iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 8 **Issue:** X **Month of publication:** October 2020

DOI: https://doi.org/10.22214/ijraset.2020.31979

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 8 Issue X Oct 2020- Available at www.ijraset.com

Microservices: Architecture and Technologies

Vishva Desai[1], Yash Koladia[2], Prof. Suvarna Pansambal[3]
[1, 2]*Department of Computer Engineering, Atharva College of Engineering, Mumbai, India.*
[3]*HOD, Department of Computer Engineering, Atharva College of Engineering, Mumbai, India.*

Abstract: ***In recent years, microservices architecture has emerged to become largely popular in DevOps practices and cloud services. Prime reasons for its meteoric rise are, its modular nature, robustness and scalability. Users today require an interactive, swift, and flexible environment on the platforms they use. These requirements can be satisfied by microservices architecture. A microservices framework consisting of microservices and containers delivers a diverse and distributed application. This architecture handles complex tasks as a collection of smaller services as opposed to the conventional monolithic architecture. In a monolithic architecture, it becomes difficult to manage the structure of code. This is made manageable by adopting microservice architecture. Furthermore, it provides scalability, easy maintenance and independent deployment of services to an application. It enables small parts of an application to be updated, replaced and scaled effortlessly. In this paper, we discuss the architecture of microservices and technology enabling its function.***
Keywords: Microservices architecture, monolithic architecture, microservices technology, Docker.

I. INTRODUCTION

In a world that demands speed and efficiency on top of easy manageability and implementation, in any application. Microservices has taken the lead in making possible all the above. The microservice architectural pattern 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 [1]. Microservices is an architectural design for building a distributed application. Microservices break an application into independent, loosely coupled, individually deployable services. This microservices architecture allows for each service to scale or update using the deployment of service proxies without disrupting other services in the application, and enables the rapid, frequent and reliable delivery of large, complex applications. [2]

In traditional monolithic architecture the entire application is developed on a single code base and deployed as a whole. A monolithic application is developed as a single unit. Figure1 shows the monolithic architecture. Components of a monolithic application cannot be deployed individually, conspicuously on every update the entire application needs to be redeployed. Monolithic applications are usually divided into modules or layers like presentation, business, database, application, etc.

Even though, this type of architecture is simple to develop it has many drawbacks. These limitations are overcome by the microservice architecture. Some limitations of monolithic architecture are listed below.

A. Every small change in application requires complete redeployment of entire application.
B. Difficult to manage due to limitation in size and complexity.
C. It is difficult to use new technologies.
D. Limitations in scaling.
E. A single defect affects the entire application.

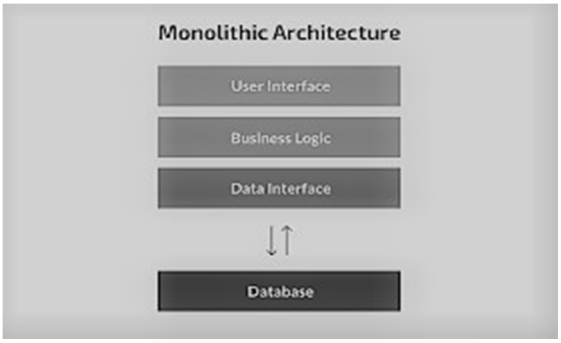


Fig.1 Monolithic Architecture

©IJRASET: All Rights are Reserved

With microservices architecture, it is possible to build a reliable platform to extend the business while taking advantage of diversity in languages. A business can standardize its microservices technology stack by choosing the programming language based on business needs [3]. Several tech giants like Netflix and Amazon have adopted microservices architecture, setting precedent in container technology. Microservices architecture is still nascent and evolving. However, DevOps, cloud and machine learning applications are using microservices to reduce the time taken from development stage to the deployment stage. This is done by using automation, continuous delivery and continuous integration.

Table.1 Comparative Analysis of Monolithic and Microservices Architecture

Monolithic Architecture	Microservices Architecture
1. It is difficult to scale monolithic application.	1. Highly scalable with the use of containers and cloud.
2. Slow deployment. All updates require redeployment of entire application.	2. Rapid Continuous Deployment. Each service can be deployed independently.
3. Uses a single technology stack as it is tightly coupled with low cohesion.	3. Can use diverse technology stack as it is loosely coupled with high cohesion.
4. A single bug can affect the entire application.	4. A single bug affects only the service in which it occurs.

