# Introduction

Many IT organizations have found that embracing service oriented and microservices architecture enables fast software delivery. Organisations have more freedom to react and make different decisions and it also facilitates the embrace of newer technologies which otherwise are difficult and challenging with monolith architecture. This shift from monolith to service oriented architecture provides flexibility, abstraction of underlying logic, loose coupling, reusability and discoverability [(Serrano et al., 2014)](https://www.zotero.org/google-docs/?HPXZXK). When the software architecture is broken down at the service level, every service may or may not need to communicate with other services in order to achieve the business outcome. The communication between the services can be achieved in different ways. There are different integration technologies, and common languages that are used for the communication between the services. The majority of this integration technology supports the general purpose programming language most commonly used for software development. The general purpose languages lack the behaviour required for the integrations technologies [(Montesi et al., 2014)](https://www.zotero.org/google-docs/?NYoSPD). This shortcoming is overcome in the programming language like Ballerina and Jolie that are specially developed for the services [(Weerawarana et al., 2018)](https://www.zotero.org/google-docs/?WHBP4b).

This research will provide a comparison between Ballerina, Jolie, and Java with the focus on the integration aspect of software development based on different parameters. The findings will help to explain which of the programming languages is the most suitable for integration technologies.

# Problems of General purpose Programming language for Integration

General-purpose languages are often used for the microservices development (Newman, 2015). Current programming languages like java, and C# do not demonstrate the capabilities required for the integration of microservices architecture, as they were not developed for building integration solutions. To overcome the shortcomings of these languages, developers use middleware systems like message brokers, ESB, transaction manager, etc. This behaviour is added as an afterthought into the middleware; hence the middleware system alone is not sufficient for all integration use cases. Sometimes developers also have to combine programming with the middleware system to achieve the business objective. This adds to the complexity of the system, which requires varied skill sets and infrastructure. Hence, it is a complex, and an error-prone endeavour to build integration solutions (Weerawarana et al., 2018).

Montesi, Guidi, and  Zavattaro (2014) believe that microservice designers have to deal effectively with both behavioural and the architecture aspects of the microservices architecture discussed earlier. This technical problem arises due to fragmentation and during integration, there should be a language specially designed for designers which let them focus on the single concept instead of the different tools and technology (Montesi et al., 2014). Thus making it easy for the developers thereby reducing the complexity and the maintenance of the application.

# Development of Jolie and Ballerina

Jolie and Ballerina are full-fledged programming languages designed for microservices architecture (Newman, 2015). Jolie enables the user to translate graphical design into code and Ballerina has graphical and textual support for programming, this helps different stakeholders with the different technical backgrounds to work and collaborate more effectively (Weerawarana et al., 2018). [MA1]

Unlike Ballerina and Jolie, other programming languages lack native constructs for workflows thus adding to the burden of middleware, ESB and API gateways (Weerawarana et al., 2018). Additionally, these languages unlike mainstream languages support visual programming and also sequence diagrams to reduce complexity. Furthermore, they are also capable of providing features like reliable delivery of messages and interruptibility which the mainstream languages fail to offer (Weerawarana et al., 2018).

However, even after the benefits that Jolie and Ballerina offer to developers, there is still reluctance to use the new languages. According to a survey carried out by DZone[MA2]  80% of the development is done in Java (Glen, 2018). Java is the oldest and most dominant software development language. There are many frameworks such as Spring Boot, Jersey, and Swagger that are developed to facilitate java for microservices development(DZone, IBM & 3Scale, 2016).

Little literature could be found dealing with the integration of the behavioral and architectural aspect of programming languages. No literature has investigated whether a single language can provide satisfactory support for the integration problems discussed earlier. According to the Ballerina’s FAQ Forum, there is no comparison yet carried out for Ballerina with other programming languages (Inc, n.d.) .

There should be a single language for developers and designers, instead of multiple tools and technology to favor one particular language.

# Research Problem

The main areas of interest for this research are the three languages - Ballerina, Jolie, and Java, and their effectiveness in integration technology for the developers.

In this research, a comparison model will be developed in all three languages. The comparison model will help to decide and evaluate each language based on different parameters. This research will help developers and designers to decide the most appropriate language for development and focus on a single language and the business logic instead of the peripheral network or integration issues(DZone, IBM & 3Scale, 2016). This research will aim to answer the below following research questions –

Which programming language is better for the development of an integration solution?

# Research Question

What programming language is best for developing microservices from the integration standpoint?

# Research Scope

To find the answer to the research question, an experiment is carried out. As this research focuses on the linguistic side of the integrations, for the services developed as part of the experiment its functionality is not crucial. Considering this, for all the test scenarios, two simple services are developed. Also, integration can be of different types based on the participating services/ systems.  For this research I am ignoring all the other types of integration but internal.

The libraries used for development of the microservices are as per the official website of each programming language. No external development framework for any language is used in the development of the services as this research focuses on the comparison of the language alone rather than the external frameworks by different service providers. [Future Scope: Oracle this June released Helidon 2.0 to ease microservices development.Current development does not make use of the Helidon server and the libraries for the microservices development]

Independent variables in the experiment methodology help to compare the language from the integration side of microservices. To understand how the language's complexity changes for microservices development the independent variable that is manipulated in the experiment is the communication styles and the type of payload sent during the communication between the services.  Different test scenarios include services written for different integration technology, the research studies the impact of every technology used in the research against different parameters. Given below are the test scenarios

1. REST with payload JSON
2. REST with payload  XML
3. AMQP with payload  JSON
4. AMQP with payload  XML
5. gRPC with payload protobuf

Dependent variables used to measure each language is derived based from two factors

1. Variables from the comparison of general purpose programming language (Verbosity, Size, Performance, Debug)
2. Variable from the linguistic side of microservices and integration

(Graphical support, ports)

Control is an important factor in the experiment methodology to make sure the output derived is purely the result of changing the independent variable. To ensure the maximum control is achieved in development, deployment and the execution of these services.

1. Development : The microservices in each language are written to be exact replicas of each other in terms of functionality.
2. Deployment: The microservices are deployed in the docker image with the same configuration for the runtime. As microservices can be shipped on any system or cloud to run.
3. Execution: All the microservices will run in the same system.

**References**

[Montesi, F., Guidi, C., & Zavattaro, G. (2014). Service-oriented programming with Jolie. In Web Services Foundations (pp. 81–107). Springer.](https://www.zotero.org/google-docs/?6666ei)

[Serrano, N., Hernantes, J., & Gallardo, G. (2014). Service-oriented architecture and legacy systems. IEEE Software, 31(5), 15–19.](https://www.zotero.org/google-docs/?6666ei)

[Weerawarana, S., Ekanayake, C., Perera, S., & Leymann, F. (2018). Bringing middleware to everyday programmers with ballerina. International Conference on Business Process Management, 12–27.](https://www.zotero.org/google-docs/?6666ei)