Spring Cloud Open Service Broker

Version 3.1.2.BUILD-SNAPSHOT

Table of Contents

Spring Cloud Open Service Broker is a framework for building Spring Boot applications that implement the Open Service Broker API.

Chapter 1. Introduction

The Open Service Broker API defines an HTTP interface between the services marketplace of a platform and a service broker.

Service brokers are responsible for:

- Advertising a catalog of their service offerings and plans
- Provisioning (creating or updating) service instances
- Creating bindings between a service instance and a client application
- Deleting bindings between a service instance and a client application
- De-provisioning (deleting) service instances

The Spring Cloud Open Service Broker project provides the scaffolding for an Open Service Broker API-compliant service broker by implementing the required Spring web controllers, domain objects, and configuration. Service broker authors can provide Spring beans that implement the appropriate interfaces.

Chapter 2. Getting started

Most service broker applications implement API or web UI endpoints beyond the Open Service Broker API endpoints. These additional endpoints might provide information about the application, provide a dashboard UI, or provide controls over application behavior. Developers may implement these additional endpoints with Spring WebFlux or Spring MVC.

The Spring Cloud Open Service Broker starter does not include a transitive dependency on Spring WebFlux or Spring MVC. A Spring Boot web starter is required to activate the auto-configuration.

2.1. Maven Dependencies

To use Spring Cloud Open Service Broker in a Spring web application, add the starter, as follows:

```
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-open-service-broker</artifactId>
            <version>${version}</version>
        </dependency>
    </dependencies>
```

2.2. Gradle Dependencies

To use Spring Cloud Open Service Broker in a Spring web application, add the starter, as follows:

```
    dependencies {
        api 'org.springframework.cloud:spring-cloud-starter-open-service-
broker:${version}'
    }
```

2.3. Configuring the Service Broker

See the Spring Boot documentation to get started building a Spring Boot application.

The framework provides default implementations of most of the components needed to implement a service broker. In Spring Boot fashion, you can override the default behavior by providing your own implementation of Spring beans, and the framework backs away from its defaults.

To start, use the `@SpringBootApplication` annotation on the service broker's main application class, as follows:

[source,XML

```
    @SpringBootApplication
    public class Application {
        public static void main(String[] args) {
            SpringApplication.run(Application.class, args);
        }
    }
```

This triggers the inclusion of the default configuration.

2.4. Using a Unique Platform ID

Every request to the service broker is able to receive a `platformInstanceId` through a path variable. This lets a service broker detect the identity of a platform to which it has been registered, as described in the Cloud Foundry documentation.

For example, an operator may register a service broker to one CF platform instance, as follows:

```
$ cf create-service-broker mybroker username password
https://mybroker.app.local/east
```

The operator may also register the same service broker to another CF platform instance, as follows:

```
$ cf create-service-broker mybroker username password
https://mybroker.app.local/west
```

The broker could then expect requests to the following paths, where the `platformInstanceId` field in the request object contains the value "east" or "west":

```
- https://username:password@mybroker.app.local/east/v2/catalog
- https://username:password@mybroker.app.local/west/v2/catalog
```

2.5. Customizing the Service Broker Path

Sometimes, it is useful to customize the prefix for the service broker path. For example, your application might be serving conflicting endpoints for another purpose. You can use the `spring.cloud.openservicebroker.base-path` property to change the prefix for your broker path, as

follows:

```
spring.cloud.openservicebroker.base-path=/broker
```

The preceding `application.properties` example changes the endpoint from `/` to `/broker/` (for example, `/broker/v2/catalog`).

Chapter 3. Advertising Services

The service broker catalog provides a set of metadata that describes the available services, along with attributes such as cost and capabilities. The catalog is made available to the platform's services marketplace through the service broker `/v2/catalog` endpoint.

The service broker can either provide a Spring bean of type `Catalog` or implement the service `CatalogService`.

3.1. Providing a Catalog Bean

You can expose the service broker catalog by creating a Spring bean and contributing it to the Spring application context. You can do so in a Spring `@Configuration` class, as follows:

```
package com.example.servicebroker;

import org.springframework.cloud.servicebroker.model.catalog.Catalog;
import org.springframework.cloud.servicebroker.model.catalog.Plan;
import org.springframework.cloud.servicebroker.model.catalog.ServiceDefinition;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class ExampleCatalogConfiguration {

    @Bean
    public Catalog catalog() {
        Plan plan = Plan.builder()
                .id("simple-plan")
                .name("standard")
                .description("A simple plan")
                .free(true)
                .build();

        ServiceDefinition serviceDefinition = ServiceDefinition.builder()
                .id("example-service")
                .name("example")
                .description("A simple example")
                .bindable(true)
                .tags("example", "tags")
                .plans(plan)
                .build();

        return Catalog.builder()
          .serviceDefinitions(serviceDefinition)
          .build();
    }
}
```

3.2. Providing a Catalog by Using Properties

You can configure a catalog with Spring Boot externalized configuration within a Java properties file or a YAML file. The catalog is parsed and made available as a `Catalog` bean during autoconfiguration.

The following YAML file configures a catalog:

```yaml
# Example Spring Boot YAML configuration
spring:
  cloud:
    openservicebroker:
      catalog:
        services:
        - id: example-service
          name: example
          description: A simple example
          bindable: true
          tags:
          - example
          - tags
          plans:
          - id: simple-plan
            name: standard
            description: A simple plan
```

The following properties file configures a catalog:

```
# Example Spring Boot properties configuration
spring.cloud.openservicebroker.catalog.services[0].id=example-service
spring.cloud.openservicebroker.catalog.services[0].name=example
spring.cloud.openservicebroker.catalog.services[0].description=A simple example
spring.cloud.openservicebroker.catalog.services[0].bindable=true
spring.cloud.openservicebroker.catalog.services[0].tags[0]=example
spring.cloud.openservicebroker.catalog.services[0].tags[1]=tags
spring.cloud.openservicebroker.catalog.services[0].plans[0].id=simple-plan
spring.cloud.openservicebroker.catalog.services[0].plans[0].name=standard
spring.cloud.openservicebroker.catalog.services[0].plans[0].description=A simple
plan
```

3.3. Implementing a Catalog Service

A service broker can take more control over the catalog by implementing the `CatalogService` interface. This might be required if some details of the catalog metadata need to be read from the environment or from an external data source.

