entities. As a new notion, there does not exist a common agreement if microservices is a completely new style for developing applications or it is just another form of SOA [17]. In addition, there is no commonly accepted unified definition for microservices, either. For the rest of this paper, we would refer to the following definition, adapted from [8]: *"The microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies."*

Main benefit is facilitation of scalability of microservice based systems operating in cloud environment, because it is easier and less expensive to scale and replicate only particular modules, which are taking the heaviest loads. In other words, there is no need to start a new instance of the whole system, which is the case of monolith applications. That enables the full use of the virtualization in the cloud – whenever it is needed, new virtual machines or containers are started and a new instance of the given microservice is executed on them. However network addresses of those replicated microservices are generated dynamically. This determines a need of a solution to discover the appropriate services within a complex heterogeneous systems, running into a distributed environment.

The goal of this paper is to present a platform for microservice discovery. Main design decisions of the platform are described, together with some details about its implementation.

Structure of the paper is as follows: Section II gives some more details about the notion of microservices and microservice architectures; Section III presents the related work; Section IV is dedicated to description of our microservice discovery platform and finally Section V concludes the paper and gives some directions for further research in the area.

II. MICROSERVICE ARCHITECTURES

The concept of microservices tries to overcome many of the limitations implied by large monolithic services.

Microservices are small in size and fine-grained. Their small codebase means less dependencies between service components, which reduces the risk or introducing bugs to the code, simplifies testing and allows developers to quickly identify and fix issues if any. This can be critical when teams support service level agreements (SLA) to their customers and they need to respond quickly to reported incidents.

In contrast to some software engineering principles, a common approach for building microservices is to copy the same components to all services and evolving the components independently. This approach benefits coordination of development teams, as it gives programmers freedom to update their services independently by reducing the external dependencies. Although this approach increases code duplication, it is believed to decrease the coupling between service components.

Microservices strictly favor Representational State Transfer (REST) [3] and simple messaging mechanisms, which eliminates the need of using complex middleware systems that translate, route and mediate messages. This way they improve interoperability and foster development of highly heterogeneous systems.

In general, microservices are a form of SOA. Since SOA practices are significantly less restrictive than microservices it is reasonable to think of microservices as a "form of SOA, perhaps service orientation done right" [11].

As discussed in the introduction, there still does not exist a consensus in the community about the definition of microservices. Nevertheless, there are some microservice characteristics that are widely accepted and are common across different existing implementations [8]. They are as follows:

- Service Componentization – Microservice systems are divided into separate components that can be independently replaced and upgraded.
- Organization around Business Capabilities – Microservice systems are designed around the business units of each company, breaking the "traditional" approach of organizing teams around system layers.
- Product Development – microservices approach is focused on developing products, not projects. Teams take ownership of the products through their entire lifetime.
- Smart Endpoints and Dumb Pipes – applications built around microservices should have communications as simple, as possible. They should eliminate the need of complex Enterprise Service Buses and method calls.
- Decentralized Governance and data management – decentralization of microservices governance results in increased flexibility in building and supporting them. Moreover, shared data stores across services should be avoided and each service should maintain a separate data store.
- Automated Infrastructure and Monitoring – microservice concept results in a big number of services, which makes it impossible to support their development manually. Therefore, highly automated infrastructures are needed to support the development, deployment and monitoring operations.
- Design for Failure – distributed nature of microservices leads to many of the common problems for distributed environments, like network outage, not delivered messages, slow network infrastructure, etc. Therefore, developers should consider failures as part of the applications themselves and design their microservices with failures in mind.
- Evolutionary Design – microservices are characterized with constant evolution as business environment and requirements are constantly changing.

The solution, presented in this paper aims to improve achievement of the "smart endpoints", "design for failure" and "automated Infrastructure" characteristics of microservice architectural style.

III. MICROSERVICE ARCHITECTURES

Academic research, related to ours concerns various solutions of service discovery, which differs based on the information used to find the service. We have identified two big groups of approaches for service discovery – based on semantic information and the second, based on information about Quality of Service (QoS).

