Learn Spring v1.5

Why Spring

Spring makes programming Java quicker, easier, and safer for everybody. Spring’s focus on speed, simplicity, and productivity has made it the world's most popular Java framework.

What can Spring do?

Microservices

Quickly deliver production-grade features with independently evolvable microservices.

Reactive

Spring's asynchronous, nonblocking architecture means you can get more from your computing resources.

Cloud

Your code, any cloud—we've got you covered. Connect and scale your services, whatever your platform.

Web apps

Frameworks for fast, secure, and responsive web applications connected to any data store.

Serverless

The ultimate flexibility. Scale up on demand and scale to zero when there's no demand.

Event Driven

Integrate with your enterprise. React to business events. Act on your streaming data in realtime.

Batch

Automated tasks. Offline processing of data at a time to suit you.

Java Code Convention

Java Code Convention

- Class
  - It should start with the uppercase letter.
  - It should be a noun such as Color, Button, System, Thread, etc.
  - Use appropriate words, instead of acronyms.

```
public class Employee {
    //code snippet
}
```

Java Code Convention

- Interface
  - It should start with the uppercase letter.
  - It should be an adjective such as Runnable, Remote, ActionListener.
  - Use appropriate words, instead of acronyms.

```
interface Printable {
    //code snippet
}
```

Java Code Convention

- Method
  - It should start with lowercase letter.
  - It should be a verb such as main(), print(), println().
  - If the name contains multiple words, start it with a lowercase letter followed by an uppercase letter such as actionPerformed().

```
class Employee {
    // method
    void draw() {
        //code snippet
    }
}
```

Java Code Convention

- Variable
  - It should start with a lowercase letter such as id, name.
  - It should not start with the special characters like & (ampersand), $ (dollar), _ (underscore).
  - If the name contains multiple words, start it with the lowercase letter followed by an uppercase letter such as firstName, lastName.
  - Avoid using one-character variables such as x, y, z.

```
class Employee {
    // variable
    int id;
    //code snippet
}
```

Java Code Convention

- Package
  - It should be a lowercase letter such as java, lang.
  - If the name contains multiple words, it should be separated by dots (.) such as java.util, java.lang.

```
//package
package com.javatpoint;

class Employee {
    //code snippet
}
```

Java Code Convention

- Constant
  - It should be in uppercase letters such as RED, YELLOW.
  - If the name contains multiple words, it should be separated by an underscore(_) such as MAX_PRIORITY.
  - It may contain digits but not as the first letter.

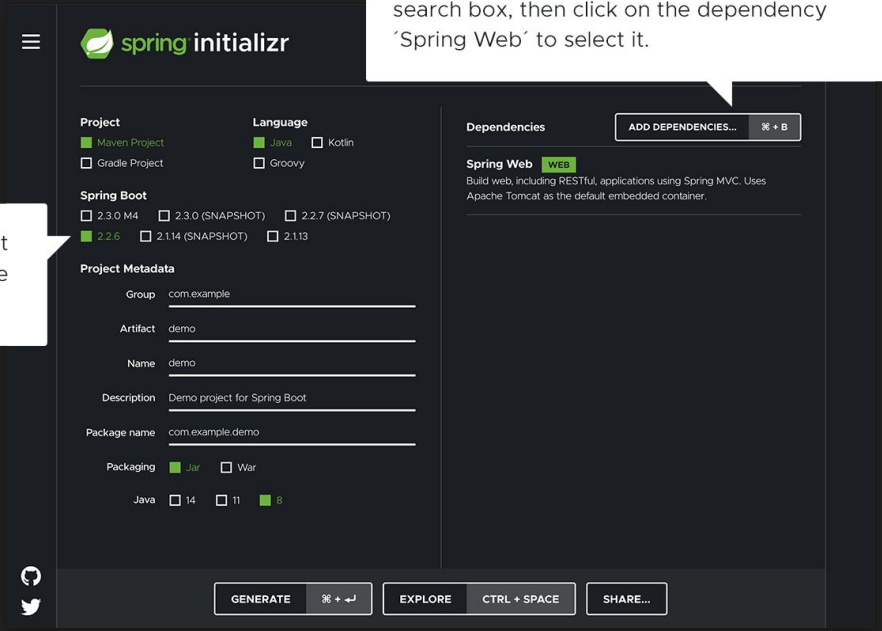
```
class Employee {
    //constant
    static final int MIN_AGE = 18;
    //code snippet
}
```

Spring Quickstart Guide

Step 1: Start a new Spring Boot project

Click on ´Add dependencies´, type `Web` in the search box, then click on the dependency ´Spring Web´ to select it.

The current version of Spring Boot changes regularly. Just choose the latest release (but not snapshot).



https://start.spring.io/

Step 2: Add your code

```
package com.example.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
@RestController
public class DemoApplication {

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

   @GetMapping("/hello")
   public String hello(@RequestParam(value = "name", defaultValue = "World") String name)
   {
      return String.format("Hello %s!", name);
   }
}
```

Step 3: Try it

```
mvnw spring-boot:run
```

Reference

https://spring.io/quickstart

Building a RESTful Web Service

What You Will Build

- You will build a service that will accept HTTP GET requests at http://localhost:8080/greeting.
- It will respond with a JSON representation of a greeting, as the following listing shows:

```
{"id":1,"content":"Hello, World!"}
```

- You can customize the greeting with an optional name parameter in the query string, as the following listing shows:

http://localhost:8080/greeting?name=User

- The name parameter value overrides the default value of World and is reflected in the response, as the following listing shows:

```
{"id":1,"content":"Hello, User!"}
```

Create POJO

```
package com.example.demo;

public class Greeting {

	private final long id;
	private final String content;

	public Greeting(long id, String content) {
		this.id = id;
		this.content = content;
	}

	public long getId() {
		return id;
	}

	public String getContent() {
		return content;
	}
}
```

Create a Resource Controller

```java
package com.example.demo;

import java.util.concurrent.atomic.AtomicLong;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class GreetingController {

	private static final String template = "Hello, %s!";
	private final AtomicLong counter = new AtomicLong();

	@GetMapping("/greeting")
	public Greeting greeting(@RequestParam(value = "name", defaultValue = "World") String name) {
		return new Greeting(counter.incrementAndGet(), String.format(template, name));
	}
}
```

References

https://spring.io/guides/gs/rest-service/

https://spring.io/guides/tutorials/rest/

Exercise I

Create REST web service that can be accessed via this endpoint:

```
POST /submit
```

and accepting request payload:

```
{
  id: 1,
  content: “Hello”
}
```

the response would be the value of `content` field, in this case `Hello`

Exercise II

Create REST web service that can be accessed via this endpoint:

`GET /hello/{name}`

and hit the endpoint using this request `GET /hello/John`

would return response: `Hello John`

Consuming a RESTful Web Service

What You Will Build

You will build an application that uses Spring's RestTemplate to retrieve a random Spring Boot quotation at https://jsonplaceholder.typicode.com/todos/1.

```
{
  "userId": 1,
  "id": 1,
  "title": "delectus aut autem",
  "completed": false
}
```

