Findings

Verbosity

When calculating verbosity, I have calculated the source line of code metric that includes different variables to be counted. These are - Physical lines that include the commented code (SLOC), the logical lines that include the executable statements, blank lines , etc [(Bhatt et al., 2012)](https://www.zotero.org/google-docs/?HwcorN). To get the count of each variable there are different automated tools. However, there was no single tool that provides support for the languages used in the experiment. Thus ,I am manually counting the line of code based on the above metric. Despite the fact that all the languages are structurally and syntactically different, as I have done the coding I have tried to be consistent in my style of coding to avoid as many differences as possible in terms of declaring, importing or calling the functions. For example -

Import statement

Java

There are two ways of importing the class either import by package.\* or package.classname. As both the ways do not have any cost or performance overhead but are for improving the readability of the developer. In this experiment my primary focus is the verbosity and with the former option I can reduce the number of lines. Therefore, I have opted for the first style of importing the class.

Ballerina

There is only one way to import the package within the class that is importing the entire module.

Jolie

There is only one way to import the package within the class that is importing the entire module.

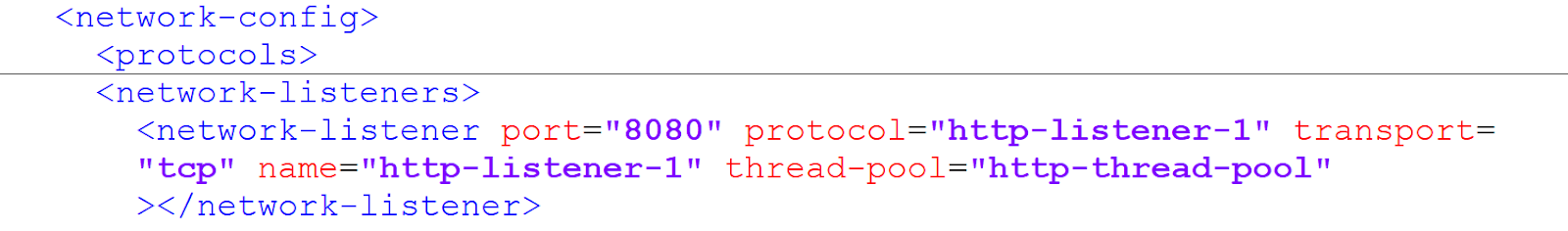
Aggregated metric link

[LOC metric](https://docs.google.com/spreadsheets/d/1YE7uL3AAqpEVxiZpRGAkCyEmGZvnlAq7hqK_ca3iRzQ/edit#gid=0)

Ports

Microservices may run in heterogeneous environments that use different communication technologies (e.g., TCP/IP sockets, Bluetooth, etc.) and data protocols (e.g., HTTPS, binary protocols, etc.). Moreover, a microservice may need to interact with many other services, each one possibly offering and/or requiring a different interface. A communication port concretely describes how some of the functionalities of a service are made available to the network, by specifying the three key elements above: interface, communication technology, and data protocol. Each service may be equipped with many ports, of two possible kinds. Input ports describe the functionalities that the service provides to the rest of the MSA. Conversely, output ports describe the functionalities that the service requires from the rest of the MSA. Ports should be specified separately from the implementation of a service, so that one can see what a service provides and what it needs without having to check its actual implementation.

In java, the language does not have the support to provide the port detail, this has to be managed as a part of the server setup.

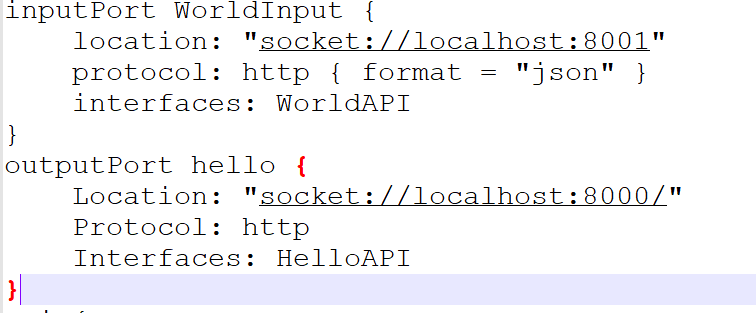


 For Ballerina, this can be handled from within the language. However it is tightly coupled to the implementation of the service.



 For a same service to be exposed on two different ports the service has to be duplicated which violates the “Dont Repeat Yourself” DRY principle.

This is well handled in Jolie, where the language has the provision to configure the port details and is separate from the implementation at the same time.



Graphical View Support

This variable is one of the important variables to easily understand the flow of data between the services. The only language that claimed to offer this support was ballerina with the help of the plugin for VS code. However, in my experiment the visual editor failed to display the flow and was blank throughout my experiment.

Debug

I tested each language for this variable in the respective IDE used for the development of the services. I tested each language for local and remote debug.

Local Debug- In this scenario, I run the microservice in the debug mode and test it for the breakpoints added.

Remote Debug - In this scenario, I run the microservice first and then attach the service to the debugger and then test it for the request and wait for the breakpoint to be hit.

Java - Support available for both local and remote.

Ballerina - Was able to run it only in the remote debug mode

Jolie - No support.

Size

I tested each language for the size of the deployable, in this experiment the size of the docker image is fetched and is compared. I have defined the name of each docker image in the docker-compose.yaml file for each deployable and run the command “docker image ls”. This command displays the name of the repository as given in the docker-compose.yaml. The output is measured in MBs.

Execution Time

Execution time is the user time in seconds which is the total CPU time spent within the process. This is the CPU time spent in executing the actual process. To obtain the user time, I have used one of the ReST Clients called Postman for the ReST and gRPC request. For AMQP requests, the same is obtained using the powershell command.

Every request is run 100 times and the average of 100 runs is compared for every language.

[Bhatt, K., Tarey, V., & Patel, P. (2012). Analysis Of Source Lines Of Code(SLOC) Metric. *IJETAE*, *2*.](https://www.zotero.org/google-docs/?CthOp9)