# 3. Building A Bot in Java with Spring Boot

This guide is based on a presentation I've written, creating a fully-functional Java bot, using the [Spring Boot Starter Symphony API](http://v) component.

By following this, you will end up with a fully functioning bot which will "echo back" what you tell it in a given room.

Apart from Step 1, which is dependent on Support, this should take around 1 hour to complete, if you don't run into any other problems.

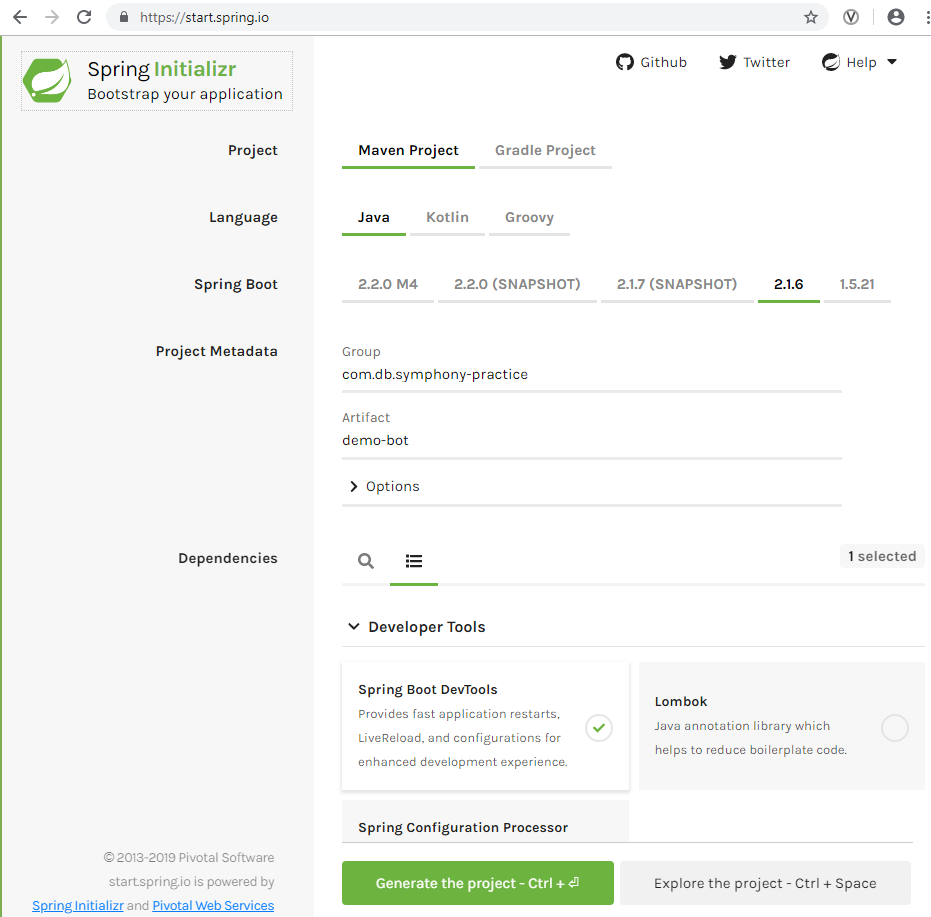
You can view [all of the code for this project here](https://stash.gto.intranet.db.com:8082/projects/SYMPHONYP/repos/spring-boot-starter-symphony-api-demo/browse).

## 1. Create a New Bot Account

To do this, follow the instructions here.  [1. Creating A Bot Account For Symphony](file:///C:\display\SYMPHONYP\1.+Creating+A+Bot+Account+For+Symphony).