Create Domain Class

```
package com.example.consumingrest;

public class Todo {

    private int id;
    private int userId;
    private String title;
    private boolean completed;

    public Todo() {
    }

    ...

}
```

Finishing the Application

```
package com.example.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.web.client.RestTemplateBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.web.client.RestTemplate;

@SpringBootApplication
public class ConsumingRestApplication {

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

	@Bean
	public RestTemplate restTemplate(RestTemplateBuilder builder) {
		return builder.build();
	}
}
```

Create a Resource Controller

```
package com.example.demo;

...

@RestController
public class TodoController {

    @Autowired
    RestTemplate restTemplate;

    @GetMapping("/todo")
    public Todo todo() {
        Todo todo = restTemplate.getForObject(
            "https://jsonplaceholder.typicode.com/todos/1", Todo.class);
        return todo;
    }
}
```

Reference

https://spring.io/guides/gs/consuming-rest/

Exercise

Create REST web service:

`GET /post/{id}`

and when it get hit it will fetch this api:

https://jsonplaceholder.typicode.com/posts?id={id}

and return the value of `body` field from post object

Consuming SOAP Web Service

Example of SOAP Web Service

- download SOAP web service example from:

https://gitlab.com/maruidea/soap-web-service/-/archive/main/soap-web-service-main.zip

- extract and run:

```
mvnw spring-boot:run
```

- test wsdl:

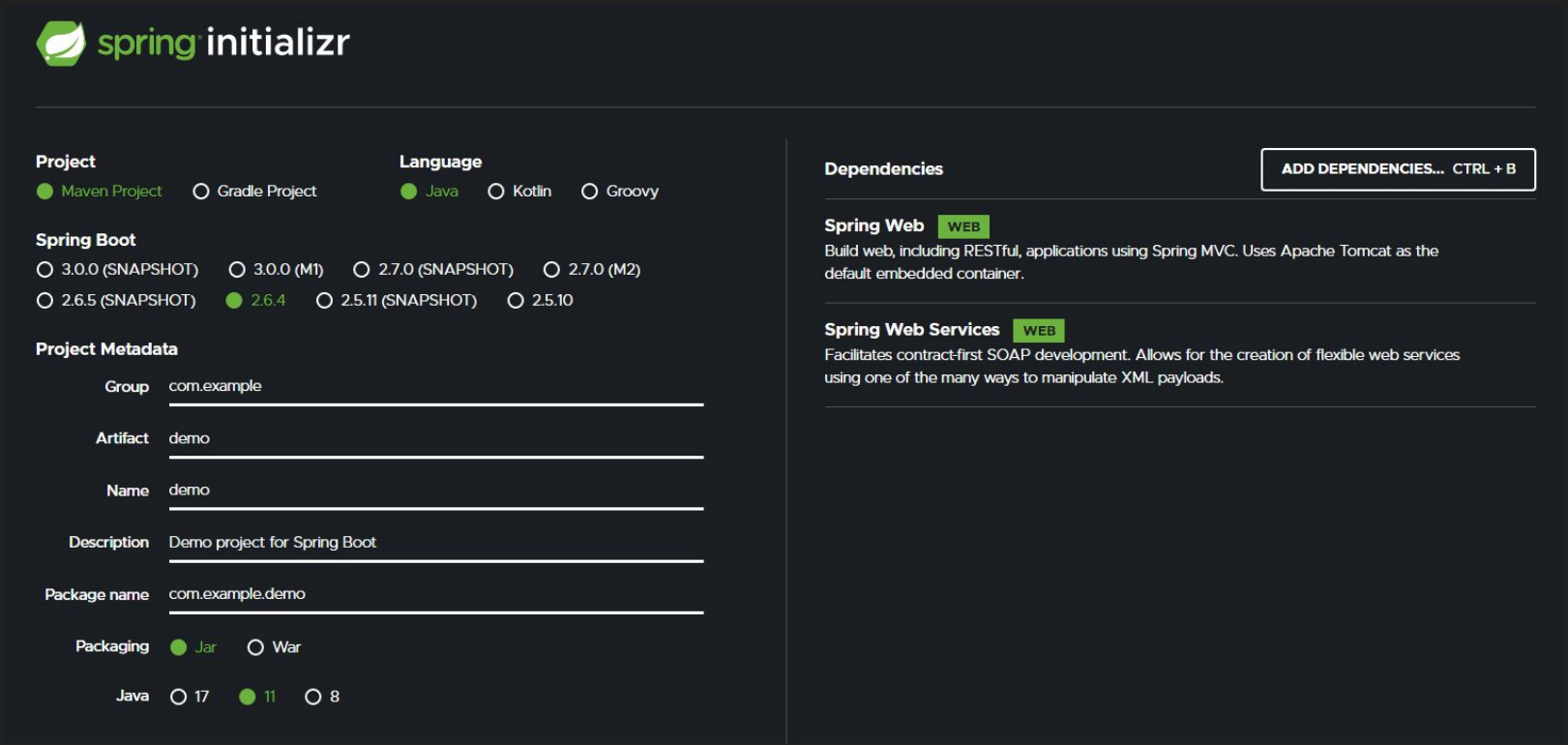
http://localhost:8080/ws/countries.wsdl

Example of SOAP Web Service

- test endpoint:

```
POST http://localhost:8080/ws

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:gs="http://spring.io/guides/gs-producing-web-service">
   <soapenv:Header/>
   <soapenv:Body>
      <gs:getCountryRequest>
         <gs:name>Spain</gs:name>
      </gs:getCountryRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Consuming using Spring Boot



Add maven dependency

```
<dependency>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-web-services</artifactId>
     <exclusions>
          <exclusion>
               <groupId>org.springframework.boot</groupId>
               <artifactId>spring-boot-starter-tomcat</artifactId>
          </exclusion>
     </exclusions>
 </dependency>
```

Add maven profile

```
<profiles>
     <profile>
          <id>java11</id>
          <activation>
               <jdk>[11,)</jdk>
          </activation>

          <dependencies>
               <dependency>
                    <groupId>org.glassfish.jaxb</groupId>
                    <artifactId>jaxb-runtime</artifactId>
               </dependency>
          </dependencies>
     </profile>
 </profiles>
```

Add maven plugin

```
<plugin>
          <groupId>org.jvnet.jaxb2.maven2</groupId>
          <artifactId>maven-jaxb2-plugin</artifactId>
          <version>0.14.0</version>
          <executions>
               <execution>
                    <goals>
                         <goal>generate</goal>
                    </goals>
               </execution>
          </executions>
          <configuration>
               <schemaLanguage>WSDL</schemaLanguage>
               <generatePackage>com.example.demo.wsdl</generatePackage>
               <schemas>
                    <schema>
                         <url>http://localhost:8080/ws/countries.wsdl</url>
                    </schema>
               </schemas>
          </configuration>
 </plugin>
```

Full pom.xml

```
<?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
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.6.4</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.example</groupId>
    <artifactId>demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>demo</name>
    <description>Demo project for Spring Boot</description>
...
```

```
...
    <properties>
        <java.version>11</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web-services</artifactId>
            <exclusions>
                <exclusion>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-starter-tomcat</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
...
```

```
...
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <profiles>
        <profile>
            <id>java11</id>
            <activation>
                <jdk>[11,)</jdk>
            </activation>

