**Redis as a Message Broker**

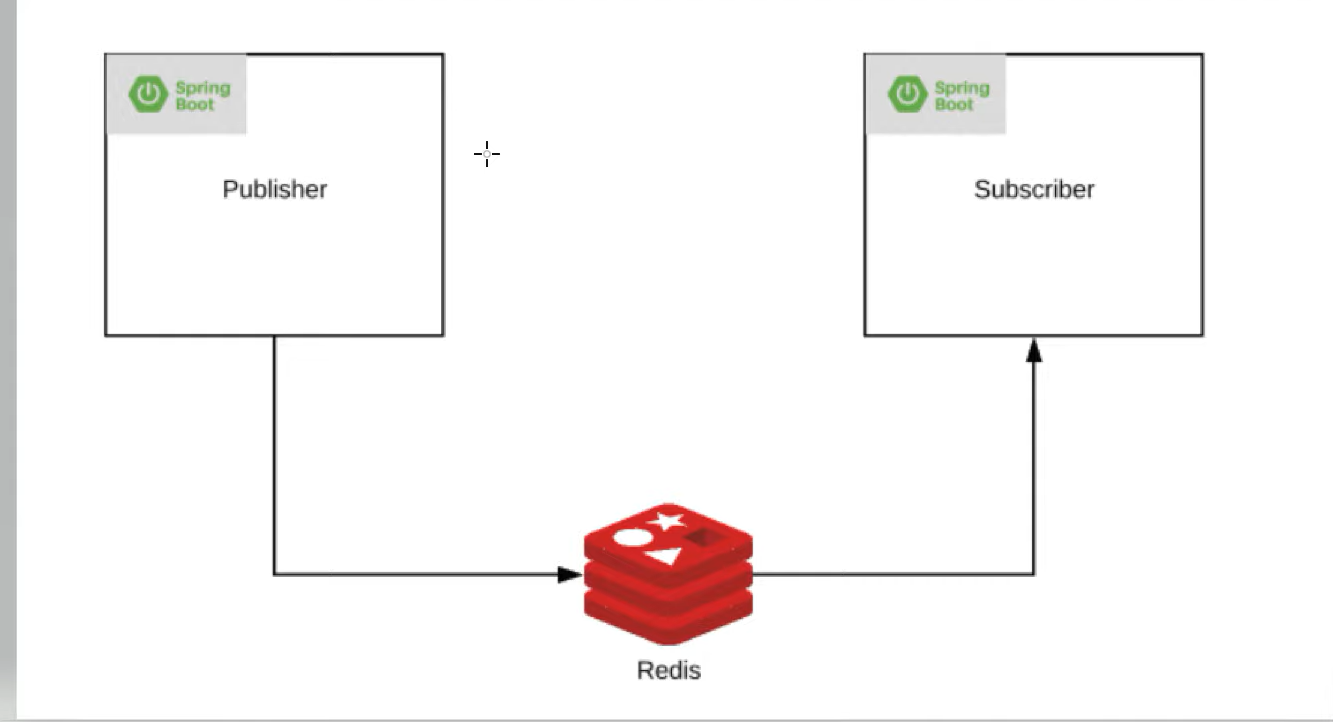
Redis Pub/Sub (Publish/Subscribe) is a messaging paradigm where message senders (publishers) send messages to channels, and message receivers (subscribers) receive messages from channels they have subscribed to. Redis provides native support for Pub/Sub, making it easy to implement real-time communication between components in a system. Here's an explanation of the key concepts and how Redis Pub/Sub works:

**Key Concepts:**

**Publisher**: A component or application that sends messages to a specific channel. Publishers are responsible for creating and broadcasting messages.

**Subscriber**: A component or application that subscribes to specific channels and receives messages published to those channels.

**Channel**: A communication mechanism in Redis where messages are sent. Messages published to a channel are received by all subscribers of that channel.



Demo project was done to illustrate Redis as Message Broker

**Dependencies used:**

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-redis</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

<version>3.3.0</version>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<version>1.18.30</version>

<scope>provided</scope>

</dependency>

**RedisConfig Class**

The connectionfactory() and template() method are the configurations which are done for connecting the Redis to the application. There is a component called publisher which publishes the event or message and there is a Subscriber which consumes the event or message. The channel acts as medium for the message. messageListnerAdapter() method will receive the message from the publisher whereas the redisMessageListenerContainer() method contains the message and is responsible for transmitting the message to subscriber.

*@Bean*

*public JedisConnectionFactory connectionFactory() {*

*RedisStandaloneConfiguration configuration = new RedisStandaloneConfiguration();*

*configuration.setHostName("localhost");*

*configuration.setPort(6379);*

*return new JedisConnectionFactory(configuration);*

*}*

*@Bean*

*public RedisTemplate<String, Object> template() {*

*RedisTemplate<String, Object> template = new RedisTemplate<>();*

*template.setConnectionFactory(connectionFactory());*

*template.setValueSerializer(new GenericToStringSerializer<Object>(Object.class));*

*return template;*

*}*

*@Bean*

*public ChannelTopic topic() {*

*return new ChannelTopic("pubsub:neokred");*

*}*

*@Bean*

*public MessageListenerAdapter messageListnerAdapter() {*

*return new MessageListenerAdapter(new Receiver());*

*}*

*@Bean*

*public RedisMessageListenerContainer redisMessageListenerContainer() {*

*RedisMessageListenerContainer container = new RedisMessageListenerContainer();*

*container.setConnectionFactory(connectionFactory());*

*container.addMessageListener(messageListnerAdapter(), topic());*

*return container;*

*}*

**Publisher class and Receiver class**

Publisher:

@PostMapping("/publish")

*public String publish (@RequestBody Product product)*

*{*

*logger.info ("Message is published by the Publisher");*

*template.convertAndSend(topic.getTopic(), product.toString());*

*return "Event published !!";*

*}*

Receiver:

*@Override*

*public void onMessage(Message message, byte[] bytes) {*

*logger.info ("Consumed event {}",message);*

*}*

The method publish is written by taking the Product object as a RequestBody with “/publish” endpoint. Whenever we hit the API, it will return the string “Event Published” as the output.

The RequestBody which is sent will be transmitted to subscriber in the “pubsub” channel which can be seen in the Redis server. (redis-cli.exe)

The message *"Message is published by the Publisher"* will be displayed as part of the publisher and “*Consumed event {}”* will be displayed as part of the receiver.

