## Selecting the Systems for Defining MiSAR Artefacts

To define the artefacts of MiSAR, we conducted an empirical. This document explains how we selected the systems from a search on Github. Along with this document please also use the excel sheet SelectionOfSystems.xslx located in https://github.com/MicroServiceArchitectureRecovery/misar/tree/main/EmpiricalStudyReplication/SelectedSystemsUsedToDefineMiSARartefacts

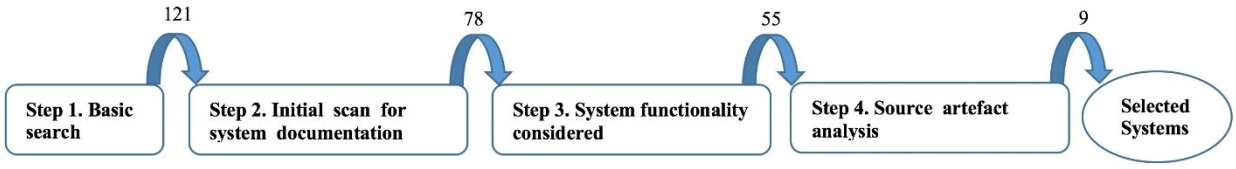


*Figure 1. Results of applying the four steps of the selection protocol*

**Selection Protocol**

The selection protocol, designed for the empirical study of microservice-based applications from GitHub, consists of four steps. Each step filters the systems further, as shown in Figure 1. The criteria for inclusion and exclusion at each step are detailed in Table 1, while Table 2 lists the systems selected for this study.

**Step 1: Basic Search** We constructed a search string that combines two populations: the Microservice population and the Frameworks technology population. For the Microservice population, we ensured the term 'Microservice' was a recurring keyword. To broaden the search, we included more general keywords with the prefix 'microservice'. Thus, the Microservice population is defined as (microservice* OR "micro service"* OR "microservice architecture"*).

The Frameworks technology population is more diverse, resulting in the string (spring* AND java* AND (docker-compose OR docker)* OR netflix* OR asynchronous* OR reactive*). The search hits were cumulatively joined, with repeated results increasing their score based on their cumulative occurrences. The result list was then sorted by hit score in descending order and reduced to the top 150 items. Any results with a score less than 3 were excluded. This step yielded a result set of 121 items, sorted alphabetically.

**Step 2: Initial Scan for System Documentation** A team member reviewed the title, description, architectural diagram, and documentation of each project. Projects were discarded if they lacked a clear description (or if the description was in a language other than English), or if they lacked any documentation, paper, or tutorial pages. This step resulted in a set of 78 items.

**In the Excel sheet SelectionOfSystems.xslx**: apply the filter for (documentation type column) by unchecking N/A, N/A[Chinese] and N/A[Spanish] which refers to systems that do not include documentation. The outcome result set for this step turned out to have a length of 78 systems.

**Step 3: Consideration of System Functionality** Two researchers were assigned to each remaining project to examine the system's functionality. We included projects that implemented at least two business/functional microservices (excluding user/account services) and excluded projects that either only implemented infrastructure services (e.g., the "Microservice Monitoring" system was excluded as it only implements monitoring infrastructure services), or only one business/functional microservice (e.g., the "Stock Price Viewer" system was excluded as it implements only one business/functional microservice named "stock-service"). This step resulted in a set of 55 items.

**In the Excel sheet SelectionOfSystems.xslx:** apply the filter for (Business microservices column) by unchecking N/A which refers to systems that do not include Business microservices and uncheck [1] which refers to systems that include only one Business/functional microservice. The outcome result set for this step turned out to have a length of 55 systems.

**Step 4: Technology Consideration** The 55 systems were attributed by technology and divided among three researchers, each responsible for analyzing the artifacts of their assigned systems to identify specific patterns or technologies.

1. **Researcher 1** focused on cases that have the essential patterns/technologies in a typical microservices architecture, such as one Eureka discovery service, one Zuul gateway service, and one data store of any technology. The cases were sorted by Score (indicating the variety of technologies implemented) and Date (indicating the recency of the application and its technologies/methodologies). The top three cases selected were 3, 5, and 9.
2. **Researcher 2** prioritized cases that implemented the asynchronous communication pattern, polyglot, and various build frameworks, selecting one case for each specific technology/framework:

    **2.1.** The first criterion was the implementation of the asynchronous communication pattern. Cases deployed using Docker Compose were selected to easily identify the existence of a RabbitMQ container in the architecture. The cases were sorted by the number of functional microservices (ascending), as selected cases need not be large hence enabling easier analysis of targeted technology, and Date (descending). The top case selected was 2. The same criteria were applied for Kafka technology (instead of RabbitMQ), resulting in the selection of case 4.

    **2.2.** The second technology targeted was the implementation of polyglot of database(s) and message broker(s). Cases that implemented at least one data store and one message broker were selected. To easily identify the existence of message broker and data store container(s) in the architecture, cases deployed using Docker Compose were selected. The cases were sorted by Score (descending) to get more variety of such technologies. The top case selected was 8.

**2.3.** The third and final criterion was to have the application built using Gradle rather than Maven, the majority. Only one case met this criterion and was selected, which was case 6.

3. **Researcher 3** targeted cases that implemented multiple configuration/profiles in addition to satisfying essential patterns/technologies in a microservices architecture, similar to Researcher 1. To ensure that there was at least one additional profile (i.e., deployment) in addition to the default profile (i.e., development), cases deployed using Docker Compose were selected. This criterion resulted in two cases only, 1 and 7.

TABLE 1
The selection criteria for the Systems Studied

| Criteria | |
|---|---|
| Inclusions | Step num |
| • Have architectural diagram and documentation. | 2 |
| • Implement business functionality. | 3 |
| • Implemented with Spring Boot/Spring Cloud framework in java language<br>• Include one or more infrastructure Netflix OSS libraries (e.g., Zuul, Eureka, Hystrix, Sidecar).<br>• Each of the modules run in a single process (Docker technology)<br>• Implement lightweight synchronous/asynchronous interaction style and smart endpoints.<br>• Consist of an accumulation of independent individual services.<br>• Have Maven POM or Gradle build file(s). | 4 |
| Exclusion | Step num |
| • Do not use Spring Boot/Spring Cloud framework.<br>• Do not have Docker Compose file(s).<br>• Do not revolve around an accumulation of independent individual services.<br>• Do not have any data processing libraries (MongoDB, MySQL or Graph).<br>• Do not have Maven POM or Gradle build file(s).<br>• Do not have Java source files (the search string "java" resulted in JavaScript). | 4 |
| • Do not have clear description, any documentation, paper or tutorial pages. | 2 |
| • Use less than two functional microservices.<br>• Only include infrastructure microservices (e.g., development tools, operation frameworks). | 3 |

TABLE 2
Selected Systems

| ID | Project Name | URL | Microservice Count |
|---|---|---|---|
| 1 | Spring-Netflix-OSS-microservices | [31] | 9 |
| 2 | Spring-RabbitMQ-microservices | [32] | 7 |
| 3 | Cloud-enabled-microservice | [33] | 7 |
| 4 | Event-sourcing-microservices | [34] | 10 |
| 5 | Spring-cloud-sidecar-polygot | [35] | 7 |
| 6 | Microservices-basics-spring-boot | [36] | 10 |
| 7 | Spring-cloud-event-sourcing | [37] | 15 |
| 8 | Spring-Boot-Graph-Processing | [38] | 9 |
| 9 | BookStoreApp-Distributed-Application | [39] | 14 |