            <dependencies>
                <dependency>
                    <groupId>org.glassfish.jaxb</groupId>
                    <artifactId>jaxb-runtime</artifactId>
                </dependency>
            </dependencies>
        </profile>
    </profiles>
...
```

```
...
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
            <plugin>
                    <groupId>org.jvnet.jaxb2.maven2</groupId>
                    <artifactId>maven-jaxb2-plugin</artifactId>
                    <version>0.14.0</version>
                    <executions>
                        <execution>
                            <goals>
                                <goal>generate</goal>
                            </goals>
                        </execution>
                    </executions>
...
```

```
...
                    <configuration>
                        <schemaLanguage>WSDL</schemaLanguage>
                        <generatePackage>com.example.demo.wsdl</generatePackage>
                        <schemas>
                            <schema>
                                <url>http://localhost:8080/ws/countries.wsdl</url>
                            </schema>
                        </schemas>
                    </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

Generate sources

Run `mvnw compile` and then look in target/generated-sources

Create a Country Service Client

```
package com.example.demo;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import org.springframework.ws.client.core.support.WebServiceGatewaySupport;
import org.springframework.ws.soap.client.core.SoapActionCallback;

import com.example.consumingwebservice.wsdl.GetCountryRequest;
import com.example.consumingwebservice.wsdl.GetCountryResponse;

public class CountryClient extends WebServiceGatewaySupport {

  private static final Logger log = LoggerFactory.getLogger(CountryClient.class);

  public GetCountryResponse getCountry(String country) {

    GetCountryRequest request = new GetCountryRequest();
    request.setName(country);

    log.info("Requesting location for " + country);
  ...
```

Create a Country Service Client

```
...

    GetCountryResponse response = (GetCountryResponse) getWebServiceTemplate()
        .marshalSendAndReceive("http://localhost:8080/ws", request,
            new SoapActionCallback(
                "http://spring.io/guides/gs-producing-web-service/"));

    return response;
  }

}
```

Configuring Web Service Components

```
package com.example.consumingwebservice;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.oxm.jaxb.Jaxb2Marshaller;

@Configuration
public class CountryConfiguration {

  @Bean
  public Jaxb2Marshaller marshaller() {
    Jaxb2Marshaller marshaller = new Jaxb2Marshaller();
    // this package must match the package in the <generatePackage> specified in
    // pom.xml
    marshaller.setContextPath("com.example.demo.wsdl");
    return marshaller;
  }

...
```

Configuring Web Service Components

```
...
  @Bean
  public CountryClient countryClient(Jaxb2Marshaller marshaller) {
    CountryClient client = new CountryClient();
    client.setDefaultUri("http://localhost:8080/ws");
    client.setMarshaller(marshaller);
    client.setUnmarshaller(marshaller);
    return client;
  }

}
```

Test Controller

```
package com.example.demo.controller;

...

@RestController
public class TestController {
    @Autowired
    CountryClient countryClient;

    @GetMapping("/test")
    private Currency test() {
        GetCountryResponse response = countryClient.getCountry("Spain");
        return response.getCountry().getCurrency();
    }
}
```

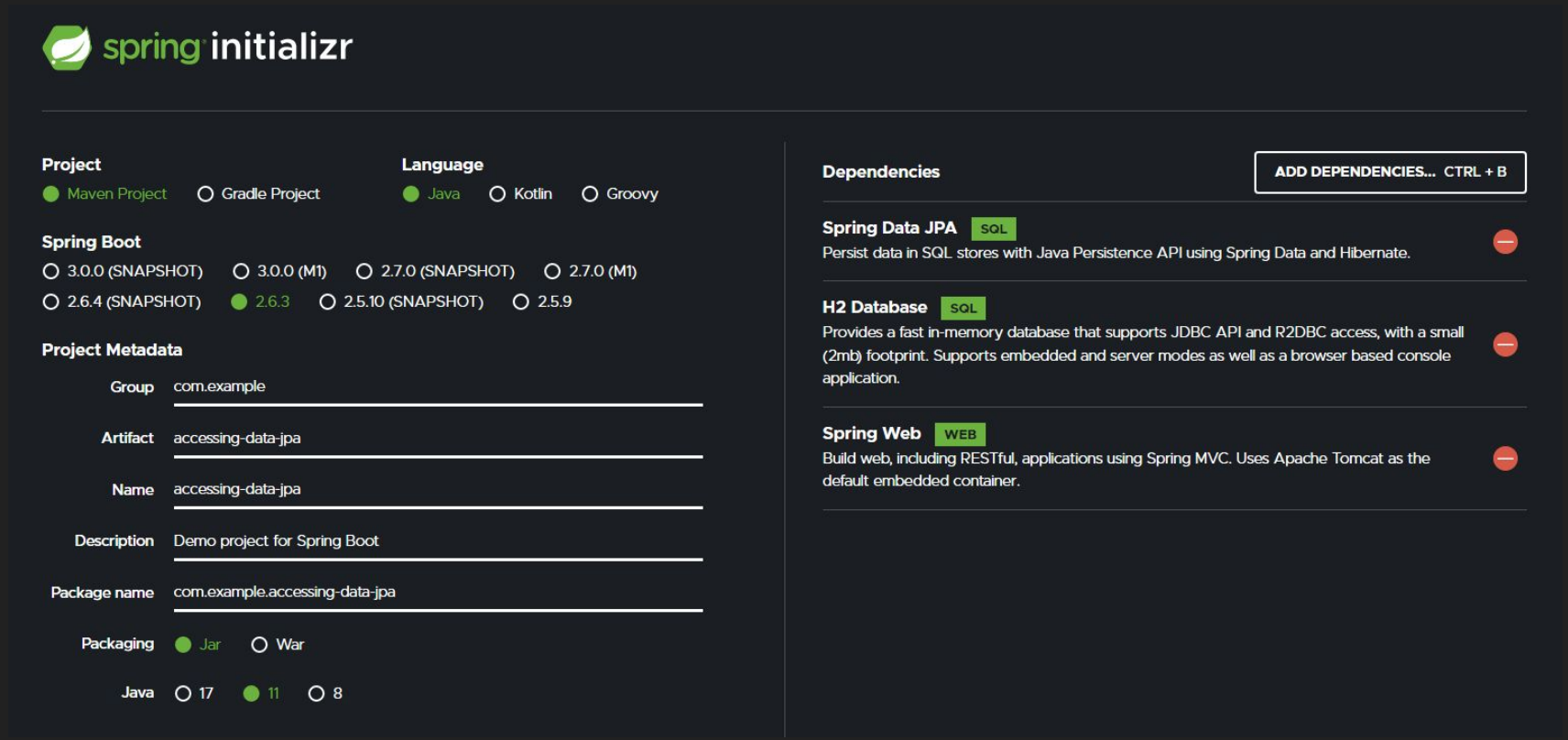
References

https://spring.io/guides/gs/consuming-web-service/

https://spring.io/guides/gs/producing-web-service/

Accessing Data with JPA

Init Spring Boot Project



Define Entity

```
package com.example.accessingdatajpa;

...

@Entity
public class Customer {

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Long id;
    private String firstName;
    private String lastName;

    protected Customer() {}

    public Customer(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }
}
```

