Introduction to Spring Cloud

Cloud Bus

LXFT LISTED NYSE

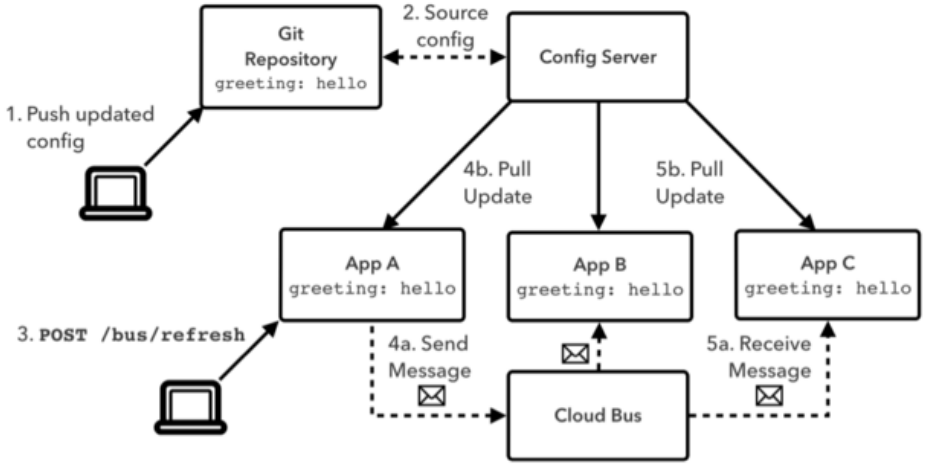
LUXOFT

Cloud Bus

- Until now we used REST APIs when we talked about microservices.
  - REST is synchronous.
  - What if we need an asynchronous approach?
- Cloud Bus is the standard event bus scaled between microservices.
  - Cloud Bus provides event bus implementation for microservices.
  - With help of Cloud Bus microservices can publish events and subscribe to them forming a single system composed of many services.

Cloud Bus

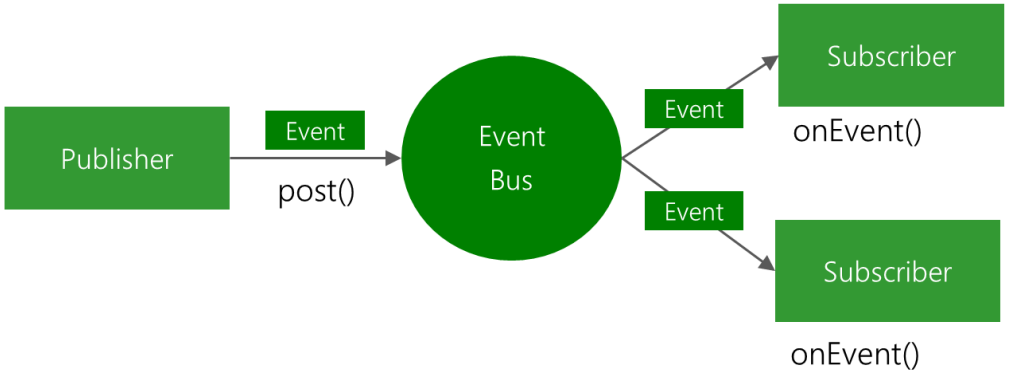
- Cloud Bus is an event bus for linking services and service instances together with distributed messaging.
  - Useful for propagating state changes across the cluster (e.g. config change events).



Spring Cloud Bus

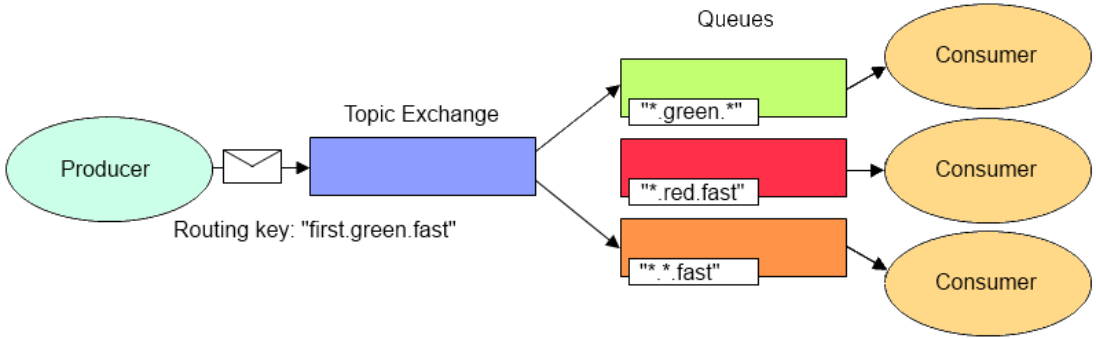
Spring Cloud Bus

- Spring Cloud Bus links nodes of a distributed system using a lightweight message broker.
  - This can be used to broadcast state changes, application events or management instructions.



