



## Lab 3

### REST web service - contract first in SwaggerHub

---

This lab will show you how you can specify a REST interface on SwaggerHub – a cloud platform that we will use for specifying our API, followed by generating code for a Spring Boot implementation. SwaggerHub does this using the Swagger/OpenAPI specification.

You can consider using SwaggerHub if you want to develop your REST services in a contract first fashion.

The lab uses the Swagger/OpenAPI 2.0 specification, and not (yet) the OpenAPI 3.0 specification: the tooling seems a bit more stable...

The starting point for this lab is to have the provided VirtualBox machine up-and-running:

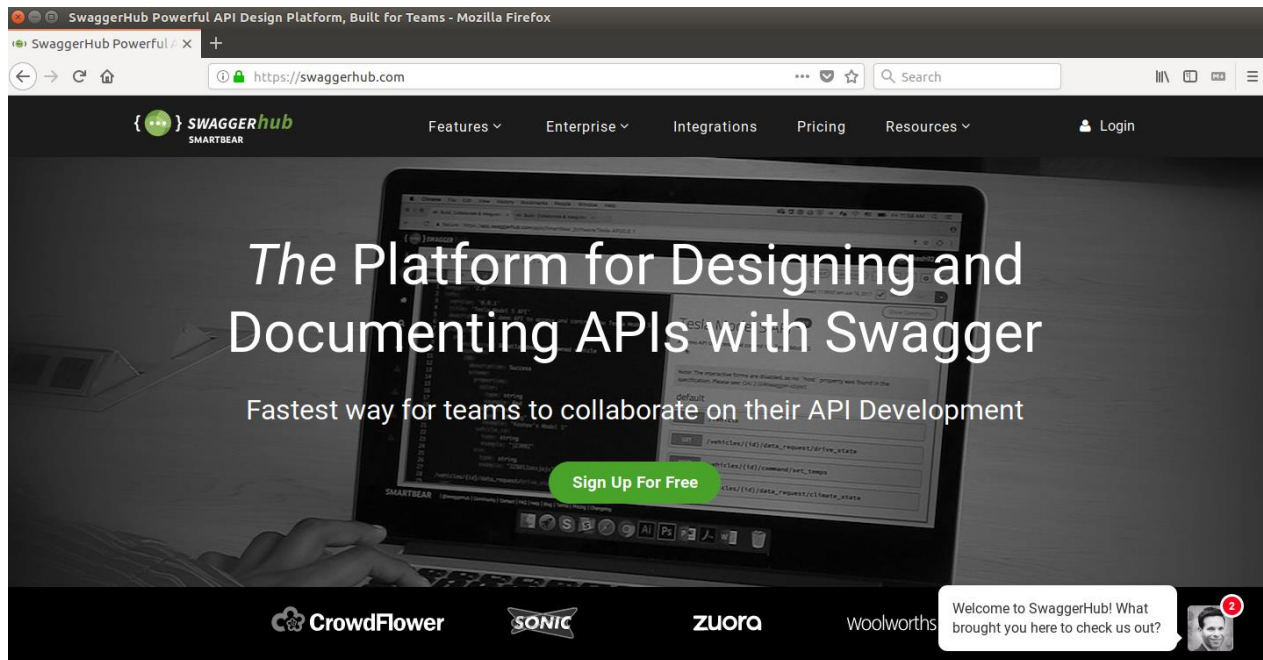
- You are logged in under user/password: developer/welcome01
- You have updated the labs running the `git pull` command in the lab workspace directory `/home/developer/projects/SIGSpringBoot101`

#### 1. Registration with SwaggerHub

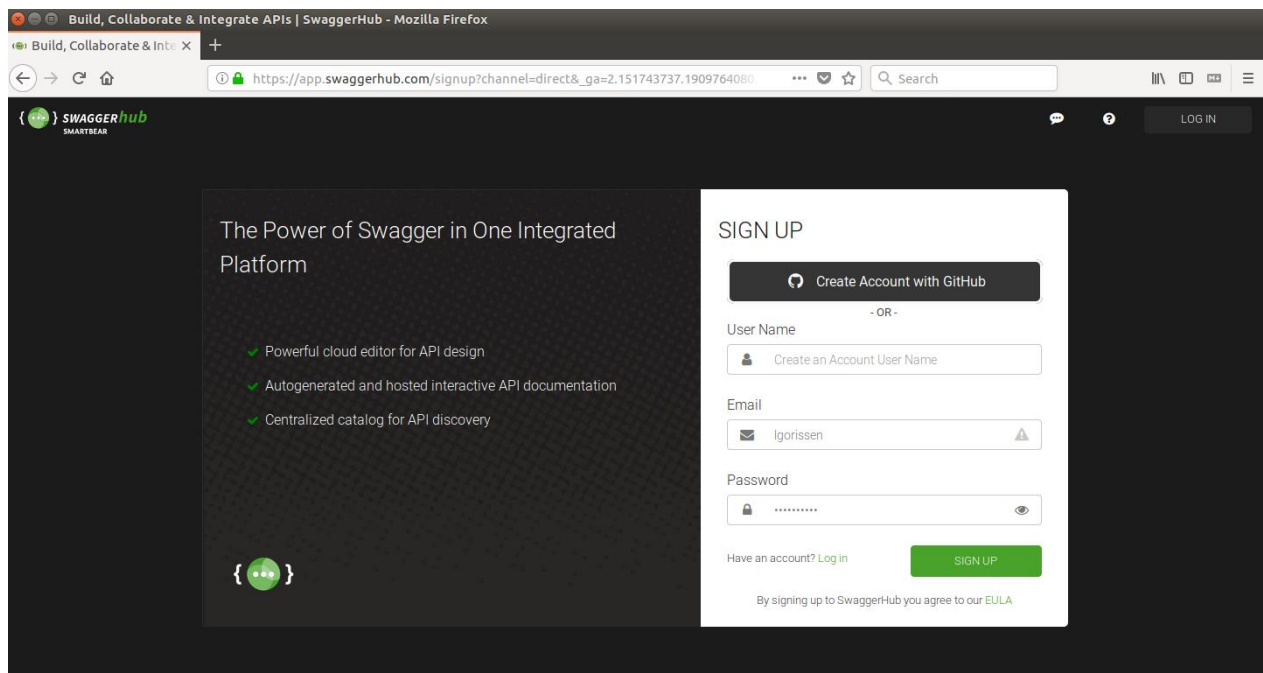
Goto [swaggerhub.com](https://swaggerhub.com) in your VirtualBox machine:



## Spring Boot 101 - intro, demo & hands on with Java, REST APIs, Containers & Cloud



Click the 'Sign Up For Free' button:

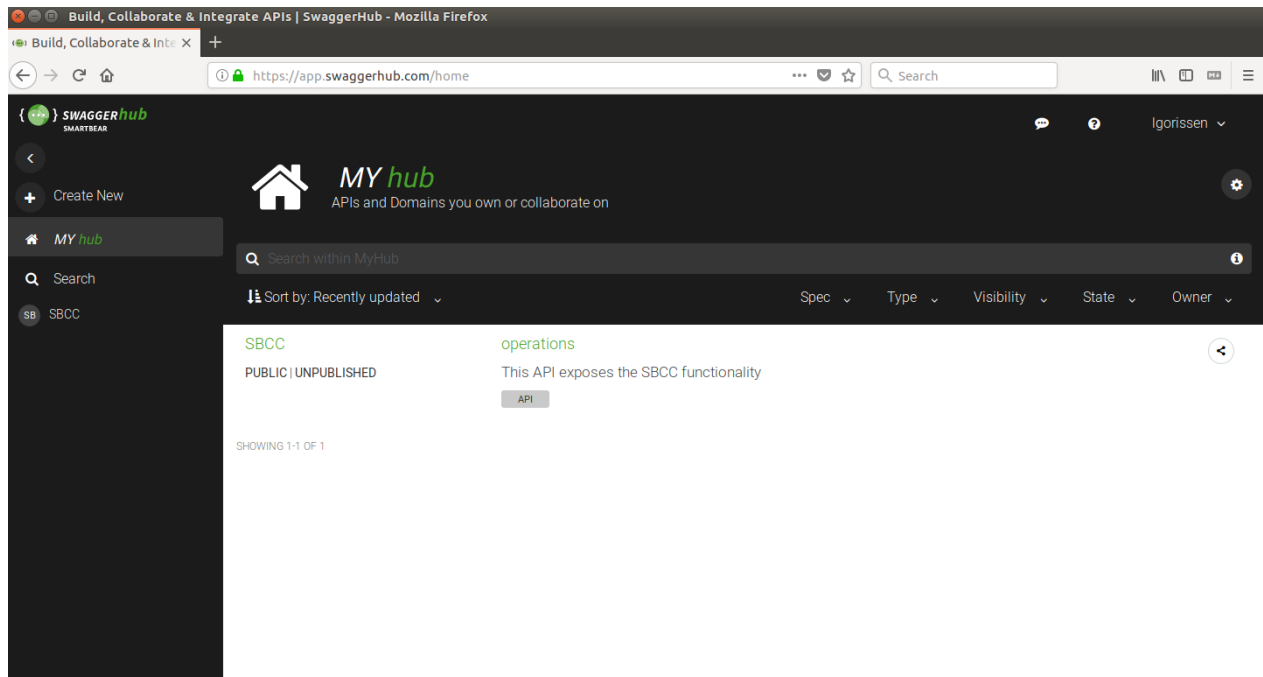


Here, you can pick your preferred option for registration.

After completing the registration and logging in, you should have a page pretty similar to the one below:



## Spring Boot 101 - intro, demo & handson with Java, REST APIs, Containers & Cloud



Note in the above screen, that there is already an interface present, named SBCC – it shouldn't be there in your screen.