Define Repository

```
package com.example.accessingdatajpa;

import java.util.List;
import org.springframework.data.jpa.repository.JpaRepository;

public interface CustomerRepository extends JpaRepository<Customer, Long> {

    List<Customer> findByLastName(String lastName);

    Customer findById(long id);
}
```

Define Controller

```
package com.example.accessingdatajpa;
...

@RestController
public class CustomerController {

    @Autowired
    CustomerRepository repository;

    @GetMapping("/init")
    public void init() {
        repository.save(new Customer("Jack", "Bauer"));
        repository.save(new Customer("Chloe", "O'Brian"));
        repository.save(new Customer("Kim", "Bauer"));
        repository.save(new Customer("David", "Palmer"));
        repository.save(new Customer("Michelle", "Dessler"));
    }
}
```

Define Controller

```
...

    @GetMapping("/all")
    public List<Customer> all() {
        return repository.findAll();
    }
    @GetMapping("/id")
    public Customer id(long id) {
        return repository.findById(id);
    }
    @GetMapping("/lastname")
    public List<Customer> lastName(String lastName) {
        return repository.findByLastName(lastName);
    }

...
```

Reference

https://spring.io/guides/gs/accessing-data-jpa/

https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods.query-creation

Exercise

Create JPA Repository for entity `Product (id, name, price)`

Create REST web service:

`GET /search/{name}` to search product by name and name like {name}

`GET /product/[asc|desc]` to list all products order by name asc or desc

`GET /product/cheaper/{price}` to list products cheaper than {price}

Properties With Spring Boot

Default Property File

Boot applies its typical convention over configuration approach to property files.

This means that we can simply put an `application.properties` file in our `src/main/resources` directory, and it will be auto-detected.

application.properties

```
myconfig.foo=this is foo
myconfig.bar=this is bar
```

Configuration POJO

```
package com.example.accessdatajpa.config;

...

@Configuration
@ConfigurationProperties(prefix = "myconfig")
public class MyConfig {
    private String foo;
    private String bar;

    // standard getters and setters
}
```

Test Configuration Controller

```java
package com.example.demo.controller;
...
@RestController
public class TestConfigController {
    @Autowired
    private MyConfig myConfig;

    @Value("${myconfig.foo}")
    private String foo;

    @Value("${myconfig.bar}")
    private String bar;

    @GetMapping("/testconfig")
    private String testConfig() {
        return foo + " " + bar;
    }

    @GetMapping("/testconfig2")
    private String testConfig2() {
        return myConfig.getFoo() + " " + myConfig.getBar();
    }
}
```

Reference

https://www.baeldung.com/properties-with-spring

https://www.baeldung.com/configuration-properties-in-spring-boot

Spring Bean

Definition

- The bean is an important concept of the Spring Framework and as such, if you want to use Spring, it is fundamental to understand what it is and how it works.
- In Spring, the objects that form the backbone of your application and that are managed by the Spring IoC container are called beans. A bean is an object that is instantiated, assembled, and otherwise managed by a Spring IoC container.
- Inversion of Control (IoC) is a process in which an object defines its dependencies without creating them. This object delegates the job of constructing such dependencies to an IoC container.

TodoController from previous demo

```
package com.example.demo;

...

@RestController
public class TodoController {

    @Autowired
    RestTemplate restTemplate;

    @GetMapping("/todo")
    public Todo todo() {
        Todo todo = restTemplate.getForObject(
            "https://jsonplaceholder.typicode.com/todos/1", Todo.class);
        return todo;
    }
}
```

Without IoC

```java
@RestController
public class TodoController {

    RestTemplate restTemplate;

    public TodoController() {
        restTemplate = new RestTemplateBuilder().build();
    }

    @GetMapping("/todo")
    public Todo todo() {
        Todo todo = restTemplate.getForObject(
            "https://jsonplaceholder.typicode.com/todos/1", Todo.class);
        return todo;
    }
}
```

What if there is a need to change settings?

```
restTemplate = new RestTemplateBuilder()
            .setReadTimeout(Duration.ofSeconds(20000l))
            .setConnectTimeout(Duration.ofSeconds(60000l)).build();
```

With IoC change happens in one place

```
    @Bean
    public RestTemplate restTemplate(RestTemplateBuilder builder) {
        return builder
            .setReadTimeout(Duration.ofSeconds(20000l))
            .setConnectTimeout(Duration.ofSeconds(60000l)).build();
    }
```

Common Annotation related to beans

1. `@Component` : class-level annotation which by default denotes a bean with the same name as the class name with a lowercase first letter. It can be specified a different name using the value argument of the annotation
2. `@Repository` : class-level annotation representing the DAO layer of an application; it enables an automatic persistence exception translation.
3. `@Service` : class-level annotation representing where the business logic usually resides and can flavor the reuse of the code.
4. `@Controller` : class-level annotation representing the controllers in Spring MVC (Model-View-Controller). See also @RestController for the REST “mode”.
5. `@Configuration` : class-level annotation to say that it can contain bean definition methods annotated with @Bean

Reference

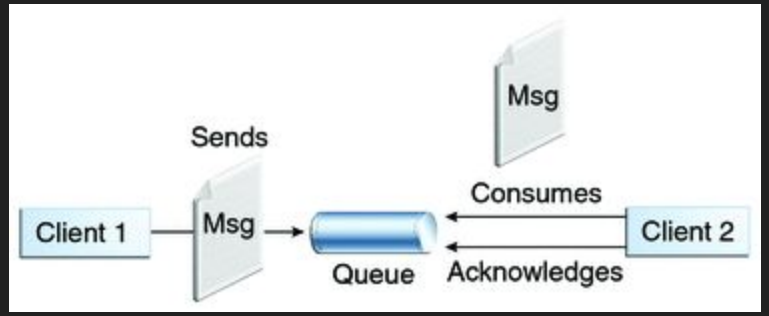
https://medium.com/javarevisited/spring-beans-in-depth-a6d8b31db8a1

