IIP-Ecosphere

Next Level Ecosphere for Intelligent Industrial Production

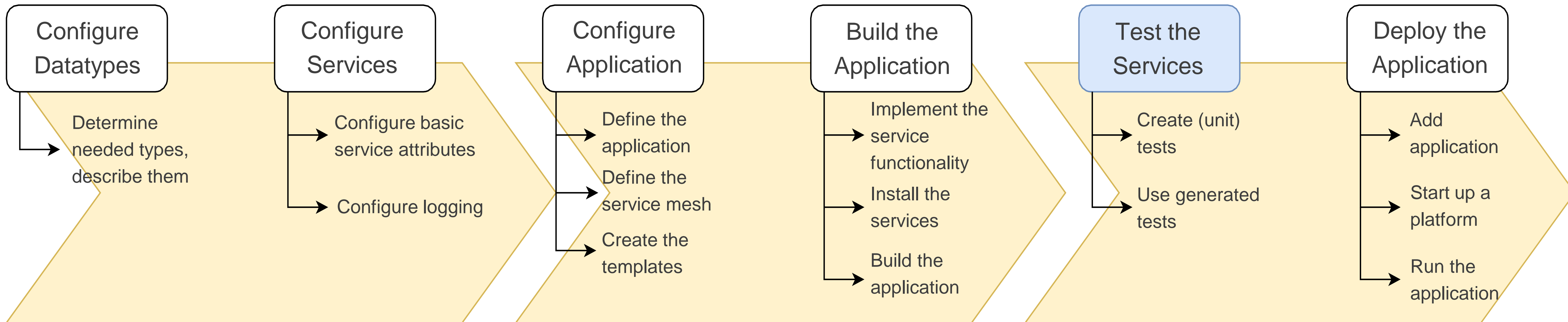
Service Integration: How to Test the Application

Gefördert durch:

Bundesministerium für Wirtschaft und Klimaschutz

IIP-Ecosphere Platform

Test the Services



© IIP-Ecosphere Konsortium

Table of Contents

- **Prerequisites**
- Introduction
- Generated Artifacts
- Creating Unit Tests
- Utilising Generated Tests

© IIP-Ecosphere Konsortium

Prerequisites

- Required:
  - Installed the platform and its dependencies or the development container
  - Installed the IDE for IIP-Ecosphere Platform (provided Eclipse Version)
  - How to configure the datatypes
  - How to configure the services
  - How to configure the application
  - How to build an application
- Optional:
  - Introduction to code generation

© IIP-Ecosphere Konsortium

Table of Contents

- Prerequisites
- **Introduction**
- Generated Artifacts
- Creating Unit Tests
- Utilising Generated Tests

© IIP-Ecosphere Konsortium

Levels of Testing

- Service Level (Before building the application)
  - Isolated services/ methods tested for their supposed functionality
  - Can be Unit testing, can be a simple main method to call the functions
  - Done before finishing the build process
- Services integrated (After building the application)
  - Tests the services and their respective input and output connection
  - Connections will be mocked as this will be done separately for each service
- App (After building the application)
  - The whole app gets tested in a simulated platform deployment
- Platform deployment (After building the application)
  - Actually starting a platform and loading the app
  - Should not fail if no test failed on the way