The following example implements the `CatalogService` interface:

```
package com.example.servicebroker;

import reactor.core.publisher.Mono;

import org.springframework.cloud.servicebroker.model.catalog.Catalog;
import org.springframework.cloud.servicebroker.model.catalog.Plan;
import org.springframework.cloud.servicebroker.model.catalog.ServiceDefinition;
import org.springframework.cloud.servicebroker.service.CatalogService;
import org.springframework.stereotype.Service;

@Service
public class ExampleCatalogService implements CatalogService {

    @Override
    public Mono<Catalog> getCatalog() {
        return getServiceDefinition("example-service")
                .map(serviceDefinition -> Catalog.builder()
                        .serviceDefinitions(serviceDefinition)
                        .build());
    }

    @Override
    public Mono<ServiceDefinition> getServiceDefinition(String serviceId) {
        return Mono.just(ServiceDefinition.builder()
            .id(serviceId)
            .name("example")
            .description("A simple example")
            .bindable(true)
            .tags("example", "tags")
            .plans(getPlan())
            .build());
    }

    private Plan getPlan() {
        return Plan.builder()
           .id("simple-plan")
           .name("standard")
           .description("A simple plan")
           .free(true)
           .build();
    }
}
```

Chapter 4. Service Instances

Service brokers are responsible for provisioning the services advertised in their catalog and managing their lifecycle in the underlying cloud platform. The services created by the broker are referred to as service instances.

Service brokers must implement the `ServiceInstanceService` interface and provide implementations of the required methods of that interface. Each method receives a single Java object parameter that contains all the details of the request from the platform and returns a Java object value that provides the details of the operation to the platform.

The service instance create, update, and delete operations can be performed synchronously or asynchronously.

- When a service broker creates, updates, or deletes a service instance synchronously, the appropriate interface method should block and return a response to the platform only when the operation completes successfully or when a failure occurs.
- When performing an operation asynchronously, the service broker can return a response to the platform before the operation is complete and indicate in the response that the operation is in progress. The platform polls the service broker to get the status of the operation when an asynchronous operation is indicated.

4.1. Service Instance Creation

The service broker must provide an implementation of the `createServiceInstance()` method.

Service brokers typically provision a resource in the platform or in another system when they create a service instance. Service brokers are responsible for keeping track of any resources associated with a service instance for future retrieval, updating, or deletion.

4.1.1. Event Registry

Service instance creation can be further customized by utilizing events. To do so:

1. Autowire the `CreateServiceInstanceEventFlowRegistry` bean.
2. Use one of the `addInitializationFlow()`, `addCompletionFlow()`, or `addErrorFlow()` methods to register custom reactive flows for execution during the various stages of creating a service instance.

4.2. Service Instance Updating

If the `plan_updateable` field is set to `true` in the services catalog, the service broker must provide an implementation of the `updateServiceInstance()` method. Otherwise, this method is never called by the platform, and the default implementation in the interface can be used.

Services brokers can modify the configuration of an existing resource when updating a service instance or deploying a new resource.

4.2.1. Event Registry

Service instance updates can be further customized by utilizing events. To do so:

1. Autowire the `UpdateServiceInstanceEventFlowRegistry` bean.
2. Use one of the `addInitializationFlow()`, `addCompletionFlow()`, or `addErrorFlow()` methods to register custom reactive flows for execution during the various stages of updating a service instance.

4.3. Service Instance Deletion

An implementation of the `deleteServiceInstance()` method must be provided by the service broker.

Any resources provisioned in the create operation should be de-provisioned by the delete operation.

4.3.1. Event Registry

Service instance deletion can be further customized by utilizing events. To do so:

1. Autowire the `DeleteServiceInstanceEventFlowRegistry` bean.
2. Use one of the `addInitializationFlow()`, `addCompletionFlow()`, or `addErrorFlow()` methods to register custom reactive flows for execution during the various stages of deleting a service instance.

4.4. Service Instance Operation Status Retrieval

If any create, update, or delete operation can return an asynchronous “operation in progress” response to the platform, the service broker must provide an implementation of the `getLastOperation()` method. Otherwise, this method is never called by the platform, and the default implementation in the interface can be used.

The platform polls this method of the service broker for a service instance that has an asynchronous operation in progress until the service broker indicates that the operation has completed successfully or a failure has occurred.

4.4.1. Event Registry

Service instance last operation requests can be further customized by utilizing events. To do so:

1. Autowire the `AsyncOperationServiceInstanceEventFlowRegistry` bean.
2. Use one of the `addInitializationFlow()`, `addCompletionFlow()`, or `addErrorFlow()` methods to register custom reactive flows for execution during the various stages of last operation retrieval.

4.5. Service Instance Retrieval

If the `instances_retrievable` field is set to `true` in the services catalog, the service broker must provide an implementation of the `getServiceInstance()` method. Otherwise, this method is never called by the platform, and the default implementation in the interface can be used.

Service brokers are responsible for maintaining any service instance state necessary to support the retrieval operation.