https://www.baeldung.com/spring-bean

https://www.baeldung.com/spring-autowire

Messaging with JMS

Basic Concepts



Install ActiveMQ Server

https://activemq.apache.org/components/classic/download/

Start ActiveMQ Server

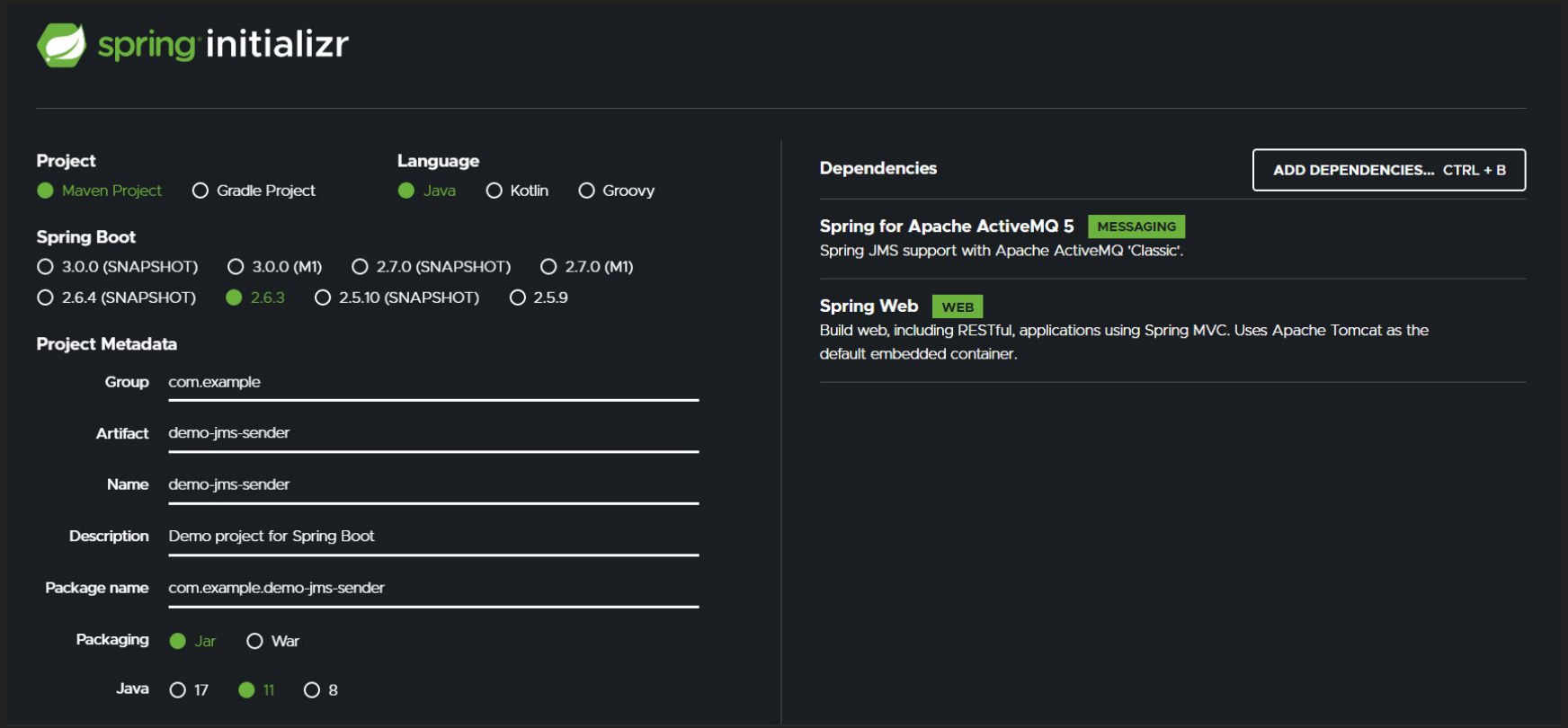
```
$ activemq start
```

ActiveMQ Console

http://127.0.0.1:8161/

admin:admin

Create Producer App



AMQ Config

```
...
@Component
@EnableJms
public class JmsConfig {

    @Bean
    public ActiveMQConnectionFactory activeMQConnectionFactory() {
        ActiveMQConnectionFactory activeMQConnectionFactory = new ActiveMQConnectionFactory();
        activeMQConnectionFactory.setBrokerURL("tcp://localhost:61616");
        activeMQConnectionFactory.setUserName("admin");
        activeMQConnectionFactory.setPassword("admin");

        return activeMQConnectionFactory;
    }
...
```

AMQ Config

```
...
    @Bean // Serialize message content to json using TextMessage
    public MessageConverter jacksonJmsMessageConverter() {
        MappingJackson2MessageConverter converter = new MappingJackson2MessageConverter();
        converter.setTargetType(MessageType.TEXT);
        converter.setTypeIdPropertyName("_type");
        return converter;
    }

}
```

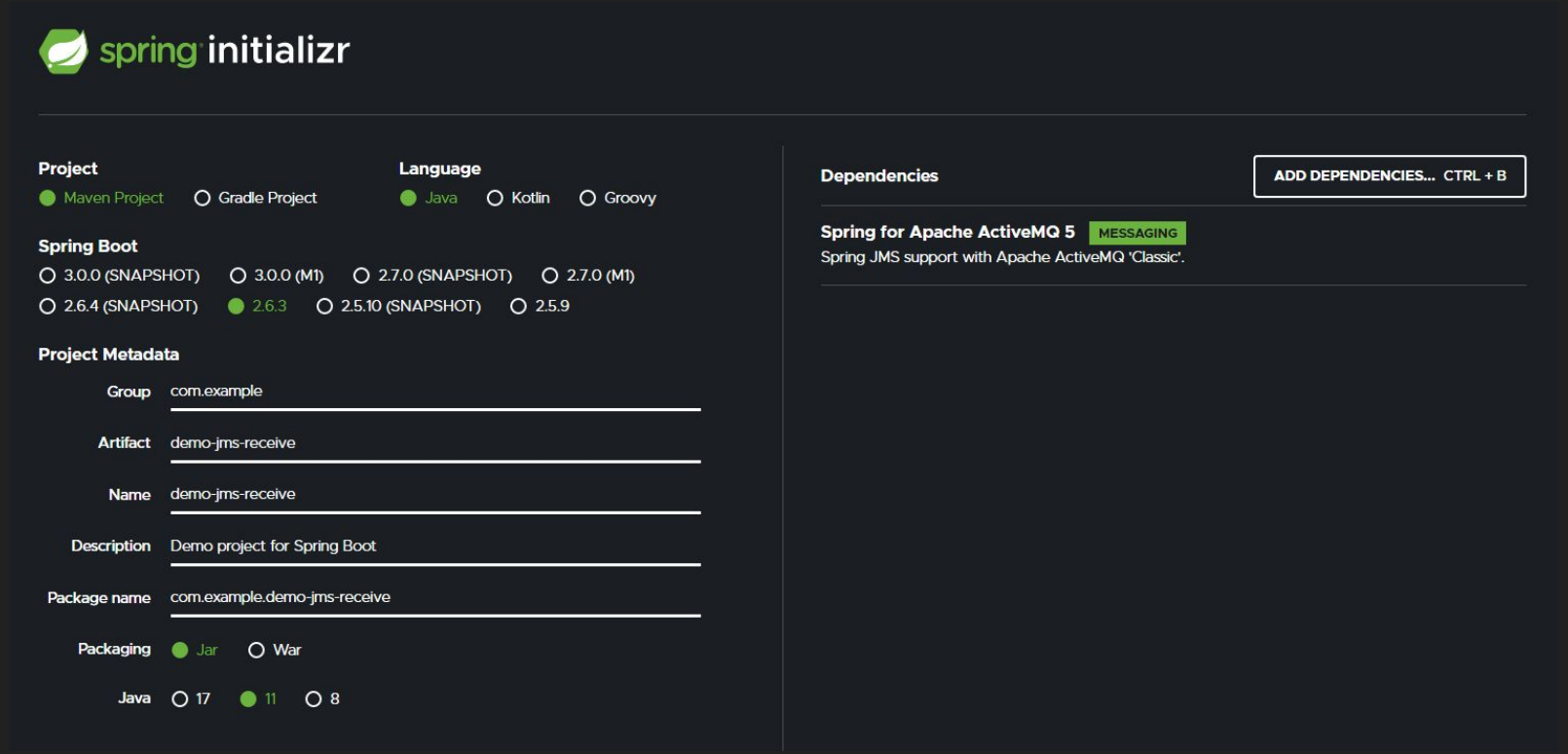
Message POJO

```
public class NotificationMessage implements Serializable {
    private String message;

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
}
```