© IIP-Ecosphere Konsortium

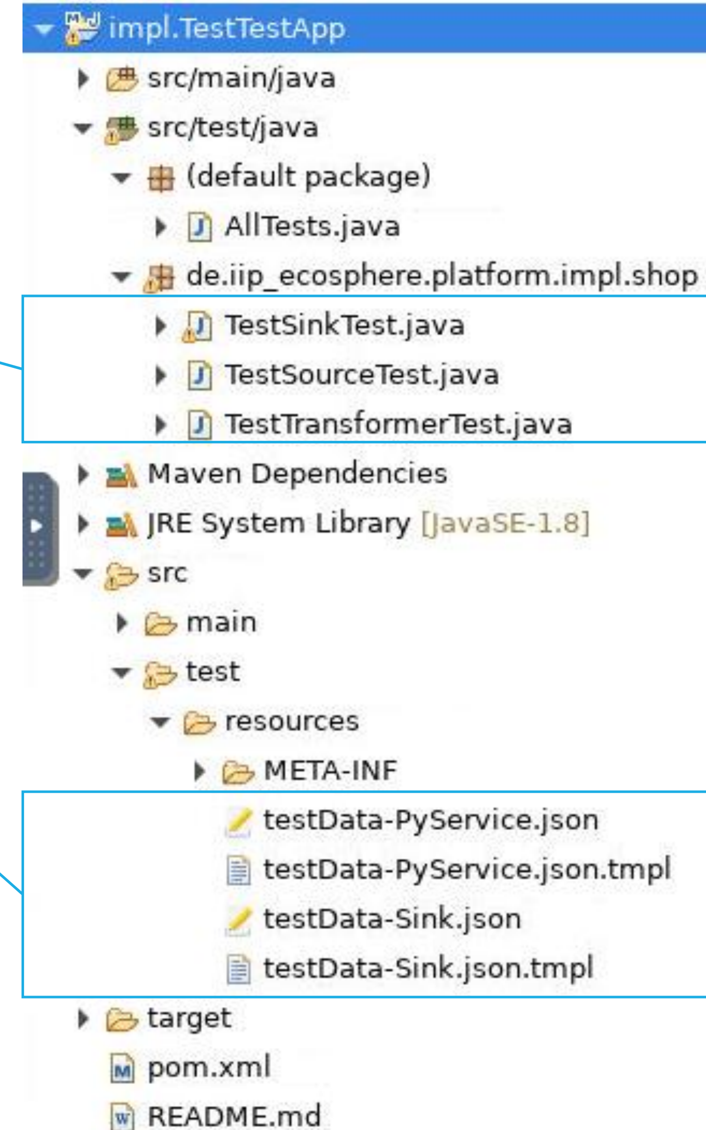
Table of Contents

- Prerequisites
- Introduction
- **Generated Artifacts**
- Creating Unit Tests
- Utilising Generated Tests

© IIP-Ecosphere Konsortium

Generated Artifacts

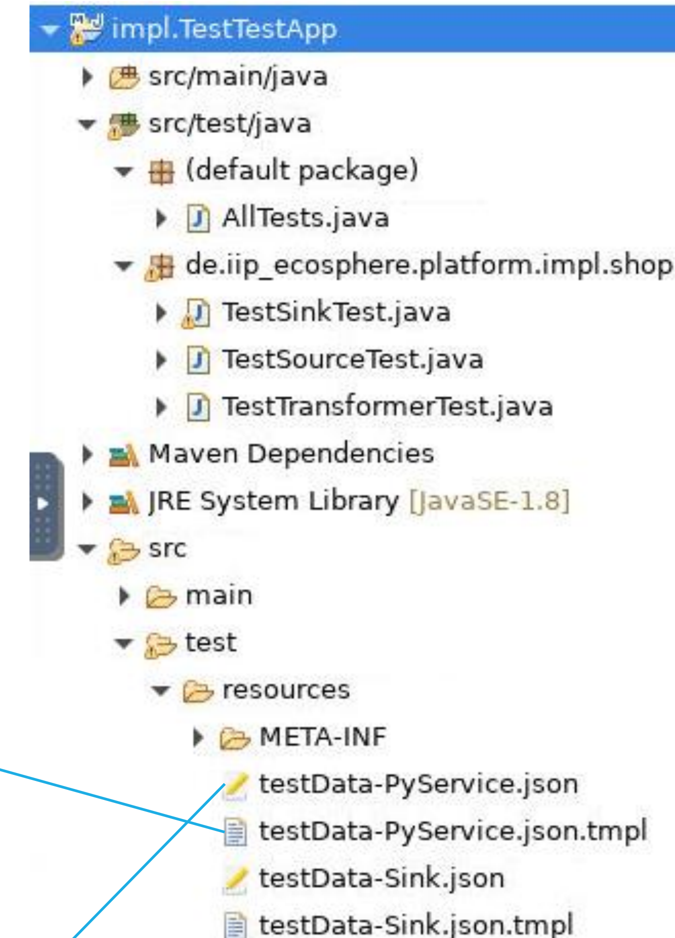
- Unit-Test Templates reduce the work to implanting test code
  - The structure of the tests is readily prepared
- Generated tests for the application with connectors
  - Allow to test services or applications in mocked environments before deployment
- Generates template files for test data to be fed into tests
  - Used by Python Unit-Tests and generated tests



