# Microservices: Integrations, Languages, Challenges

## Microservices Architecture

* History of software architecture [different architecture and problems faced]
* Microservices architecture in brief
* How problems with Service Oriented Architecture is overcome with microservices

## Integration

* Introduction
* Categories of microservices based on integration - Internal and External
* Internal is within the microservices
* External are again categorised into two -

1. Consumer of the service
2. Provider from the service [Service to External System ] Scope of this is large

## Development of Microservices

* Problem with the general purpose programming language
* Language for microservices
* Popular languages used are general purpose language despite the challenges faced.

## Gap in Literature

* No structured comparison available between the microservices languages like other general purpose language based on the integration aspect of microservices

## Research Question

* Which programming language is better for microservices development for communication between the microservices?

## Research Scope

* This research will focus on the integration side of the microservices.  Hence what the microservice achieves is not of any concern for the comparison. The microservices built for the research are simple hello world returning services.
* The scope of integration for this research is only the communication between the microservices. Microservices built for research will demonstrate two ways of communication between each other - Request response and messaging.
* The microservices developed are all the REST API and will be called using the Advanced Rest Client.
* The methodology used for comparison is Experiment.
* 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 and also that increases the scope of the research. [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 will 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.  that is changed in the experiment to identify the impact of these variables on the dependent variables.

1. Code studied for Communication Style (Request - response) REST with payload JSON
2. Code studied for Communication Style (Request - response) REST with payload  XML
3. Code studied for Communication Style (Messaging) AMQP with payload  JSON
4. Code studied for Communication Style (Messaging ) AMQP with payload  XML

* 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, services & interfaces as first class citizen)

* 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.

- Development : The microservices in each language are written to be exact replicas of each other in terms of functionality.

-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.

-Execution: All the microservices will run in the same system.

## Introduction

Today microservices architecture is very popular and widely adopted (WSO2, CA technogies, Runscope, 2018, p. 2). Microservices manages complexity by making services completely independent in development and deployment. With the growth of the microservices architecture, integration of different applications and services have become a critical factor for businesses and consumers (DZone, IBM & 3Scale, 2016). Integration solutions are often developed using the general-purpose programming language. The languages like Java and C# fail to offer the capabilities needed for the microservices integration solution (Weerawarana et al., 2018). There are other new languages emerging to solve the integration related problems. Jolie and Ballerina are the emerging full-fledged programming languages designed for microservices architecture (Newman, 2015). The research aims to identify the most appropriate programming language for the development of integration solutions for microservices.

Why these languages? The main objective of the research is to compare three programming languages – Ballerina, Jolie, and Java for different integration technologies.

## Language Overview & Version used

### Java –

Java is both programming language and platform. Java language is a high-level object-oriented programming language. Java platform is an environment on which Java programming language applications run (*Differences between Java EE and Java SE - Your First Cup: An Introduction to the Java EE Platform*, n.d.). There are four different platforms of Java programming language and all the platforms contain Java Virtual Machine and an application programming interface (API).

1.  Java Platform, Standard Edition (Java SE),

2.  Java Platform, Enterprise Edition (Java EE)

3.  Java Platform, Micro Edition (Java ME)

4.  JavaFX

According to Oracle, microservices can be built using Java EE(Enterprise Edition) (Krill, 2018). Java Platform, Enterprise Edition is the standard in the community driven enterprise software (<https://www.oracle.com/java/technologies/java-ee-glance.html>). All the programs for comparison will be built on and run on the latest available version of Java EE as of today which is Java EE 8.

### Ballerina

Like Java, Ballerina is a language and a platform. However, unlike Java, Ballerina is not an object-oriented programming language. It is a traditional declarative language like C, as most things that pass over the network are not object. Ballerina also supports object oriented style of coding. As Ballerina was designed specially to solve the integration related issues, WSO2 designed this language as an API first programming language.  Ballerina platform runs on java and the runtime for ballerina application is called jBallerina. jBallerina internally transforms the Ballerina sources to Java bytecode and run on a JVM<https://ballerina.io/learn/installing-ballerina/#installing-via-the-ballerina-language-zip-file> . Ballerina requires Java version 1.8 or above to run the ballerina programs. All the programs in Ballerina will be build using the version 1.2.6 which is the latest stable version of ballerina (<https://ballerina.io/downloads/release-notes/>).