4.6. Example Implementation

The following example shows a service instance implementation:

```
package com.example.servicebroker;

import java.util.Map;

import reactor.core.publisher.Mono;

import
org.springframework.cloud.servicebroker.model.instance.CreateServiceInstanceReques
t;
import
org.springframework.cloud.servicebroker.model.instance.CreateServiceInstanceRespon
se;
import
org.springframework.cloud.servicebroker.model.instance.DeleteServiceInstanceReques
t;
import
org.springframework.cloud.servicebroker.model.instance.DeleteServiceInstanceRespon
se;
import
org.springframework.cloud.servicebroker.model.instance.GetLastServiceOperationRequ
est;
import
org.springframework.cloud.servicebroker.model.instance.GetLastServiceOperationResp
onse;
import
org.springframework.cloud.servicebroker.model.instance.GetServiceInstanceRequest;
import
org.springframework.cloud.servicebroker.model.instance.GetServiceInstanceResponse;
import org.springframework.cloud.servicebroker.model.instance.OperationState;
import
org.springframework.cloud.servicebroker.model.instance.UpdateServiceInstanceReques
t;
import
org.springframework.cloud.servicebroker.model.instance.UpdateServiceInstanceRespon
se;
```

```
import org.springframework.cloud.servicebroker.service.ServiceInstanceService;
import org.springframework.stereotype.Service;

@Service
public class ExampleServiceInstanceService implements ServiceInstanceService {

    @Override
    public Mono<CreateServiceInstanceResponse> createServiceInstance
(CreateServiceInstanceRequest request) {
        String serviceInstanceId = request.getServiceInstanceId();
        String planId = request.getPlanId();
        Map<String, Object> parameters = request.getParameters();

        //
        // perform the steps necessary to initiate the asynchronous
        // provisioning of all necessary resources
        //

        String dashboardUrl = ""; /* construct a dashboard URL */

        return Mono.just(CreateServiceInstanceResponse.builder()
                .dashboardUrl(dashboardUrl)
                .async(true)
                .build());
    }

    @Override
    public Mono<UpdateServiceInstanceResponse> updateServiceInstance
(UpdateServiceInstanceRequest request) {
        String serviceInstanceId = request.getServiceInstanceId();
        String planId = request.getPlanId();
        String previousPlan = request.getPreviousValues().getPlanId();
        Map<String, Object> parameters = request.getParameters();

        //
        // perform the steps necessary to initiate the asynchronous
        // updating of all necessary resources
        //

        return Mono.just(UpdateServiceInstanceResponse.builder()
                .async(true)
                .build());
    }

    @Override
    public Mono<DeleteServiceInstanceResponse> deleteServiceInstance
(DeleteServiceInstanceRequest request) {
        String serviceInstanceId = request.getServiceInstanceId();
        String planId = request.getPlanId();

        //
```

```
        // perform the steps necessary to initiate the asynchronous
        // deletion of all provisioned resources
        //

        return Mono.just(DeleteServiceInstanceResponse.builder()
                .async(true)
                .build());
    }

    @Override
    public Mono<GetServiceInstanceResponse> getServiceInstance
(GetServiceInstanceRequest request) {
        String serviceInstanceId = request.getServiceInstanceId();

        //
        // retrieve the details of the specified service instance
        //

        String dashboardUrl = ""; /* retrieve dashboard URL */

        return Mono.just(GetServiceInstanceResponse.builder()
                .dashboardUrl(dashboardUrl)
                .build());
    }

    @Override
    public Mono<GetLastServiceOperationResponse> getLastOperation
(GetLastServiceOperationRequest request) {
        String serviceInstanceId = request.getServiceInstanceId();

        //
        // determine the status of the operation in progress
        //

        return Mono.just(GetLastServiceOperationResponse.builder()
                .operationState(OperationState.SUCCEEDED)
                .build());
    }
}
```

4.7. Example Event Flow Configuration

There are multiple ways to configure service instance event flows. One option is to autowire one or more registries and interact with the registry directly. Another option is to define beans for specific flows. These beans are automatically identified and added to the appropriate registry. A final option is to declare a new registry bean. However, be aware that defining a new registry bean overrides the provided auto-configuration.