## 2. A HelloWorld API in SwaggerHub

Now that you have your account in SwaggerHub and you are logged in, you are ready to get to work. We will first create a simple API in SwaggerHub.

The following steps will be done:

- Step 1: create the API specification in SwaggerHub
- Step 2: generating code in SwaggerHub
- Step 3: import into eclipse and add business logic in the code
- Step 4: run and test the API

### Step 1: create the API specification in SwaggerHub

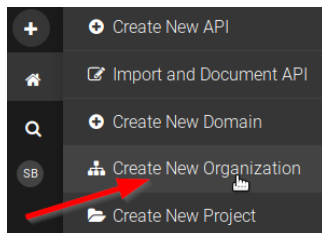
Log in into SwaggerHub, and click the + icon to start adding a new interface:



## Spring Boot 101 - intro, demo & handson with Java, REST APIs, Containers & Cloud



Start by creating a new organization: Create New Organization



Complete the form as shown below, taking two things into account:

- Organization Account Name: use your own name, e.g. DroneBuzzersXYZ where XYZ are your initials
- Organization e-mail: use your own e-email address

### Add Organization ✕

Please enter a unique Organization Account Name, Organization Email and a Title (Optional).

Organization Account Name

Organization Email

Title

Note: You will start a trial for this organization. Your personal plan will not be changed.

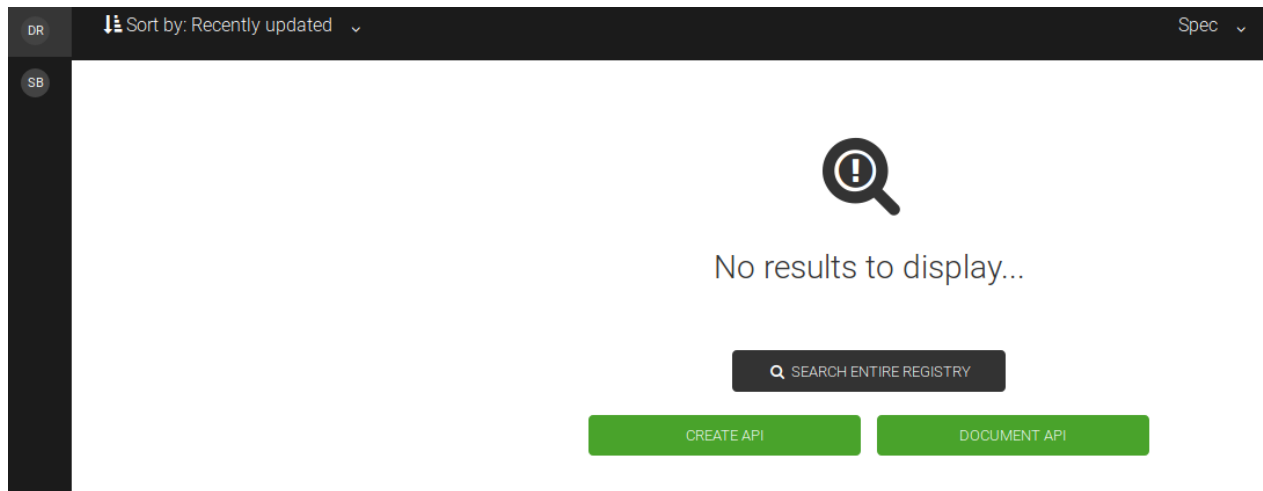
CANCELADD ORGANIZATION

**Important:** the Organization Account Name that you entered in the above form will appear as part of the URL of the service endpoint. Please note that the screenshots may refer to DroneBuzzers, where you will have the organization name that you entered yourself. Where necessary, we will address the difference...

After clicking the 'Add Organization' button, the screen should look like:



## Spring Boot 101 - intro, demo & handson with Java, REST APIs, Containers & Cloud



Click on the 'CREATE API' button to start adding the API. Complete the resulting form as shown below:

Select a Template or create a Blank API ✕

---

Enter a unique name for your API definition below and select a Template  
Select None for a Blank API.

OpenAPI Version 2.0 ▾

Template Simple API ▾

Name

Visibility Public ▾

Owner DroneBuzzers ▾

Auto Mock API ON ☐

CANCEL CREATE API

With respect to the above form some remarks:

- The OpenAPI version is set to 2.0. This version is also known as 'Swagger 2.0'.
- The template is 'Simple API'. For a good overview of how a more realistic interface specification can look like, it is good to examine the Petstore example that is also available. Just don't do that now ;-)
- Visibility is set to Public as the number of private APIs that can be created with a free account is fixed to 1

Click CREATE API and wait for the magic to finish:



## Spring Boot 101 - intro, demo & hands on with Java, REST APIs, Containers & Cloud

DroneBuzze... | HelloWorld | 1.0.0

PUBLIC UNPUBLISHED

Last Saved: 12:48:37 pm Mar 10, 2018 VALID Save

Editor Split UI

```
1 swagger: 2.0
2 info:
13
14 # tags are used for organizing operations
15 tags:
16 - name: admins
17 - name: developers
18
19
20
21 paths:
22 /inventory:
23 get:
24 post:
25
26 definitions:
27 InventoryItem:
28 Manufacturer:
29
30 # Added by API Auto Mocking Plugin
31 host: virtserver.swaggerhub.com
32 basePath: /DroneBuzzers/HelloWorld/1.0.0
33 schemes:
34 - https
```

Contact the developer  
Apache 2.0

Show Comments

Schemes  
HTTPS

admins Secured Admin-only calls

POST /inventory adds an inventory item

developers Operations available to regular developers

GET /inventory searches inventory

Models

InventoryItem {  
id\* string(\$uuid)  
example: d290f1ee-6c54-4b01-90e6-d701748f0851  
name\* string  
example: Widget Adapter  
releaseDate\* string(\$int32)  
example: OrderedMap {}  
manufacturer\* Manufacturer > {...} <←  
}

Now is a good moment to spend some time to understand what we're looking at:

- The screen is now in split view (red oval on top): it shows the OpenAP/Swagger 2.0 interface definition on the left. On the right is the description of the API in Swagger documentation style.
- The top left green rectangle shows the tags that can be used for logical grouping of operations
- The middle left green rectangle describes the paths to the endpoints and for each endpoint the operations and parameters
- The bottom left green rectangle has the definitions of the data types that can be consumed/produced by the operations
- The right side of the screen shows the Swagger documentation corresponding to the interface definition

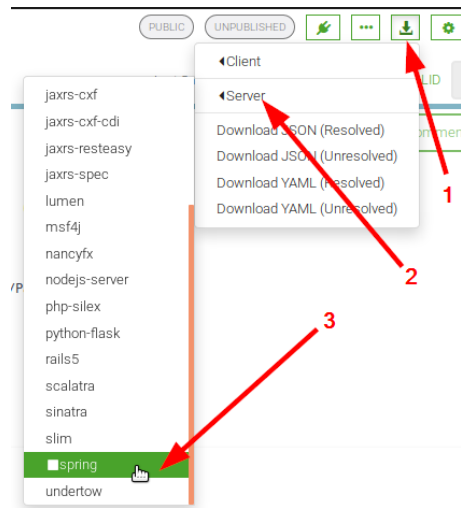
Note how the comments and examples for the Swagger documentation are incorporated into the API definition.

### Step 2: generating code in SwaggerHub

SwaggerHub can also generate code for an API definition: for both client and server side. And for many languages. These code generation options are a bit difficult to find: the 3 magic clicks are shown below:

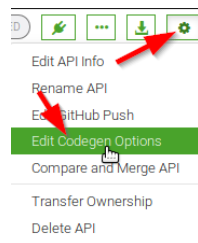


## Spring Boot 101 - intro, demo & handson with Java, REST APIs, Containers & Cloud



Not how many language options there are: impressive!

Before we start our 'Spring' code generation, we will first set the code generation options. Go to the code generation options as shown below:



In the pop-up window, select spring in the Servers section and then complete the settings as shown in the table below – and in the screenshot shown underneath:

Setting	Value
<b>useTags</b>	<i>not checked</i>
<b>implicitHeaders</b>	<i>not checked</i>
<b>configPackage</b>	com.dronebuzzers.helloworld.service.config
<b>interfaceOnly</b>	<i>not checked</i>
<b>artifactVersion</b>	1.0.0
<b>sortParamsByRequiredFlag</b>	<i>not checked</i>
<b>useOptional</b>	<i>not checked</i>
<b>singleContentTypes</b>	<i>not checked</i>
<b>sourceFolder</b>	/src/main/java
<b>serializableModel</b>	<i>not checked</i>
<b>artifactDescription</b>	DroneBuzzers HelloWorld
<b>delegatePattern</b>	<i>checked</i>



<b>scmDeveloperConnection</b>	
<b>apiPackage</b>	com.dronebuzzers.helloworld.service.api
<b>licenseName</b>	
<b>invokerPackage</b>	com.dronebuzzers.helloworld.service.invoker
<b>dateLibrary</b>	
<b>artifactId</b>	helloworld-rest-service
<b>licenseUrl</b>	
<b>swaggerDocketConfig</b>	<i>checked</i>
<b>useBeanValidation</b>	<i>not checked</i>
<b>withXml</b>	<i>not checked</i>
<b>responseWrapper</b>	
<b>developerEmail</b>	<u><i>insert your own email</i></u>
<b>developerOrganizationUrl</b>	<u><a href="https://www.amis.nl">https://www.amis.nl</a></u>
<b>fullJavaUtil</b>	<i>not checked</i>
<b>bigDecimalAsString</b>	<i>not checked</i>
<b>ensureUniquerParams</b>	<i>not checked</i>
<b>basePackage</b>	com.dronebuzzers.helloworld.service
<b>developerName</b>	<i>&lt;insert your own name&gt;</i>
<b>allowUnicodeIdentifiers</b>	<i>not checked</i>
<b>java8</b>	<i>checked</i>
<b>Title</b>	DroneBuzzers HelloWorld
<b>localVariablePrefix</b>	<i>not checked</i>
<b>groupId</b>	com.dronebuzzers.helloworld
<b>Library</b>	Spring-boot Server application using the SpringFox integration
<b>scmConnection</b>	
<b>hideGenerationTimestamp</b>	
<b>Async</b>	<i>not checked</i>
<b>modelPackage</b>	com.dronebuzzers.helloworld.service.model
<b>developerOrganization</b>	AMIS
<b>artifactUrl</b>	

And in screenshots:





## Spring Boot 101 - intro, demo & hands-on with Java, REST APIs, Containers & Cloud

### Edit Codegen Options

☒ Hide Hidden Options

jaxrs-resteasy

jaxrs-spec

lumen

msf4j

nancyfx

nodejs-server

php-silex

python-flask

rails5

scalatra

sinatra

slim

spring

undertow

— Clients

akka-scala

android

clojure

server / spring

☐ useTags

use tags for creating interface and controller classnames

☐ implicitHeaders

Use of @ApiImplicitParams for headers.

configPackage

com.dronebuzzers.helloworld.service.config

configuration package for generated code

☐ interfaceOnly

Whether to generate only API interface stubs without the server files.

artifactVersion

1.0.0

artifact version in generated pom.xml

☐ sortParamsByRequiredFlag

Sort method arguments to place required parameters before optional parameters.

