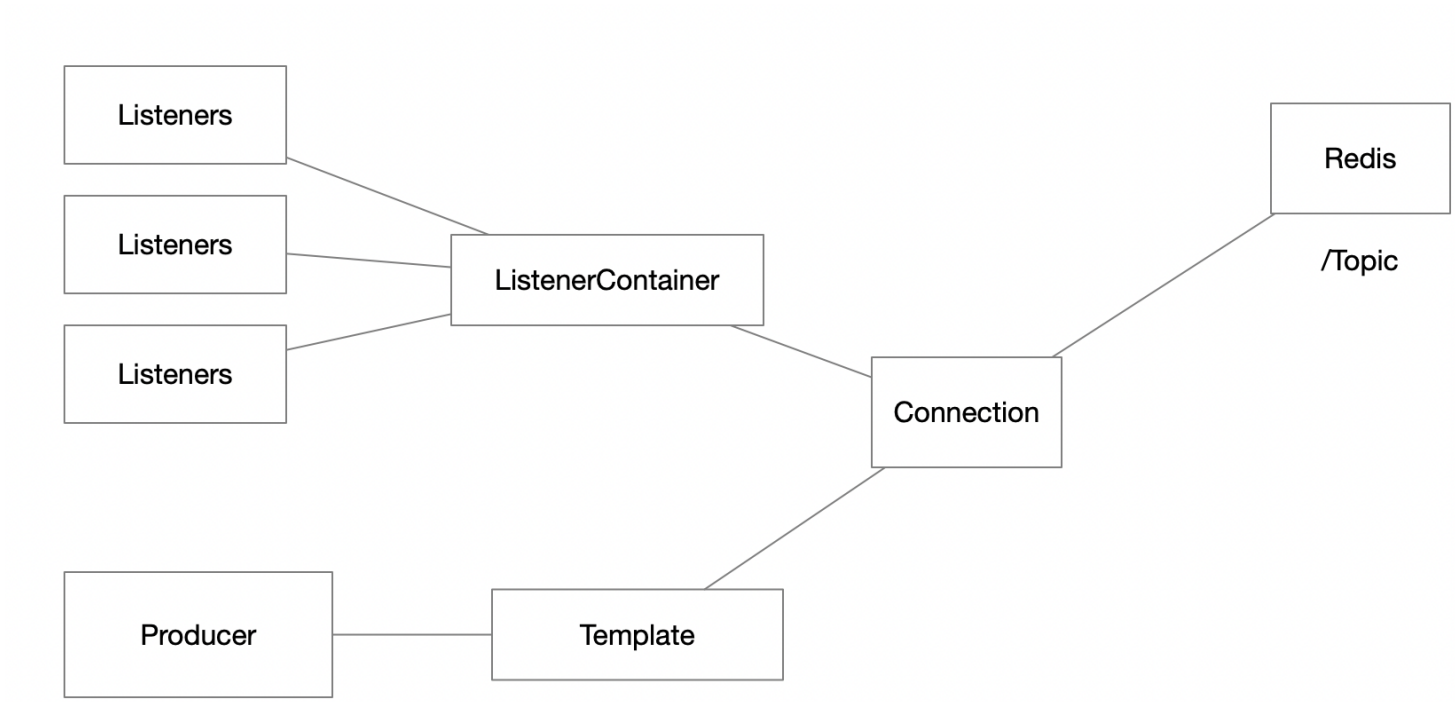
Tutorial REDIS Publish Subscribe

La idea de este tutorial es usar REDIS como una message broker que implementa el servicio de publish-subscribe. Concretamente vamos a implementar la siguiente arquitectura. En esta arquitectura un productor envía varios mensajes que se replican a los escuchadores ("listeners") que están escuchando en el tópico especificado. PAra crear la aplicación usaremos el framework de aplicaciones empresariales Spring.



Parte 1. Prepare una aplicación MAVEN.