4.7.1. Option 1: Autowire Registries

The following example shows a configuration for service instance event flows:

```
package com.example.servicebroker;

import reactor.core.publisher.Mono;

import
org.springframework.cloud.servicebroker.model.instance.CreateServiceInstanceReques
t;
import
org.springframework.cloud.servicebroker.model.instance.CreateServiceInstanceRespon
se;
import
org.springframework.cloud.servicebroker.model.instance.DeleteServiceInstanceReques
t;
import
org.springframework.cloud.servicebroker.model.instance.DeleteServiceInstanceRespon
se;
import
org.springframework.cloud.servicebroker.model.instance.GetLastServiceOperationRequ
est;
import
org.springframework.cloud.servicebroker.model.instance.GetLastServiceOperationResp
onse;
import
org.springframework.cloud.servicebroker.model.instance.UpdateServiceInstanceReques
t;
import
org.springframework.cloud.servicebroker.model.instance.UpdateServiceInstanceRespon
se;
import
org.springframework.cloud.servicebroker.service.events.AsyncOperationServiceInstan
ceEventFlowRegistry;
import
org.springframework.cloud.servicebroker.service.events.CreateServiceInstanceEventF
lowRegistry;
import
org.springframework.cloud.servicebroker.service.events.DeleteServiceInstanceEventF
lowRegistry;
import
org.springframework.cloud.servicebroker.service.events.UpdateServiceInstanceEventF
lowRegistry;
import
org.springframework.cloud.servicebroker.service.events.flows.AsyncOperationService
InstanceCompletionFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.AsyncOperationService
InstanceErrorFlow;
```

```
import
org.springframework.cloud.servicebroker.service.events.flows.AsyncOperationService
InstanceInitializationFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.CreateServiceInstance
CompletionFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.CreateServiceInstance
ErrorFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.CreateServiceInstance
InitializationFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.DeleteServiceInstance
CompletionFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.DeleteServiceInstance
ErrorFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.DeleteServiceInstance
InitializationFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.UpdateServiceInstance
CompletionFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.UpdateServiceInstance
ErrorFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.UpdateServiceInstance
InitializationFlow;
import org.springframework.context.annotation.Configuration;

@Configuration
public class ExampleServiceInstanceEventFlowsConfiguration {

    private final CreateServiceInstanceEventFlowRegistry createRegistry;

    private final UpdateServiceInstanceEventFlowRegistry updateRegistry;

    private final DeleteServiceInstanceEventFlowRegistry deleteRegistry;

    private final AsyncOperationServiceInstanceEventFlowRegistry asyncRegistry;

    public ExampleServiceInstanceEventFlowsConfiguration
(CreateServiceInstanceEventFlowRegistry createRegistry,
            UpdateServiceInstanceEventFlowRegistry updateRegistry,
            DeleteServiceInstanceEventFlowRegistry deleteRegistry,
            AsyncOperationServiceInstanceEventFlowRegistry asyncRegistry) {
        this.createRegistry = createRegistry;
        this.updateRegistry = updateRegistry;
        this.deleteRegistry = deleteRegistry;
```

```
        this.asyncRegistry = asyncRegistry;

        prepareCreateEventFlows()
                .then(prepareUpdateEventFlows())
                .then(prepareDeleteEventFlows())
                .then(prepareLastOperationEventFlows())
                .subscribe();
    }

    private Mono<Void> prepareCreateEventFlows() {
        return Mono.just(createRegistry)
                .map(registry -> registry.addInitializationFlow(new
CreateServiceInstanceInitializationFlow() {
                    @Override
                    public Mono<Void> initialize(CreateServiceInstanceRequest
request) {
                        //
                        // do something before the instance is created
                        //
                        return Mono.empty();
                    }
                })
                        .then(registry.addCompletionFlow(new
CreateServiceInstanceCompletionFlow() {
                            @Override
                            public Mono<Void> complete
(CreateServiceInstanceRequest request,
                                    CreateServiceInstanceResponse response) {
                                //
                                // do something after the instance is created
                                //
                                return Mono.empty();
                            }
                        }))
                        .then(registry.addErrorFlow(new
CreateServiceInstanceErrorFlow() {
                            @Override
                            public Mono<Void> error(CreateServiceInstanceRequest
request, Throwable t) {
                                //
                                // do something if an error occurs while creating
an instance
                                //
                                return Mono.empty();
                            }
                        })))
                .then();
    }

    private Mono<Void> prepareUpdateEventFlows() {
        return Mono.just(updateRegistry)
```

```
                .map(registry -> registry.addInitializationFlow(new
UpdateServiceInstanceInitializationFlow() {
                    @Override
                    public Mono<Void> initialize(UpdateServiceInstanceRequest
request) {
                        //
                        // do something before the instance is updated
                        //
                        return Mono.empty();
                    }
                })
                        .then(registry.addCompletionFlow(new
UpdateServiceInstanceCompletionFlow() {
                            @Override
                            public Mono<Void> complete
(UpdateServiceInstanceRequest request,
                                    UpdateServiceInstanceResponse response) {
                                //
                                // do something after the instance is updated
                                //
                                return Mono.empty();
                            }
                        }))
                        .then(registry.addErrorFlow(new
UpdateServiceInstanceErrorFlow() {
                            @Override
                            public Mono<Void> error(UpdateServiceInstanceRequest
request, Throwable t) {
                                //
                                // do something if an error occurs while updating
an instance
                                //
                                return Mono.empty();
                            }
                        })))
                .then();
    }

    private Mono<Void> prepareDeleteEventFlows() {
        return Mono.just(deleteRegistry)
                .map(registry -> registry.addInitializationFlow(new
DeleteServiceInstanceInitializationFlow() {
                    @Override
                    public Mono<Void> initialize(DeleteServiceInstanceRequest
request) {
                        //
                        // do something before the instance is deleted
                        //
                        return Mono.empty();
                    }
                })
```

```
                        .then(registry.addCompletionFlow(new
DeleteServiceInstanceCompletionFlow() {
                            @Override
                            public Mono<Void> complete
(DeleteServiceInstanceRequest request,
                                    DeleteServiceInstanceResponse response) {
                                //
                                // do something after the instance is deleted
                                //
                                return Mono.empty();
                            }
                        }))
                        .then(registry.addErrorFlow(new
DeleteServiceInstanceErrorFlow() {
                            @Override
                            public Mono<Void> error(DeleteServiceInstanceRequest
request, Throwable t) {
                                //
                                // do something if an error occurs while deleting
an instance
                                //
                                return Mono.empty();
                            }
                        })))
                .then();
    }

    private Mono<Void> prepareLastOperationEventFlows() {
        return Mono.just(asyncRegistry)
                .map(registry -> registry.addInitializationFlow(new
AsyncOperationServiceInstanceInitializationFlow() {
                    @Override
                    public Mono<Void> initialize(GetLastServiceOperationRequest
request) {
                        //
                        // do something before returning the last operation
                        //
                        return Mono.empty();
                    }
                })
                        .then(registry.addCompletionFlow(new
AsyncOperationServiceInstanceCompletionFlow() {
                            @Override
                            public Mono<Void> complete
(GetLastServiceOperationRequest request,
                                    GetLastServiceOperationResponse response) {
                                //
                                // do something after returning the last operation
                                //
                                return Mono.empty();
                            }
```

```
                        }))
                        .then(registry.addErrorFlow(new
AsyncOperationServiceInstanceErrorFlow() {
                            @Override
                            public Mono<Void> error(GetLastServiceOperationRequest
request, Throwable t) {
                                //
                                // do something if an error occurs while
processing the last operation response
                                //
                                return Mono.empty();
                            }
                        })))
                .then();
    }

}
```