Spring Cloud Bus

- The Spring Cloud Bus adds a management backplane to your application instances.
  - It is currently implemented as a client-side binding to AMQP exchanges and queues, and is designed to be pluggable.

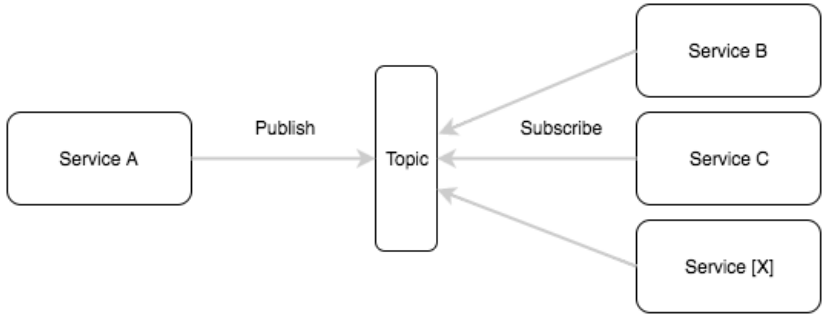


Spring Cloud Bus

- Any Spring Cloud application can have the Spring Cloud Bus plugged-in.
  - Whenever the Spring Cloud Bus is activated, the Spring Cloud application will be immediately connected to all other application.
- Spring Cloud Bus connected applications are servers and clients at the same time.
  - Any application can generate events.
  - Any application can listen for events.

Spring Cloud Bus

- Spring Cloud Bus works automatically out of the box.
  - All you need to do to enable the bus is to add one of available dependencies and Spring Cloud takes care of the rest.
- Available dependencies are:
  - spring-cloud-starter-bus-amqp
  - spring-cloud-starter-bus-kafka



Spring Cloud Bus

- Make sure the broker (RabbitMQ or Kafka) is available and configured.
  - Running on localhost you shouldn't have to do anything, but if you are running remotely, define the broker credentials.
  - E.g. for Rabbit:

```
spring:
  rabbitmq:
    host: rabbit.broker.com
    port: 5672
    username: user
    password: secret
```

Spring Cloud Bus

- With spring cloud bus you may send messages
  - To all nodes listening at the exact moment.
  - To all nodes for a particular service.
  - To exact node for a particular service.
- This is done using Application Context ID which is similar to what is defined by Eureka.

Spring Cloud Bus

- By default, Spring Boot sets the Application Context ID for you as a combination of:
  - spring.application.name
  - server.port
  - active profiles
- The String representation may look like
  - `ServiceName:8080:profileName`

Management Endpoints

Management Endpoints

- The Cloud Bus adds additional management endpoints to your application.
  - These are delivered with Spring Boot Actuator and will be available under management context.
  - By default, management context is available within application context, however, if you use spring security, it makes sense to separate them.

Management Endpoints

- There many settings that can be applied to management context, however they are out of scope of Spring Cloud training, as they are part of Spring Boot itself.
  - For Spring Cloud training we need to set a different management port as application context is secured with OAuth2.

```
management.port: 9000
management.security.enabled: false
```

  - Management context security is also disabled for sake of simplicity of testing.

Management Endpoints

- The /bus/* endpoints are available under actuator namespace.
- There are currently two implemented:
  - POST /actuaror/bus-env
    - Sends key/value pairs to update each node's Spring Environment.
    - Accepts "destination" parameter.
  - POST /actuator/bus-refresh
    - Reloads application configuration, just as if it has been pinged on /refresh endpoint.
    - Accepts "destination" parameter.

Management Endpoints

- The HTTP endpoints accept a "destination" parameter where the destination is an ApplicationContext ID.
  - The "destination" parameter is used in a Spring PathMatcher, with the path separator as a colon ":“, to determine if an instance will process the message.
    - POST /actuator/bus-refresh?destination=ServiceName:8080
    - If no destination is provided, all application will be considered.
  - If the ID is owned by an instance on the Bus, then it will process the message (all other instances will ignore it).