Test Controller

```
...

@RestController
public class TestMessageController {

    @Autowired
    JmsTemplate jmsTemplate;

    @GetMapping("/send")
    private void send() {
        NotificationMessage notif = new NotificationMessage();
        notif.setMessage("Ada pesan baru");

        jmsTemplate.convertAndSend("nama-queue", notif);
    }
}
```

Create Consumer App



AMQ Config

```
...
@Component
@EnableJms
public class JmsConfig {

    @Bean
    public ActiveMQConnectionFactory activeMQConnectionFactory() {
        ActiveMQConnectionFactory activeMQConnectionFactory = new ActiveMQConnectionFactory();
        activeMQConnectionFactory.setBrokerURL("tcp://localhost:61616");
        activeMQConnectionFactory.setUserName("admin");
        activeMQConnectionFactory.setPassword("admin");

        return activeMQConnectionFactory;
    }

...
```

AMQ Config

```
...
    @Bean
    public JmsListenerContainerFactory<?> myFactory(ConnectionFactory connectionFactory,
                                DefaultJmsListenerContainerFactoryConfigurer configurer) {
      DefaultJmsListenerContainerFactory factory = new DefaultJmsListenerContainerFactory();
      // This provides all boot's default to this factory, including the message converter
      configurer.configure(factory, connectionFactory);
      // You could still override some of Boot's default if necessary.
      return factory;
    }

    @Bean // Serialize message content to json using TextMessage
    public MessageConverter jacksonJmsMessageConverter() {
        MappingJackson2MessageConverter converter = new MappingJackson2MessageConverter();
        converter.setTargetType(MessageType.TEXT);
        converter.setTypeIdPropertyName("_type");
        return converter;
    }
}
```

Message POJO

```
public class NotificationMessage implements Serializable {
    private String message;

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
}
```

Receiver

```
@Component
public class Receiver {

  @JmsListener(destination = "nama-queue", containerFactory = "myFactory")
  public void receiveMessage(NotificationMessage notification) {
    System.out.println("Received <" + notification.getMessage() + ">");
  }

}
```

application.properties

```
server.port=8081
```

Reference

https://spring.io/guides/gs/messaging-jms/

Exercise

Create producer app and consumer app where producer app has endpoint:

```
GET /ping
```

When it get hit it will send message:

```
{
  “message”: “ping”
}
```

to queue named “ping-request”.

Exercise

Consumer app will consume the message and send back

```
{
  “message”: “pong”
}
```

to queue named “pong-request”.

Producer app then consume the message and print the message “pong” to console.

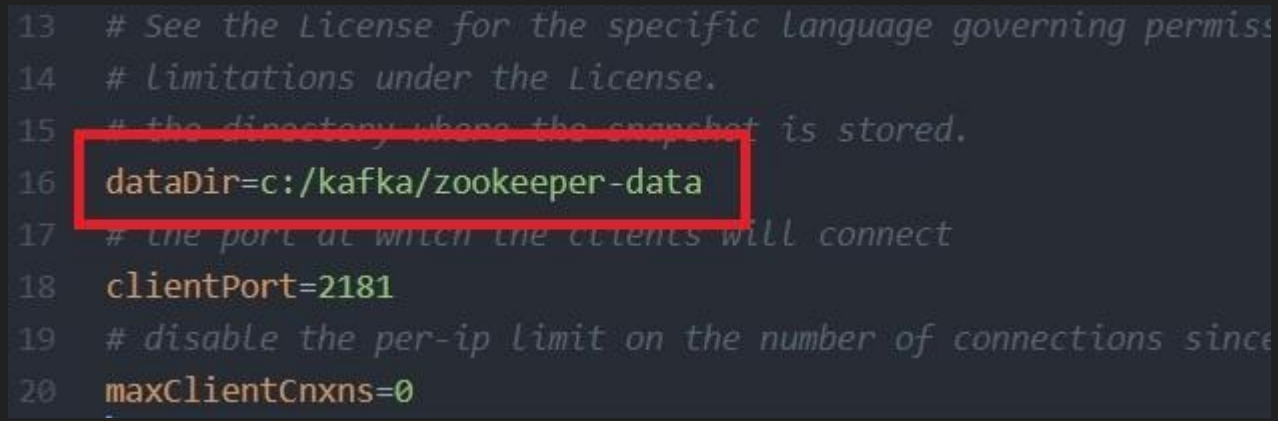
Messaging with Kafka

Install Kafka

https://kafka.apache.org/downloads

Config Zookeeper

- Copy the path of the Kafka folder
- Go to config inside kafka folder and open `zookeeper.properties` file
- Paste the path against the field dataDir and add /zookeeper-data to the path