Once you have had the bot account approved, you can [Test That IT Works In Bash](https://confluence.intranet.db.com/display/SYMPHONYP/Testing+Connectivity)

## 2. Create a Spring Boot App



* The best way to do this is via the website, <https://start.spring.io>
* As you can see, I've created a maven project, with developer tools.
* When you click download, it will deliver a zip file which you can unpack in your filesystem

## 3. Import Into Your IDE

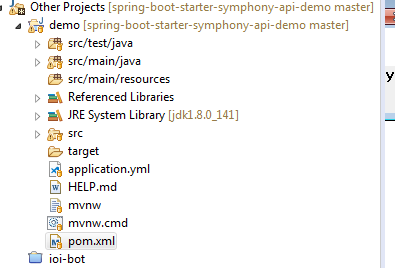
## C:\1bba1565bef33d43ef03d69dbefce03b

I am going to use eclipse, which means I have to run:

mvn eclipse:eclipse -DdownloadSources

This will allow me to import my project into eclipse.  If you are using a different IDE, substitute your step here.

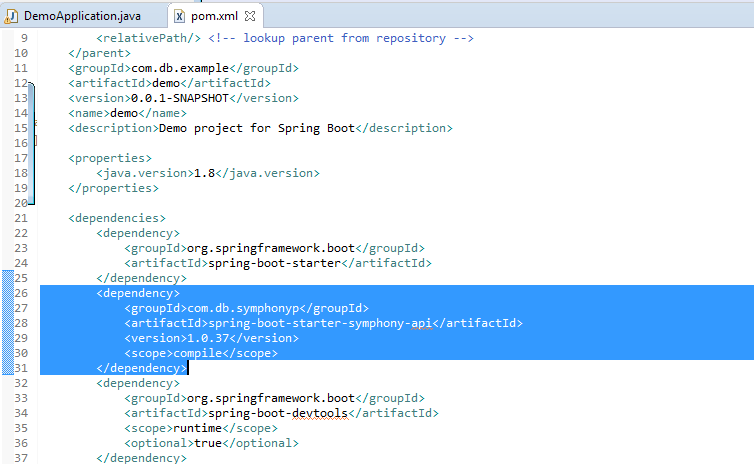
Here is my project in eclipse:



* If I run the project now, I should see Spring start, and then finish (as we haven't actually added any code).

## 4. Add The Spring-Boot-Starter-Symphony-Api Dependency

In pom.xml add this dependency (please check artifactory for later versions)



<dependency>  
 <groupId>com.db.symphonyp</groupId>  
 <artifactId>spring-boot-starter-symphony-api</artifactId>  
 <version>check artifactory</version>  
 <scope>compile</scope>  
</dependency>

## 5.  Add Jax-RS Dependency

JAX-RS provides the code to call REST endpoints.  You can either add Jersey, or CXF.  Both are detailed on this page under Step 2: [Choose a JAX-RS Implementation.](https://stash.gto.intranet.db.com:8082/projects/SYMPHONYP/repos/spring-boot-starter-symphony-api/browse)

#### CXF

Include this dependency:

<dependency>

<groupId>org.apache.cxf</groupId>

<artifactId>cxf-rt-rs-client</artifactId>

<version>...</version>

<scope>compile</scope>

</dependency>

#### Jersey

Include this:

<dependency>

<groupId>org.glassfish.jersey.core</groupId>

<artifactId>jersey-common</artifactId>

</dependency>

<dependency>

<groupId>org.glassfish.jersey.core</groupId>

<artifactId>jersey-client</artifactId>

</dependency>

<dependency>

<groupId>org.glassfish.jersey.inject</groupId>

<artifactId>jersey-hk2</artifactId>

</dependency>

<dependency>

<groupId>org.glassfish.jersey.media</groupId>

<artifactId>jersey-media-json-jackson</artifactId>

</dependency>

<dependency>

<groupId>org.glassfish.jersey.connectors</groupId>

<artifactId>jersey-apache-connector</artifactId>

</dependency>

<dependency>

<groupId>org.glassfish.jersey.media</groupId>

<artifactId>jersey-media-multipart</artifactId>

</dependency>

You can use the Jersey BOM to avoid specifying version numbers like so:

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.glassfish.jersey</groupId>

<artifactId>jersey-bom</artifactId>

<version>...</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

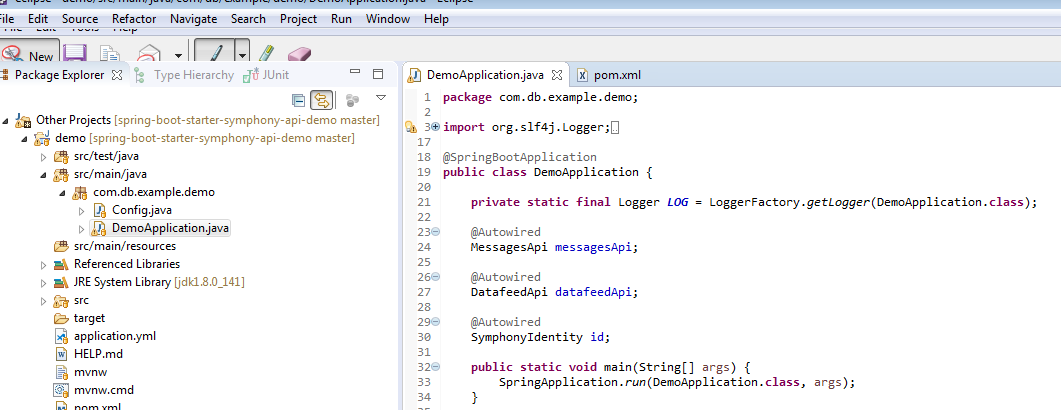
## 6. Rebuild your Project

mvn eclipse:eclipse -DdownloadSources

* Then, refresh in eclipse.

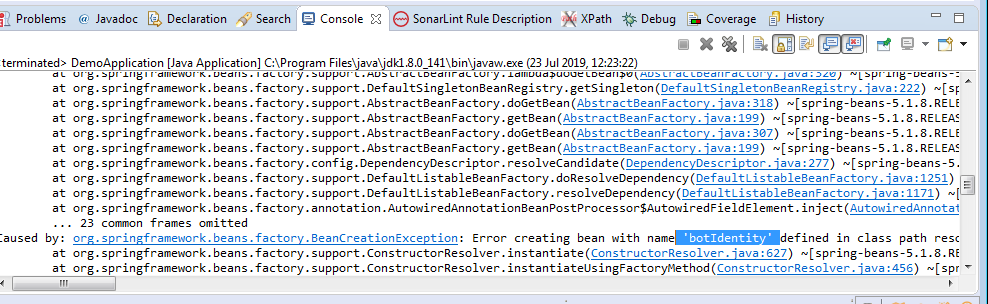
## 7.  Autowiring!

If the dependencies loaded correctly, you should now be able to auto-wire some beans:



Here, I have auto-wired the **messagesApi, datafeedApi and the symphonyIdentity.**The apis are provided by the [symphony-java-client-bindings](https://stash.gto.intranet.db.com:8082/projects/SYMPHONYP/repos/symphony-java-client-parent/browse/bindings) project, while the ID is a bean exposed by the [Spring Boot Starter Symphony API](http://v/) configuration, containing the id of your bot.

What happens if we run now?



Disaster!  We now need to configure our application..

## 8. Bot Configuration

If you have completed step 1, you should have the bot certificate, email address and private key.  Create an **application.yml** file and add them like so:

symphony:   
 bot:  
 identity:  
 email: [symphony-practice-bot3@list.db.com](mailto:symphony-practice-bot3@list.db.com)  
 privateKey: |  
 -----BEGIN RSA PRIVATE KEY-----  
 MIIJKQ...  
 ...neY  
 -----END RSA PRIVATE KEY-----  
 certificates:   
 - |  
 -----BEGIN CERTIFICATE-----  
 MIIG...  
 ...IIU6A==  
 -----END CERTIFICATE-----  
 - |  
 -----BEGIN CERTIFICATE-----  
 MIIFv...  
 ...sN0=  
 -----END CERTIFICATE-----  
 - |  
 -----BEGIN CERTIFICATE-----  
 MIIGH...EWu3rzA=  
 -----END CERTIFICATE-----

### Some Notes

* I am using YAML's block format here to allow me to paste the keys and certificates in.  There are many ways to format this in YAML, you can enclose in a string on a single line if you want to.
* I am providing the entire certificate chain from the certificate bundle provided back in the PKI process, but I actually think you can get away with just the first one.
* You can use a p12 file if you want to.  Review the docs in [Spring Boot Starter Symphony API](http://v/) for details.
* Column formatting is very important in yaml!  Make sure everything lines up.