Utilice el siguiente comando para crear una aplicación MAVEN desde la línea de comandos.

```
mvn archetype:generate -DgroupId=co.edu.escuelaing.arsw -DartifactId=Publish-Subscribe-Redis -
DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

Parte 2. Configure el POM.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>co.edu.escuelaing</groupId>
    <artifactId>RedisPubSubPrimer</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.8</maven.compiler.source>
```

```
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-redis</artifactId>
            <version>2.3.1.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>javax.inject</groupId>
            <artifactId>javax.inject</artifactId>
            <version>1</version>
            <type>jar</type>
        </dependency>
    </dependencies>
</project>
```

Parte 3. Cree la aplicación.

1. Primero cree una clase principal para cargar Spring y arrancar la aplicación.

```
package co.edu.escuelaing.redispubsubprimer;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class PSRedisPrimerAppStarter {

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

2. Ahora cree una clase para configurar los parámetros de conexión a REDIS y para crear una fábrica de conexiones que genere conexiones a REDIS.

```
package co.edu.escuelaing.redispubsubprimer.connection;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.data.redis.connection.RedisStandaloneConfiguration;
import org.springframework.data.redis.connection.lettuce.LettuceConnectionFactory;

@Configuration
@PropertySource("application.properties")
public class PSRedisConnectionConfiguration {
```

```
    @Value("${redis.broker.hostname}")
    private String redisHostName;

    @Value("${redis.broker.port}")
    private int redisPort;

    @Bean
    public LettuceConnectionFactory redisConnectionFactory() {
        LettuceConnectionFactory connectionFactory
                = new LettuceConnectionFactory(new RedisStandaloneConfiguration(redisHostName,
redisPort));
        return connectionFactory;
    }

}
```

Asegúrese que en el directorio *src/main/resources* está el archivo de configuraciones "application.properties", este archivo lo lee directamente spring para cargar valores de propiedades que usted quiera pasar a la aplicación. El archivo se debe ver de la siguiente manera:

```
#----------------------------
#CONFIGURE Broker CACHE USING REDIS.

#Configure the BB Cache with Redis.
redis.broker.hostname=localhost

#This is the default port of a redis instance.
#redis.broker.port=6379

#You can use other port if your redis instance is published in other port.
redis.broker.port=45000
```

3. Defina un tipo de componente que sirve de intermediario de conexión a los diferentes listeners. De la página del API de Spring:

"Contenedor que proporciona un comportamiento asincrónico para los escuchas de mensajes de Redis. Maneja los detalles de bajo nivel de escucha, conversión y envío de mensajes.

A diferencia del Redis de bajo nivel (una conexión por suscripción), el contenedor usa solo una conexión que está 'multiplexada' para todos los oyentes registrados, y el envío de mensajes se realiza a través del ejecutor de tareas.

Tenga en cuenta que el contenedor usa la conexión de manera diferida (la conexión se usa solo si al menos un oyente está configurado).

Agregar y eliminar oyentes al mismo tiempo tiene resultados indefinidos. Se recomienda encarecidamente sincronizar / ordenar estos métodos en consecuencia."

```
package co.edu.escuelaing.redispubsubprimer.connection;

import javax.inject.Inject;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.listener.PatternTopic;
import org.springframework.data.redis.listener.RedisMessageListenerContainer;
import org.springframework.data.redis.listener.adapter.MessageListenerAdapter;
import org.springframework.stereotype.Component;

@Component
public class PSRedisListenerContainer extends RedisMessageListenerContainer{

