ServiceRegistry System SysDD

**Abstract**

This document provides an overview on the ServiceRegistry System within Arrowhead generation 4.0.

This System is primarily implemented based on the DNS-SD BIND server, but registration, update and delete methods are only allowed through the REST interface of this System. Lookup is allowed via the DNS standard.

1. System Design Description Overview

Table 1 System Information

|  |  |
| --- | --- |
| **Full Name** | ServiceRegistry System |
| **Owner** | Csaba Hegedűs, AITIA Inc., hegeduscs@aitia.ai |

This System provides the database, which stores information related to the currently actively offered Services within the Local Cloud.

In this generation of the framework, this System is implemented twice, using two different database technologies:

* DNS-SD BIND server with a Java-bases REST interface
* The same Java-based REST interface, but using a MySQL database

The purpose of this System is therefore to allow:

* Application Systems to register what Services they offer at the moment, making this announcement available to other Application Systems on the network.
* They are also allowed to remove or update their entries when it is necessary.
* All Application Systems can utilize the lookup functionality of the Registry to find appropriate Service offerings in the network.

However, it is worth noting, that within this generation the lookup functionality of Services is integrated within the “orchestration process”. Therefore, in the primary scenario, when an Application System is looking for a Service to consume, it shall ask the Orchestrator System via the Orchestration Service to locate one or more suitable Service Providers and help establish the connection based on metadata submitted in the request. Direct lookups from Application Systems within the network is not advised in this generation, due to security reasons. There are only two exceptions to this guideline.

* Application Systems shall use the DNS-based, insecure ServiceRegistry interface to look up the DeviceRegistry and SystemRegistries upon bootstrapping (*not implemented yet!)*
* They also can look up the Orchestration Service using the DNS-based interface at all times.

However, the lookup of other Application Systems and Services directly is not within the primary use, since access will not be given without the Authorization token. The use of the TokenGeneration is restricted to the Orchestrator for general System accountability reasons. Yet again, this restriction is not mandatory, just a default setting (within Authorization System properties file).

The ServiceRegistry assumes that either the BIND server or the MySQL database is properly configured and set up when used. The setup guide for this module doesn’t contain guidelines on that, however, for the DNS server set up it is advised:

* the TSIG authentication schema is activated
* only the REST bridge possesses the TSIG keys: WRITE rights are only allowed through the REST interface
* all other Application Systems only have READ rights to the DNS server

There are no additional requirements for the MySQL-based Registry, besides the initialization with the supplied SQL scripts.

1. Services and Use-cases

This System only provides one Core Service:

* ServiceDiscovery

There are two use case scenarios connected to the Service Registry.

* Service registration, de-registration
* Service Registry querying (lookup)

There is another functionality that does not bound to any Services, just an internal part of the Service Registry. There are two optional cleanup tasks within the Service Registry, which can be used to remove old, inactive service offerings. The first task is based on pinging the service provider, and if the provider does not respond to the ping, its offered services will be deleted. The second task is based on a feature, called “Time to Live”. Service providers upon registration can provide a “ttl” number, which specifies how long the service will be offered by the provider, making the service de-registrations unnecessary, if this task is active. The task is used to remove expired services. Both of these internal tasks can be configured in the app.properties file.

1. Internal structure

This module is a Java jar executable. It uses the config folder contents, where the configuration files are. The code includes the following classes:

* **ServiceRegistryMain**: instantiates the HTTP or HTTPS server based on the properties files and command line arguments
* **ServicerRegistryResource**: contains the offered REST interface functions
* **ServiceRegistryApi**: contains the REST interface functions for manipulating the underlying MySQL database table, this interface is reserved for system operators (Local Cloud admins).
* **ServiceRegistry**: contains the “business logic” behind the REST interface (e.g. related to registration or query)
* **RegistryUtils**: contain utility functions for the ServiceRegistry class
* **PingProvidersTask**: is responsible for pinging registered Service Providers and check whether they are active on the network or not (if not, they are auto-removed the database)
* **RemoveExpiredServicesTask**: is responsible for periodically checking the database for expired services (and then removing them), based on the “time to live” field the provider given upon registration.

The SQL and DNS based implementations differ in some classes.

# Usage

Start the module as a Java executable. The following command line arguments are available:

* “-tls”: starts the Core System in secure (HTTPS) mode, using the certificates which were set in the app.properties file.
* “-daemon” (Linux only!): starts the module in daemon mode, kill signals will prompt a normal shutdown, and the core system will de-register its services from the Service Registry.
* “-d”: starts the module in debug mode, which means every incoming REST request (URL + payload) and the corresponding response will be printed to the console output.

1. Security

This System can be secured via the HTTPS protocol. If it is started in secure mode, it verifies whether the Application System possesses a proper X.509 identity certificate and whether that certificate is Arrowhead compliant in its making. This certificate structure and creation guidelines ensure:

* Application System is properly bootstrapped into the Local Cloud
* The Application System indeed belongs to this Local Cloud
* The Application System then automatically has the right to register its Services in the Registry.

If these criteria are met, the Application System’s registration or removal message is processed. An Application System can only delete or alter entries that contain the App. System as the Service Provider in the entry.

The BIND server is TSIG-secured and Application Systems only have READ access through the DNS interface. All other (CUD) methods can happen via the REST HTTPS interface, secured by SSL and X.509 identities.

1. Revision history

# Amendments

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. | Date | Version | Subject of Amendments | Author |
| 1 | 2017-09-29 | M2 | Initial | Csaba Hegedűs |
| 2 | 2018-05-23 | G4.0 | Updated to G4.0 | Zoltán Umlauf |

# Quality Assurance

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