☐ useOptional

Use Optional container for optional parameters

☐ singleContentTypes

Whether to select only one produces/consumes content-type by operation.

sourceFolder

/src/main/java

source folder for generated code

☐ serializableModel

boolean - toggle "implements Serializable" for generated models

artifactDescription

DroneBuzzers HelloWorld

artifact description in generated pom.xml

☒ delegatePattern

Whether to generate the server files using the delegate pattern

scmDeveloperConnection

SCM developer connection in generated pom.xml

apiPackage

com.dronebuzzers.helloworld.service.api

package for generated api classes

licenseName

The name of the license

invokerPackage

com.dronebuzzers.helloworld.service.invoker

And:



## Spring Boot 101 - intro, demo & hands-on with Java, REST APIs, Containers & Cloud

root package for generated code

dateLibrary

Option: Date library to use

artifactId

artifactId in generated pom.xml

licenseUrl

The URL of the license

☒ swaggerDocketConfig

Generate Spring Swagger Docket configuration class.

☐ useBeanValidation

Use BeanValidation API annotations

☐ withXml

whether to include support for application/xml content type and include XML annotations in the model (works with libraries that provide support for JSON and XML)

scmUrl

SCM URL in generated pom.xml

responseWrapper

wrap the responses in given type

(Future, Callable, CompletableFuture, ListenableFuture, DeferredResult, HystrixCommand, RxObservable or fully qualified type)

developerEmail

developer email in generated pom.xml

developerOrganizationUrl

developer organization URL in generated pom.xml

☐ fullJavaUtil

whether to use fully qualified name for classes under java.util. This option only works for Java API client

☐ bigDecimalAsString

Treat BigDecimal values as Strings to avoid precision loss.

☐ ensureUniqueParams

Whether to ensure parameter names are unique in an operation (rename parameters that are not).

basePackage

base package (invokerPackage) for generated code

developerName

developer name in generated pom.xml

☐ allowUnicodeIdentifiers

boolean, toggles whether unicode identifiers are allowed in names or not, default is false

☒ java8

use java8 default interface

title

server title name or client service name

localVariablePrefix



## Spring Boot 101 - intro, demo & handson with Java, REST APIs, Containers & Cloud

And:

prefix for generated code members and local variables

groupid

groupid in generated pom.xml

library

library template (sub-template) to use

scmConnection

SCM connection in generated pom.xml

hideGenerationTimestamp

hides the timestamp when files were generated

☐ async

use async Callable controllers

modelPackage

package for generated models

developerOrganization

developer organization in generated pom.xml

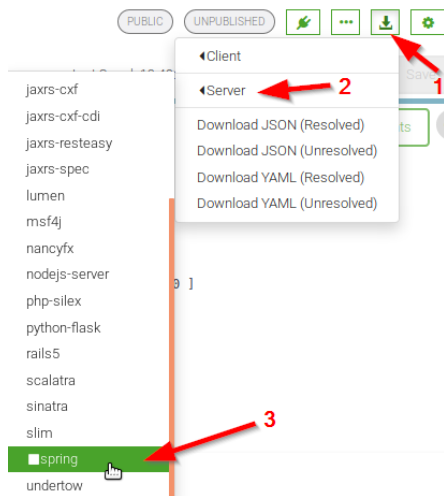
artifactUrl

artifact URL in generated pom.xml

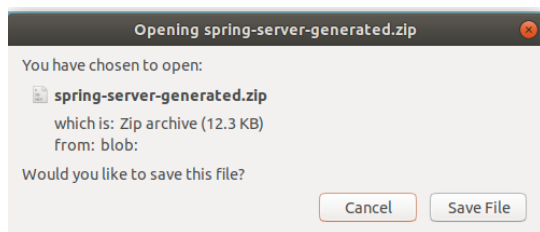
After completing the list, click 'SAVE OPTIONS' and close this screen. Now, the server side code can be generated. Follow these steps:



## Spring Boot 101 - intro, demo & hands-on with Java, REST APIs, Containers & Cloud



This will result in a zip file with the generated server side implementation of the API:



Save the file to the local file system in the VM.

### Step 3: import into eclipse and add business logic in the code

First, unpack the zip file in the right location: lab 3/helloworld. You can either use your own downloaded zip file from the previous step, or use the one provided:

```
/home/developer/projects/SIGSpringBoot101/lab 3/input/helloworld-server-generated.zip
```



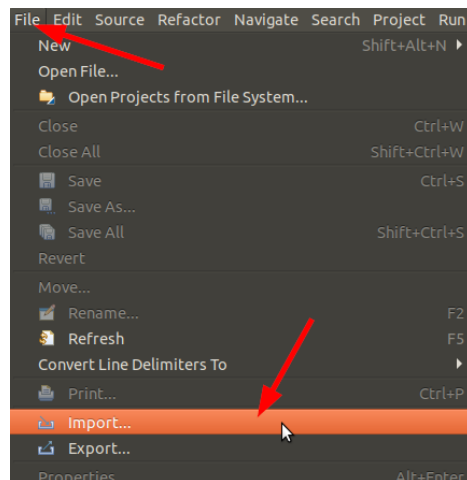
## Spring Boot 101 - intro, demo & hands on with Java, REST APIs, Containers & Cloud

```
developer@developer-VirtualBox: ~/projects/SIGSpringBoot101/lab 3/helloworld
developer@developer-VirtualBox:~/projects/SIGSpringBoot101/lab 3/helloworld$ pwd
/home/developer/projects/SIGSpringBoot101/lab 3/helloworld
developer@developer-VirtualBox:~/projects/SIGSpringBoot101/lab 3/helloworld$ cp ../input/helloworld-server-generated.zip .
developer@developer-VirtualBox:~/projects/SIGSpringBoot101/lab 3/helloworld$ unzip helloworld-server-generated.zip
Archive:  helloworld-server-generated.zip
  inflating: src/main/java/com/dronebuzzers/helloworld/service/model/Manufacturer.java
  inflating: src/main/java/com/dronebuzzers/helloworld/service/model/InventoryItem.java
  inflating: src/main/java/com/dronebuzzers/helloworld/service/invoke/RFC3339DateFormat.java
  inflating: src/main/java/com/dronebuzzers/helloworld/service/invoke/Swagger2SpringBoot.java
  inflating: src/main/java/com/dronebuzzers/helloworld/service/config/HomeController.java
  inflating: src/main/java/com/dronebuzzers/helloworld/service/config/SwaggerDocumentationConfig.java
  inflating: src/main/java/com/dronebuzzers/helloworld/service/config/CustomInstantDeserializer.java
  inflating: src/main/java/com/dronebuzzers/helloworld/service/config/JacksonConfiguration.java
  inflating: src/main/java/com/dronebuzzers/helloworld/service/api/ApiException.java
  inflating: src/main/java/com/dronebuzzers/helloworld/service/api/InventoryApiController.java
  inflating: src/main/java/com/dronebuzzers/helloworld/service/api/NotFoundException.java
  inflating: src/main/java/com/dronebuzzers/helloworld/service/api/InventoryApi.java
  inflating: src/main/java/com/dronebuzzers/helloworld/service/api/ApiOriginFilter.java
  inflating: src/main/java/com/dronebuzzers/helloworld/service/api/ApiResponseMessage.java
  inflating: src/main/java/com/dronebuzzers/helloworld/service/api/InventoryApiDelegate.java
  inflating: src/main/resources/application.properties
  inflating: README.md
  inflating: pom.xml
  inflating: .swagger-codegen/VERSION
developer@developer-VirtualBox:~/projects/SIGSpringBoot101/lab 3/helloworld$
```

Remove the helloworld-server-generated.zip file from the  
~/projects/SIGSpringBoot101/lab 3/helloworld directory.