4.7.2. Option 2: Event Flow Beans

Optionally, you can configure beans for the individual flows, as follows:

```
package com.example.servicebroker;

import reactor.core.publisher.Mono;

import
org.springframework.cloud.servicebroker.model.instance.CreateServiceInstanceReques
t;
import
org.springframework.cloud.servicebroker.model.instance.CreateServiceInstanceRespon
se;
import
org.springframework.cloud.servicebroker.model.instance.DeleteServiceInstanceReques
t;
import
org.springframework.cloud.servicebroker.model.instance.DeleteServiceInstanceRespon
se;
import
org.springframework.cloud.servicebroker.model.instance.GetLastServiceOperationRequ
est;
import
org.springframework.cloud.servicebroker.model.instance.GetLastServiceOperationResp
onse;
import
org.springframework.cloud.servicebroker.model.instance.UpdateServiceInstanceReques
t;
import
```

```
org.springframework.cloud.servicebroker.model.instance.UpdateServiceInstanceRespon
se;
import
org.springframework.cloud.servicebroker.service.events.flows.AsyncOperationService
InstanceCompletionFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.AsyncOperationService
InstanceErrorFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.AsyncOperationService
InstanceInitializationFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.CreateServiceInstance
CompletionFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.CreateServiceInstance
ErrorFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.CreateServiceInstance
InitializationFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.DeleteServiceInstance
CompletionFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.DeleteServiceInstance
ErrorFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.DeleteServiceInstance
InitializationFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.UpdateServiceInstance
CompletionFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.UpdateServiceInstance
ErrorFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.UpdateServiceInstance
InitializationFlow;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class ExampleServiceInstanceEventFlowsConfiguration2 {

    //
    // Create Service Instance flows
    //

    @Bean
    public CreateServiceInstanceInitializationFlow
createServiceInstanceInitializationFlow() {
```

```
        return new CreateServiceInstanceInitializationFlow() {
            @Override
            public Mono<Void> initialize(CreateServiceInstanceRequest request) {
                //
                // do something before the instance is created
                //
                return Mono.empty();
            }
        };
    }

    @Bean
    public CreateServiceInstanceCompletionFlow
createServiceInstanceCompletionFlow() {
        return new CreateServiceInstanceCompletionFlow() {
            @Override
            public Mono<Void> complete(CreateServiceInstanceRequest request,
                    CreateServiceInstanceResponse response) {
                //
                // do something after the instance is created
                //
                return Mono.empty();
            }
        };
    }

    @Bean
    public CreateServiceInstanceErrorFlow createServiceInstanceErrorFlow() {
        return new CreateServiceInstanceErrorFlow() {
            @Override
            public Mono<Void> error(CreateServiceInstanceRequest request,
Throwable t) {
                //
                // do something if an error occurs while creating an instance
                //
                return Mono.empty();
            }
        };
    }

    //
    // Update Service Instance flows
    //

    @Bean
    public UpdateServiceInstanceInitializationFlow
updateServiceInstanceInitializationFlow() {
        return new UpdateServiceInstanceInitializationFlow() {
            @Override
            public Mono<Void> initialize(
                    UpdateServiceInstanceRequest request) {
```

```
                //
                // do something before the instance is updated
                //
                return Mono.empty();
            }
        };
    }

    @Bean
    public UpdateServiceInstanceCompletionFlow
updateServiceInstanceCompletionFlow() {
        return new UpdateServiceInstanceCompletionFlow() {
            @Override
            public Mono<Void> complete(UpdateServiceInstanceRequest request,
                    UpdateServiceInstanceResponse response) {
                //
                // do something after the instance is updated
                //
                return Mono.empty();
            }
        };
    }

    @Bean
    public UpdateServiceInstanceErrorFlow updateServiceInstanceErrorFlow() {
        return new UpdateServiceInstanceErrorFlow() {
            @Override
            public Mono<Void> error(UpdateServiceInstanceRequest request,
Throwable t) {
                //
                // do something if an error occurs while updating an instance
                //
                return Mono.empty();
            }
        };
    }

    //
    // Delete Service Instance flows
    //

    @Bean
    public DeleteServiceInstanceInitializationFlow
deleteServiceInstanceInitializationFlow() {
        return new DeleteServiceInstanceInitializationFlow() {
            @Override
            public Mono<Void> initialize(
                    DeleteServiceInstanceRequest request) {
                //
                // do something before the instance is deleted
                //
```

```
                return Mono.empty();
            }
        };
    }

    @Bean
    public DeleteServiceInstanceCompletionFlow
deleteServiceInstanceCompletionFlow() {
        return new DeleteServiceInstanceCompletionFlow() {
            @Override
            public Mono<Void> complete(DeleteServiceInstanceRequest request,
                    DeleteServiceInstanceResponse response) {
                //
                // do something after the instance is deleted
                //
                return Mono.empty();
            }
        };
    }

    @Bean
    public DeleteServiceInstanceErrorFlow deleteServiceInstanceErrorFlow() {
        return new DeleteServiceInstanceErrorFlow() {
            @Override
            public Mono<Void> error(DeleteServiceInstanceRequest request,
Throwable t) {
                //
                // do something if an error occurs while deleting the instance
                //
                return Mono.empty();
            }
        };
    }

    //
    // Get Last Operation flows
    //

    @Bean
    public AsyncOperationServiceInstanceInitializationFlow
getLastOperationInitializationFlow() {
        return new AsyncOperationServiceInstanceInitializationFlow() {
            @Override
            public Mono<Void> initialize(
                    GetLastServiceOperationRequest request) {
                //
                // do something before getting the last operation
                //
                return Mono.empty();
            }
        };
```

```
    }

    @Bean
    public AsyncOperationServiceInstanceCompletionFlow
getLastOperationCompletionFlow() {
        return new AsyncOperationServiceInstanceCompletionFlow() {
            @Override
            public Mono<Void> complete(GetLastServiceOperationRequest request,
                    GetLastServiceOperationResponse response) {
                //
                // do something after getting the last operation
                //
                return Mono.empty();
            }
        };
    }

    @Bean
    public AsyncOperationServiceInstanceErrorFlow getLastOperationErrorFlow() {
        return new AsyncOperationServiceInstanceErrorFlow() {
            @Override
            public Mono<Void> error(GetLastServiceOperationRequest request,
Throwable t) {
                //
                // do something if an error occurs while getting the last
operation
                //
                return Mono.empty();
            }
        };
    }

}
```

Chapter 5. Service Bindings

Service brokers can provide information to a consumer of a service instance through a service binding. Service bindings are often used to expose credentials for service instance resources to an application.

If the `bindable` field is set to `true` for any plan in the service catalog, the service broker must provide an implementation of the `ServiceInstanceBindingService` interface. Otherwise, the binding methods of the service broker are not called by the platform, and a default implementation of this interface can be used. Each method receives a single Java object parameter that contains all the details of the request from the platform and returns a Java object value that provides details of the operation to the platform.

The service binding create and delete operations can be performed synchronously or asynchronously.

- When a service broker creates or deletes a service binding synchronously, the appropriate interface method should block and return a response to the platform only when the operation completes successfully or when a failure occurs.
- When performing an operation asynchronously, the service broker can return a response to the platform before the operation is complete and indicate in the response that the operation is in progress. The platform polls the service broker to get the status of the operation when an asynchronous operation is indicated.

5.1. Service Binding Creation

The service broker must provide an implementation of the `createServiceInstanceBinding()` method.

Two types of bindings are supported:

- App bindings can be used to provide credentials, log drains, and volume services to applications.
- Route bindings can be used to provide routes for the platform to use when proxying requests.

The response from this method lets one of two Java object types be returned, reflecting the two types of supported bindings.

Service brokers can generate one set of credentials for all binding requests or provide unique credentials for each binding request.

5.1.1. Event Registry

You can use events to further customize service binding creation. To do so:

1. Autowire the `CreateServiceInstanceBindingEventFlowRegistry` bean.
2. Use one of the `addInitializationFlow()`, `addCompletionFlow()`, or `addErrorFlow()` methods to register custom reactive flows for execution during the various stages of creating a service binding.

5.2. Service Binding Deletion

The service broker must provide an implementation of the `deleteServiceInstanceBinding()` method.

Any credentials provisioned in the create operation should be de-provisioned by the delete operation.

5.2.1. Event Registry

You can use events to further customize service binding deletion. To do so: . Autowire the `DeleteServiceInstanceBindingEventFlowRegistry` bean.

