The Gateway System

System Design Document

**Abstract**

This document describes the Gateway automation support Core System.

1. System Design Description Overview

Table 1 System Information

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

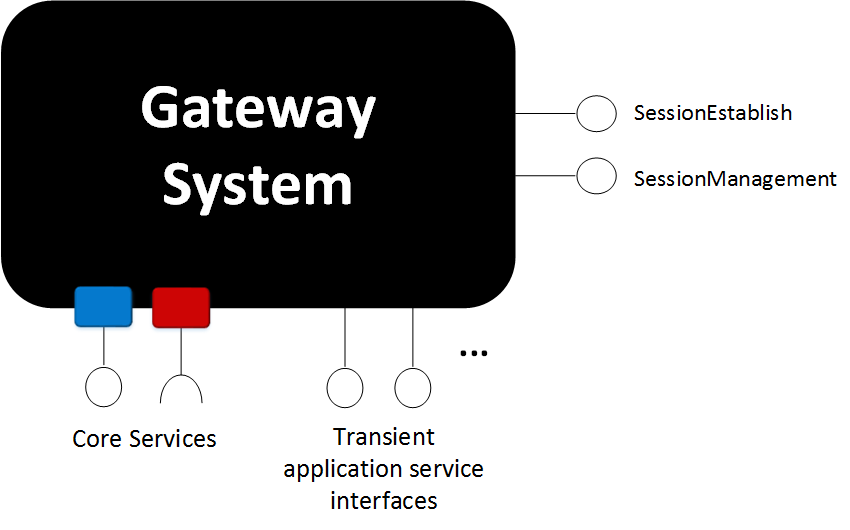
The Figure 1 depict an overview of the Gateway System.

Figure 1 The Gateway Core System

This System provides two Core Services:

* Session Establish
* Session Management

The Session Establish Service has two interfaces:

* Connect to Consumer
* Connect to Provider

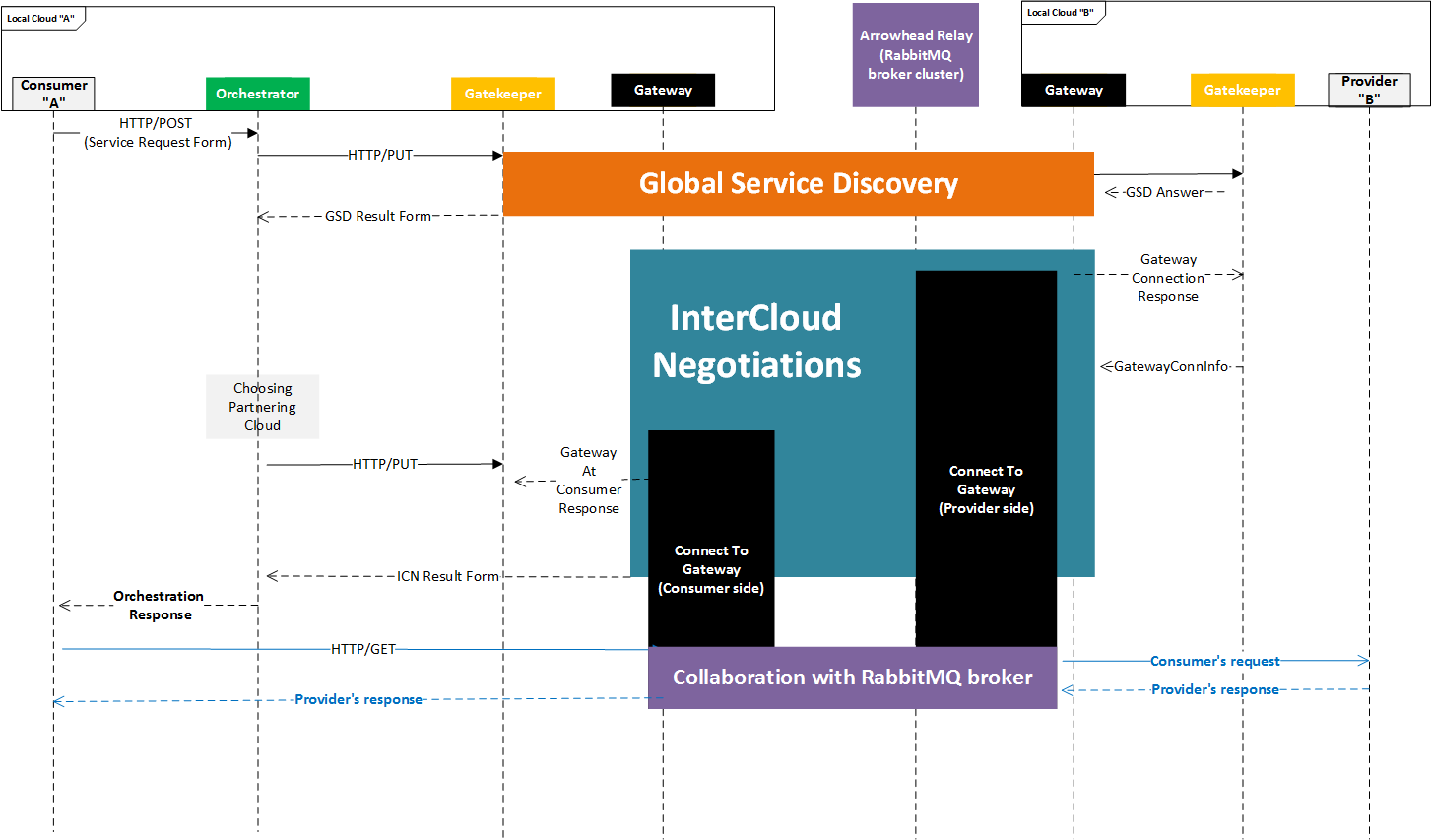
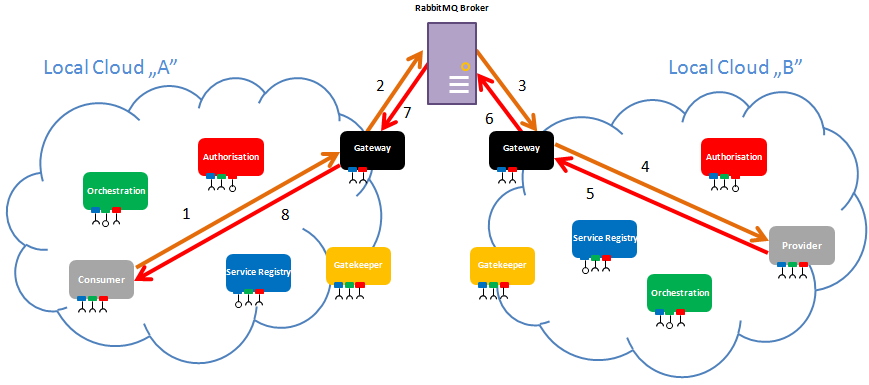
These Services are part of the inter-Cloud orchestration process (Figure 2).

Figure 2 The inter-cloud orchestration process

The built up data path is depicted in Figure 3.

Figure 3 The datapath



1. Use-cases

Table 2. Use-case description for Connect To Consumer

|  |
| --- |
| **Name of the Use-case** |
| **ID**: Connect-To-Consumer |
| **Brief description**:  The Gateway is tasked to connect to the Consumer and mediate between the Broker and the Consumer. |
| **Primary actors**:  Gatekeeper |
| **Secondary actors**:  Arrowhead compliant AMQP Broker |
| **Preconditions**:  InterCloud orchestration process was started by a consuming Application System. |
| **Main flow**:  1- The Gatekeeper sends a ConnectToConsumerRequest to the Gateway.  2- The Gateway internally adds a new ActiveSession object to the activeSessions HashMap.  3- The Gateway starts a new thread (secure/insecure based on connection mode).  4- The Gateway sends a ConnectToConsumerResponse to the Gatekeeper.  In the thread:  5- The Gateway creates a serverSocket/sslServerSocket.  6 – The Consumer connects to the port of the serverSocket.  7 - The Gateway accepts the connection and creates a socket/SSLSocket for the Consumer.  8 – The Gateway gets the request from the Consumer through the socket/SSLSocket and forwards it to the Broker.  9- The Gateway gets the response from the Provider through the Broker and forwards it to the Consumer through the socket towards the consumer.  10- The Gateway checks the control messages from Broker.  11- Repeat the steps 8-10. until one of the sockets gets closed or it gets a “close” message from Broker via “controlQueue”. |
| **Alternative flows**:  In secure mode, the Gateway encrypts every message. Before forwarding them, it generates a new unique and random AES (Advanced Encryption Standard) Key and initialization vector, encrypts them with RSA and send them to the Provider’s Gateway through the Broker. When a new message arrives from Broker, the Gateway decrypts it with the AES Key. |