Next, open Eclipse STS 

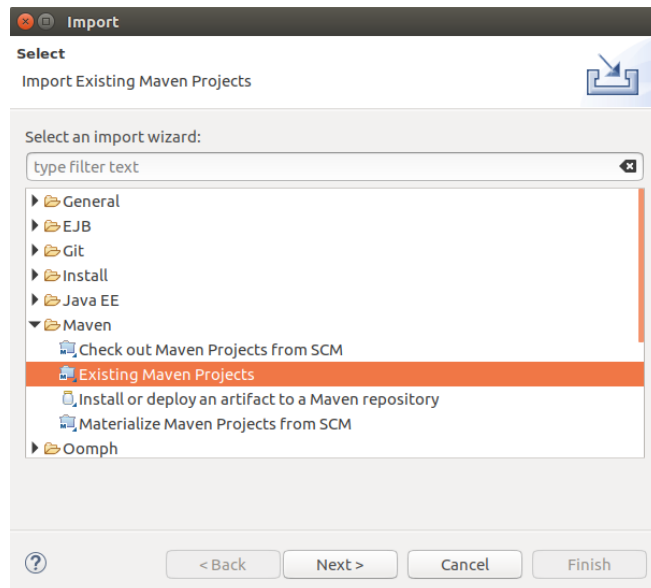
In the Eclipse STS menu bar, click on File and then Import:



In the resulting Import pop-up, select 'Existing Maven Projects':

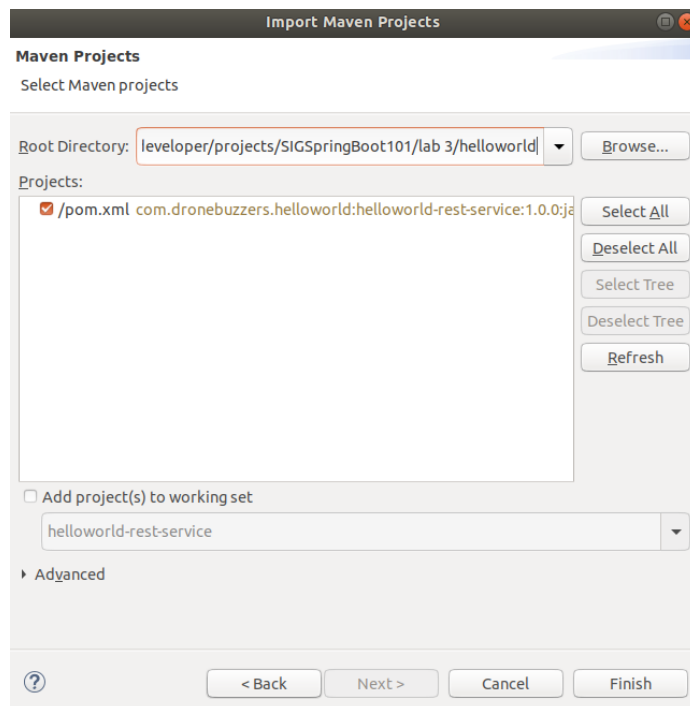


## Spring Boot 101 - intro, demo & hands-on with Java, REST APIs, Containers & Cloud



Click Next and then:

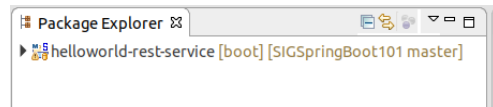
- set the Root Directory: `/home/developer/projects/SIGSpringBoot101/lab 3/helloworld`
- select the pom file



Click Finish and the helloworld-rest-service project should become visible in the Package Explorer:



## Spring Boot 101 - intro, demo & hands on with Java, REST APIs, Containers & Cloud



This is a good moment to take some time to examine all the server side code that SwaggerHub has generated.

It is important to understand that the project that we have generated only has the API implementation: all business logic is missing. Now, we will continue to add the business logic.

The business logic that we will add will be an implementation of the generated interface InventoryApi:

```
InventoryApi.java
/* NOTE: This class is auto generated by the swagger code generator program (1.0.11).
package com.dronebuzzers.helloworld.service.api;

import com.dronebuzzers.helloworld.service.model.InventoryItem;
import io.swagger.codegen.languages.SpringCodegen;
import java.util.Date;

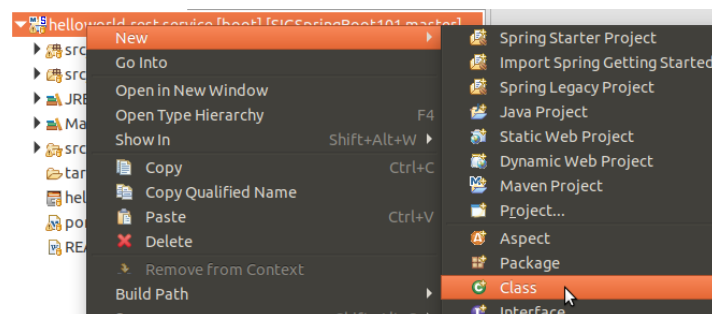
@Api(value = "inventory", description = "the inventory API")
public interface InventoryApi {

    @ApiOperation(value = "adds an inventory item", nickname = "addInventory", notes = "Adds a new inventory item")
    @ApiResponse(code = 201, message = "item created"),
    @ApiResponse(code = 400, message = "invalid input, object invalid"),
    @ApiResponse(code = 409, message = "an existing item already exists")
    @RequestMapping(value = "/inventory", produces = { "application/json" }, consumes = { "application/json" }, method = RequestMethod.POST)
    ResponseEntity<Void> addInventory(@ApiParam(value = "Inventory item to add" ) @Valid @RequestBody InventoryItem inventoryItem);

    @ApiOperation(value = "searches inventory", nickname = "searchInventory", notes = "By pass search criteria")
    @ApiResponse(code = 200, message = "search results matching criteria", response = InventoryItem.class),
    @ApiResponse(code = 400, message = "bad input parameter")
    @RequestMapping(value = "/inventory", produces = { "application/json" }, method = RequestMethod.GET)
    ResponseEntity<List<InventoryItem>> searchInventory(@ApiParam(value = "pass an optional search criteria" ) @Valid @RequestParam(value = "searchCriteria", required = false) String searchCriteria);
}
```

So, we will add a new Java class named InventoryApiDelegateService in the package com.dronebuzzers.helloworld.service.api.impl.

Right-click the project, select New and select Class:





## Spring Boot 101 - intro, demo & handson with Java, REST APIs, Containers & Cloud

---

Complete like shown below:

- Package: com.dronebuzzers.helloworld.service.api.impl
- Name: InventoryApiDelegateService

**New Java Class**

Create a new Java class.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected  
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)  
☐ Constructors from superclass  
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

Copy-paste in the sample code that can be found in file:

```
/home/developer/projects/SIGSpringBoot101/lab 3/input/InventoryApiDelegateService.java
```

That should make it look like shown below:





## Spring Boot 101 - intro, demo & hands on with Java, REST APIs, Containers & Cloud

```
InventoryApiDelegateService.java

package com.dronebuzzers.helloworld.service.api.impl;

import com.dronebuzzers.helloworld.service.api.InventoryApiDelegate;
import com.dronebuzzers.helloworld.service.model.*;
import com.fasterxml.jackson.databind.ObjectMapper;

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

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.List;
import java.util.UUID;

import javax.servlet.http.HttpServletRequest;

/**
 * A delegate to be called by the {@link CaseactivityApiController}.
 * Implement this interface with a {@link org.springframework.stereotype.Service} annotated class.
 */
@javax.annotation.Generated(value = "io.swagger.codegen.languages.SpringCodegen", date = "2018-01-02T11:12:28.4")
@Service
public class InventoryApiDelegateService implements InventoryApiDelegate {

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

    private final ObjectMapper objectMapper;

    private final HttpServletRequest request;
}
```

The code is now completed. It defines a single manufacturer (AMIS) with a single inventory item (SIG Spring Boot 101)

### Step 4: run and test the API

Before actually running the code, have a look at the application.properties file. There are 2 changes that you need to make:

1. Change the server port:  
server.port=8090
2. Change the server contextPath as this will refer to your organization's name:  
server.contextPath=/DroneBuzzers/HelloWorld/1.0.0

```
Package Explorer
helloworld-rest-service [boot] [SIGSpringBoot101 master]
├── src/main/java
│   ├── com.dronebuzzers.helloworld.service.api
│   ├── com.dronebuzzers.helloworld.service.api.impl
│   │   ├── InventoryApiDelegateService.java
│   │   ├── com.dronebuzzers.helloworld.service.config
│   │   └── com.dronebuzzers.helloworld.service.invoker
└── application.properties

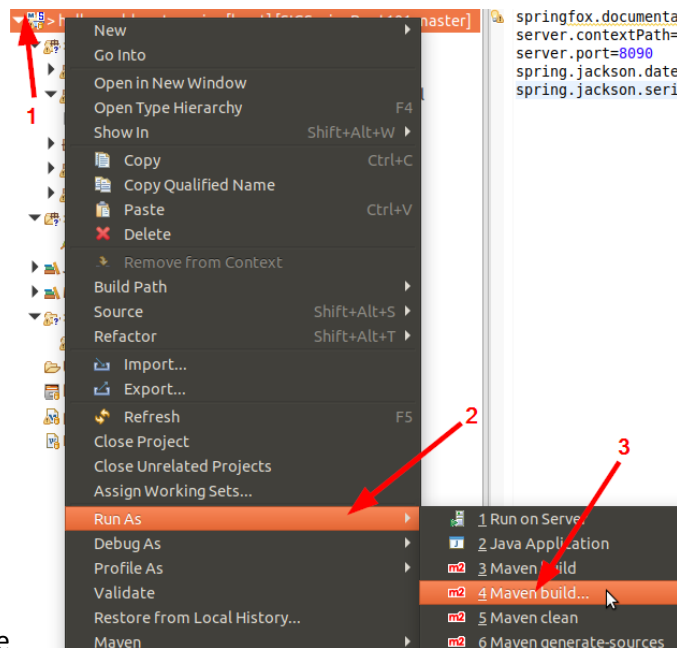
springfox.documentation.swagger.v2.path=/api-docs
server.contextPath=/DroneBuzzers/HelloWorld/1.0.0
server.port=8090
spring.jackson.date-format=com.dronebuzzers.helloworld.service.invoker.RFC3339DateFormat
spring.jackson.serialization.WRITE_DATES_AS_TIMESTAMPS=false
```

We change the above settings so the provided Postman requests will work without changes.

