Service Description (SD) System Discovery

**Abstract**

This document describes an Arrowhead compliant System Discovery service. System Discovery is used to register, unregister, and read metadata about a specific system in an Arrowhead cloud.

Table of Contents

[1. Service Description Overview 3](#_Toc105596969)

[1.1. Significant Prior Art 3](#_Toc105596970)

[1.2. How This Service Is Meant To Be Used 3](#_Toc105596971)

[1.3. Important delimitations and dependencies 3](#_Toc105596972)

[2. Service Interface 3](#_Toc105596974)

[2.1. Operation Register (RegistrationItem) 4](#_Toc105596979)

[2.2. Operation Unregister (Instance) 4](#_Toc105596981)

[2.3. Operation Lookup(MatchingInfo): InstanceList 4](#_Toc105596982)

[2.4. Service Discovery Sequence 4](#_Toc105596984)

[3. Information Model 5](#_Toc105596985)

[3.1. Struct RegistrationItem 5](#_Toc105596986)

[3.2. Struct Instance 5](#_Toc105596987)

[3.3. Struct MatchingInfo 5](#_Toc105596988)

[4. Non-functional Requirements 6](#_Toc105596989)

[5. References 6](#_Toc105596990)

[6. Revision history 6](#_Toc105596991)

[6.1. Amendments 6](#_Toc105596992)

[6.2. Quality Assurance 6](#_Toc105596993)

1. Service Description Overview

This document describes an Arrowhead compliant System Discovery service. The System Discovery service is used to update and discover metadata about a specific system which is crucial for day-to-day operations and runtime governance. The metadata contains information about the identity of the system, the owner and vendor of the system, SLAs, what service instances are provided by the system and more. In practice, the service will primarily be used by other systems to update their own system metadata. This since the unique identifier of the system (system ID) needs to be known for the service operations.

# Significant Prior Art

The need for an inventory of systems in an IT environment is not new. However, with modern Microservice Architectures (MSA) the number of systems, services and APIs deployed to an environment has increased. Furthermore, the autonomy of teams in a large organization leads to this system setup being ever changing each day. This has led to a lot of developments in the field of system inventories, also known as software catalogs.

There exist several systems which solves this problem and, thus, provides the system discovery service among others. For example, Backstage by Spotify (Backstage Project Authors, 2022) is a software catalog and a developer portal that allows organizations to track ownership and system composition. The system model of Backstage also strongly resembles the system composition model of Arrowhead Framework.

There is also some overlap between the problem the system discovery service solves and API management. It is not uncommon for API management tools to include owner and vendor of the APIs and similar metadata, e.g., SLAs. An example of such tools could be the WSO2 API Management.

# How This Service Is Meant to Be Used

The service is meant to be used by systems to register and unregister themselves to advertise their presence in the Arrowhead local cloud. The service could also be used to obtain information about a specific system in case of issues during runtime, e.g. to alert the system owner.

# Important delimitations and dependencies

The system as a concept is not used for service operations, i.e., in the process to exchange information between systems. To discover services and how to access them (their address) is the role of the service registry. An important delimitation is that the system discovery service only operates on a specific system identified by the system ID. For similar operations on the collection of systems in the local cloud, refer to the System Registry administration service.

1. Service Interface

The abstract Service Discovery interface has three operations for maintaining the metadata for the system in the Arrowhead local cloud. The operations are Register, Unregister and Lookup.

When a system wants to advertise its presence in the Arrowhead local cloud it uses the register operation. Similarly, when a system is removed it uses the unregister operation. It is also possible to read the metadata of a system using its system id and the lookup operation. Note that the registration and deregistration do not necessarily correspond to a software instance being started or terminated. The timing of the operations is up to the design of the system and the system of systems. For example, if the system of systems is in a factory context and the system is a sensor then it is more likely that it registers itself. However, if the system of systems is in a cloud native context and is hosted using e.g., serverless functions then the system is probably registered outside of the lifecycle of individual software components.

# Operation Register (SystemID, SystemMetadata)

The Register operation is used by a system to register itself in the Arrowhead Local Cloud. The registration operation is idempotent and thus can be used as an upsert operation. The registration requires a unique identifier known as the system ID and the metadata to register.

An implementing service must ensure that only authorized parties can register a system. This could, for example but not limited to, be achieved using client certificates mapped to the system ID.

# Operation Unregister (SystemID)

The Unregister operation in used to deregister a system from the Arrowhead local cloud. The operation uses a system identifier as target system to be removed.

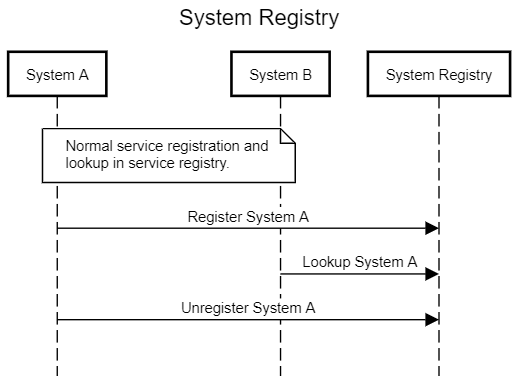
An implementing service must ensure that only authorized parties can deregister a system. This could, for example but not limited to, be achieved using client certificates mapped to the system ID.

# Operation Lookup(SystemID): SystemMetadata

The lookup operation is used to read system metadata for a specific system, identified by the system identifier.

This operation is not subject to any object-level authorization; however, an implementing service might require that the service consumer identifies itself before completing the operation.

# System Discovery Sequence



1. Information Model

# SystemID

The system identifier is a unique identifier for a system. An implementing service may choose how this identifier is represented but common techniques could be a simple string identifier, a sequential number, a URI or a public certificate.

# SystemMetadata

The data structure used to represent the metadata about a system. The metadata must contain a human readable name of the system. An implementing service could add more required metadata. Examples of metadata to consider are owner, vendor, SLAs, link to documentation, disaster recovery objectives, data classification (e.g. if the system handles personal data), GDPR related information etc.

1. Non-functional Requirements

No non-functional requirements have been defined.

1. References

No references have been identified.

1. Revision history

# Amendments

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. | Date | Version | Subject of Amendments | Author |
| 1 | 2022-09-09 | 0.1 | Initial draft of specification | David Rutqvist |
| 2 | 2022-10-03 | 0.2 | Added sequence in ch 2.4 | Per Olofsson |

# Quality Assurance

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Date | Version | Approved by |
| 1 |  |  |  |
| 2 |  |  |  |