Authors of [7] present a tool called OWLS-MX for automated web service discovery, based on ontology based semantic description of services with OWL-S language. The tool uses hybrid matching filters that elaborate logic based reasoning and content based information retrieval to find relevant services. Another method described in [10] aims to improve existing service discovery approaches by extending semantic representation through analysis of service WSDL and grouping similar services in clusters. Caching mechanism that acquires knowledge from previous searches is applied in order to increase the performance of semantic discovery in [14]. A research effort [6], focused into the domain of sensor webs, presents method for discovery of sensor services, which is again based on semantic observations. In [13], a logic based matchmaking for semantic web services is proposed for services described with WSDL 2.0. A continuous based degrees of match, formed by linear regression are used to discover services, in distinction with majority of other approaches that uses discrete degrees.

Approaches, based on QoS aim to facilitate users to select a service between several alternatives that best fits their needs. Some of them share a common idea, based on user feedback to analyze QoS, while others are based on soft computing or statistical methods for this purpose. For example, another semantic approach, based on peer-to-peer service registries and augmented with QoS information about services is presented in [15]. To express QoS characteristics of services, author propose a user feedback mechanism. User feedback data to augment standard UDDI registries is used also in [16], where a discovery agent calculates reputation score in order to find appropriate services. A novel approach, based on probability and fuzzy set theory for service selection is proposed in [5], which takes into account also uncertainty in user specifications.

There exist some platforms implemented for industrial use, which although not specifically targeted for microservice discovery, may be tailored in order to be utilized for this purpose. Among them are Consul [18], ZooKeeper [21], etcd [19], etc. However, more scrutinized description of such platforms falls out of the scope of this paper.

Our work does not fit into any of these categories and is more likely to complement the above listed approaches. Main application area of the platform that we have developed is to find appropriate replica of a given

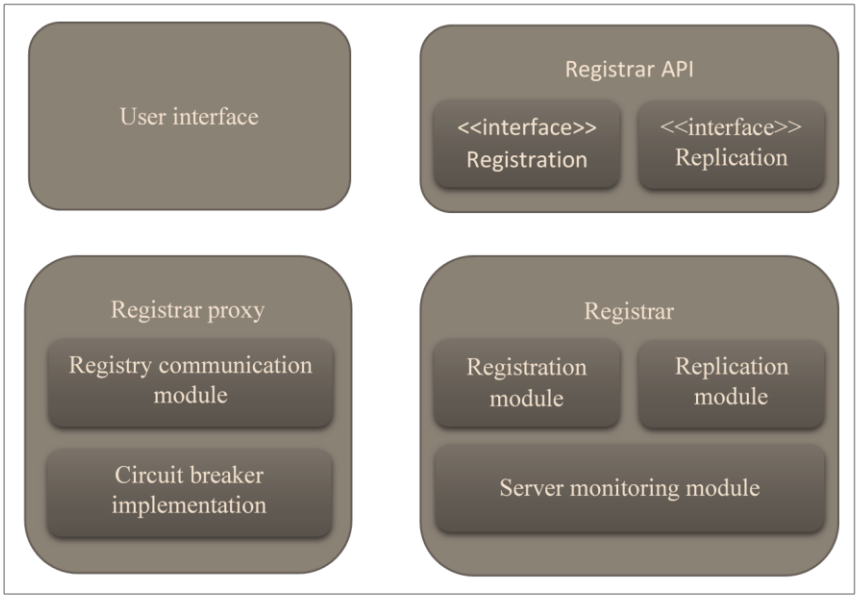

Figure 1. Architecture of the microservice discovery platform

microservice, and more specifically its address, because when operating into a distributed cloud environment, the addresses may become a subject of constant change.

In next section of the paper, we continue with more detailed description of our solution for microservice discovery.