1. Use one of the `addInitializationFlow()`, `addCompletionFlow()`, or `addErrorFlow()` methods to register custom reactive flows for execution during the various stages of deleting a service binding.

5.3. Service Binding Operation Status Retrieval

If any create or delete operation can return an asynchronous “operation in progress” response to the platform, the service broker must provide an implementation of the `getLastOperation()` method. Otherwise, this method is never called by the platform, and the default implementation in the interface can be used.

The platform polls this method of the service broker for a service instance that has an asynchronous operation in progress until the service broker indicates that the operation has completed successfully or a failure has occurred.

5.3.1. Event Registry

You can use events to further customize service binding last operation requests. To do so: . Autowire the `AsyncOperationServiceInstanceBindingEventFlowRegistry` bean.

1. Use one of the `addInitializationFlow()`, `addCompletionFlow()` or `addErrorFlow()` methods to register custom reactive flows for execution during the various stages of last operation retrieval.

5.4. Service Binding Retrieval

If the `bindings_retrievable` field is set to `true` in the services catalog, the service catalog must provide an implementation of the `getServiceInstanceBinding()` method. Otherwise, this method is never called by the platform, and the default implementation in the interface can be used.

Service brokers are responsible for maintaining any service binding state necessary to support the retrieval operation.

5.5. Example Implementation

The following example implements a service binding:

```
package com.example.servicebroker;

import reactor.core.publisher.Mono;

import
org.springframework.cloud.servicebroker.model.binding.CreateServiceInstanceAppBind
ingResponse;
import
org.springframework.cloud.servicebroker.model.binding.CreateServiceInstanceBinding
Request;
import
org.springframework.cloud.servicebroker.model.binding.CreateServiceInstanceBinding
Response;
import
org.springframework.cloud.servicebroker.model.binding.DeleteServiceInstanceBinding
Request;
import
org.springframework.cloud.servicebroker.model.binding.DeleteServiceInstanceBinding
Response;
import
org.springframework.cloud.servicebroker.model.binding.GetServiceInstanceAppBinding
Response;
import
org.springframework.cloud.servicebroker.model.binding.GetServiceInstanceBindingReq
uest;
import
org.springframework.cloud.servicebroker.model.binding.GetServiceInstanceBindingRes
ponse;
import
org.springframework.cloud.servicebroker.service.ServiceInstanceBindingService;
import org.springframework.stereotype.Service;

@Service
public class ExampleServiceBindingService implements ServiceInstanceBindingService
{

    @Override
    public Mono<CreateServiceInstanceBindingResponse>
createServiceInstanceBinding(CreateServiceInstanceBindingRequest request) {
        String serviceInstanceId = request.getServiceInstanceId();
        String bindingId = request.getBindingId();

        //
        // create credentials and store for later retrieval
        //

        String url = new String(/* build a URL to access the service instance */);
        String bindingUsername = new String(/* create a user */);
        String bindingPassword = new String(/* create a password */);
```

```
        CreateServiceInstanceBindingResponse response =
CreateServiceInstanceAppBindingResponse.builder()
                .credentials("url", url)
                .credentials("username", bindingUsername)
                .credentials("password", bindingPassword)
                .bindingExisted(false)
                .async(true)
                .build();

        return Mono.just(response);
    }

    @Override
    public Mono<DeleteServiceInstanceBindingResponse>
deleteServiceInstanceBinding(DeleteServiceInstanceBindingRequest request) {
        String serviceInstanceId = request.getServiceInstanceId();
        String bindingId = request.getBindingId();

        //
        // delete any binding-specific credentials
        //

        return Mono.just(DeleteServiceInstanceBindingResponse.builder()
                .async(true)
                .build());
    }

    @Override
    public Mono<GetServiceInstanceBindingResponse> getServiceInstanceBinding
(GetServiceInstanceBindingRequest request) {
        String serviceInstanceId = request.getServiceInstanceId();
        String bindingId = request.getBindingId();

        //
        // retrieve the details of the specified service binding
        //

        String url = new String(/* retrieved URL */);
        String bindingUsername = new String(/* retrieved user */);
        String bindingPassword = new String(/* retrieved password */);

        GetServiceInstanceBindingResponse response =
GetServiceInstanceAppBindingResponse.builder()
                .credentials("username", bindingUsername)
                .credentials("password", bindingPassword)
                .credentials("url", url)
                .build();

        return Mono.just(response);
    }
```

```
}
```

5.6. Example Event Flow Configuration

There are multiple ways to configure service binding event flows. One option is to autowire one or more registries and interact with the registry directly. Another option is to define beans for specific flows. These beans are automatically identified and added to the appropriate registry. A final option is to declare a new registry bean. However, be aware that defining a new registry bean overrides the provided auto-configuration.