Event Broadcasting

Event Broadcasting

- Spring Cloud Bus can carry any event of RemoteApplicationEvent type.
- The default transport is JSON
  - When an event is sent to Cloud Bus, it is serialized to JSON
  - When an event is received from Cloud Bus, it is deserialized from JSON
- The deserializer needs to know which types are going to be used ahead of time.
  - Event classes should be registered at both, publisher and listener sides.

Event Broadcasting

- Event Example

```
package com.luxoft.training.spring.cloud;

import org.springframework.cloud.bus.event.RemoteApplicationEvent;

public class MyCustomEvent extends RemoteApplicationEvent {
    public MyCustomEvent() {
        super();
    }

    public MyCustomEvent(String originService, String destinationService) {
        super(new Object(), originService, destinationService);
    }
}
```

Event Broadcasting

- Event classes can be registered with the deserializer by adding annotation @RemoteApplicationEventScan to the configuration class.
  - By default, the package of the class where @RemoteApplicationEventScan is used will be scanned for RemoteApplicationEvent subclasses.
  - You can also explicitly specify the packages to scan using the value, basePackages or basePackageClasses properties of event scan annotation.

Event Tracing

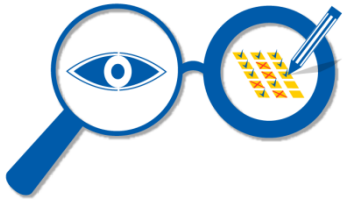
Event Tracing

- Bus events, subclasses of RemoteApplicationEvent, can be traced by setting:
  - spring.cloud.bus.trace.enabled: true
- The /trace endpoint will show each event sent and all the acks from each service instance.
  - Any Bus application can trace acks, but sometimes it will be useful to do this in a central service that can do more complex queries on the data, or forward it to a specialized tracing service.

Push Configurations

Push Configurations

- When a centralized configuration repository is used, with a special monitor, Spring Cloud Bus is capable to watch for configuration changes and push notifications to services signaling them to reload the bootstrap context.
  - This is achieved by using the actuator refresh feature.



Push Configurations

- Many source code repository providers like Github, Gitlab or Bitbucket will notify you of changes in a repository through a web-hook.
  - You can configure the web-hook via the provider's user interface as a URL and a set of events in which you are interested.
  - For instance, Github will POST to the web-hook with a JSON body containing a list of commits, and a header "X-Github-Event" equal to "push".

Push Configurations

- Spring Cloud Config project has a special monitor that watches for configuration changes and pushes notifications to Spring Cloud Bus.
- If you add a dependency on the spring-cloud-config-monitor library and activate the Spring Cloud Bus in your Config Server:
  - The config server will start to watch for configuration changes and will notify other application by Spring Cloud Bus.
  - Additionally a /monitor endpoint is enabled.

Push Configurations

- When the webhook is activated the Config Server will send a RefreshRemoteApplicationEvent targeted at the applications it thinks might have changed.
  - The change detection strategy looks for changes in files that match the application name
  - E.g. “SomeService.yml" is targeted at the “SomeService" application, and "application.yml" is targeted at all applications.

Push Configurations

- The default configuration works out of the box with Github, Gitlab or Bitbucket.
- In addition to the JSON notifications from Github, Gitlab or Bitbucket you can trigger a change notification by POSTing to /monitor with a JSON body {"path" : "file_name"}.
  - This will broadcast to applications matching the "file_name" pattern
  - File name can contain wildcards and the list of file names.
    - For instance, {"path": “*Service"} or {"path": ["Card*","Hist*"]}

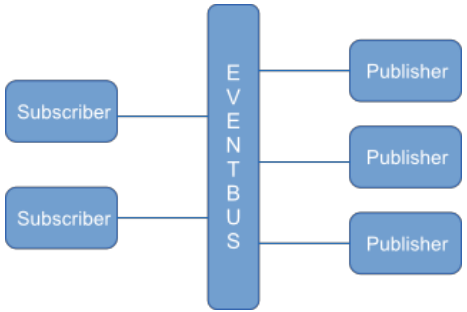
Push Configurations

- The RefreshRemoteApplicationEvent will only be transmitted if the spring-cloud-bus is activated in the Config Server and all client applications.
- The default configuration also detects file system changes in local git repositories.
  - the web-hook is not used in this case, but as soon as you edit a config file a refresh will be broadcast.

Lab 10
Cloud Bus

Lab 10 – Cloud Bus

- In this lab we will:
  - Add push notifications for configuration changes, so that our services will automatically reload the bootstrap context with no server restart.
  - Introduce a new History Service that will keep history of clients' fund and withdraw operations.



Thank You!

LXFT
LISTED
NYSE

LUXOFT