First step is to build the code: right-click the project, click 'Run As' and select the option 'Maven build...':



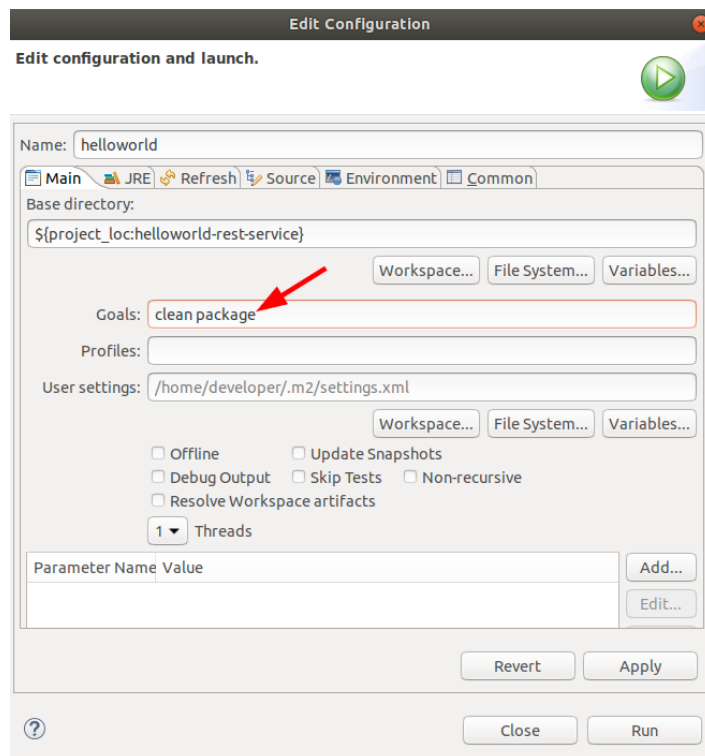
## Spring Boot 101 - intro, demo & hands on with Java, REST APIs, Containers & Cloud



Clean package

The pop-up window appears as shown below. Complete by setting the Goals:

- Goals: clean package



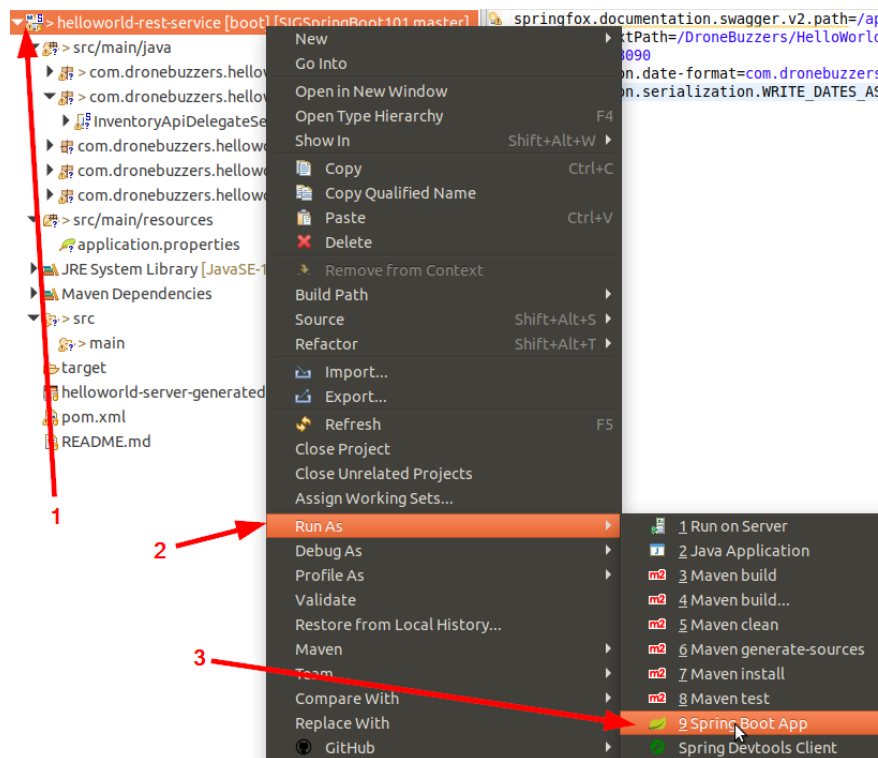


## Spring Boot 101 - intro, demo & hands-on with Java, REST APIs, Containers & Cloud

Complete like shown above and click Run. Check in the console that the code is built successfully:

```
Problems  Javadoc  Declaration  Console  [x]
<terminated> helloworld (2) [Maven Build] /usr/lib/jvm/java-8-openjdk-amd64/bin/java (Mar 11, 2018, 10:26:37 AM)
[INFO] --- spring-boot-maven-plugin:1.5.4.RELEASE:repackage (default) @ helloworld-rest-service ---
[INFO] BUILD SUCCESS
[INFO] Total time: 5.065 s
[INFO] Finished at: 2018-03-11T10:26:43+01:00
[INFO] Final Memory: 26M/207M
```

Now that the code is built, it is time to run it. Right-click the project, click 'Run As' and then 'Spring Boot App':



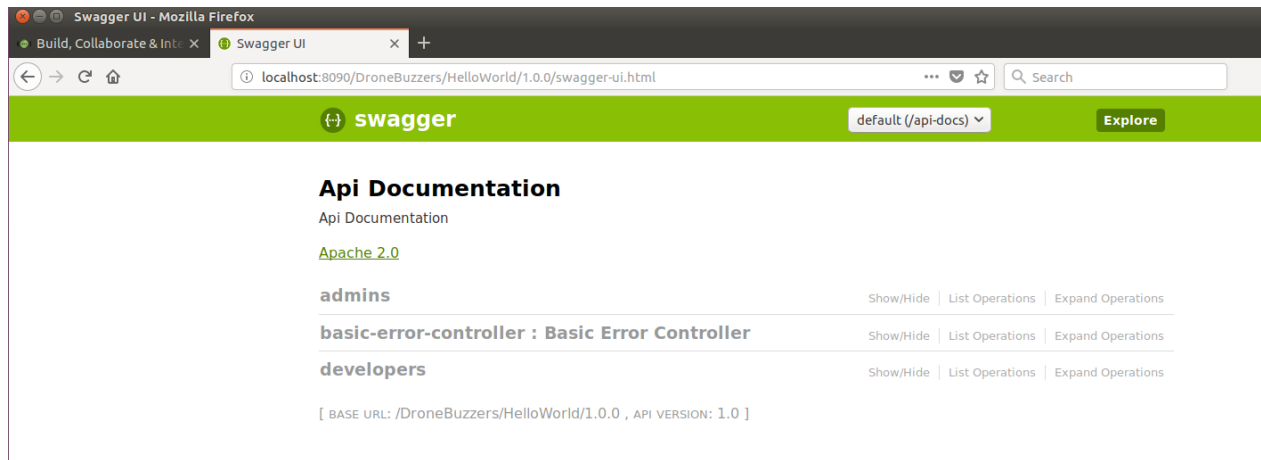
Verify that the code is running by going to url

<http://localhost:8090/DroneBuzzers/HelloWorld/1.0.0/swagger-ui.html> in your browser:

The result should look like:



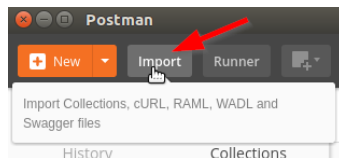
## Spring Boot 101 - intro, demo & hands-on with Java, REST APIs, Containers & Cloud



You can check url <http://localhost:8090/DroneBuzzers/HelloWorld/1.0.0/api-docs> for the raw output...

Now that the code is running, we will use Postman again to test it.

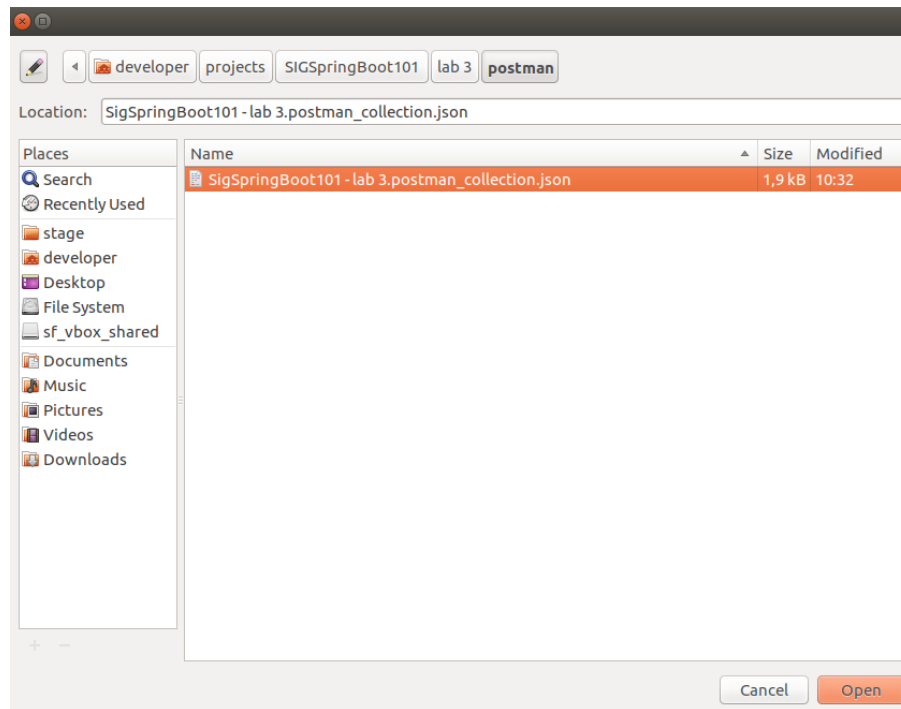
Start Postman  and import the Collection of Postman tests for lab 3:



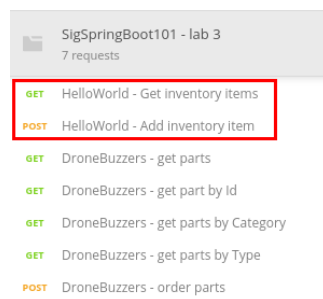
The collection is in the `/home/developer/projects/SIGSpringBoot101/lab 3/postman` directory:



## Spring Boot 101 - intro, demo & handson with Java, REST APIs, Containers & Cloud



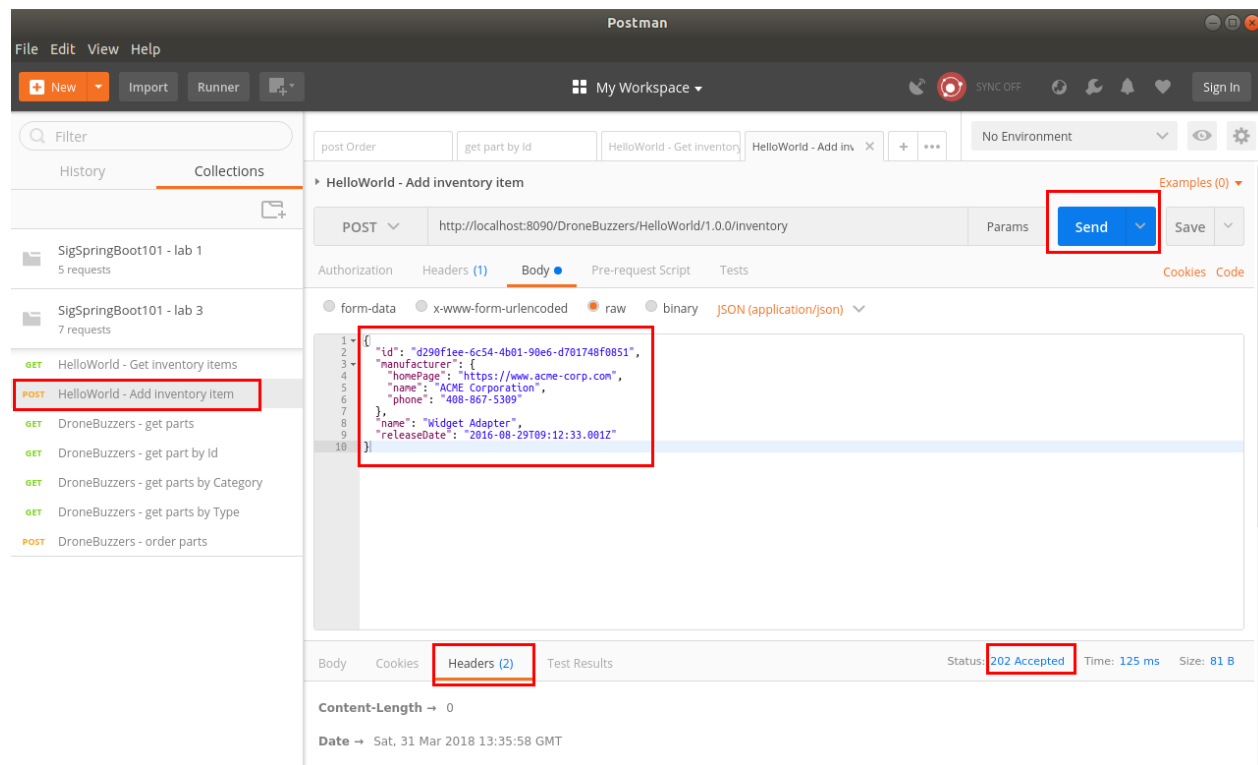
NOTE: the imported collection has 7 requests: in this stage, you should use the first two HelloWorld requests. The other requests will be used later on in this lab.



Next, test both operations:



## Spring Boot 101 - intro, demo & hands-on with Java, REST APIs, Containers & Cloud



### 3. The DroneBuzzers API in SwaggerHub

The HelloWorld API in the previous section was a simple API that illustrated all the steps to develop a REST service in a contract-first style.

The same steps can now be done for the DroneBuzzers API, which is a bit more detailed interface.

Similar to the HelloWorld API, the following steps will be done:

- Step 1: create the API specification in SwaggerHub
- Step 2: generating code in SwaggerHub
- Step 3: import into STS Eclipse and add business logic in the code
- Step 4: run and test the API

The steps should be familiar, so in this section they will be described with less detail..

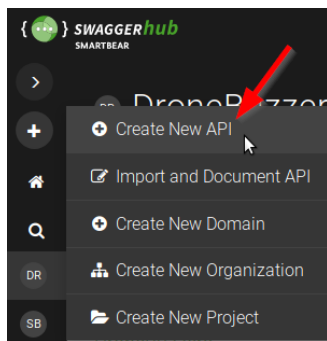


## Spring Boot 101 - intro, demo & hands-on with Java, REST APIs, Containers & Cloud

Should you not want to do the complete exercise, some intermediate results are made available in the `/home/developer/projects/SIGSpringBoot101/lab 3/input` directory. Like this text, intermediate results are marked in a box.

### Step 1: create the API specification in SwaggerHub

In SwaggerHub, create a new API:



Complete the form for the API named Parts as shown below:

Select a Template or create a Blank API

Enter a unique name for your API definition below and select a Template  
Select None for a Blank API.

OpenAPI Version

2.0

Template

Simple API

Name

Parts

Visibility

Public

Owner

DroneBuzzers

Auto Mock API

ON

CANCEL

CREATE API

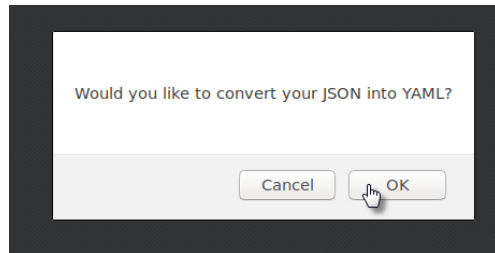
In the editor, replace the contents with the contents of file

```
/home/developer/projects/SIGSpringBoot101/lab 3/input/DroneBuzzers_Parts_1.0.0_swagger.json
```

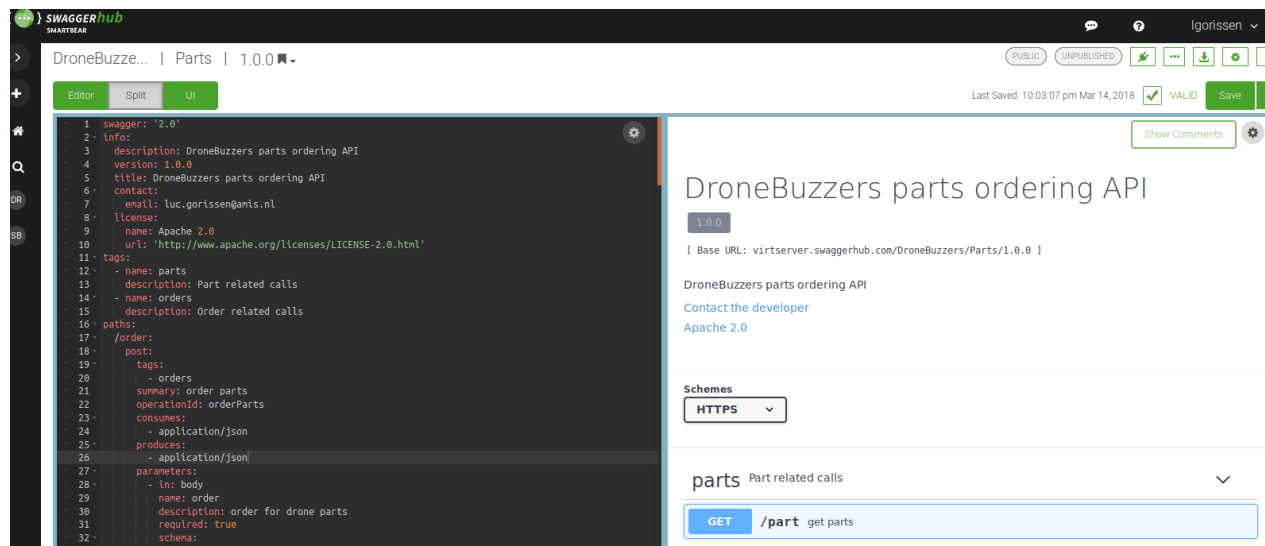
The result in SwaggerHub should look somewhat like below:



## Spring Boot 101 - intro, demo & handson with Java, REST APIs, Containers & Cloud



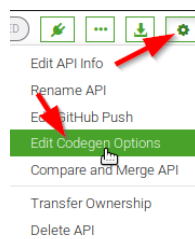
Click OK to let Swagger do the JSON to YAML conversion for our interface definition.



**NOTE:** also in this part of the exercise, the same considerations apply for the organization's name: DroneBuzzers is assumed throughout the screenshots, whereas you will have your own organization's name

### Step 2: generating code in SwaggerHub

Before generating the code for the server side of the DroneBuzzers Parts API, go to the code generation options as shown below:







## Spring Boot 101 - intro, demo & hands-on with Java, REST APIs, Containers & Cloud

In the pop-up window, select spring in the Servers section and then complete the settings as shown in the table below. This is again quite some work, but can't be avoided: Swagger has separate code generation settings for each API. That does make sense as things like the package names are different for each API.