© IIP-Ecosphere Konsortium

Generated testData Files

- Utilised in generated tests and Python Unit tests
- Generated as "*testData-<ServiceName>.json.tmpl*"
- Need to be renamed from "*.json.tmpl*" to "*.json*"
  - Remove the instructions in the files
  - Add values into the files




```
This file represents service/connector test input data. To use it, rename this file from .json.tmpl to .json and
delete these trailing lines until the JSON structure and add your own input data structures following the
example JSON. A structure contains at least one input data item named according to the type as configured for
the service/connector in the configuration model. Multiple input data items of different types are ingested in
the same step. The sub-structure of the input data item corresponds to the respective data structure in the
configuration model. No specific values are given rather than "m" for mandatory and "o" for optiona. Further,
meta attributes on top-level allow controlling the input behavior. Currently we support:
  - $period: p ms to wait until ingestion of that line
  - $repeats: r repetitions of that line; 0 = emit once, no repetitions, positive = number of repetitions,
    negative = endless

{"inData": {"intExample": m, "floatExample": m, "stringExample": m, "doubleExample": m}, "$period": p, "$repeats": r}
```

```
{"inData": {"intExample": 1, "floatExample": 2, "stringExample": "WORD", "doubleExample": 4}, "$period": 300, "$repeats": 1}
```

period defines the time to wait bevor the test is started

repeats defines how often the defined data point is used

© IIP-Ecosphere Konsortium

Table of Contents

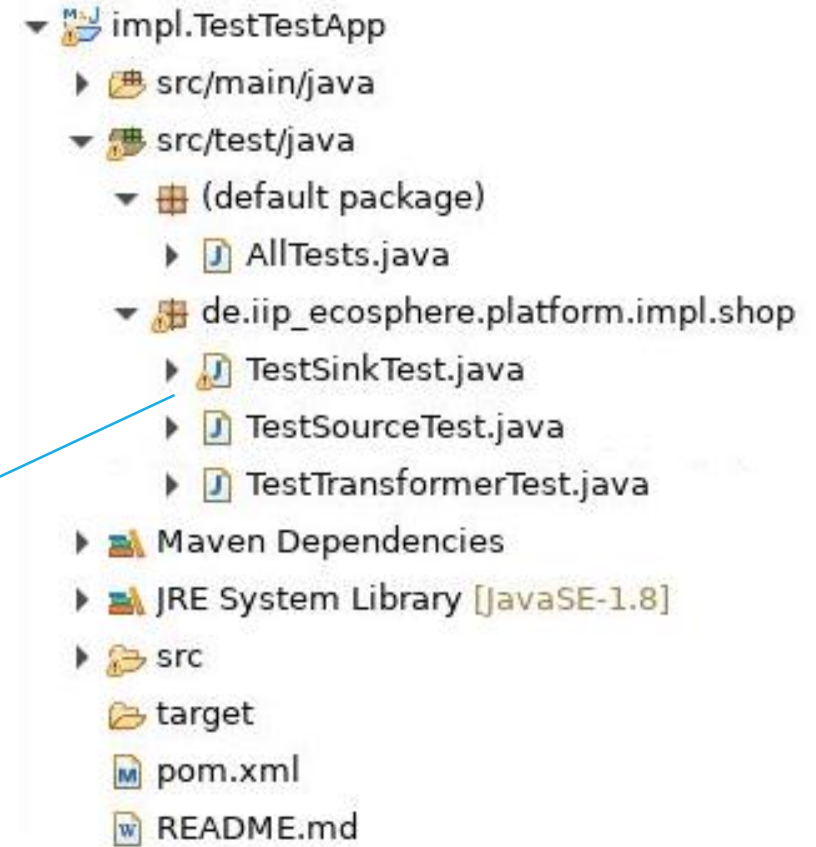
- Prerequisites
- Introduction
- Generated Artifacts
- **Creating Unit Tests**
- Utilising Generated Tests

© IIP-Ecosphere Konsortium

IIP-Ecosphere

Creating Unit Tests

- Templates reduce the work to implanting the actual test code
- Edit the method annotated by “*@Test*”
  - Add the functionality you want to test
  - Add assert methods e.g. “*Assert.assertTrue()*” to confirm the functionality