IV. PLATFORM FOR MICROSERVICE DISCOVERY

The most rigorous requirements towards the platform concern availability and reliability. This is due to the fact that in real world application environment it is a bottleneck if many other users (both machine and human) rely on it in order to find appropriate service to process their request. This way, design and implementation of the platform should be presented by its architecture and with respect to the two main challenges that concern the process of service registration and the algorithm to select the active instance of the platform.

A. Architecture of the platform

The service discovery platform is comprised of four main sub-systems (Fig. 1): User interface, Registrar API, Registrar proxy and Registrar. All of the modules are themselves implemented as microservices.

The registrar subsystem is responsible to connect with the registry to store data about service registrations. It contains three modules, as follows:

The registration module contains all functionality about service registration and unregistration. It also connects with the services registry.

The Replication module is responsible for replication of the server in order to achieve scalability, availability and fault-tolerance. Replication process is described in more details in the next subsection of the paper.

The server monitoring module provides REST endpoints for observation of server state, hardware load and also provides statistics about the raw data for service invocations - successful and unsuccessful requests, response time, etc.

The Registrar API provides the Application Programming Interface (API) for the services that should register in the discovery platform. All functionality is provided as REST endpoints.

The Registry proxy subsystem is responsible to register the service, to track its availability and unregister the service if it appears to be repeatedly down. Here is implemented also the functionality for discovering a specific service and obtaining its URL. For the purpose to track service availability, an implementation of the circuit breaker pattern [4] is used. This subsystem should be integrated in each microservice that is going to be registered into the discovery platform. Currently, this is possible only for Java implementations, via Maven [20], which offers flexibility when building complex software projects.

The user interface subsystem has the responsibility to visualize the registrar state in order to facilitate administration of the whole platform.

B. Service registration

Usually services, including microservices are identified by their location (or endpoint) and description of their interface. For this purpose our platform should implement methods to register the service, when it is active and subsequently - unregister when it becomes inactive. Two main approaches to do this are possible in our context [12]: (1) service self-registration and (2) third party service registration.

The first of these approaches suppose the service to internally implement the registration algorithm. Its main advantages are architectural simplicity, as no additional components need to be added and availability, as each service may support keep-alive messages to notify the register about its load and readiness to deliver functionality.