## 9. Pod Configuration

Add the pod information to you **application.yml** too:

symphony:  
 apis:  
 - id: dbtest   
 pod:  
 url: <https://db-test.symphony.com/pod>  
 proxy:  
 host: [userproxy.intranet.db.com](http://userproxy.intranet.db.com)  
 sessionauth:   
 url: <https://db-test-api.symphony.com:443/sessionauth>  
 proxy:  
 host: [userproxy.intranet.db.com](http://userproxy.intranet.db.com)  
 keyauth:   
 url: <https://ipsmpu02.de.db.com:8444/keyauth>  
 agent:  
 url: <https://ipsmpu01.de.db.com:8444/agent>

### Some Notes

* This configuration is also described in [Spring Boot Starter Symphony API](http://v/).
* You have separate proxy entries for each of the main endpoints, **pod, sessionauth, keyauth**and **agent.**

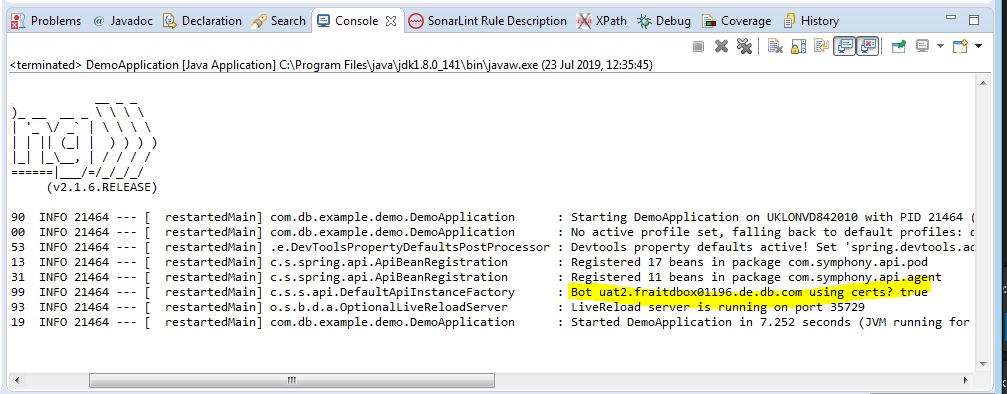
## 10. Object Mapper

Out-of-the-box, Spring Boot contains Jackson for doing marshalling to JSON, however without spring-boot-starter-web, it doesn't expose a bean for this.  Since we're going to need to marshal to JSON for calling symphony APIs, we need to provide one:

@Configuration  
public class Config {

@Bean  
 public ObjectMapper objectMapper() {  
 return new ObjectMapper();  
 }  
}

