Other

- Regex Notes
- RMQ Scenarios
- RabbitMQ Cloud Service RPC Conventions

Regex Notes

Good site for testing Regex matching strings: https://regex101.com/r/iY5hU9/1

RMQ Scenarios

Change Events

A service that needs to publish change events for a domain.

An example of this would be a security directory that receives a change request for a user account, to change its username. This update would need to be propagated to other services that use or cache the user's username, and to client devices that display it.

In such a case, the security directory can publish the change event to a fanout exchange, and subscribers can deal with it appropriately.

- Caches can invalidate the appropriate record

RPC Requests

Cloud services that make requests needing deterministic responses use RPC over RMQ.

An implementation of this has been integration tested in the libraries, and provides an abstraction similar to a local method call.

See this for details: RabbitMQ Cloud Service RPC Conventions

RabbitMQ Cloud Service RPC Conventions

Here’s a list of conventions followed by cloud services for inter-service RPC communication.

RPC, being a request-response type of comms, like a REST call (request and response), requires some a standardized convention when implemented over an async queue transport.

Here’s a guide for simplified implementation of client RPC request spin-wait logic, and server side logic:

RabbitMQ tutorial - Remote procedure call (RPC) | RabbitMQ

Current Implementation

Currently, the RPC request and response implementation is integration tested.

The reference RPC client is here:

https://github.com/ogauto/OGA.RabbitMQClient.Lib/blob/main/OGA.RabbitMQClient.Lib/RMQ_Tests/HelperClasses/RMQClient_Test_RPCClient.cs

The reference RPC server is here:

https://github.com/ogauto/OGA.RabbitMQClient.Lib/blob/main/OGA.RabbitMQClient.Lib/RMQ_Tests/HelperClasses/RMQClient_Test_RPCServer.cs

RPC Exchanges

Two exchanges exist in the cluster for RPC comms.

RPC Request Exchange

This is a direct exchange where all RPC Requests are sent.

It is named, Cloud.Service.RPC1.Requests

Clients will publish requests to this exchange, with the relevant routingkey string for the request.

Servers that handle RPC requests will start a consumer on a persistent queue for their service type, with bindings of routingkeys for requests they handle.

RPC Response Exchange

This is a direct exchange where all RPC Responses are sent.

It is named, Cloud.Service.RPC1.Responses

Clients will bind to a local-queue to it with a unique routing key.

Servers that handle RPC requests will publish to this exchange with the routing key given in the client's request.

Request RoutingKey Convention

The naming convention for RPC routingkeys shall be:

RPC.Cloud.<ServiceType>.<Resource>.<Action>

A service can bind to the RPC Request exchange a number of ways.

It can service RPC requests for all instances of its service type:
“RPC.Cloud.<ServiceType>.*”

It can service RPC requests for a resource type:
“RPC.Cloud.*.<Resource>.*”

Messages

Two types of messages shall be accepted by these exchanges.

They are considered wrappers for the specific class types of the request.

All requests and responses shall use these two types. This keeps things a little more agnostic and easier to migrate to another message bus, if need be.

Requests

All requests will use the RPCRequestDTO struct.

The RPC client set its routingkey in each request, so the server can send a response.

The RPC client set a requestid in each request, so the server can provide idempotency.

The RPC client set a corelationId in each request so it can reconcile responses.

The RPC client will serialize its request as json and save to the message payload.

The RPC client will set the request class to the serialized class name.

The RPC client may use the message type to provide multiple uses for the same request class.

Responses

All responses will use the RPCResponseDTO struct.

The RPC server will set the responseid, so the client can provide idempotency.

The RPC server will set the corelationId to what the client gave.

The RPC server will serialize its response as json and save to the message payload.

The RPC server will set the response class to the serialized class name.

The RPC server will set the response type if needed to distinguish meaning of a common message class.

The RPC server will send a response for success or failure.

Failures will include an appropriate status and err string.