5.6.1. Option 1: Autowire Registries

The following example configures service binding event flows:

```
package com.example.servicebroker;

import reactor.core.publisher.Mono;

import
org.springframework.cloud.servicebroker.model.binding.CreateServiceInstanceBinding
Request;
import
org.springframework.cloud.servicebroker.model.binding.CreateServiceInstanceBinding
Response;
import
org.springframework.cloud.servicebroker.model.binding.DeleteServiceInstanceBinding
Request;
import
org.springframework.cloud.servicebroker.model.binding.DeleteServiceInstanceBinding
Response;
import
org.springframework.cloud.servicebroker.model.binding.GetLastServiceBindingOperati
onRequest;
import
org.springframework.cloud.servicebroker.model.binding.GetLastServiceBindingOperati
onResponse;
import
org.springframework.cloud.servicebroker.service.events.AsyncOperationServiceInstan
ceBindingEventFlowRegistry;
import
org.springframework.cloud.servicebroker.service.events.CreateServiceInstanceBindin
gEventFlowRegistry;
import
org.springframework.cloud.servicebroker.service.events.DeleteServiceInstanceBindin
gEventFlowRegistry;
import
org.springframework.cloud.servicebroker.service.events.flows.AsyncOperationService
```

```
InstanceBindingCompletionFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.AsyncOperationService
InstanceBindingErrorFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.AsyncOperationService
InstanceBindingInitializationFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.CreateServiceInstance
BindingCompletionFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.CreateServiceInstance
BindingErrorFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.CreateServiceInstance
BindingInitializationFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.DeleteServiceInstance
BindingCompletionFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.DeleteServiceInstance
BindingErrorFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.DeleteServiceInstance
BindingInitializationFlow;
import org.springframework.context.annotation.Configuration;

@Configuration
public class ExampleServiceBindingEventFlowsConfiguration {

    private final CreateServiceInstanceBindingEventFlowRegistry createRegistry;

    private final DeleteServiceInstanceBindingEventFlowRegistry deleteRegistry;

    private final AsyncOperationServiceInstanceBindingEventFlowRegistry
asyncRegistry;

    public ExampleServiceBindingEventFlowsConfiguration(
            CreateServiceInstanceBindingEventFlowRegistry createRegistry,
            DeleteServiceInstanceBindingEventFlowRegistry deleteRegistry,
            AsyncOperationServiceInstanceBindingEventFlowRegistry asyncRegistry) {
        this.createRegistry = createRegistry;
        this.deleteRegistry = deleteRegistry;
        this.asyncRegistry = asyncRegistry;

        prepareCreateEventFlows()
                .then(prepareDeleteEventFlows())
                .then(prepareLastOperationEventFlows())
                .subscribe();
    }
```

```
    private Mono<Void> prepareCreateEventFlows() {
        return Mono.just(createRegistry)
                .map(registry -> registry.addInitializationFlow(new
CreateServiceInstanceBindingInitializationFlow() {
                    @Override
                    public Mono<Void> initialize
(CreateServiceInstanceBindingRequest request) {
                        //
                        // do something before the instance is created
                        //
                        return Mono.empty();
                    }
                })
                        .then(registry.addCompletionFlow(new
CreateServiceInstanceBindingCompletionFlow() {
                            @Override
                            public Mono<Void> complete
(CreateServiceInstanceBindingRequest request,

CreateServiceInstanceBindingResponse response) {
                                //
                                // do something after the instance is created
                                //
                                return Mono.empty();
                            }
                        }))
                        .then(registry.addErrorFlow(new
CreateServiceInstanceBindingErrorFlow() {
                            @Override
                            public Mono<Void> error
(CreateServiceInstanceBindingRequest request, Throwable t) {
                                //
                                // do something if an error occurs while creating
an instance
                                //
                                return Mono.empty();
                            }
                        })))
                .then();
    }

    private Mono<Void> prepareDeleteEventFlows() {
        return Mono.just(deleteRegistry)
                .map(registry -> registry.addInitializationFlow(new
DeleteServiceInstanceBindingInitializationFlow() {
                    @Override
                    public Mono<Void> initialize
(DeleteServiceInstanceBindingRequest request) {
                        //
                        // do something before the instance is deleted
                        //
```

```
                        return Mono.empty();
                    }
                })
                        .then(registry.addCompletionFlow(new
DeleteServiceInstanceBindingCompletionFlow() {
                            @Override
                            public Mono<Void> complete
(DeleteServiceInstanceBindingRequest request,

DeleteServiceInstanceBindingResponse response) {
                                //
                                // do something after the instance is deleted
                                //
                                return Mono.empty();
                            }
                        }))
                        .then(registry.addErrorFlow(new
DeleteServiceInstanceBindingErrorFlow() {
                            @Override
                            public Mono<Void> error
(DeleteServiceInstanceBindingRequest request, Throwable t) {
                                //
                                // do something if an error occurs while deleting
an instance
                                //
                                return Mono.empty();
                            }
                        })))
                .then();
    }

    private Mono<Void> prepareLastOperationEventFlows() {
        return Mono.just(asyncRegistry)
                .map(registry -> registry.addInitializationFlow(new
AsyncOperationServiceInstanceBindingInitializationFlow() {
                    @Override
                    public Mono<Void> initialize
(GetLastServiceBindingOperationRequest request) {
                        //
                        // do something before the instance is deleted
                        //
                        return Mono.empty();
                    }
                })
                        .then(registry.addCompletionFlow(new
AsyncOperationServiceInstanceBindingCompletionFlow() {
                            @Override
                            public Mono<Void> complete
(GetLastServiceBindingOperationRequest request,
                                    GetLastServiceBindingOperationResponse
response) {
```

```
                                //
                                // do something after the instance is deleted
                                //
                                return Mono.empty();
                            }
                        }))
                        .then(registry.addErrorFlow(new
AsyncOperationServiceInstanceBindingErrorFlow() {
                            public Mono<Void> error
(GetLastServiceBindingOperationRequest request, Throwable t) {
                                //
                                // do something if an error occurs while deleting
an instance
                                //
                                return Mono.empty();
                            }
                        })))
                .then();
    }

}
```