II. BACKGROUND

The term, 'micro web services' was used back in 2005, by Dr. Peter Rogers at a conference on cloud computing. 'Microservice' was used for the first time at an event for software architects in 2011, near Venice. The microservice architecture was first presented by Fred George and this approach was first implemented by Adrian Cockcroft as a fine-grained SOA. Lewis and Fowler in 2014 extensively discussed microservices being about functional decomposition in a domain-driven design (DDD) context. This architecture uses lightweight communication protocols like HTTPS and the microservices are usually deployed using containers instead of VM and provides speed, robustness, and flexibility. Microservices gained momentum due to the stimulus given by the companies like Amazon and Netflix. This style of architecture develops complex applications using small, individual services that communicate using language-independent interfaces.

III. KEY BENEFITS OF MICROSERVICES ARCHITECTURE

The greatest benefit is manageability. It is easier to optimize and maintain the code, when the application is compiled of smaller tasks. Due to each service being different in composition concerning programming and system environments, it is easy to manage the code. Other benefits include:

1) *Scalability:* Each task being independent form the other makes it simple to scale only required components, saving time as well as resources.
2) *Autonomy:* Developer teams can work independently, increasing speed to market.
3) *Failure Isolation:* Failure of a single service does not affect other segments of an application
4) *Decentralization:* There is no specific way to model the services. All microservices should do a single task, using any technology stack of choice [1].

IV. MICROSERVICES ARCHITECTURE AND COMPONENTS

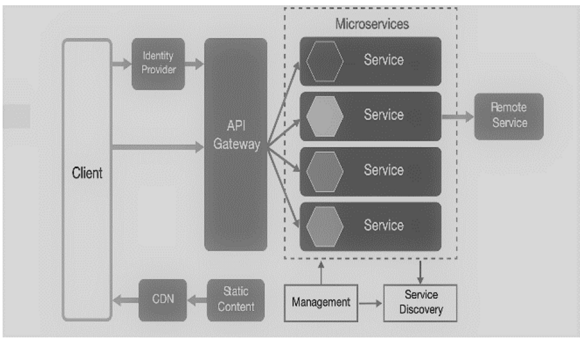


Fig.2 Microservice Architecture

©IJRASET: All Rights are Reserved

A. Client

Clients start of the architecture by requesting services from different sources to perform varied tasks.

B. Identity Provider

Client requests need to be authenticated and authorized. It is important for all services to be secure so as to not be exploited. Every service needs to be identified on service call with the role defined so that the caller is authorized. Identity providers are used to store and verify user ID for communication between client and system. The requests are then passed forward to API Gateways to connect the client to the services.

C. API Gateway

An Application Program Interface Gateway in a microservice architecture provides singular entrance to the system. It is capable of creating multiple custom API's for a microservice depending on the platform being used and its requirements.

The API gateway juggles several responsibilities like authentication, request management, caching, load-balancing and monitoring. Apart from being able to translate between protocols, the gateway enables services and clients to use protocols as per their requirements, increasing speed and productivity.

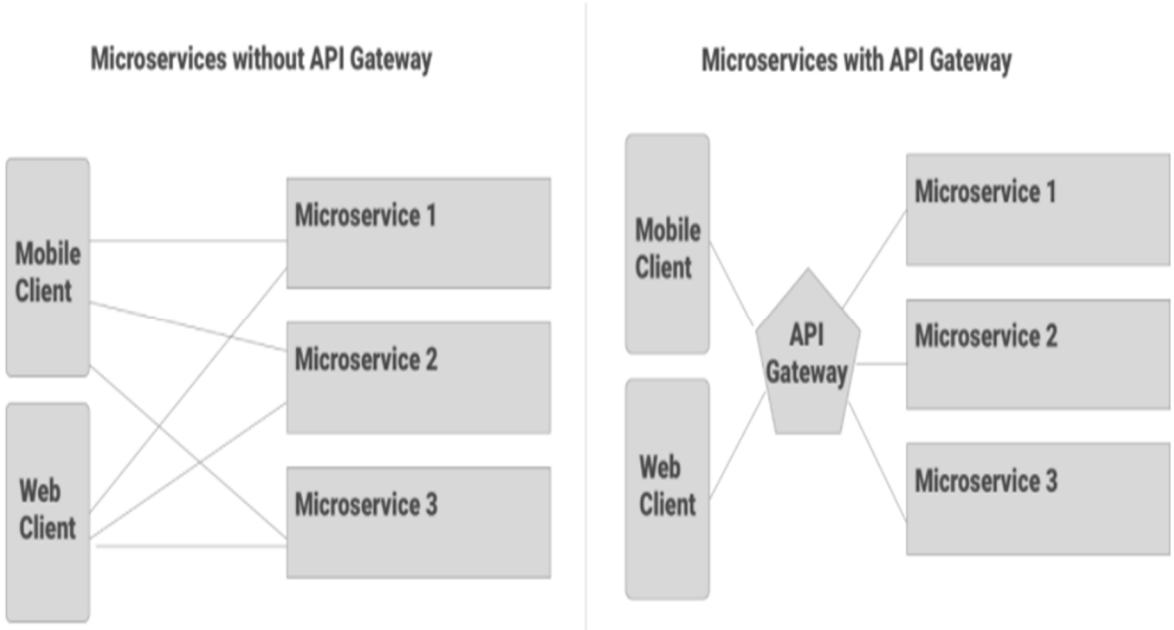


Fig.3 API Gateway

An API gateway takes into consideration the number of requests a client or device can handle. To illustrate, a desktop can handle more requests without compromising performance. However, for the same request, a mobile or tablet will perform poorly due to low bandwidth. Depending on whether service is being called by a mobile or a computer, API gateway offers a finer grained API to clients using a computer and a coerce, grained API to clients using mobile devices [5].

D. Database

In microservices architecture each service has its own database to store data. Each service uses its private database to complete its service specific task. This is done to hold up its decentralization aspect, however, in doing so, it becomes difficult to manage these databases. The problem is one service cannot update the database of another service, making it difficult to implement query operations among several services.

Microservices prefer letting each service manage its own database, either different instances of the same database technology, or entirely different database systems — an approach called Polyglot Persistence [1]. There are some patterns to be considered when adopting databases for microservices like Database-per-Service and Database-is-the-service [6].

©IJRASET: All Rights are Reserved

E. Containers

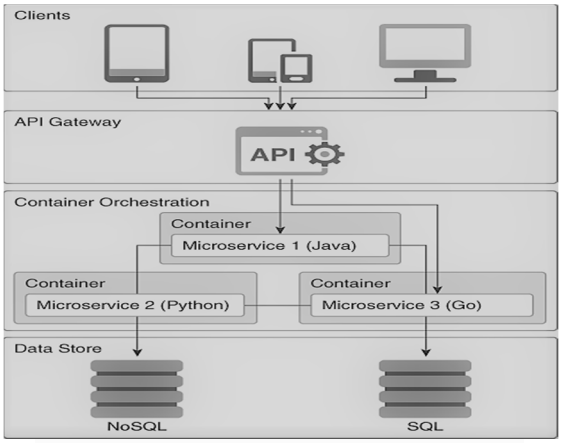


Fig.4 Containers in microservice architecture

Containers need not necessarily be a part of microservice architecture but significantly ameliorate speed and efficiency. They are packages consisting of a singular part of system i.e. microservices. Also, they are not allowed access to rest of the architecture, but only microservices and its dependencies. Containers are independent like microservices, thereby aiding in scaling of particular features.

F. Service Mesh

There is rapid growth of service mesh adoption in microservices architecture with no intention of slowing down. Service mesh is used for secure inter-communication between different microservices in a distributed application. For smooth and safe communication, service mesh residing in the infrastructure layer help in networking by managing, observing and securing services. Some positives of service mesh are as follows.

1) Traffic Management
2) Access Control
3) Routing
4) Load balancing

The service mesh component is made up of a data plane and a control plane [7]. The data plane deals with the actual traffic between services and networking aspects of service requests. The control panel links the data planes to form a distributed network [7]. Service mesh operates in a sidecar proxy pattern. The sidecar proxy is deployed along with a service instance. If two services need to communicate, the proxy at the source intercepts the request and routes it to the destination proxy before passing it on to the destination service [8].

G. Service Discovery

Another component in the microservice architecture is the service discovery. Service discovery is the process of locating existing services that are relevant for a given request based on the description of their functional and non-functional semantics [11]. It maintains a record of services and aids a service to find the route between services. Service discovery is made up of

1) Service directory
2) Service provider
3) Service consumer

Service providers register their services with one central directory or multiple distributed service directories. Service consumers are informed about available services in the network through the directory [11]. While making a request there are two patterns in service discovery

a) *Client-side Discovery:* Scans the directory for provider then selects suitable service to make a request using load balancing
b) *Server-side Discovery:* Request is made via a router which then searches the directory for required service. Once the service is available, the router forwards the request.

Service discovery handles the load during fluctuating workloads or updates by keeping track of microservices in the network.

©IJRASET: All Rights are Reserved

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 8 Issue X Oct 2020- Available at www.ijraset.com

V. TRANSITIONING TO MICROSERVICES ARCHITECTURE

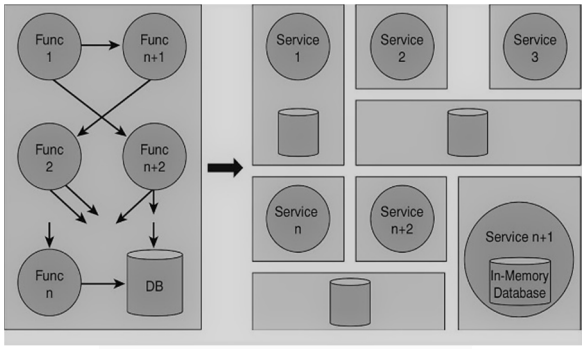


Fig.5 Monolithic to Microservices

Wanting to adopt a microservice architecture for your monolithic application is understandable given the benefits offered by the former. Figure5 visualizes the transition to microservices architecture. To adopt a microservice architecture we need to consider the following steps

A. Split the Monolith

The first step in migrating to microservices architecture involves analysing the monolithic application and identifying the components and inter-relation between various components. Identify components that require frequent updates and reconfigure them as a new service. This is a tedious process that needs to be done slowly without making major changes to data and logic. However, if monolithic is apt for a component, there is no need to force it into a microservice.

B. Inter-service Communication

The divided monolith components need to communicate with each other. Simple communication protocol should be used for inter-service communication, like REST. An API gateway can be used to commune between microservices. It is used for transport security, authentication, request dispatching, etc. API gateway and its functions have been discussed in the prior section in this paper.

C. Divide the data

It is likely that a monolithic application will have a central database, or some components of the application might have shared databases. Nonetheless, it is required for microservices to have private databases. In doing so, data is duplicated and to avoid duplicated data across multiple databases event driven architecture should be used. In EDA all databases are updated based on the action performed by services. We can also refactor the databases to separate the schemas [10].

D. Prepare for Failure

Failure is unavoidable in any system and one always needs to be prepared to avoid affecting the entire system. Microservices architecture provides with failure isolation by putting in place mechanisms to deal with such errors. Some of the mechanisms are timeouts and circuit breakers. Timeouts are something it is easy to overlook, but in a downstream system they are important to get right. How long can I wait before I can consider a downstream system to actually be down? Put timeouts on all out-of-process calls and pick a default timeout for everything. Log when timeouts occur, look at what happens, and change them accordingly [10].
Circuit breakers shields a service by monitoring service calls. When failure occurs, breaker is tripped calls are redirected and a default service or error message substitutes the failed service. The circuit breaker periodically allows request to check if service has recovered. If yes, then the service is reset. Some other mechanisms for failure isolation are rate limiters and bulkheads [10].

E. Containers for Testing

At every point during the transition, it is necessary to test the system in order to avoid failures in microservices and to make sure that the function of component is unchanged. Container orchestration is one way to do this, where integration tests can be performed without affecting the entire application. Docker provides containers for testing in a uniform isolated environment.

©IJRASET: All Rights are Reserved

F. Continuous Delivery

Microservices architecture allow for services to be integrated and deployed independently. Continuous delivery is a method to deploy microservices into fine-grained architectures. Continuous delivery is the approach whereby we get constant feedback on the production readiness of each and every service, and furthermore treat each and every service as a release candidate [10]. When a service is developed or modified, a CD pipeline execution is automatically triggered. The service passes through a series of stages that check the quality of service code. If the change fails to pass any of the pipeline stages, the pipeline process is aborted, and the developers are notified to fix the problems. If the service passes all these stages, then the service can be deployed [12].

G. Monitoring

When we move to microservices from monolithic, we divide the structure into small parts as a result, it becomes necessary to monitor multiple servers, multiple log files to sift through, and multiple places where network latency could cause problems. To monitor services Splunk and Logstash are tools available for log management and storage. S. Newman provides substantial methods for monitoring in his book *Building microservices: designing fine-grained systems* [10]. Splunk and Logstash are tools available for monitoring.

VI. IMPLEMENTATION TOOLS

A. Programming Languages

For microservices, few things one should consider before selecting programming languages are.

1) To ensure time conservation, it is important that one selects the language which has less code to type. So, C should be avoided.

2) Choose a language that gives you speed and can control traffic.

Some characteristics to consider, for programming languages before implementing Microservices are:

a) Consumer first approach

b) Independent Deployment

c) Cultural Automation

d) Highly Observable

Some of the top languages to use for microservices are as follows-

- *Golang:* Golang or GO is one of the trending languages for developing microservices. This language has easy syntax which is easier for developers. Also, this language has the capacity to manage highly loaded apps, so the developers can make apps which can handle high load. Go also has high speeds. Go also has inbuild standard libraries and has several packages which helps developer to implement their products faster. There are few cons about Go like there is no manual memory management. Also, it focuses on speed rather than safety. Go Micro and Go kit are the frameworks available for microservices [3].
- *Python:* Python is a language which is easy to learn, easy to write clean indented code. One advantage of python is the availability of copious inbuilt functions which helps the developer to write less code than many other languages. Python provides frameworks like Flask, Falcon and CherryPy for microservices development [3]. It also has great speed and performance due to strong enhanced control capabilities and unit testing framework. Monitoring, testing and integration becomes uncomplicated with python. Nevertheless, there are some disadvantages of python like lack of security and slower execution speed.
- *Java:* Java is one of the most popular language as it has many libraries which are popular among microservices developers. It is very useful when working with complex systems as it provides readability. Java virtual machine provides developers to use a different language anywhere else. Frameworks available are Spring Boot, Restlet and Spark [3].
- *Ruby:* Ruby is an object-oriented language and is compatible with several other languages. It has great speed, flexibility, and quality of code. It is reliable and consistent due to which is used in many startups.

B. Technologies

1) *Docker and Kubernetes:* Docker is a software platform used for building, testing and deploying software as containers. These containers can be used anywhere, and all the containers contain all the resources needed for running. Kubernetes is the complement to Docker. It is used to solve some operation complexities when scaling multiple containers. Kubernetes dynamically allocates resources to improve infrastructure utilization. All dependencies of microservices are placed into a Docker container and isolated from the rest. Docker is a containerization tool used in contrast to virtualization. Docker comes with its own ecosystem. Due to the technologies built round it is a viable option for companies that decided to use container to run microservices. They are known to facilitate processes like rolling updates and automated scaling. [13]

©IJRASET: All Rights are Reserved

2) *Redis:* Redis is one of the most popular NO-SQL databases. It is an in-memory data structure store, but it performs well as compared to traditional database systems. In Redis you don't need to worry about record locks for database. Redis uses its own hashing mechanism. Redis is popular for speed to query for any data.
3) *Consul:* Consul ensures communication between microservices. Consul is better than other services discovery solutions concerning features like:

a) HTTP Rest API and support for DNS.
b) It can auto generate configuration files.

Consul can be used with many technologies due to availability of DNS interface and Consul template. Consul offers the following services:

- *Service Discovery:* This feature is useful for adding new technologies to your microservices.
- *Configuration:* Consul is used for the configuration of microservices.
- *Load Balancing:* Consul uses Load Balancing transparently with the DNS server with the help of Consul DNS.

4) *Prometheus:* Prometheus is an open source monitoring solution. It is developed by SoundCloud. It is mostly used to store and query 'time-series data'. Tools like Grafana is often combined with Prometheus to visualize time data and provide dashboards.

VII. PITFALLS OF MICROSERVICES ARCHITECTURE

Like every other development method, microservices architecture too has drawbacks. Even though microservice architecture offers a myriad of benefits, it has pitfalls that need to be considered before rushing into this architecture.
Firstly, inter-service communication becomes complex when there are multiple services in play, and it becomes difficult to handle requests which might lead to network latency and poor performance. Multiple databases and traffic control do not make things easy and multiple microservices require just as many resources. Testing every service becomes a laborious task due to multiple nodes. It becomes time-consuming and complicated to use microservices architecture service on small scale.
Even though, microservices outstrips monolithic, it does have its cons. Despite the drawbacks, microservices offer a better development path.

VIII. FUTURE SCOPE

Despite getting a significant amount of attention lately, the microservice architecture is fairly young. Nevertheless, it shows immense promise in the development of applications. This architecture improves the performance of an application along with its efficiency giving it an edge. Further on, improving routing, inter-service communication and service discovery on top of smoothness in management and delivery, monolith will find it difficult to keep up. The future of this implement is multifaceted and copious, beginning with deeper integration in the cloud domain. With microservices, developers will move past locally hosted apps and enter the cloud. Microservices will provide scalability to cloud and its services [9]. Microservices too will embrace the cloud by going serverless, avoiding pre-allocated server resources and instead use only the server resources you need, saving cost [9].
With more businesses using microservice architecture, sharing of microservices seems like a likely outcome. To be able to share microservices to be reused or repurposed by tweaking the existing microservice and its dependencies. This reusability saves resources development time [9]. Based on user response, the need to update the applications frequently can be met without much difficult.

IX. CONCLUSION

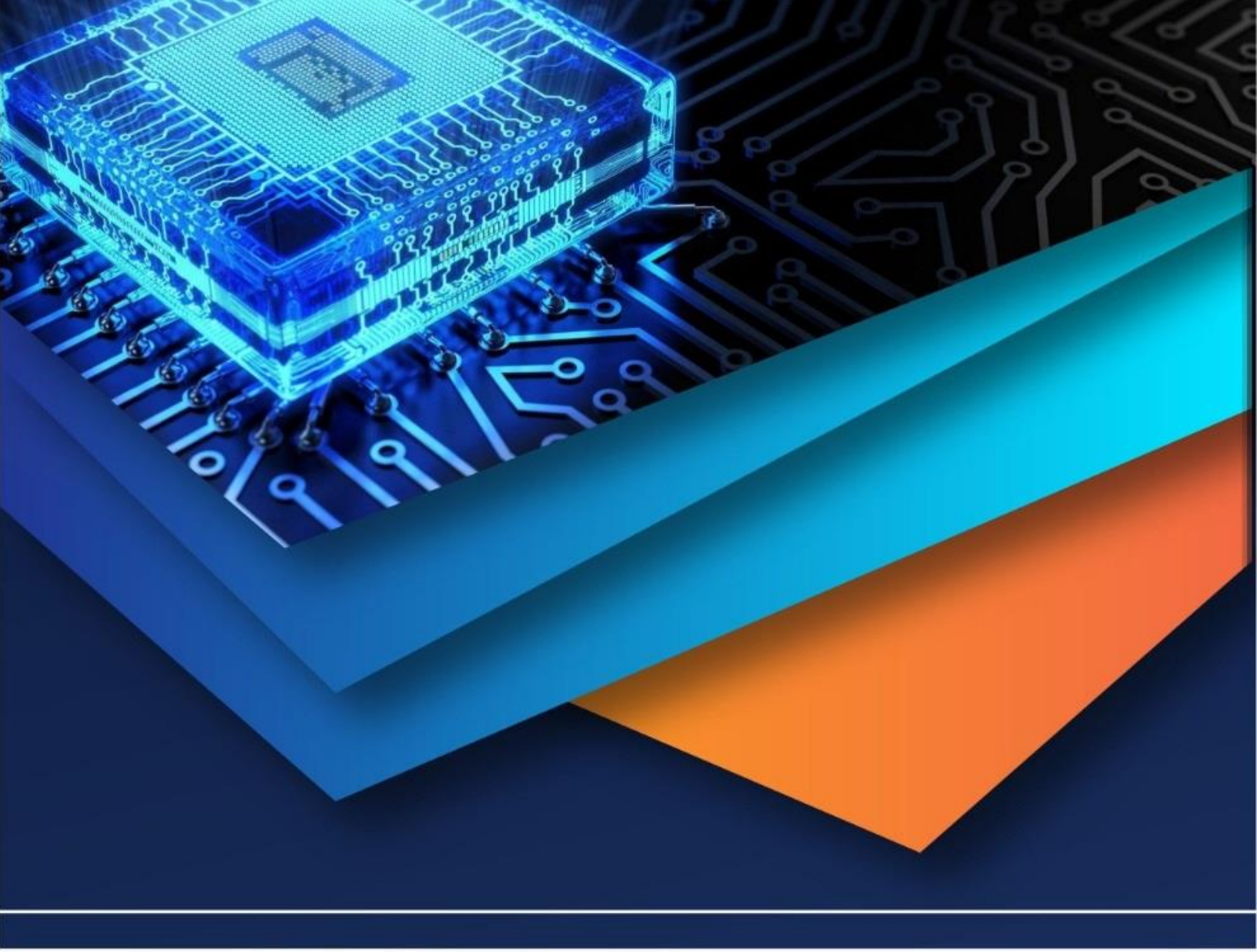
Microservices architecture serves as an alternative to monolithic architecture. In monolithic architecture, the entire application is developed using a single code base for all components. This architecture has many limitations. Thereby, we adopt the microservices architecture which offers scalability, flexibility, resilience and autonomy.
In this paper, we take a look at the architectural components of microservices system and things to do and consider when migrating to a microservice system from a monolithic application. We expatiate about inter – service communication, containers and tools required to put into effect microservices style of architecture. These tools include programming languages like Golang, Python and their frameworks for development. Furthermore, technologies like Prometheus and Docker & Kubernetes for different components of microservice architecture.
With ever evolving technology and demanding user experience, microservices will provide robust solutions in software development. Microservice is still adolescent and with immense scope of development and research, is likely to have a bright future ahead.