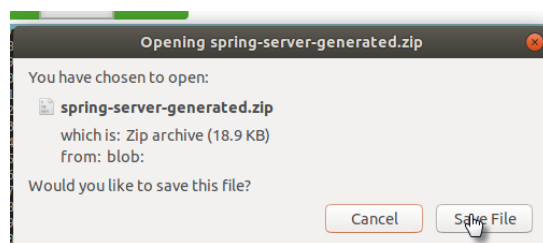
Setting	Value
useTags	<i>not checked</i>
implicitHeaders	<i>not checked</i>
configPackage	com.dronebuzzers.parts.service.config
interfaceOnly	<i>not checked</i>
artifactVersion	1.0.0
sortParamsByRequiredFlag	<i>not checked</i>
useOptional	<i>not checked</i>
singleContentTypes	<i>not checked</i>
sourceFolder	/src/main/java
serializableModel	<i>not checked</i>
artifactDescription	DroneBuzzers Parts
delegatePattern	<i>checked</i>
scmDeveloperConnection	
apiPackage	com.dronebuzzers. parts.service.api
licenseName	
invokerPackage	com.dronebuzzers. parts.service.invoker
dateLibrary	
artifactId	dronebuzzers-rest-service
licenseUrl	
swaggerDocketConfig	<i>checked</i>
useBeanValidation	<i>not checked</i>
withXml	<i>not checked</i>
responseWrapper	
developerEmail	<insert your own e-mail address>
developerOrganizationUrl	<a href="https://www.amis.nl">https://www.amis.nl</a>
fullJavaUtil	<i>not checked</i>
bigDecimalAsString	<i>not checked</i>
ensureUniquerParams	<i>not checked</i>
basePackage	com.dronebuzzers.parts.service
developerName	<insert your own name>
allowUnicodeIdentifiers	<i>not checked</i>
java8	<i>Checked</i>
title	DroneBuzzers Parts
localVariablePrefix	<i>not checked</i>
groupId	com.dronebuzzers. parts
library	Spring-boot Server application using the SpringFox integration



## Spring Boot 101 - intro, demo & hands-on with Java, REST APIs, Containers & Cloud

scmConnection	
hideGenerationTimestamp	
async	not checked
modelPackage	com.dronebuzzers.parts.service.model
developerOrganization	AMIS
artifactUrl	

Now, the server side code can be generated. Download the zip file with the generated code for the server side:



Should you want to skip this step: the generated code is also present in:

```
/home/developer/projects/SIGSpringBoot101/lab 3/input/dronebuzzers-server-generated.zip
```

### Step 3: import into eclipse and add business logic in the code

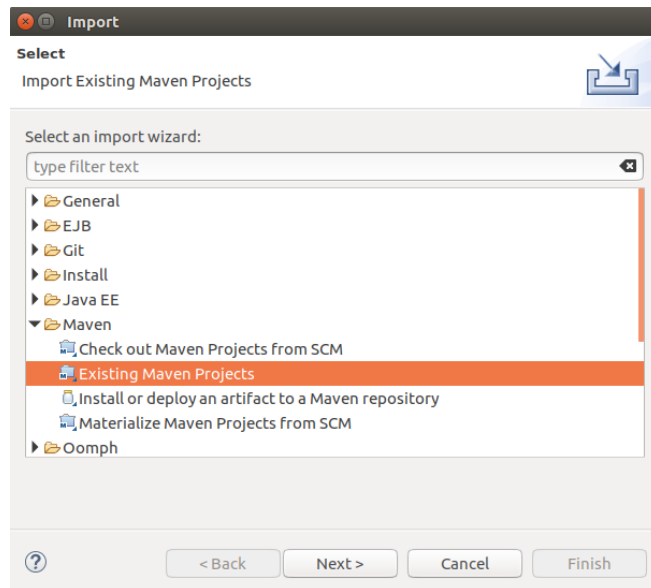
The generated code can now be unzipped to directory `../lab 3/dronebuzzers`:

```
developer@developer-VirtualBox: ~/projects/SIGSpringBoot101/lab 3/dronebuzzers
developer@developer-VirtualBox:~/projects/SIGSpringBoot101/lab 3/dronebuzzers$ pwd
/home/developer/projects/SIGSpringBoot101/lab 3/dronebuzzers
developer@developer-VirtualBox:~/projects/SIGSpringBoot101/lab 3/dronebuzzers$ cp ../input/dronebuzzers-server-generated.zip .
developer@developer-VirtualBox:~/projects/SIGSpringBoot101/lab 3/dronebuzzers$ unzip dronebuzzers-server-generated.zip
Archive:  dronebuzzers-server-generated.zip
  inflating: src/main/java/com/dronebuzzers/parts/service/model/Parts.java
  inflating: src/main/java/com/dronebuzzers/parts/service/model/OrderSummary.java
  inflating: src/main/java/com/dronebuzzers/parts/service/model/Order.java
  inflating: src/main/java/com/dronebuzzers/parts/service/model/OrderSummaryLine.java
  inflating: src/main/java/com/dronebuzzers/parts/service/model/Part.java
  inflating: src/main/java/com/dronebuzzers/parts/service/model/OrderLine.java
  inflating: src/main/java/com/dronebuzzers/parts/service/invoke/RFC3339DateFormat.java
  inflating: src/main/java/com/dronebuzzers/parts/service/invoke/Swagger2SpringBoot.java
  inflating: src/main/java/com/dronebuzzers/parts/service/config/HomeController.java
  inflating: src/main/java/com/dronebuzzers/parts/service/config/SwaggerDocumentationConfig.java
  inflating: src/main/java/com/dronebuzzers/parts/service/api/OrderApiController.java
  inflating: src/main/java/com/dronebuzzers/parts/service/api/OrderApi.java
  inflating: src/main/java/com/dronebuzzers/parts/service/api/ApiException.java
  inflating: src/main/java/com/dronebuzzers/parts/service/api/PartApiDelegate.java
  inflating: src/main/java/com/dronebuzzers/parts/service/api/PartApiController.java
  inflating: src/main/java/com/dronebuzzers/parts/service/api/NotFoundException.java
  inflating: src/main/java/com/dronebuzzers/parts/service/api/OrderApiDelegate.java
  inflating: src/main/java/com/dronebuzzers/parts/service/api/PartApi.java
  inflating: src/main/java/com/dronebuzzers/parts/service/api/ApiOriginFilter.java
  inflating: src/main/java/com/dronebuzzers/parts/service/api/ApiResponseMessage.java
  inflating: src/main/resources/application.properties
  inflating: README.md
  inflating: pom.xml
  inflating: .swagger-codegen/VERSION
developer@developer-VirtualBox:~/projects/SIGSpringBoot101/lab 3/dronebuzzers$
```



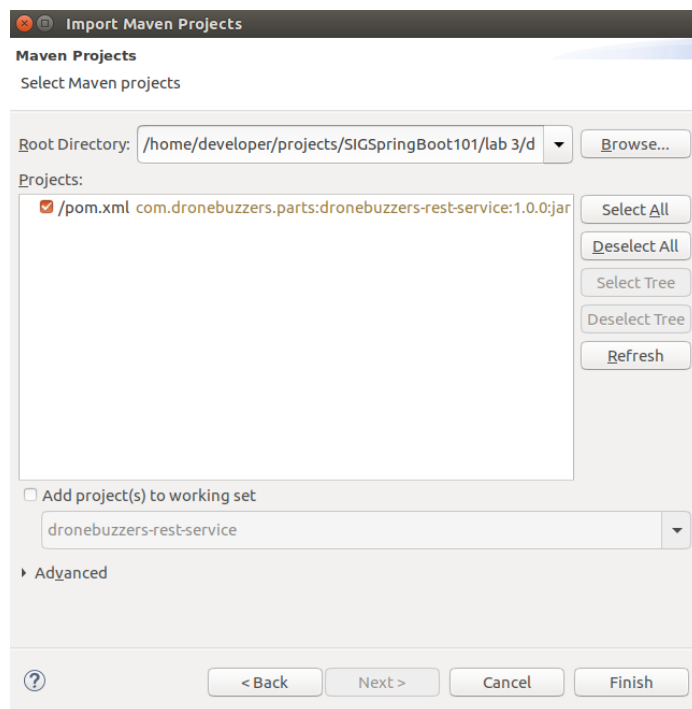
## Spring Boot 101 - intro, demo & hands-on with Java, REST APIs, Containers & Cloud

Now, start Eclipse and then import the maven project:



Select the pom file from your project directory:

/home/developer/projects/SIGSpringBoot101/lab 3/dronebuzzers/pom.xml



Now, we need to copy the business logic into the code: the impl directory in the input directory



## Spring Boot 101 - intro, demo & handson with Java, REST APIs, Containers & Cloud

lab 3/dronebuzzers/impl

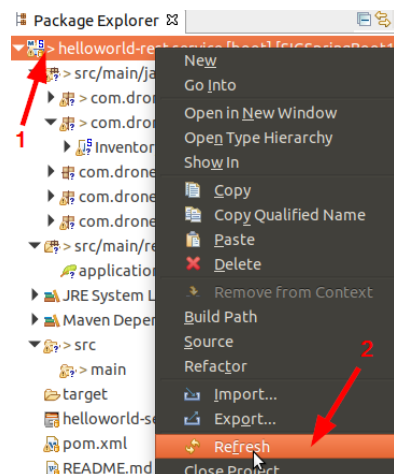
must now be copied to the right directory in the project:

lab 3/dronebuzzers/src/main/java/com/dronebuzzers/parts/service/api

Illustrated in the figure below:

```
developer@developer-VirtualBox: ~/projects/SIGSpringBoot101/lab 3/dronebuzzers/src/main/java/com/dronebuzzers/parts/service/api
developer@developer-VirtualBox:~/projects/SIGSpringBoot101/lab 3/dronebuzzers/src/main/java/com/dronebuzzers/parts/service/api$ pwd
/home/developer/projects/SIGSpringBoot101/lab 3/dronebuzzers/src/main/java/com/dronebuzzers/parts/service/api
developer@developer-VirtualBox:~/projects/SIGSpringBoot101/lab 3/dronebuzzers/src/main/java/com/dronebuzzers/parts/service/api$ cp -R /home/develop
eloper/projects/SIGSpringBoot101/lab\ 3/input/impl/ .
developer@developer-VirtualBox:~/projects/SIGSpringBoot101/lab 3/dronebuzzers/src/main/java/com/dronebuzzers/parts/service/api$ ls -al impl/
total 56
drwxr-xr-x 2 developer developer 4096 mrt 14 22:25 .
drwxr-xr-x 3 developer developer 4096 mrt 14 22:25 ..
-rw-r--r-- 1 developer developer 1284 mrt 14 22:25 MockedPartsDAO.java
-rw-r--r-- 1 developer developer 3497 mrt 14 22:25 MockedPartsStore.java
-rw-r--r-- 1 developer developer 4418 mrt 14 22:25 OrderApiDelegateService.java
-rw-r--r-- 1 developer developer 1085 mrt 14 22:25 Order.java
-rw-r--r-- 1 developer developer 560 mrt 14 22:25 OrderLine.java
-rw-r--r-- 1 developer developer 3079 mrt 14 22:25 OrderSummary.java
-rw-r--r-- 1 developer developer 591 mrt 14 22:25 OrderSummaryLine.java
-rw-r--r-- 1 developer developer 3847 mrt 14 22:25 PartApiDelegateService.java
-rw-r--r-- 1 developer developer 1280 mrt 14 22:25 Part.java
-rw-r--r-- 1 developer developer 497 mrt 14 22:25 PartsDAO.java
-rw-r--r-- 1 developer developer 301 mrt 14 22:25 Parts.java
developer@developer-VirtualBox:~/projects/SIGSpringBoot101/lab 3/dronebuzzers/src/main/java/com/dronebuzzers/parts/service/api$
```

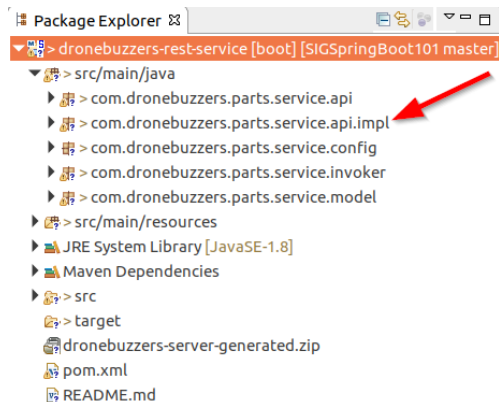
Right-click the project in eclipse and clicking Refresh should make the impl package visible:



Look at the Package Explorer



## Spring Boot 101 - intro, demo & hands-on with Java, REST APIs, Containers & Cloud



The completed code is also available in:

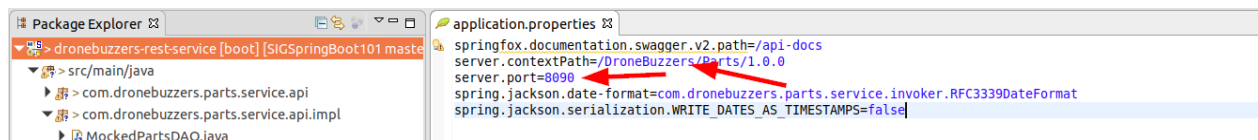
```
/home/developer/projects/SIGSpringBoot101/lab_3/dronebuzzers-completed
```

### Step 4: run and test the API

**Note:** if you have the HelloWorld project still running from earlier in this lab, then now is it a good time to stop it.

Before actually running the code, have a look at the application.properties file. There are 2 changes that you need to make:

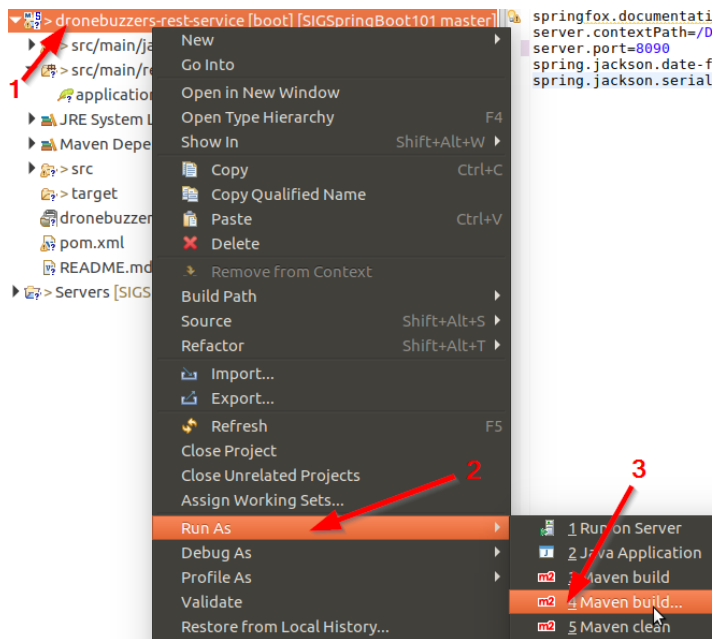
1. Change the server port:  
server.port=8090
2. Change the server contextPath as this will refer to your organization's name:  
server.contextPath=/DroneBuzzers/HelloWorld/1.0.0



To build the code: right-click the project, click 'Run As' and select the option 'Maven build...':

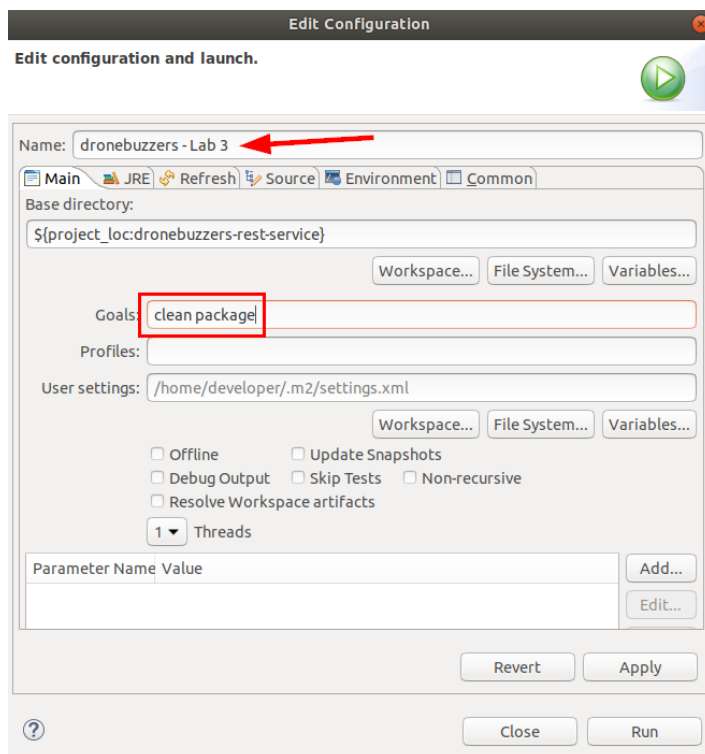


## Spring Boot 101 - intro, demo & hands on with Java, REST APIs, Containers & Cloud



The pop-up window will be shown. Complete as shown below:

- Name: dronebuzzers – Lab 3
- Goals: clean package



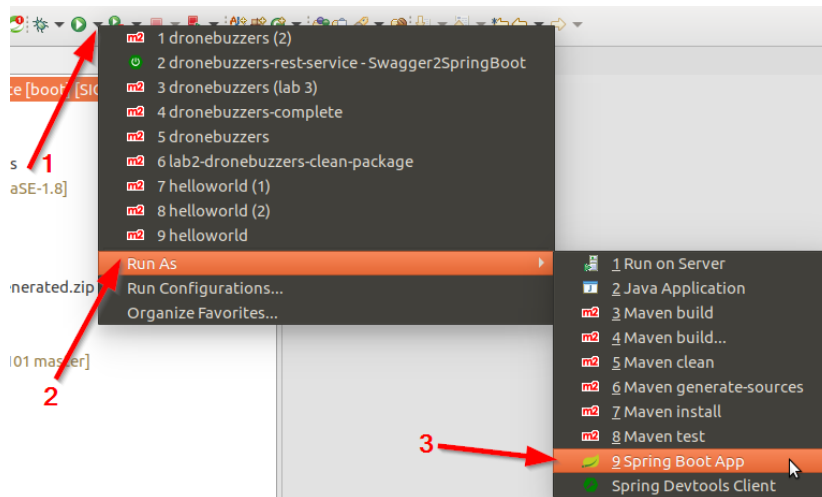


## Spring Boot 101 - intro, demo & hands on with Java, REST APIs, Containers & Cloud

Complete like shown above and click Run.

Check in the console that the code is built successfully:

Now that the code is built, it is time to run it:



Verify that the code is running by going to url

<http://localhost:8090/DroneBuzzers/Parts/1.0.0/swagger-ui.html> in your browser:

**Api Documentation**

Api Documentation

Apache 2.0

**basic-error-controller : Basic Error Controller**

Show/Hide | List Operations | Expand Operations

**orders**

Show/Hide | List Operations | Expand Operations

POST /order [order parts](#)

**parts**

Show/Hide | List Operations | Expand Operations

GET /part [get parts](#)

GET /part/category/{category} [get parts by category](#)

GET /part/type/{type} [get parts by type](#)

GET /part/{id} [get part details](#)

[ BASE URL: /DroneBuzzers/Parts/1.0.0 , API VERSION: 1.0 ]



## Spring Boot 101 - intro, demo & handson with Java, REST APIs, Containers & Cloud

---

For testing, start Postman .

If you have not done this during the HelloWorld example: import the Collection of Postman tests for lab 3 from location

`/home/developer/projects/SIGSpringBoot101/lab 3/postman`

Test the interface with the last 5 operations:

