# Reactive Microservices Architecture Using a Framework of Fault Tolerance Mechanisms

**Conference Paper** · August 2021

DOI: 10.1109/ICESC51422.2021.9532893



**2 authors:**

J. ABDUL Rasheedh

**5** PUBLICATIONS **8** CITATIONS

SEE PROFILE

Dr S Saradha

Vels University

**7** PUBLICATIONS **19** CITATIONS

SEE PROFILE

Proceedings of the Second International Conference on Electronics and Sustainable Communication Systems (ICESC-2021)
IEEE Xplore Part Number: CFP21V66-ART; ISBN: 978-1-6654-2867-5

# Reactive Microservices Architecture Using a Framework of Fault Tolerance Mechanisms

J Abdul Rasheedh,
Research Scholar,
Department of Computer Science, VISTAS,
Pallavaram,Chennai, India
abdul_rasheedh@yahoo.co.in

Dr. S. Saradha[2],
Research Supervisor,
Department of Computer Science, VISTAS,
Pallavaram,Chennai,India
saradha.research@gmail.com

2021 Second International Conference on Electronics and Sustainable Communication Systems (ICESC) | 978-1-6654-2867-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICESC51422.2021.9532893

***Abstract*—In Cloud Computing, microservices have been recently introduced for enabling the development of large-scale structures, which are scalable, agile and especially suitable for meeting the emerging demands. The asynchronous communication has facilitated using reactive system which managed in interaction challenges and even variation of load in modern systems. There are certain features of microservices like elasticity and resilience have considered for messages can be progressed through Reactive Microservices (RM) which consists of segregated components over event stream that can able to perform individually or shown with various microservices for reaching the event at final stages. The reactive principle in microservices have involved for individual possibility in every microservice components reference architecture include the components of microservices, which have the ability to develop, release, organize, scale, upgrade and retired individually. It has infused the necessary redundancy into the network for avoiding failure cascading, which maintains the system reactive in case of any failure. This paper has explored the RM architecture application and provides insights by assisting in reducing the developing demand to create resilient and scalable systems. The basic use case of microservice framework implementation has been illustrated by Vert.x, which act as the general toolkit to create RM and also the performance of both reactive and nonreactive implementation are compared as an alternate. The performance comparison can reveal how RM performs better than non-reactive microservices.**

***Keywords—Reactive Microservices (RM), fault tolerance, Vert.x, failures***

## I. INTRODUCTION

During earlier, applications have been established by monolithic methods, which represent a single code-base utilized for executing the whole applications. When the monolithic application design is considered to be slightly complex compared with high distribution method but executing it has faced several challenges. There are some inadequacy in significant features of monolithic applications such as elasticity and scalability [1]. In case of monolithic application which gets enhanced in a single piece that creates complex for modifying and this complex progress from high coupled nature of those applications. However, scaling of monolithic application is other main obstacle because of its huge volume of data. Therefore, the application of monolithic has been scaled completely while a specific portion required for scaling. Moreover, these problems may urge for introducing a novel approaches that has driven the software industry to search for an option whereas the microservices are considered to be a best solution in addressing various preceding problems. Thus, microservices has performed as set of services which assigned to manage together for designing an applications and every services can built for executing a single project [2]. In case of the smaller sizes implementation and restoration is simple and it empower the organization for accomplishing tasks but in other hand complex if feasible with monolithic applications. Technology diversity is the major feature in architecture of microservices that contribute better performance and it enhance to the programming language. Moreover, the service has been considered along with its technique in implementing the embodied ability using the components of microservices in the reference architecture and the communication pattern among the components. The application and system has implemented a reactive principle which is flexible, reliable, loosely coupled and scalable. It consists of tolerant in failures and also failure occurs in responded gracefully instead of catastrophic crash. In addition, the reactive system is generally high reactive that offer with high user experience which is loosely-coupled, scalable and flexible. "The Reactive Manifesto" [3] has discussed about four significant characteristics of reactive systems are given below

*Responsive:* This assist in predictability with high degree.

*Event-driven:* This helps in passing the asynchronous message with subscribed model or published model.

*Scalable:* This characteristic has the ability for performing service with growing load when maximizing the usage resources.

*Resilience:* This assist in isolating the faults.

Hence, it makes quite easier for developing and responsive to modify whereas RM is an independent which has capability in accommodating to the services of availability or absence that surrounding them. Thus, the isolation is corresponding to independence and RM has ability to manage failure locally, performs independently and gets collaborated with other based on the requirement. The utilization of RM for interacting an asynchronous message that passed is communicate with it peers and receives messages which even have the capability in generating responses for those messages. This study explored an introduction of the microservices architecture by reactive applications are the feasible solution and presents novel possibilities to minimize the improving demand for flexible and robust structures to be created. It also highlights a specific use case for implementing microservices, contrasting the performance of alternatives to both reactive and non-reactive implementation. The organization of this paper is as follows: Section 2 describes the related survey regarding technique based methodological contributions from existing work, Section 3 describes the proposed methodology based on RM Architecture via Vert.X , Section 4, discusses performance based on throughput , Section 5 concludes the work.

## II. LITERATURE REVIEW

Santana et.al has proposed a model by enhancing IoT application reliability for RM based on availability aspect [4]. Similarly the researcher Lirade Santana et.al is discussing about microservices that may be distributed at the network edge for acknowledging system resilience and elasticity [5]. Rosa Righi et.al has addressed the ability to withstand failures of both internal and external based on flexibility and elasticity. It gets associated with the systems capability for performing in accordance with demand variations [6]. The framework may utilize asynchronous messages for guarantee less coupling, segregation and contact habit transparency to resilient. DevOps, cloud, virtualization and its relationship with microservice are the three main platform plays a major role to implement microservice is introduced by Hamzehloui.et.al [7]. Moreover, the microservices are associated with virtualization but it may release the actual microservice power. The applications of traditional development method is executed with DevOps may be complex in execution. Thus, the distributed components by cloud may be quiet easy when communicating microservice interest. Samanta et.al[8] has proposed FLAVOUR which is a novel scalable and latency-optimal microservice that act as a scheduling method to an edge computing network for accomplishing minimum service latency, when provided with approved transmission rate to decrease the MicroService Completion Times (MSCT). Power and Kotonya has suggested a pluggable platform depending upon microservices architecture which integrates FT support as two comparable microservices: In Real-time FT detection has utilized dynamic event analysis in other hand Online ML is utilized for predicting fault patterns and resolving faults pre-emptively before they can be accessed [9]. Sun et al.[10] has proposed an architecture with microservices based IoT which disintegrate the network into nine units which interact in a REST interface whereas the core microservice is controlling the other eight that provide storage, security, bigdata analytics, etc. They have utilized microservices as it allows the platform to quickly extend, develop, and incorporate third party applications for enabling interoperability and scalability, with better capability to implement fault tolerance over scale. Celesti et al. has discovered a service for containerized microservices in monitoring software that is built on IoT devices as middleware [11]. The failure of microservice has repaired or replaced with clone whereas the solution has illustrated an adequate overhead at recovery. The researchers Goel and Nayak have introduced an architecture of microservice which focused on event streams and it has a decentralized the framework of microservice coordination [12]. The microservices architecture enables for enhancing information over event stream without any limitations on time or versatility. Orchestration includes a conductor, which act as a central service which sends requests to other services and by receiving replies supervises the operation. In order to build collaboration, choreography performs as a design that has not considered centralization which utilized the mechanism of events and publishes or subscribes [13] [14] [15]. The researcher Dastjerdi, R. Buyya have addressed reactive fault tolerance whereas this method get initiated with strategy of error recovery once an error recognized which need quick detection and decision making with the connection of low-latency for the hardware or software at fault. The fog can able to provide service within cloud platform for network edge with data analytics of low-latency providing it an optimal applicant to analyze stream data [16]. Al-Fuqaha et.al has addressed several architecture of microservice that communicate among services gets managed through a style of RESTful architecture whereas the data is shared by JSON format. There are some other protocols applicable to IoT involves CoAP, MQTT and XMPP, but the benefit of REST can be considered to all cloud services that provide by accomplishing it with an adequate option to promote interoperability over IoT system [17]. The researchers Pedro and Luca has developed particularly over IoT middleware layer when this paper proposal focused on the IoT architecture of application layer [18] [19]. Similarly, this researcher has focused on this research based on the Microservices orchestration is incorporated in containers which can be conducted from Egde to Fog and Cloud [20].

In addition, the RM is considered to be elastic which assist the system by providing an ability to handle the load of several instance which signify constraints set namely eliminating in-memory state, sharing of state among instance when needed or it has capability in routing messages for the similar instance to massive services. It cannot be suited for existing legacy or prevailing systems.

## III. MICROSERVICES ARCHITECTURE

Microservices architecture is architectural pattern to develop an application of software as a limited suite with autonomous services. It deliberately implemented on various solution stacks and also executed in several programming languages that can able to execute together by its certain process. In general, lightweight communication protocol isusedtocommunicatethe architecturebyeitherHTTPrequestre sponseswithAPIresourcesorlightweightmessaging. Microservicearchitecturaltechniqueiscomparativelybetterthan monolithicarchitecturaltechniquefor application development whereas, a single deployable unit is deployed and scaled to the full framework. The company logics in monolithic architecture have been bundled over a single package and implemented as a single operation. The applications are thus scaled by horizontally executing many instances.

### Reactive Systems

Systems or applications are created based on reactive principles with loosely-coupled, flexible, scalable and reliable which generate it as simple as to improve and responsible to modify. Therefore, the failure tolerance is highly significant and it gets gracefully while the failure occurs instead of catastrophically failing. In addition, the reactive system is highly responsive in providing users with a better user experience.

The reactive technique features has determine the reactive manifesto using four basic principles:

- *Responsive:* Responds promptly in a timely manner and respond frequently has established its upper bound reliability and delivered a constant Quality of Services (QoS).
- *Resilient:* According to the reactive failure, the presence of resilience may get accomplished by isolation, containment, replication and delegation.
- *Elastic:* Is responsive while varying workloads and has ability in reacting to modify with request load

using decrease or increase of resources which allocated for servicing these requests.

- *Message driven:* In order to establish the boundary between components using asynchronous message passing. It supports several characteristics like location transparency, isolation, and loose coupling.

*A. Reactive Microservices:*

The principle of reactive microservice technique has provided specific possibilities in which each part of the reference architecture present in microservice is involved. Components, are independently scale, develop, retire, released, deploy and update which has introduced the needed resilience into the method to avoid failure cascading which maintain the method as responsive while failure occurs. In addition, the communication of asynchronous has aided using reactive technique with communication challenges and load variation in modern systems. The implementation of reactive microservice framework is Node.js, Qbit, Netflix Spring Cloud, etc. but Vert.x has been considered as a basic toolkit to create RM.

*B. Implementating Reactive Microservices Architecture Via Vertx:The word "data" is plural, not singular.*

The toolkit Vert.x has performed to create reactive and distributed applications with the top of Java Virtual Machine (VM) by non-blocking asynchronous growth model. It is created for implementing an event bus that act as a broker for lightweight message and it's enabled several components of the application in communicating over non-blocking and thread safe manner. Vert.x has offered several components to create RM based applications.

The Vert.xVerticle is a logical code unit has capable for implementing over Vert.x which is proportionate to actor in the actor model. The similar Vert.x application has been collected in several verticle cases available in each vert.x. Therefore, the verticle has ability for communicating each by sending messages over event bus.

The Vert.xevent Bus perform as a lightweight distributing messaging technique which involves various application parts or dissimilar applications and services for communicating with each other over loosely coupled method. The event bus is experienced with several messaging such as point-to-point, request response and publish subscribe. In the event bus, the verticle has performed to address send and listen information whereas it is also called as channel. If a message is sent to the specific address then all Verticles which observe over specific address that has been received the message. The key features of Vert.x are suited for microservices architecture

- *Lightweight:*Based on the distribution and runtime footprint, the Vert.x core has used limited size as 650kB and light weighted which is completely integrated with existing applications.
- Scalable: Vert.x has ability to scale horizontally as well as vertically and it create cluster by Hazelcast or JGroups. It has the capability to use all CPUs in the Processor for the machine or cores.
- *Polyglot:*Java, JavaScript, Ruby, Python and Groovy can be executed by Vert.x. through an event bus presented in different languages and the Vert.x components have communicate with each other easily.
- *Fast, Event-Driven and Non-blocking:*However, there is no thread in Vert.x APIs block. Therefore, the Vert.x application can able to maintain a better adequacy with less thread counts. Thus, it provide an experts thread for handling the blocking calls.
- *Modular:*The runtime of Vert.x is segregated into several components whereas it is needed and available to the specific implementation to utilize.
- *Unopinionated:*Vert.x is not a container or disruptive method but doesn't support an appropriate way of describing the application. Conversely, the Vert.x is offering various modules and empowered developers to build their individual applications.

Moreover,
thisimplementationincludestheinvokeofanexternallyappli cablemicroservice,
requestingfiveindividualmicroservices,
computingtheresponsefromallthesesservicesandreturntheca llertothecompositeresponse.TheFlightInfoServiceis
a hybrid service which intended to be deployed by web portals and smartphone applications. Hence, it is intended to access information related to a respective flight such as flight timetable, current status of the flight, individual aircraft, and weather reports at the airports of arrival and departure.

Our Approach

*Setup: Structural and core infrastructure, along with all organizational government initiative (for application navigation, tracking, call monitoring, application development, hierarchical log tracking and security), were deployed in remote backup virtual servers as individual Java applications.*

*Process: the FlightInfoService composite service was invoked using Apache JMeter. With identical components running on the same virtual machines, the three scenarios (Spring Cloud Netflix-Synch, Spring Cloud Netflix Asynch and Vert.x) were executed serially in the same environment.*

*Configuration settings: Composite and core services with their default configuration options, along with all governance components, have been executed.*

The implementation of asynchronous and reactive message in Vert.x has exchanged the pattern among core and composite services is shown in Figure I. The microservices are implemented as Vert.xVerticles.

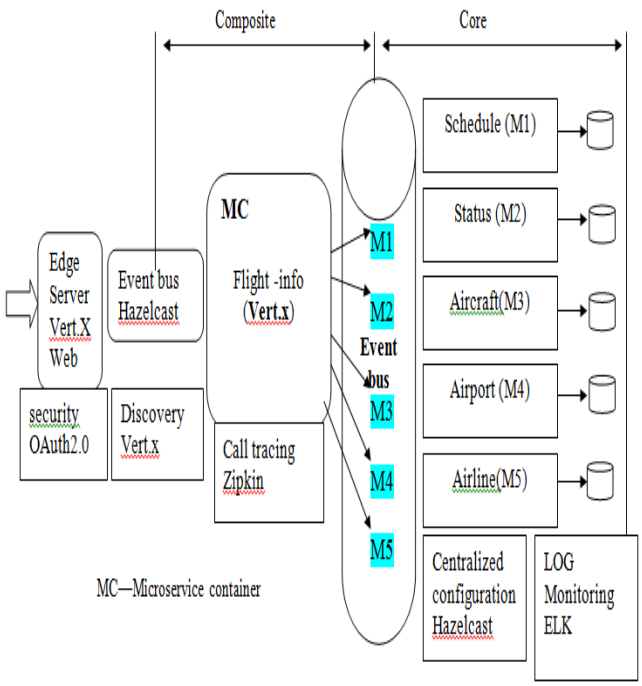


Fig. 1. Block diagram for Reactive Microservices Architecture Via Vert.X

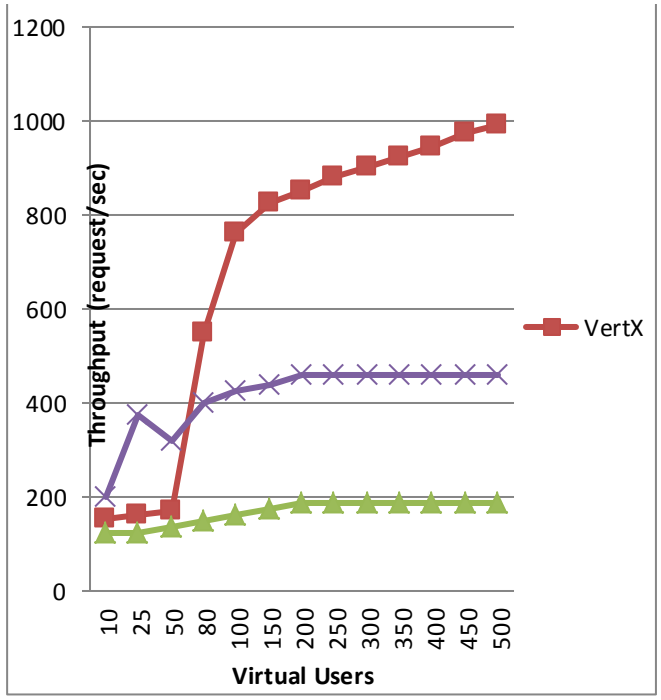


Fig. 2. Comparison of throughput performance between nonreactive and reactive implementation alternatives

The design of RM frameworks by Vert.x consists of subsequent features in which certain features are promising for the deployment of enterprise-scale microservices:

The components present in the vert.xmicroservice ecosystem which consists of composite and core microservices and various operational components for communicating each other Via the Event Bus introduced by a Hazelcast cluster.

Vert.x has facilitates polyglot programming, indicating that multiple microservices, namely Java, JavaScript, Groovy, Ruby and Ceylon, are being developed in various languages.

1. Vert.xhas generated automatic fail-over as well as load balancing out of the box.

2. Vert.xassists in establishing various components of usual microservices reference architecture need to be created. Thus, the microserviceshave been implemented as Verticles.

## IV. Comparison Based on Throughput

Figure IIshows the throughput comparison between Spring Cloud Netflix (Sync and Async-REST) and the Vert.x implementation

*Spring Cloud Netflix–Sync implementation:* The request number progressed per second has been observed as the least from the three sequences of events. Hence, it illustrate that the pools of server request thread are enervated as a complex for core service have invocate by blocking and synchronous. Thus, the processor and use of memory are not significant. Similarly, the I/O operations are blocked by the central and composite services.

*Spring Cloud Netflix–Async-REST implementation:* The Asynch API of the Spring Framework and RxJavaare assist for incrementing the progress of maximum requestsper second. This is due to implement of core and composite services are utilized in the Spring Framework's alternative technique for asynchronous request processing.

*Vert.x implementation:* The request number progressed per second has been observed as the least from the three sequences of events because of Vert.x'sevent-driven and non-blocking in nature. Vert.x uses a paradigm of an actor-based interoperability which considers it as a multiple pattern which utilizes various case sequences through available cores on VMs. Hence, the edge server is based on the non-blocking Vert.x Web module. Rather than conventional synchronous HTTP, Async RPC has been utilized for compositing the core service requests that resulted with better throughput. When comparing with blocking JDBC requests, asynchronous API is performing faster in interaction with database for Vert.x JDBC clients. Thus, the entire scales are better with incremental of virtual users.

## V. Conclusion

The principle of reactive is not the recent one but it has been utilized and determined over a decade. The efficient influence of RM concepts with modern platform of software and hardware is for delivering autonomous microservices, loosely coupled and single responsibility by asynchronous message passed to

interservice communication. This research has been implemented by highlighting the adoptive reactive microservices feasibility applicable for implementing ecosystem of microservice to an enterprise application with the development of Vert.x as a fascinating reactive toolkit. Vert.x provides its own fault tolerance mechanisms in Vert.x Circuit Breaker project. It is a simple implementation of the circuit breaker pattern for Vert.x that keeps track of the number of failures and opens the circuit when a threshold is reached. This research has underscored the better operational effectiveness, which can be acquired from reactive microservices.

## ACKNOWLEDGMENT *(Heading 5)*

The preferred spelling of the word “acknowledgment” in America is without an “e” after the “g”. Avoid the stilted expression “one of us (R. B. G.) thanks ...”. Instead, try “R. B. G. thanks...”.Put sponsor acknowledgments in the unnumbered footnote on the first page.