**Table 3. Use-case description for Connect To Provider**

|  |
| --- |
| **Name of the Use-case** |
| **ID**: Connect-To-Provider |
| **Brief description**:  The Gateway is tasked to connect to the Consumer and mediate between the Broker and the Provider. |
| **Primary actors**:  Gatekeeper |
| **Secondary actors**:  Arrowhead compliant AMQP Broker |
| **Preconditions**:  Inter-Cloud orchestration process was started by a consuming Application System. |
| **Main flow**:  1- The Gatekeeper sends a ConnectToProviderRequest to the Gateway.  2- The Gateway creates a unique queueName and controlQueueName based on a random number and the current time.  3- The Gateway adds a new ActiveSession object to the activeSessions HashMap.  4- The Gateway starts a new thread (secure/insecure based on connection mode).  5- The Gateway sends a ConnectToProviderResponse to the Gatekeeper  In the thread:  5- The Gateway creates a socket/SSLSocket for the Provider.  8 – The Gateway gets the request from the Consumer through the Broker.  9- The Gateway gets the response from the Provider through the socket and forwards it to the Consumer’s Gateway through AMQP.  10- The Gateway checks the control messages from its partner Gateway.  11- Repeat the steps 8-10. until one of the sockets get closed or the Gateway gets a “close” message from Broker via the controlQueue. |
| **Postconditions**: |
| **Alternative flows**:  In secure mode, the Gateway encrypts every message. Before forwarding them, it generates a new unique and random AES (Advanced Encryption Standard) Key and initialization vector, encrypts them with RSA and sends them to the Provider’s Gateway through the Broker. When a new message arrives from Broker, the Gateway decrypts it with the AES Key. |

The payload encryption is depicted in Figure 4.

Figure 4 The payload encryption

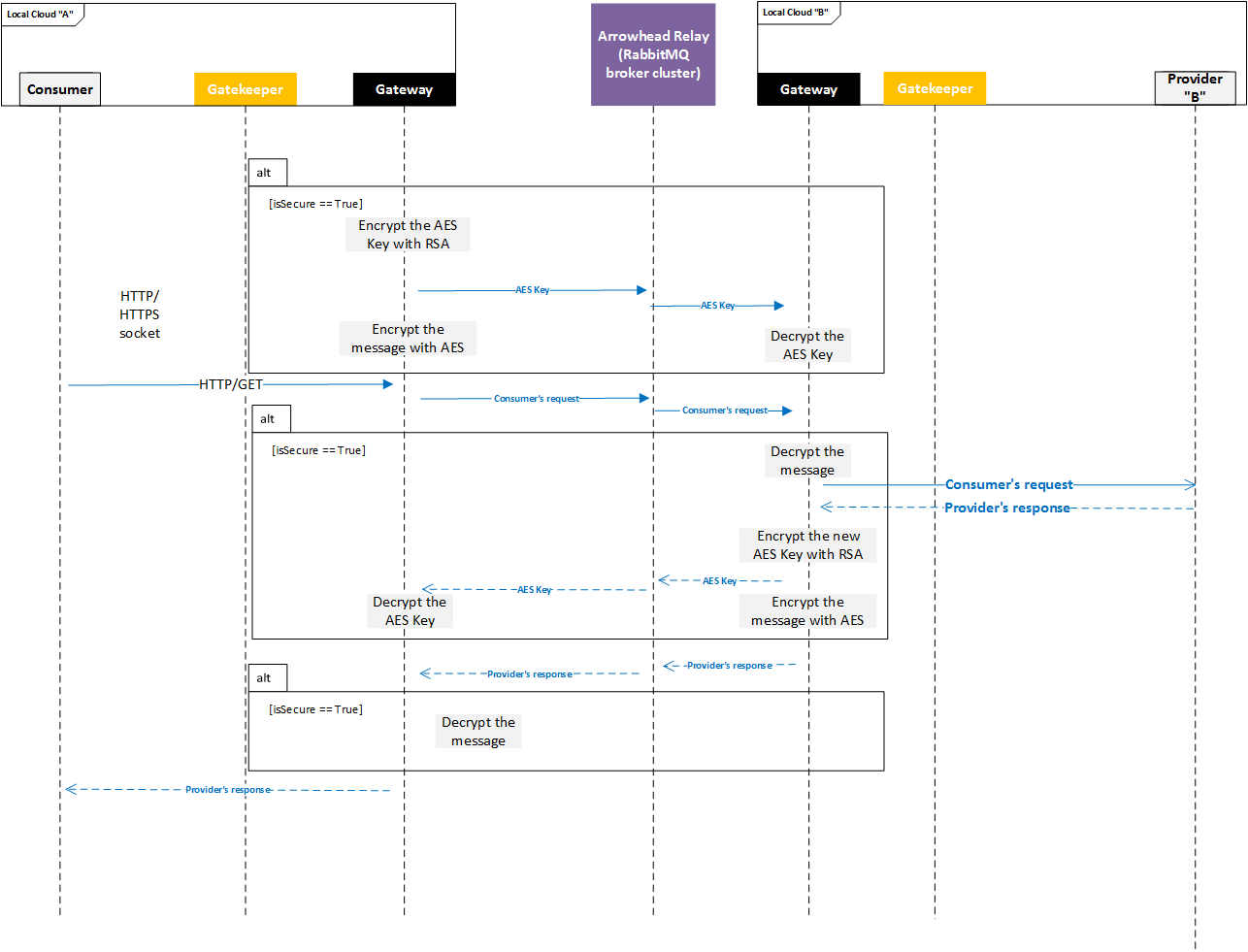


Table 4. Use-case description for Session Management

|  |
| --- |
| **Name of the Use-case** |
| **ID**: Session-Management |
| **Brief description**:  The Gateway is tasked to present the active sessions in JSON format |
| **Primary actors**:  User / operator |
| **Secondary actors**: |
| **Preconditions**:  The Gateway module is running. |
| **Main flow**:  1- User connects to Gateway via REST GET request.  2- Gateway checks the activeSessions HashMap  3- Gateway presents the active sessions’ list in JSON format. |
| **Postconditions**: |
| **Alternative flows**:  If the current sessions’ HashMap is empty, Gateway presents a default message. |

1. Internal structure

This module is a simple Java jar executable. It uses the ”config” folder contents, where the configuration files are there.

The code includes the following classes:

* **GatewayMain**: starts the HTTP and/or the HTTPS servers based on the properties files and command line arguments
* **GatewayResource**: contains the REST interface functions
* **GatewayService**: Contains miscellaneous helper functions for the Gateway (e.g. creating channel to the Broker, encrypting/decrypting messages, properly closing sockets)
* **AccessControlFilter**: implements CN-based access control when the HTTPS (secure) server is started
* **InsecureServerSocketThread:** a new thread which mediates between the Consumer and the Broker through sockets (insecure mode)
* **SecureServerSocketThread**: a new thread which mediates between the Consumer and the Broker through sockets (secure mode)
* **InsecureSocketThread:** a new thread whichmediates between the Provider and the Broker through sockets (insecure mode)
* **SecureSocketThread**: a new thread which mediates between the Provider and the Broker through sockets (secure mode)

Used libraries:

* Java Jersey API
* Grizzly servlet container
* RabbitMQ AMQP-client library

# Usage

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

* “-m <mode>” sets the server mode
  + “secure”: using HTTPS
  + “insecure”: using simple HTTP (default mode)
  + or “both” when two ports are used, one for simple HTTP and one for HTTPS server
* “-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. Revision history

# Amendments

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. | Date | Version | Subject of Amendments | Author |
| 1 | 2018-01-31 | 0.1 | Initial | Nikolett Szeles |

# Quality Assurance

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