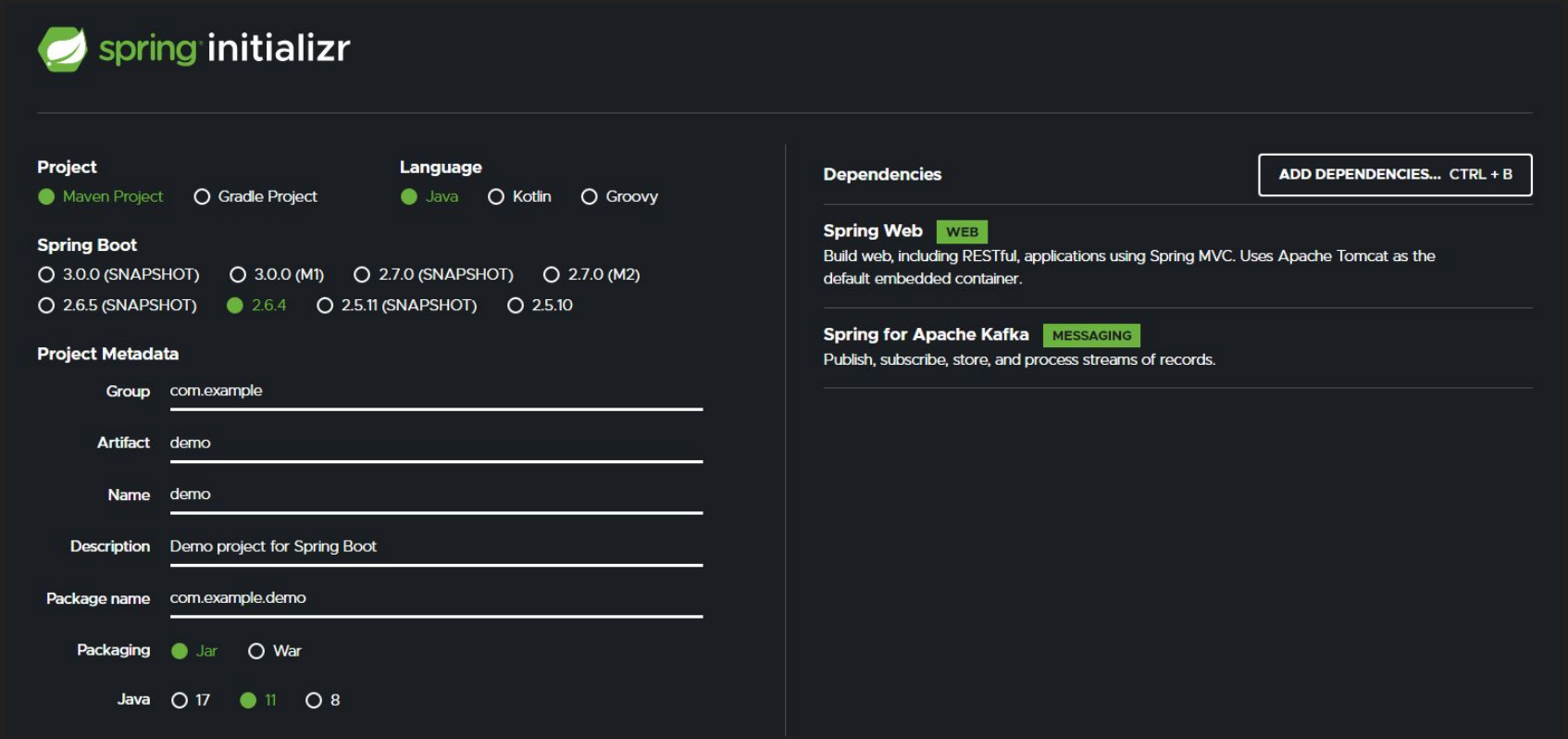
Config Kafka

- Now in the same folder config open `server.properties` and scroll down to `log.dirs` and paste the path.
- To the path add /kafka-logs

Run Kafka

```
.\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties

.\bin\windows\kafka-server-start.bat .\config\server.properties
```

Create Producer App



Kafka Config

```
package com.example.demo.config;
...
@Configuration
public class KafkaConfig {
    @Bean
    public ProducerFactory<String, String> producerFactory() {
        Map<String, Object> config = new HashMap<>();
        config.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "127.0.0.1:9092");
        config.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
        config.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class);

        return new DefaultKafkaProducerFactory<>(config);
    }

    @Bean
    public KafkaTemplate<String, String> kafkaTemplate() {
        return new KafkaTemplate<>(producerFactory());
    }
}
```

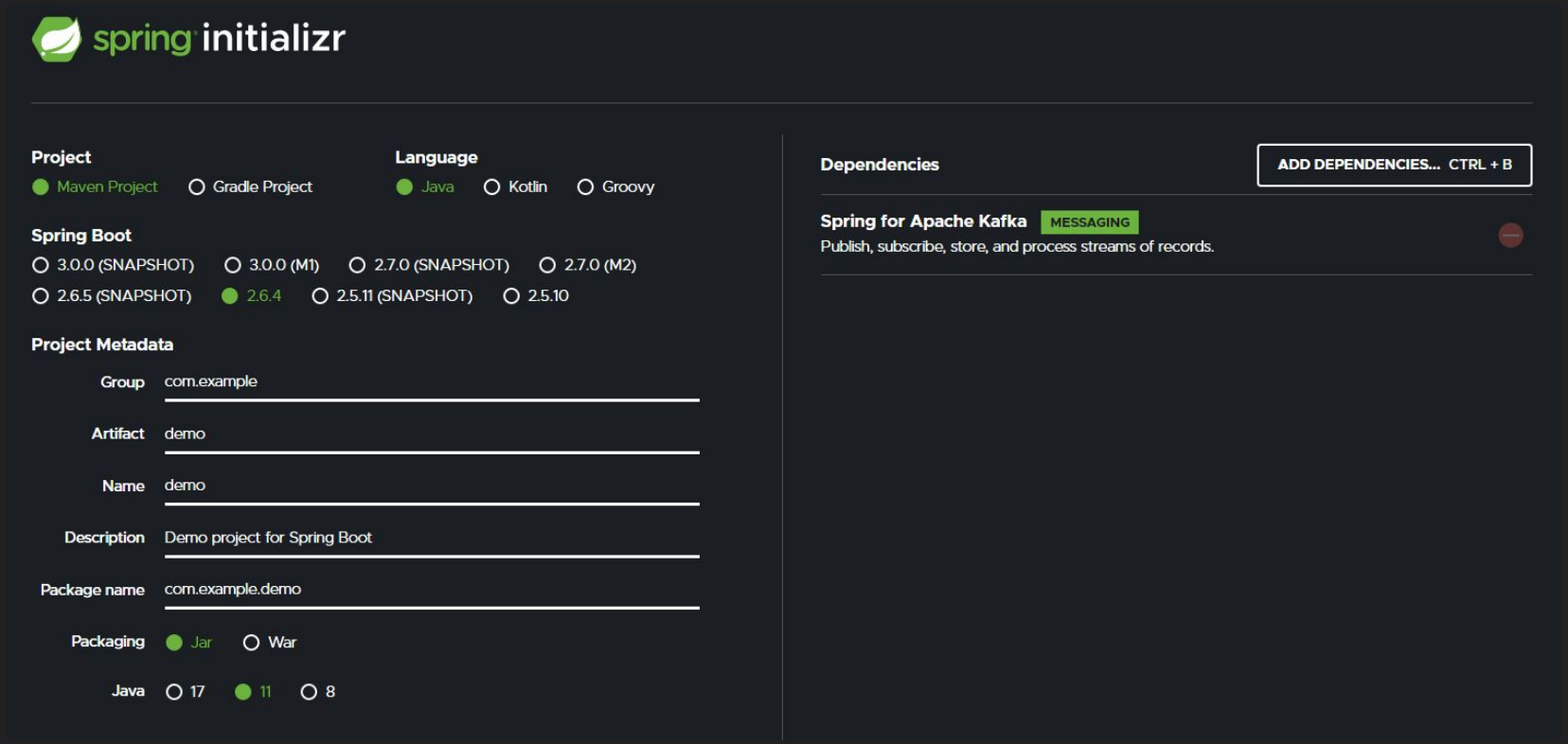
Test Controller

```
package com.example.demo.controller;
...
@RestController
public class TestController {
    // Autowiring Kafka Template
    @Autowired
    KafkaTemplate<String, String> kafkaTemplate;

    @GetMapping("/publish/{message}")
    public String publishMessage(@PathVariable String message) {

        // Sending the message
        kafkaTemplate.send("MyTopic", message);

        return "Published Successfully";
    }
}
```

Listen to the messages

```
.\bin\windows\kafka-console-consumer.bat --bootstrap-server localhost:9092
--topic MyTopic --from-beginning
```

Create Consumer App



Kafka Config

```
@Configuration
@EnableKafka
public class KafkaConfig {
    @Bean
    public ConsumerFactory<String, String> consumerFactory() {
        Map<String, Object> config = new HashMap<>();

        config.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "127.0.0.1:9092");
        config.put(ConsumerConfig.GROUP_ID_CONFIG, "group_id");
        config.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
        config.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);

        return new DefaultKafkaConsumerFactory<>(config);
    }
 ...
```