## REFERENCES

[1] D. Jaramillo, D. V. Nguyen, and R. Smart, “Leveraging microservices architecture by using Docker technology,” SoutheastCon, 2016.
[2] A. D. Camargo, I. Salvadori, R. Mello, S. Dos, and F. Siqueira, “An architecture to automate performance tests on microservices,” in Proc. 18th Int. Conf. Inf. Integr. Web-based Appl. Serv, 2016.
[3] J. Bon´er, D. Farley, R. Kuhn, and M. Thompson, “The reactive manifesto,”
Dosegljivo: http://www. reactivemanifesto. org/.[Dostopano: 21. 08. 2017], 2014.
[4] Cleber Santana, Leandro Andrade, Brenno Mello, Ernando Batista, José VitorSampaio and CássioPrazeres, “A Reliable Architecture Based on Reactive Microservices for IoT applications”, Association for Computing Machinery, ISBN 978-1-4503-6763-9/19/10, 2019.
[5] Cleber Jorge Lirade Santana, Brennode Mello Alencar, and CássioV.SerafimPrazeres, “Reactive Microservices for the Internet of Things:A Case Study in Fog Computing”,2019, https://doi.org/10.1145/3297280.3297402
[6] Rodrigo da Rosa Righi, Everton Correa, M Ãąrcio Miguel Gomes and Cristiano AndrÃľda Costa, “Enhancing performance of IoT applications with load prediction and cloud elasticity”, Future Generation Computer Systems, 2018, pp-1–13. https://doi.org/10.1016/j.future.2018.06.026.
[7] Mohammad SadeghHamzehloui, ShamsulSahibuddin, and ArdavanAshabi, “A Study on the Most Prominent Areas of Research in Microservices”,International Journal of Machine Learning and Computing, Vol. 9, No. 2, April 2019.
[8] Amit Samanta, Yong Li and Flavio Esposito, “Battle of Microservices: Towards Latency-Optimal Heuristic Scheduling for Edge Computing”, IEEE,2019.
[9] Alexander Power and Gerald Kotonya, “ AMicroservices Architecture for Reactive and Proactive Fault Tolerance in IoT Systems”, 2018.
[10] L. Sun, Y. Li, and R. A. Memon, “An open iot framework based on microservices architecture,” China Communications, vol. 14, no. 2, pp. 154–162, 2017.
[11] A. Celesti, L. Carnevale, A. Galletta, M. Fazio, and M. Villari, “A watchdog service making container-based micro-services reliable in iot clouds,” in 2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud). IEEE, 2017, pp. 372–378.
[12] DivyaGoel and Amaresh Nayak, “Reactive Microservices in Commodity Resources”, IEEE, 2019.
[13] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, “Microservices: yesterday, today, and tomorrow,” in Present and ulterior software engineering. Springer, 2017, pp. 195–216.
[14] C. K. Rudrabhatla, “Comparison of event choreography and orchestration techniques in microservice architecture,” International Journal of Advanced Computer Science and Applications, vol. 9, no. 8, pp. 18–22, 2018.
[15] J. P. Macker and I. Taylor, “Orchestration and analysis of decentralized workflows within heterogeneous networking infrastructures,” Future Generation Computer Systems, vol. 75, pp. 388–401, 2017.
[16] A. V. Dastjerdi and R. Buyya, “Fog computing: Helping the internet of things realize its potential,” Computer, vol. 49, no. 8, pp. 112–116, 2016.
[17] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, “Internet of things: A survey on enabling technologies, protocols, and applications,”IEEE Communications Surveys & Tutorials, vol. 17, no. 4, pp. 2347–2376, 2015.
[18] Pedro M.TaverasNÃžÃśez, “A Reactive Microservice Architectural Model with asynchronous Programming and Observable Streams as an Approach to Developing IoT Middleware”, 2017.
[19]Lucas M.C.e Martins, Francisco L. de Caldas Filho,RafaelT.deSousaJúnior, William F. Giozza, and João Paulo C.L. da Costa, “Increasing the Dependability of IoT Middleware with Cloud Computing and Microservices”, In Companion Proceedings of the10th International Conference on Utility and Cloud Computing,2017.
[20] Salman Taherizadeh, VladoStankovski, and Marko Grobelnik, “A Capillary Computing Architecture for Dynamic Internet of Things:Orchestration of Microservices from Edge Devices to Fog and Cloud Providers”, Sensors vol-18,issues-9, 2018, pp-13–22. https://doi.org/10.3390/s18092938.