## 11. Run It

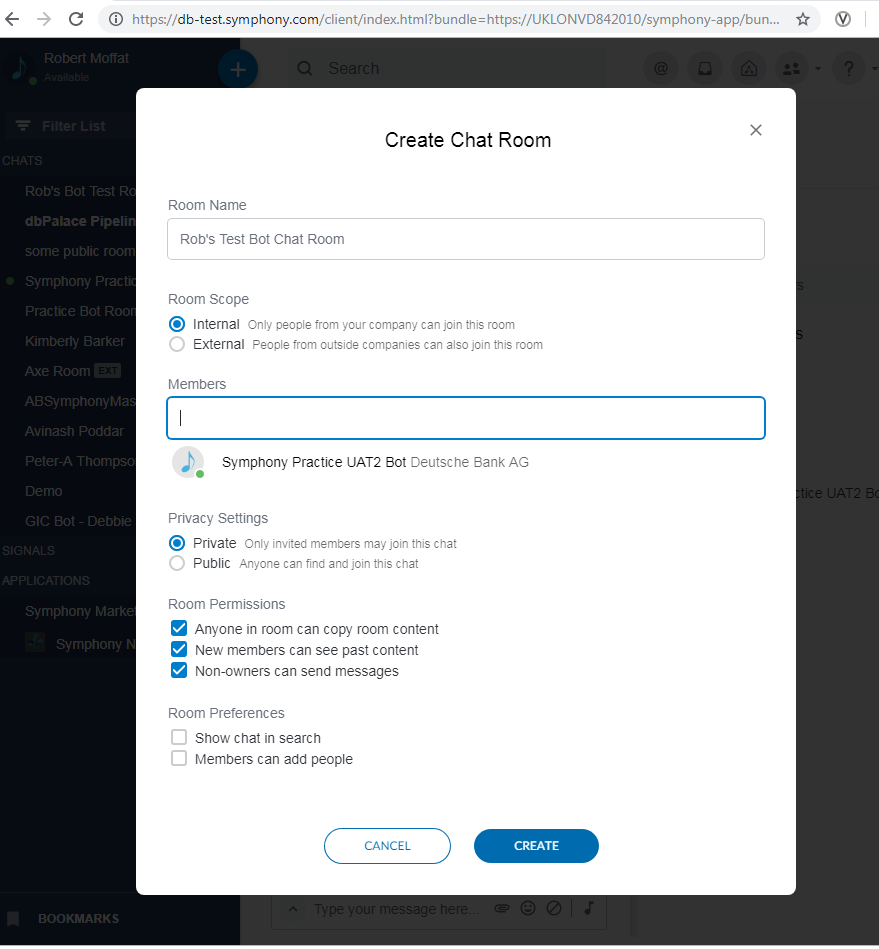


At this point, you should be able to start your app, and the autowiring should complete successfully.

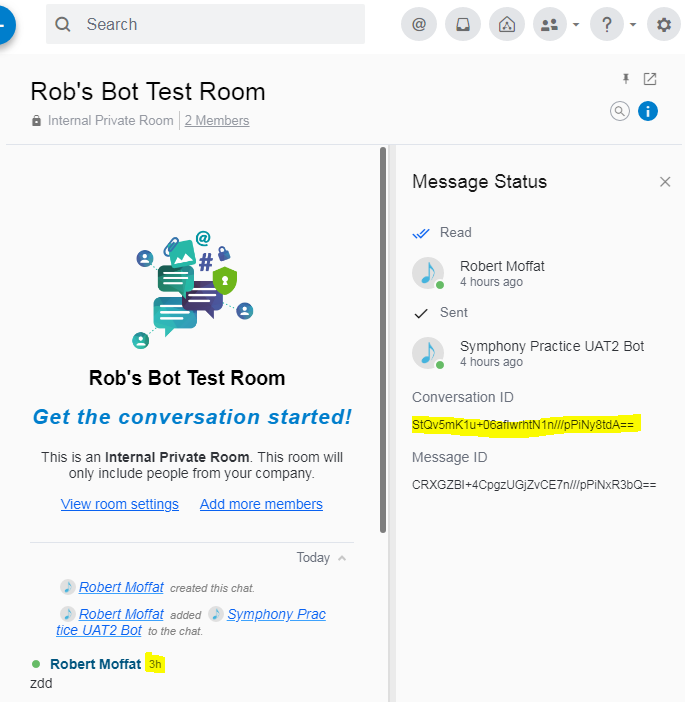
You'll get some information on the screen about the bot you've autowired.

## 12.  Create A Chat Room

Create a chat room and add the bot



You'll need to know the chat room stream ID, which you can find by clicking the time next to your name:

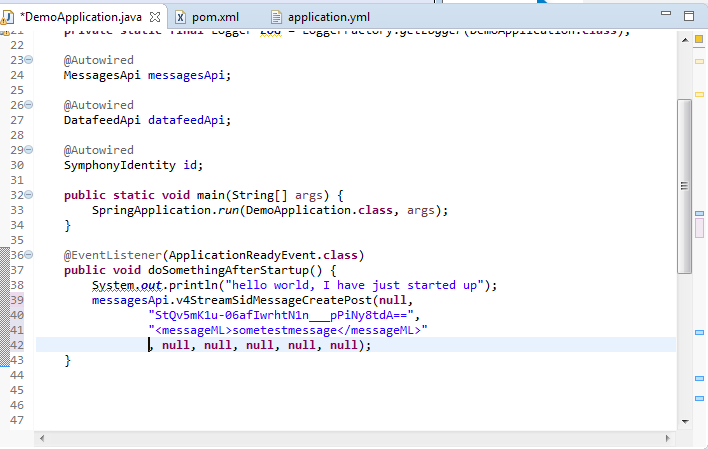


The stream ID (or conversation ID) appears on the right.

StQv5mK1u+06afIwrhtN1n///pPiNy8tdA==

## 12. Send A Message

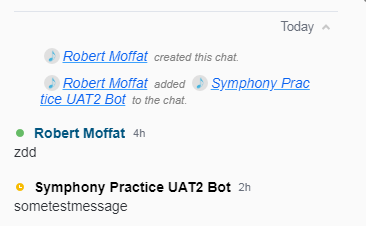
We're going to add an event listener, so that the bot will send an event when the room is created.



Add this code into your main class somewhere:

@EventListener(ApplicationReadyEvent.class)  
public void doSomethingAfterStartup() {  
 System.out.println("hello world, I have just started up");  
 messagesApi.v4StreamSidMessageCreatePost(null,   
 "<your url-safe stream Id>",   
 "<messageML>sometestmessage</messageML>"  
 , null, null, null, null, null);  
}

When you start the application now, you'll see the test message in Symphony:



### Some Notes

* You'll need to make your stream ID URL-Safe, by replacing / with \_ and + with -.  It's documented here.<https://stackoverflow.com/questions/47874229/symphony-embedded-chat-module-not-working>
* This is the method we are calling from Symphony: <https://developers.symphony.com/restapi/reference#create-message-v4>
* [Symphony-Java-Client-Bindings](https://stash.gto.intranet.db.com:8082/projects/SYMPHONYP/repos/symphony-java-client-parent/browse/bindings) will handle **sessionTokens** and **keyManagerTokens**, so you can leave those parameters blank.

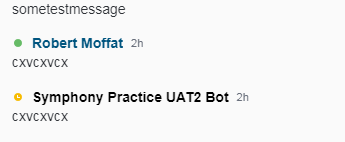
## 13. Listening For Messages

We're going to add code to listen to messages sent to the bot.

to do this, add the following in your **doSomethingAfterStartup** method:

// create a datafeed  
Datafeed df = datafeedApi.v4DatafeedCreatePost(null, null);  
   
Streams.createWorker(() -> datafeedApi.v4DatafeedIdReadGet(df.getId(), null, null, 50), e -> e.printStackTrace()))  
 .stream()  
 .filter(e -> e.getType().equals("MESSAGESENT"))  
 .map(e -> e.getPayload().getMessageSent().getMessage())  
 .filter(m -> !m.getUser().getEmail().equals(id.getEmail()))  
 .forEach(m -> messagesApi.v4StreamSidMessageCreatePost(null, "<url safe stream id>", m.getMessage(), null, null, null, null, null));

When you run now, you should see your messages echoed back to you:



### Some Notes

* Streams is a class from [Symphony-Java-Client-Bindings](https://stash.gto.intranet.db.com:8082/projects/SYMPHONYP/repos/symphony-java-client-parent/browse/bindings).  You can read about it there.  It allows you to create a stream from symphony REST calls, handling tokens and so on. The first parameter is the call to get some events from the stream, the second parameter is a function to handle exceptions.  You do what you want here.
* It returns a stream.  We are filtering this so that we only get the "MESSAGESENT" event, and then pulling the message out.
* We ignore messages from ourselves (otherwise, an infinite loop ensues)
* At the end, we write the message back as it came in.

## For Bonus Points

* Add Spring-Boot-Starter-Web to your application and view the /health and /metrics endpoints as you go.  This will show you the state of the bot, the agent and the pod, and details of the api calls you have made.