5.6.2. Option 2: Event Flow Beans

Optionally, you can configure beans for the individual flows, as follows:

```
package com.example.servicebroker;

import reactor.core.publisher.Mono;

import
org.springframework.cloud.servicebroker.model.binding.CreateServiceInstanceBinding
Request;
import
org.springframework.cloud.servicebroker.model.binding.CreateServiceInstanceBinding
Response;
import
org.springframework.cloud.servicebroker.model.binding.DeleteServiceInstanceBinding
Request;
import
org.springframework.cloud.servicebroker.model.binding.DeleteServiceInstanceBinding
Response;
import
org.springframework.cloud.servicebroker.model.binding.GetLastServiceBindingOperati
onRequest;
import
org.springframework.cloud.servicebroker.model.binding.GetLastServiceBindingOperati
onResponse;
```

```
import
org.springframework.cloud.servicebroker.service.events.flows.AsyncOperationService
InstanceBindingCompletionFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.AsyncOperationService
InstanceBindingErrorFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.AsyncOperationService
InstanceBindingInitializationFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.CreateServiceInstance
BindingCompletionFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.CreateServiceInstance
BindingErrorFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.CreateServiceInstance
BindingInitializationFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.DeleteServiceInstance
BindingCompletionFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.DeleteServiceInstance
BindingErrorFlow;
import
org.springframework.cloud.servicebroker.service.events.flows.DeleteServiceInstance
BindingInitializationFlow;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class ExampleServiceBindingEventFlowsConfiguration2 {

    //
    // Create Service Instance Binding flows
    //

    @Bean
    public CreateServiceInstanceBindingInitializationFlow
createServiceInstanceBindingInitializationFlow() {
        return new CreateServiceInstanceBindingInitializationFlow() {
            @Override
            public Mono<Void> initialize(CreateServiceInstanceBindingRequest
request) {
                //
                // do something before the service instance binding completes
                //
                return Mono.empty();
            }
        };
    }
```

```
    @Bean
    public CreateServiceInstanceBindingCompletionFlow
createServiceInstanceBindingCompletionFlow() {
        return new CreateServiceInstanceBindingCompletionFlow() {
            @Override
            public Mono<Void> complete(CreateServiceInstanceBindingRequest
request,
                    CreateServiceInstanceBindingResponse response) {
                //
                // do something after the service instance binding completes
                //
                return Mono.empty();
            }
        };
    }

    @Bean
    public CreateServiceInstanceBindingErrorFlow
createServiceInstanceBindingErrorFlow() {
        return new CreateServiceInstanceBindingErrorFlow() {
            @Override
            public Mono<Void> error(CreateServiceInstanceBindingRequest request,
Throwable t) {
                //
                // do something if an error occurs while creating a service
instance binding
                //
                return Mono.empty();
            }
        };
    }

    //
    // Delete Service Instance Binding flows
    //

    @Bean
    public DeleteServiceInstanceBindingInitializationFlow
deleteServiceInstanceBindingInitializationFlow() {
        return new DeleteServiceInstanceBindingInitializationFlow() {
            @Override
            public Mono<Void> initialize(DeleteServiceInstanceBindingRequest
request) {
                //
                // do something before the service instance binding is deleted
                //
                return Mono.empty();
            }
        };
    }
```

```
    @Bean
    public DeleteServiceInstanceBindingCompletionFlow
deleteServiceInstanceBindingCompletionFlow() {
        return new DeleteServiceInstanceBindingCompletionFlow() {
            @Override
            public Mono<Void> complete(DeleteServiceInstanceBindingRequest
request,
                    DeleteServiceInstanceBindingResponse response) {
                //
                // do something after the service instance binding is deleted
                //
                return Mono.empty();
            }
        };
    }

    @Bean
    public DeleteServiceInstanceBindingErrorFlow
deleteServiceInstanceBindingErrorFlow() {
        return new DeleteServiceInstanceBindingErrorFlow() {
            @Override
            public Mono<Void> error(DeleteServiceInstanceBindingRequest request,
Throwable t) {
                //
                // do something if an error occurs while deleting a service
instance binding
                //
                return Mono.empty();
            }
        };
    }

    //
    // Get Last Service Instance Binding Operation flows
    //

    @Bean
    public AsyncOperationServiceInstanceBindingInitializationFlow
getLastOperationServiceInstanceBindingInitializationFlow() {
        return new AsyncOperationServiceInstanceBindingInitializationFlow() {
            @Override
            public Mono<Void> initialize(GetLastServiceBindingOperationRequest
request) {
                //
                // do something before getting the last operation
                //
                return Mono.empty();
            }
        };
    }
```

```
    @Bean
    public AsyncOperationServiceInstanceBindingCompletionFlow
getLastOperationServiceInstanceBindingCompletionFlow() {
        return new AsyncOperationServiceInstanceBindingCompletionFlow() {
            @Override
            public Mono<Void> complete(GetLastServiceBindingOperationRequest
request,
                    GetLastServiceBindingOperationResponse response) {
                //
                // do something after getting the last operation
                //
                return Mono.empty();
            }
        };
    }

    @Bean
    public AsyncOperationServiceInstanceBindingErrorFlow
getLastOperationServiceInstanceBindingErrorFlow() {
        return new AsyncOperationServiceInstanceBindingErrorFlow() {
            @Override
            public Mono<Void> error(GetLastServiceBindingOperationRequest request,
Throwable t) {
                //
                // do something if an error occurs while getting the last
operation
                //
                return Mono.empty();
            }
        };
    }
}
```

Chapter 6. API version verification

Platforms are required to provide an HTTP header in each call to the Open Service Broker API, to indicate the version of the API specification that the platform supports. You can configure Spring Cloud Open Service Broker to verify the version provided by the platform on each call to the service broker. By default, this version verification is configured to allow any API version.

To customize the version verification, set the `apiVersion` property that specifies the API version required by the service broker, as follows:

```
spring.cloud.openservicebroker.apiVersion=2.13
```

Alternatively, you can provide a `BrokerApiVersion` Spring bean, as follows:

```
package com.example.servicebroker;

import org.springframework.cloud.servicebroker.model.BrokerApiVersion;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class ExampleApiVersionConfiguration {
    @Bean
    public BrokerApiVersion brokerApiVersion() {
        return new BrokerApiVersion("2.13");
    }
}
```

In the case of both a Spring Bean and a property being configured, the Spring Bean takes precedence over the property.

If an API version is specified and the platform provides a different version in the `X-Broker-API-Version` header, the framework returns a `412 Precondition Failed` error to the platform.

As mentioned earlier, the default version verification is configured to allow any API version. However, to disable version verification entirely, you can set the `api-version-check-endabled` property to `false`, as follows:

```
spring.cloud.openservicebroker.api-version-check-enabled = false
```

Chapter 7. Service Broker Security

Authentication and authorization of service broker endpoints is not specified in the Open Service Broker API specification, but some platforms require or let basic authentication or OAuth2 credentials be provided when a service broker is registered to the platform.

The Spring Cloud Open Service Broker project does not implement any security configuration. Service broker application endpoints can be secured with Spring Security and Spring Boot security configuration by applying security to application endpoints with the path-matching pattern: `/v2/**`.

7.1. Example Configuration

The following example implements a security configuration:

```
package com.example.servicebroker;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity
;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfig
urerAdapter;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.provisioning.InMemoryUserDetailsManager;

@Configuration
@EnableWebSecurity
public class ExampleSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
                .csrf().disable()
                .authorizeRequests()
                .antMatchers("/v2/**").hasRole("ADMIN")
                .and()
                .httpBasic();
    }

    @Bean
    public InMemoryUserDetailsManager userDetailsService() {
        return new InMemoryUserDetailsManager(adminUser());
    }

    private UserDetails adminUser() {
        return User
                .withUsername("admin")
                .password("{noop}supersecret")
                .roles("ADMIN")
                .build();
    }
}
```

Chapter 8. Example Service Broker Application

The Bookstore Service Broker project implements a simple service broker that adheres to the Open Service Broker API by using the Spring Cloud Open Service Broker framework. It can be deployed to either Cloud Foundry or Kubernetes and can be registered as a service broker to either platform. View the project README for more information.