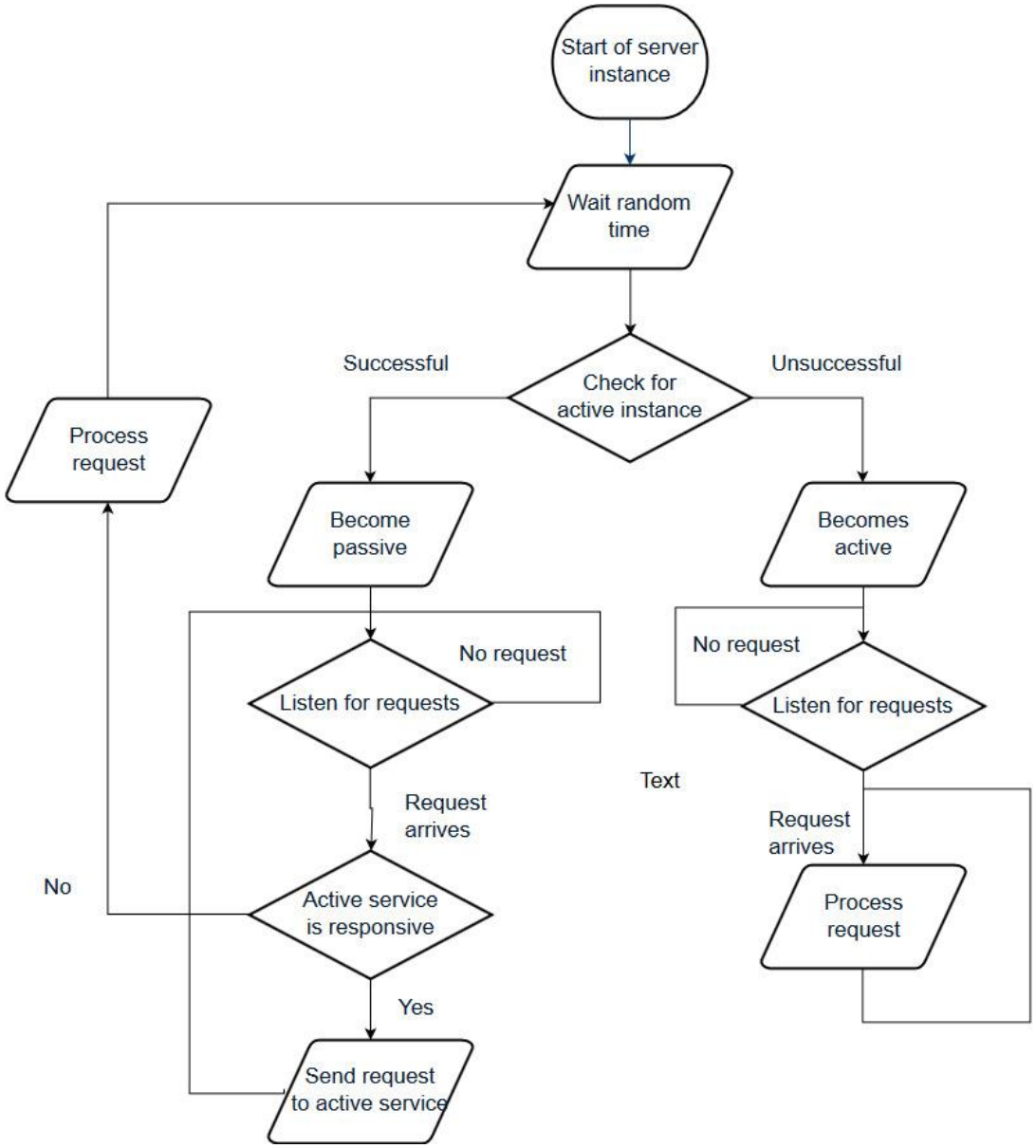

Figure 2. Architecture of the microservice discovery platform

However, this approach have also the disadvantage to complicate the implementation of the service, as additional functionality about registration, which is also dependent on the programming language, should be developed in each service. The second approach for service registration relies on additional component, external for the service to register and unregister it. In this case, service implementation is simplified, however additional mechanisms should be implemented to notify the registrar about service availability and state. Moreover, the registrar itself may appear as a bottleneck about the over reliability of the service discovery platform.

In our platform we are going to use a combination of the approaches described above. More specifically, the platform will export a communication module, which acts as a proxy and is going to be integrated into each service that should be registered into the platform. It is added as a dependency to each service which requires registration. It should be noted, that with such approach programming language independence may be achieved if multiple instances of the communication module are developed at each language of interest.

C. Server replication

Main requirement towards the platform are its availability and reliability. For this purpose, it uses passive redundancy for the registrar module [1].

In our implementation of passive replication, there exists one leading instance of the replicated platform server that process the requests to register, search and unregister and the others synchronize their state with it. This way if enough instances are started one may achieve ultra-high availability of the platform. However, in this case, there need to be a consensus between the instances which one should be the leading one. For this purpose each server instance, when started waits random time and then asks for

existing leader. If such does not exists then it should become active and process all requests. However, all instances operate in parallel and when a request is received from one of the passive services, than it should first the check if the active server is functional, if not – the request is processed and the initialization is started over. The whole process is shown on Fig. 2.

In order for the process described above to be functional, addresses of all platform server instances to be known and configured by an administration.

V. CONCLUSION

The paper described the some design and implementation details about a platform for discovery of running microservice instances. Microservices is a recently emerged architectural style for development of complex and heterogeneous distributed software systems. Usually such systems tend run in dynamic environment and individual microservices are subject of continuous replication. In this case, their physical address also changes in dynamic manner and there raises a need for a tool, like the one proposed here platform, which facilitates discovery of running microservice instances.

The platform for microservice discovery would be a critical component of a system which is using it, so it has been designed for maximum availability and performance. To achieve that, a client-server architecture has been used with maintaining passive redundancy. That allows for improving the availability of the platform.

Test have shown that developed platform is capable of handling production traffic. It is also planned to achieve an even better performance in the future development of the platform by enabling load balancing between both passive replicas of the platform server and also between running microservice instances. Another direction for future work include augmenting the discovery capabilities of the platform by adding semantic and QoS aware discovery mechanisms.

ACKNOWLEDGMENT

Research, presented in this paper was partially supported by the DFNI I02-2/2014 (ДФНИ И02-2/2014) project, funded by the National Science Fund, Ministry of Education and Science in Bulgaria.

REFERENCES

[1] Bass, L., P. Clements and R. Kazman. Software architecture in practice. Addison-Wesley Professional. 2013.

[2] Evans, E. Domain-driven design: tackling complexity in the heart of software. Addison-Wesley Professional, 2004.

[3] Fielding, Roy Thomas. Architectural styles and the design of network-based software architectures. Diss. University of California, Irvine, 2000.

[4] Fowler, M. Circuit Breaker. Martin Fowler professional blog, March 2014, available at: https://martinfowler.com/bliki/CircuitBreaker.html

[5] Georgieva, O. and D. Petrova-Antonova. QoS-Aware Web Service Selection Accounting for Uncertain Constraints. 40th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA). IEEE, 2014.

[6] Jirka, S., A. Bröring and C. Stasch. Discovery mechanisms for the sensor web. Sensors 9.4 (2009): 2661-2681.

[7] Klusch, M., B. Fries, and K. Sycara. Automated semantic web service discovery with OWLS-MX." Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems. ACM, 2006.

[8] Lewis, J. and M. Fowler. Microservices: A definition of this new architectural term. Martin Fowler professional blog, March 2014, available at: http://martinfowler.com/articles/microservices.html

[9] Namiot, Dmitry, and Manfred Sneps-Sneppe. On micro-services architecture. International Journal of Open Information Technologies 2.9 (2014): 24-27.

[10] Nayak, R. and B. Lee. Web service discovery with additional semantics and clustering. Web Intelligence, IEEE/WIC/ACM International Conference on. IEEE, 2007.

[11] Richards, M. Microservices vs. Service-Oriented Architecture. O'Reilly Media, Inc., 2015

[12] Richardson, C., Building Microservices: Inter-Process Communication in a Microservices Architecture. Ngnix technical paper, May 2015, available at https://www.nginx.com/blog/building-microservices-inter-process-communication/

[13] Schulte, Stefan, et al. LOG4SWS. KOM: self-adapting semantic web service discovery for SAWSDL. 6th World Congress on Services. IEEE, 2010.

[14] Stollberg, M., M. Hepp and J. Hoffmann. A caching mechanism for semantic web service discovery. The Semantic Web (2007): 480-493.

[15] Vu, Le-Hung. M. Hauswirth and K. Aberer. Towards p2p-based semantic web service discovery with qos support. International Conference on Business Process Management. Springer Berlin Heidelberg, 2005.

[16] Xu, Ziqiang, et al. Reputation-enhanced QoS-based web services discovery. International Conference on Web Services. IEEE, 2007.

[17] InfoQ EMAG "Microservices: Decomposing applications for Deployability and Scalability", 2014

[18] HashiCorp Consul website, Introduction to Consul, available at: https://www.consul.io/intro/index.html.

[19] etcd website, Getting started with etcd, available at: coreos.com/etcd

[20] What is Maven?, Apache Maven Project Website. available at: https://maven.apache.org/what-is-maven.html

[21] Apache Zookeeper website, ZooKeeper: A Distributed Coordination Service for Distributed Applications, available at: http://zookeeper.apache.org/doc/trunk/zookeeperOver.html