        @Inject
        PSRedisListenerContainer(RedisConnectionFactory connectionFactory) {
                this.setConnectionFactory(connectionFactory);
        }
}
```

4. Defina una clase template de Redis que permita recibir mensajes de tipo String. Es una generalización del template de Redis que simplifica la codificación de acceso a REDIS. Del API de Spring 2.5.2:

"`public class RedisTemplate<K,V>`
`extends RedisAccessor`
`implements RedisOperations<K,V>, BeanClassLoaderAware`
Helper class that simplifies Redis data access code.
Performs automatic serialization/deserialization between the given objects and the underlying binary data in the Redis store. By default, it uses Java serialization for its objects (through JdkSerializationRedisSerializer). For String intensive operations consider the dedicated StringRedisTemplate.
The central method is execute, supporting Redis access code implementing the RedisCallback interface. It provides RedisConnection handling such that neither the RedisCallback implementation nor the calling code needs to explicitly care about retrieving/closing Redis connections, or handling Connection lifecycle exceptions. For typical single step actions, there are various convenience methods.
Once configured, this class is thread-safe.
Note that while the template is generified, it is up to the serializers/deserializers to properly convert the given Objects to and from binary data.
This is the central class in Redis support."

```
package co.edu.escuelaing.redispubsubprimer.connection;

import javax.inject.Inject;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.core.StringRedisTemplate;
import org.springframework.stereotype.Component;

@Component
public class PSRedisTemplate extends StringRedisTemplate{
```

```
        @Inject
        public PSRedisTemplate(RedisConnectionFactory connectionFactory) {
                super(connectionFactory);
        }
}
```

5. Defina una clase receptor que recibirá los mensajes concretos.

```
package co.edu.escuelaing.redispubsubprimer.receiver;

import java.util.concurrent.atomic.AtomicInteger;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import static org.springframework.beans.factory.config.BeanDefinition.SCOPE_PROTOTYPE;
import org.springframework.context.annotation.Scope;
import org.springframework.data.redis.listener.adapter.MessageListenerAdapter;
import org.springframework.stereotype.Component;

@Component
@Scope(SCOPE_PROTOTYPE)
public class Receiver extends MessageListenerAdapter {

    private static final Logger LOGGER = LoggerFactory.getLogger(Receiver.class);

    private AtomicInteger counter = new AtomicInteger();

    public Receiver() {
        this.setDefaultListenerMethod("receiveMessage");
    }

    public void receiveMessage(String message) {
        LOGGER.info(this.hashCode() + ". Received <" + message + ">");
        counter.incrementAndGet();
    }

    public int getCount() {
        return counter.get();
    }

}
```

6. Cree el productor

```
package co.edu.escuelaing.redispubsubprimer.producer;

import co.edu.escuelaing.redispubsubprimer.connection.PSRedisListenerContainer;
import javax.inject.Inject;
import co.edu.escuelaing.redispubsubprimer.connection.PSRedisTemplate;
import co.edu.escuelaing.redispubsubprimer.receiver.Receiver;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

```
import org.slf4j.LoggerFactory;
import org.springframework.boot.CommandLineRunner;
import org.springframework.context.ApplicationContext;
import org.springframework.data.redis.listener.PatternTopic;
import org.springframework.stereotype.Component;

@Component
public class Producer implements CommandLineRunner  {

    private static final Logger LOGGER = LoggerFactory.getLogger(Producer.class);

    @Inject
    private ApplicationContext appContext;

    @Inject
    PSRedisTemplate template;

    //@Inject
    //Receiver receiver;

    @Inject
    PSRedisListenerContainer container;

    public void run(String... args) throws Exception {


        for( int i = 0; i <7 ; i++){
            Receiver receiver = appContext.getBean(Receiver.class);
            container.addMessageListener(receiver, new PatternTopic("PSChannel"));
        }

        Integer i =0;
        Thread.sleep(500);

        while (i < 6) {
            i= i+ 1;
            LOGGER.info("Sending message... " + i);
            template.convertAndSend("PSChannel", "Hello from Redis! Message " + i);
            Thread.sleep(500);
        }

        System.exit(0);
    }

}
```

Parte 4. Ejecute la aplicación

Para ejecutar la aplicación necesita una instancia de REDIS corriendo. En este tutorial usaremos una instancia ejecutándose en una instancia local de Docker. Para instalar REDIs en docker use el siguiente comando:

```
docker run --name some-redis -p 45000:6379 -d redis
```