©IJRASET: All Rights are Reserved

REFERENCES

[1] Fowler, M. and Lewis, J. (March 25, 2014.) "Microservices," https://martinfowler.com/articles/microservices.html
[2] What are Microservices and Containers. https://avinetworks.com/what-are-microservices-and-containers/
[3] 5 Best Technologies to build Microservices Architecture. https://www.clariontech.com/blog/5-best-technologies-to-build-microservices-architecture
[4] Pahl, Claus & Jamshidi, Pooyan. (2016). "Microservices: A Systematic Mapping Study," 137-146. 10.5220/0005785501370146.
[5] Pachghare, Vinod. (2016). "Microservices Architecture for Cloud Computing," Journal of Information Technology and Sciences. 2. 13.
[6] Messina, Antonio & Rizzo, Riccardo & Storniolo, Pietro & Urso, Alfonso. (2016). "A Simplified Database Pattern for the Microservice Architecture," 10.13140/RG.2.1.3529.3681.
[7] George Miranda, "The Service Mesh," 2018, ISBN: 978-1-4920-3131-4 [Online]. Available: http://www.safaribooksonline.com/library/view//978149203131/?ar
[8] Lee Calcote, "The enterprise path to service mesh architecture: decoupling at layer 5,". 2018 [Online]. Available: http://proquest.safaribooksonline.com/?fpi=9781492041795.
[9] Tom Smith, "What is the future of microservice?" 3.2018 [Online]. Available: https://dzone.com/articles/what-is-the-future-of-microservices
[10] S. Newman, "Building microservices: designing fine-grained systems," Sebastopol, California: O'Reilly Media, 2015
[11] Klusch, Matthias. (2014). "Service Discovery," 10.1007/978-1-4614-6170-8_121.
[12] Chen, Lianping. (2018). "Microservices: Architecting for Continuous Delivery and DevOps," 10.1109/ICSA.2018.00013.
[13] Hamzehloui, Mohammad & Sahibuddin, Shamsul & Ashabi, Ardavan. (2019). "A Study on the Most Prominent Areas of Research in Microservices," International Journal of Machine Learning and Computing. 9. 242-247. 10.18178/ijmlc.2019.9.2.793.

©IJRASET: All Rights are Reserved



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)