Kafka Config

```
...
    // Creating a Listener
    @Bean
    public ConcurrentKafkaListenerContainerFactory<String, String>
kafkaListenerContainerFactory() {
        ConcurrentKafkaListenerContainerFactory<String, String> factory
                = new ConcurrentKafkaListenerContainerFactory<>();
        factory.setConsumerFactory(consumerFactory());
        return factory;
    }
}
```

Kafka Consumer

```java
package com.example.demo;

import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.stereotype.Component;

@Component
public class KafkaConsumer {

    @KafkaListener(topics = "MyTopic", groupId = "group_id")
    public void consume(String message) {
        System.out.println("message = " + message);
    }
}
```

application.properties

```
server.port=8081
```

References

https://www.geeksforgeeks.org/how-to-install-and-run-apache-kafka-on-windows/

https://www.geeksforgeeks.org/spring-boot-kafka-producer-example/

https://www.geeksforgeeks.org/spring-boot-kafka-consumer-example/

https://www.baeldung.com/spring-kafka

Exercise

Create producer app and consumer app where producer app has endpoint:

`GET /ping`

When it get hit it will send message: “`ping`” to topic named “`ping-request`”.

Exercise

Consumer app will consume the message and send back message: “pong” to topic named “pong-response”.

Producer app then consume the message and print the message “pong” to console.

Redis for Caching

What is Redis?

Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache, and message broker. Redis provides data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs, geospatial indexes, and streams.

https://redis.io/

Install Redis Server

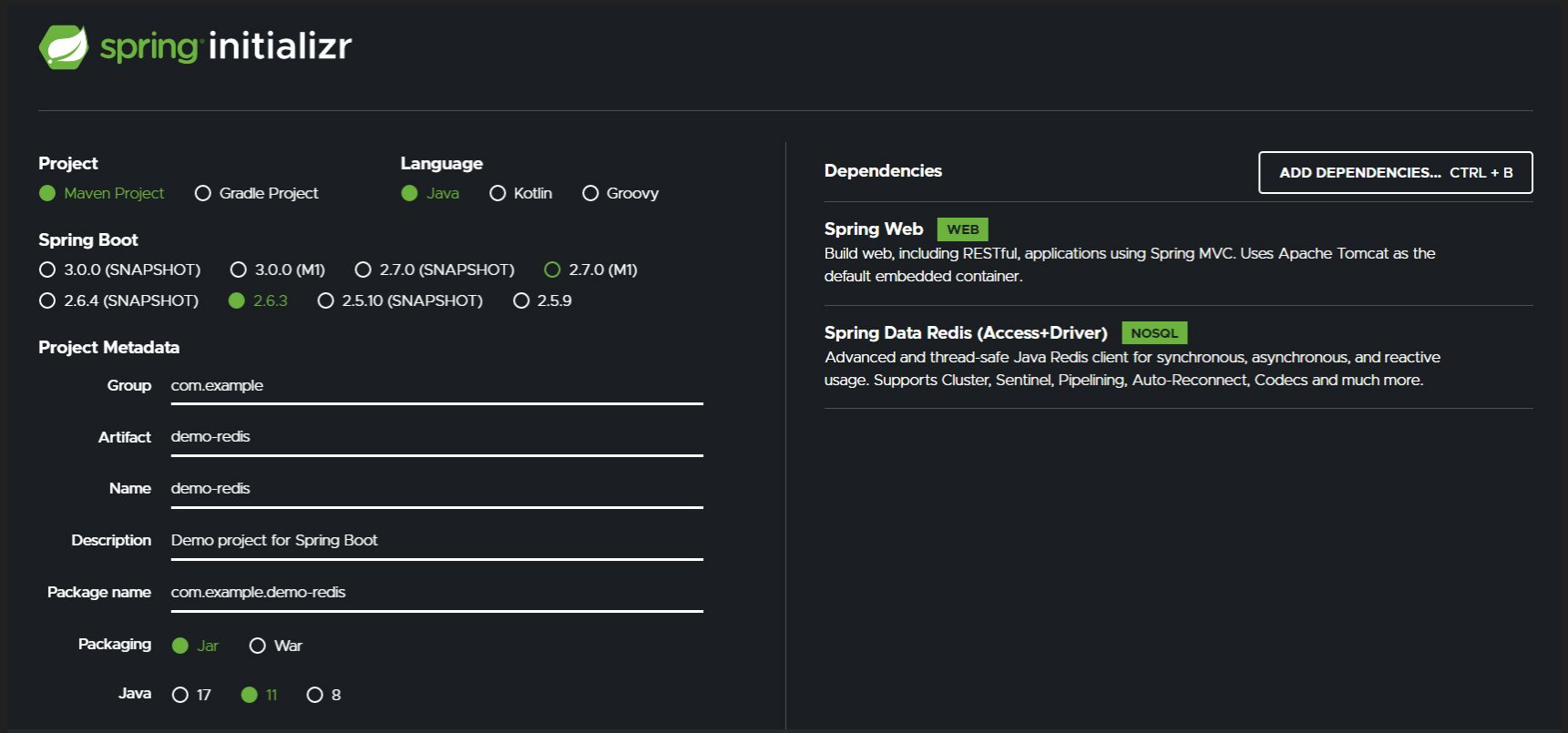
https://redis.io/download

https://hub.docker.com/_/redis/

Our Redis Server

`108.160.135.163:6379`

Create Redis App



Additional Dependency

```
<dependency>
    <groupId>redis.clients</groupId>
    <artifactId>jedis</artifactId>
</dependency>
```

Redis Config

```java
@Configuration
public class RedisConfig {

    @Bean
    public JedisConnectionFactory jedisConnectionFactory() {
        RedisStandaloneConfiguration redisStandaloneConfiguration =
                new RedisStandaloneConfiguration("108.160.135.163", 6379);
        return new JedisConnectionFactory(redisStandaloneConfiguration);
    }

    @Bean
    public RedisTemplate<String, Object> redisTemplate() {
        RedisTemplate<String, Object> template = new RedisTemplate<String, Object>();
        template.setConnectionFactory(jedisConnectionFactory());
        template.setKeySerializer(new StringRedisSerializer());
        template.setValueSerializer(new GenericJackson2JsonRedisSerializer());

        return template;
    }

}
```

Test Controller

```
@RestController
public class TestRedisController {

    @Autowired
    RedisTemplate<String, Object> redisTemplate;

    @GetMapping("/initcache1")
    private void initCache() {
        redisTemplate.boundValueOps("city").set("surabaya");
    }

    @GetMapping("/getcache1")
    private String getCache() {
        return redisTemplate.opsForValue().get("city").toString();
    }
...
}
```

Test Controller

```
...
    @GetMapping("/initcache2")
    private void initCache2() {
        redisTemplate.boundHashOps("person").put("firstname", "john");
        redisTemplate.boundHashOps("person").put("lastname", "smith");
    }

    @GetMapping("/getcache2")
    private String getCache2() {
        return redisTemplate.opsForHash().get("person", "firstname").toString();
    }
}
```

Reference

https://medium.com/@hulunhao/how-to-use-redis-template-in-java-spring-boot-647a7eb8f8cc