### Jolie

Jolie is the first microservice-oriented programming language. It is Java Orchestration Language Interpreter Engine (JOLIE). Jolie is a service-oriented programming language with the syntax like C language. Unlike Java and Ballerina, Jolie does not have its own platform. It requires Java to run. Latest version of Jolie runs on Java 8 or later versions of Java. As of today, the latest version of Jolie is 1.8.<https://www.jolie-lang.org/downloads.html>.

## Methodology

According to Nanz and Furia (2015) a reliable answer to what programming language is the best, can be given only by analyzing empirically with the help of artifacts built in those programming language. In this research, the same experimental approach is used to find out the best programming language from the three chosen languages – Ballerina, Jolie and Java.

Repository called Rosetta Code  has 1059 programs written in 791 languages and is available to the developer community to contribute and researchers to study ((*Rosetta Code*, n.d.). These source code for a programming language is used from this repository for the comparison purpose (Nanz & Furia, 2015) (Nanz et al., 2013). There are other research conducted to compare the programming languages where the researchers have built their own artifacts for comparison based on the benchmark problem statement (Nanz et al., 2013). My research will also use similar experimental methodology with same parameters identified for comparison with additional parameters that focus on the integration aspect of the programming language to answer the research question.

According to (Jackson, 2014), the experiment design approach “allows a researcher not only to describe and predict but also to establish a cause and effect relationship through manipulation of a variable and control of the situation.” The main foundation of the experimental approach is that the researcher gains control as much as possible by setting up the common runtime environment for all three languages. This group of variables is identified as a control group, which includes system configuration and runtime environment variables for each language. This can be achieved using the same CPU processor, RAM, assigning equal memory to the runtime environment for every execution and the type of connectivity, etc.  In the experimental process, certain factors are selected and deliberately varied in a controlled manner to understand their effects on the response of interest (Wang & Wan, 2009). These are called independent variables. To understand the effect of different integration scenarios, it is important to test the programs for different integration type scenarios for different formats of data sent over the network during the integration of the services. The different integration scenarios used for this research are - REST, SOAP, AMQP and Database (Newman, 2015). The format of data used for the experiment are XML and JSON as  they are the common  format and widely adopted for RESTful services and the SOAP service (Newman, 2015).

The variable that is measured is the dependent variable. In this experiment, there is a list of dependent variables that are used from the previous research done for comparison of programming languages (Nanz & Furia, 2015) and the variables derived from the linguistic approach focusing on the integration aspect of microservices. The variables from the linguistic approach for microservices are interfaces or services as they separate the behavior from deployment (Guidi et al., 2017). This can be possible if the programming language treats Interfaces or Services as the first-class entity. Besides Interfaces, Ports and Workflows are also important for microservices. The other factors important for the general purpose programming language is verbosity of the source code, size of the executables and the execution time of the code (Nanz & Furia, 2015).

### Dependent Variables

The parameters from the linguistic approach for microservices are interfaces or services as they separate the behavior from deployment (Guidi et al., 2017). This can be possible if the programming language treats Interfaces or Services as the first-class entity. Besides Interfaces, Ports and Workflows are also important for microservices. The other factors important for the general purpose programming language is verbosity of the source code, size of the executables and the execution time of the code (Nanz & Furia, 2015).

The variables identified are measured by running the programs for different integration scenarios.

#### Verbosity:

Term Verbose means explaining in words more than needed. According to the developer community StackOverFlow, majority of the developers supported that a good code should be easy to comprehend at a glance (Why Is Verbosity Bad for a Programming Language?, n.d.). Also, according to the Gupta (2004), excess verbosity can cause the programmers at the beginner level to miss the conceptual learning in order to get the grasp of the huge code. When a programming language is less verbose it also mean that it is capable to do a task in less number of lines which also saves the development time with respect to writing code (Krishna et al., 2017). Considering all this factors, the line of code for each program will be measured and the languages will be evaluated on the numbers obtained from the experiment.

#### Size:

Size of the executable has a great impact on the deployment of the code. Reduced size of executable has several benefits like faster deployment, smaller disk size, small server space, reduced cost of hardware and easy management of code (Cepa, 2005). The factors that influence the size of the code is the use of third-party libraries or the external frameworks to support a development. However, the same can be avoided if the language development environment itself provides those libraries. (How Ballerina  Is Different From Other Programming Languages - DZone Integration, n.d.). In this study the executable built for every language will be evaluated against size of the executables for every integration scenario.

#### Execution Time

It is a well-known fact that faster applications are preferred over the slow performing applications. Execution time forms an important parameter to measure the performance of any application. A new technology is always assessed on its performance and execution time is one of the parameter to measure the performance (Pongnumkul et al., 2017). In this research, for every program the execution time will be recorded and compared for all the languages.

#### Debug

Debugging is useful in various stages of software development like design where the error made in writing the code is identified, secondly it is also helpful in the later stages of testing. Debugging is also helpful in the error diagnostics in the production environment (Cheng et al., 2017).

Breakpoint support (Davis et al., 2020)

Conditional Breakpoint support (Davis et al., 2020)

Remote debugging support (Zinkovsky et al., 2015)

#### Interfaces and Port

Microservices are deployed as the “black boxes” whose implementation details are hidden in order to support modular programming. The details of the functionality of the services is provided by the interfaces. Interfaces describe the set of operations that can be remotely invoked. Thus, interfaces become the first-class citizen of the microservices. Thus, it is important for a microservices languages to provide the set-theoretical operators like union and intersection that can work with the interfaces (Guidi et al., 2017).

Besides Interfaces, ports are important as communication port describes how the services are made available to the network. Each service may be equipped with many ports thus ports should be separate from the implementation of the service. Hence, the language for microservices should provide the capability to separate the port from the implementation of the service (Guidi et al., 2017).

This research will identify what language treats Interfaces and Ports as the first-class citizen. This can be seen from the artifact built in all three languages.

#### Graphical View Support

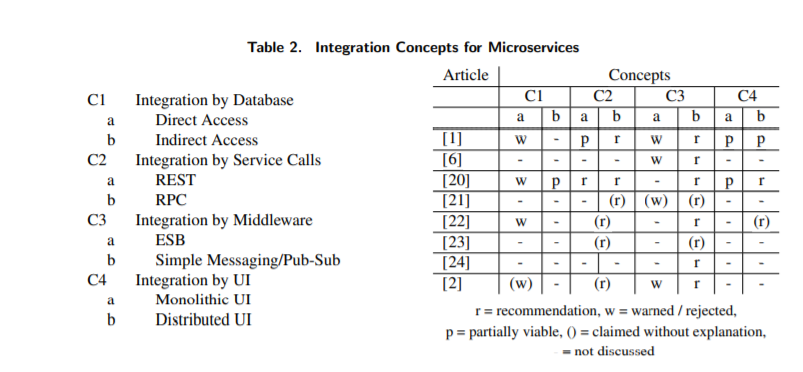
It is a common practise to draw the sequence diagram for the complex integration scenarios (Weerawarana et al., 2018). Also, there are different workflow languages like BPEL being used by the industry experts to simplify the complex integration scenarios (Guidi et al., 2017). Thus, it is helpful for the microservices language to provide the visual support or the workflow development options to support the development of microservices. This research will test the three languages for its capability to support visual designing. This can be seen from the artifact built in all three languages.

### Independent Variable

To identify the difference between the programming language it is important to test each service for different integration scenario for different data type.

#### Microservices Integration

The integration model for microservices was identified by (Schwarz & Riehle, 2020). According to this model there was a different integration technology that microservices communicates with. Some of them are the best technology recommended by many researchers while some are to be avoided as it is not a best design decision from the microservices architecture perspective. (Fowler & Lewis, 2014; Newman, 2015; Thones, 2015) This research will use the technology that is recommended by the researchers for the experiment.



Eight researchers have discouraged the integration with the database, ESB and Monolithic UI [Ignoring p and w]. This research will test for the integration scenario that has been widely recommended.

1.  Integration by REST

2.  Integration by RPC

3.  Integration by Messaging

4.  Integration by Distributed UI

All the programs developed for the integration scenario is saved in the Github repository –

<https://github.com/ushakotian/Ballerina_Java_Jolie-/tree/master/workspace/>

#### Data Payload

The microservices use REST mechanism, XML and JSON are the main data-interchange formats for the microservice API (Dragoni et al., 2017).  This experiment will use XML and JSON as the two data types for writing the service.

### Control

To achieve the maximum control, the program in all the three language will be the exact replica of one other in terms of performing a given task. Apart from that all the programs will connect to the same integration service and will run in the same system configuration.

#### Writing programs for comparison

Microservices architecture follows smart endpoints and dumb pipeline approach. The application built from the microservices architecture should be highly decoupled.

According to Martin Fowler [<https://martinfowler.com/articles/microservices.html#SmartEndpointsAndDumbPipes>] the applications built from the microservices should use the principles and protocols that the world wide web is built on. All the microservices that will be used for comparison will be the RESTful api’s written in Java, Ballerina and Jolie.

Java source code will be used from the examples listed in the official website of Oracle. For Ballerina and Jolie, the examples will be coded as the Java example. To code Ballerina and Jolie example the official website documentation and tutorial of both the languages will be used. I will log all the issues that I come across while writing and running the program and that will also form the basis for comparison as user experience.

#### System configuration

### Expected Output

All these dependent variables and the integration technology will lead to a matrix-like structure to analyze the results. The figure below demonstrates how the comparison model will look.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Independent Variable | Microservices Integration | Data Type | Control | | |
| Replica REST services  Deployed in Container with same configuration | | |
| Dependent Variable | Ballerina | Jolie | Java |
| Verbosity  (No. of lines) | Rest | JSON | ? | ? | ? |
| XML | ? | ? | ? |
| RPC | JSON | ? | ? | ? |
| XML | ? | ? | ? |
| Messaging | JSON | ? | ? | ? |
| XML | ? | ? | ? |
| UI | JSON | ? | ? | ? |
| XML | ? | ? | ? |
| Performance  (Seconds) – | Rest | JSON | ? | ? | ? |
| XML | ? | ? | ? |
| RPC | JSON | ? | ? | ? |
| XML | ? | ? | ? |
| Messaging | JSON | ? | ? | ? |
| XML | ? | ? | ? |
| UI | JSON | ? | ? | ? |
| XML | ? | ? | ? |
| Size | Rest | JSON | ? | ? | ? |
| XML | ? | ? | ? |
| RPC | JSON | ? | ? | ? |
| XML | ? | ? | ? |
| Messaging | JSON | ? | ? | ? |
| XML | ? | ? | ? |
| UI | JSON | ? | ? | ? |
| XML | ? | ? | ? |
| Graphical Support | Rest | JSON | ? | ? | ? |
| XML | ? | ? | ? |
| RPC | JSON | ? | ? | ? |
| XML | ? | ? | ? |
| Messaging | JSON | ? | ? | ? |
| XML | ? | ? | ? |
| UI | JSON | ? | ? | ? |
| XML | ? | ? | ? |
| Interfaces/Ports as first-class citizen | Rest | JSON | ? | ? | ? |
| XML | ? | ? | ? |
| RPC | JSON | ? | ? | ? |
| XML | ? | ? | ? |
| Messaging | JSON | ? | ? | ? |
| XML | ? | ? | ? |
| UI | JSON | ? | ? | ? |
| XML | ? | ? | ? |

### Experiment Task

1.  Finding/Coding all the integration scenarios can be any technology

Using simple hello world example for comparison

First REST Service returning hello

Second REST Service returning world

For the external client, This research will use postman as to call the endpoints. Main focus of the research is only the integration of the two services.

Both the services will follow different communication styles.

Since microservices is decentralized, both the services should be capable of talking to each other instead of relying on the middleware.

1.  Two REST services communicating using REST - Done

2.  Two REST services communicating using Messaging - Done

Different messaging protocols and brokers available. This research is not focusing on which messaging mechanism is the best hence I have used the latest and most recommended messaging protocol and broker - AMQP with rabbitmq broker.

3.  Two REST services communicating using RPC

There are different

Database – Users

Rest Style

Rest Service1 – Queries db

Rest Service 2 – calls Service1 and users the response from service1

Messaging Style –

Service1 sends message to RabbitMQ Service2 consumes

Client – Calls Service1 & Service2

Calls Service1 & Service2

3.  Deploying the services in containers with same configuration

4.  Calling the service from external client

### **Experiment Logs**

Daily Logs are maintained in Coda

<https://coda.io/d/Research-Notes_dc8RIZO9bxI/Research-Notes_sudcz#_luf8Y>

# References

Cepa, V. (2005). Using Reflection to Reduce the Size of. NET Executables. *J. Object Technol.*, *4*(7), 51–64.

Cheng, C.-T., Lee, Y.-H., Liu, C.-H., & Wu, M.-T. (2017). *Error diagnostic in a production environment*. Google Patents.

Davis, J., Billing, N. M., & Joglekar, N. R. (2020). *Conditional debugging of server-side production code*. Google Patents.

*Differences between Java EE and Java SE - Your First Cup: An Introduction to the Java EE Platform*. (n.d.). Retrieved August 5, 2020, from https://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: Yesterday, today, and tomorrow. In *Present and ulterior software engineering* (pp. 195–216). Springer.

DZone, IBM & 3Scale. (2016). *Integration: Microservices, APIs, and Platforms—DZone—Research Guides*. https://dzone.com/guides/integration-microservices-apis-and-platforms

Fowler, M., & Lewis, J. (2014). *Microservices a definition of this new architectural term*. Martinfowler.Com. https://martinfowler.com/articles/microservices.html

Guidi, C., Lanese, I., Mazzara, M., & Montesi, F. (2017). Microservices: A language-based approach. In *Present and Ulterior Software Engineering* (pp. 217–225). Springer.

Gupta, D. (2004). What is a good first programming language? *XRDS: Crossroads, The ACM Magazine for Students*, *10*(4), 7–7. https://doi.org/10.1145/1027313.1027320

*How Ballerina  Is Different From Other Programming Languages—DZone Integration*. (n.d.). Retrieved June 9, 2020, from https://dzone.com/articles/ballerinawhy-it-is-different-from-other-programmin

Jackson, S. L. (2014). *Research methods: A modular approach*. Cengage Learning.

Krill, P. (2018, September 12). *Oracle forges a Java microservices framework*. InfoWorld. https://www.infoworld.com/article/3305758/oracle-forges-a-java-microservices-framework.html

Krishna, R., Menzies, T., & Layman, L. (2017). Less is more: Minimizing code reorganization using XTREE. *Information and Software Technology*, *88*, 53–66.

Nanz, S., & Furia, C. A. (2015). A comparative study of programming languages in rosetta code. *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, *1*, 778–788.

Nanz, S., West, S., Da Silveira, K. S., & Meyer, B. (2013). Benchmarking usability and performance of multicore languages. *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 183–192.

Newman, S. (2015). *Building microservices: Designing fine-grained systems* (First Edition). O’Reilly Media.

Pongnumkul, S., Siripanpornchana, C., & Thajchayapong, S. (2017). Performance analysis of private blockchain platforms in varying workloads. *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, 1–6.

*Rosetta Code*. (n.d.). Retrieved July 6, 2020, from http://rosettacode.org/wiki/Rosetta\_Code

Schwarz, G.-D., & Riehle, D. (2020). What Microservices Can Learn From Enterprise Information Integration. *HICSS*, 1–10.

Thones, J. (2015). Microservices. *IEEE Software*, *32*(1), 116–116. https://doi.org/10.1109/MS.2015.11

Wang, J., & Wan, W. (2009). Experimental design methods for fermentative hydrogen production: A review. *International Journal of Hydrogen Energy*, *34*, 235–244.

Weerawarana, S., Ekanayake, C., Perera, S., & Leymann, F. (2018). Bringing middleware to everyday programmers with ballerina. *International Conference on Business Process Management*, 12–27.

*Why is verbosity bad for a programming language?* (n.d.). Software Engineering Stack Exchange. Retrieved June 8, 2020, from https://softwareengineering.stackexchange.com/questions/141175/why-is-verbosity-bad-for-a-programming-language

WSO2, CA technogies, Runscope. (2018). *API Management: Comparative Views of Real-World Design—DZone—Research Guides*. https://dzone.com/guides/api-management-comparative-views-of-real-world-des

Zinkovsky, I. A., Sawicki, M., Stall, J. M., Crider, A. L., & Quinn Jr, T. E. (2015). *Remote debugging of server side code without blocking or high-level privileges*. Google Patents.