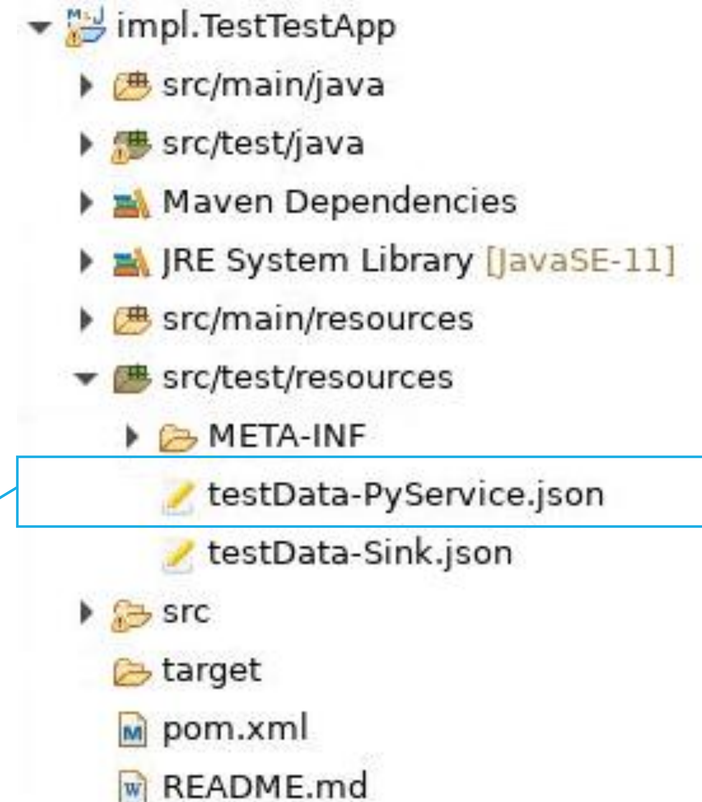



```java
public class TestSinkTest {

    private TestSink service;

    /**
     * Tests processing a data item of type "OutData" asynchronously.
     */
    @Test
    public void testProcessOutData() {
        OutData data = new OutDataImpl();
        data.setResult(2);
        data.setStringExample("Testing");
        service.processOutData(data);
        // no direct output for a sink. may be it's generating a log, a file, etc. to assert
    }
```


© IIP-Ecosphere Konsortium

Creating Unit Tests Python (1)

- The basic setup of the Unit Test is prepared
  - The data is taken from the testData file named after the service
  - Data can also be added manually in the test itself



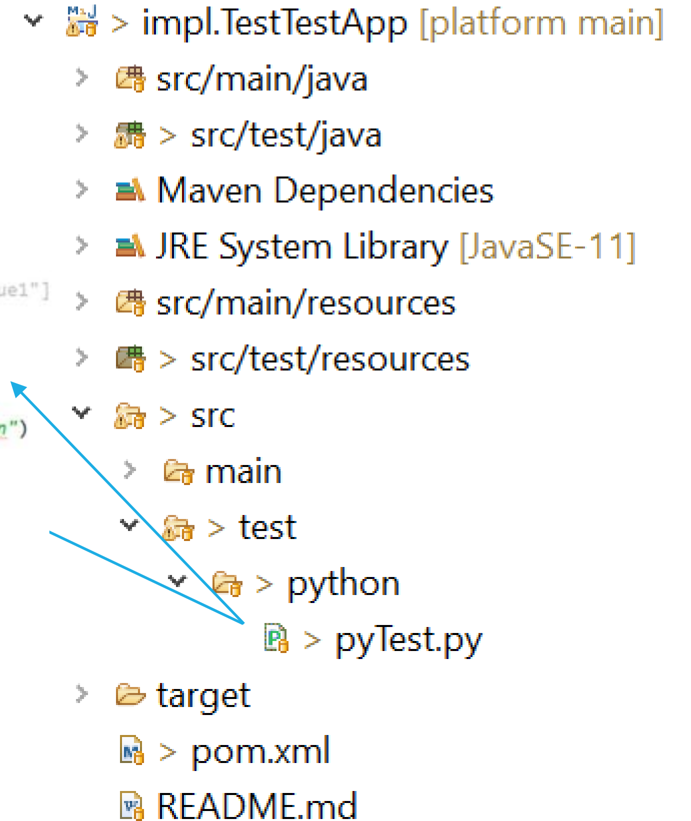
{"inData": {"intExample": 1, "floatExample": 2, "stringExample": "WORD", "doubleExample": 4}, "$period": 300, "$repeats": 1}

© IIP-Ecosphere Konsortium

Creating Unit Tests Python (2)

- The data is taken from generated `.json` files
- Create an instance of your input datatype and put the values from the json in it
- Create an instance of your Service and feed your data in the method you want to test

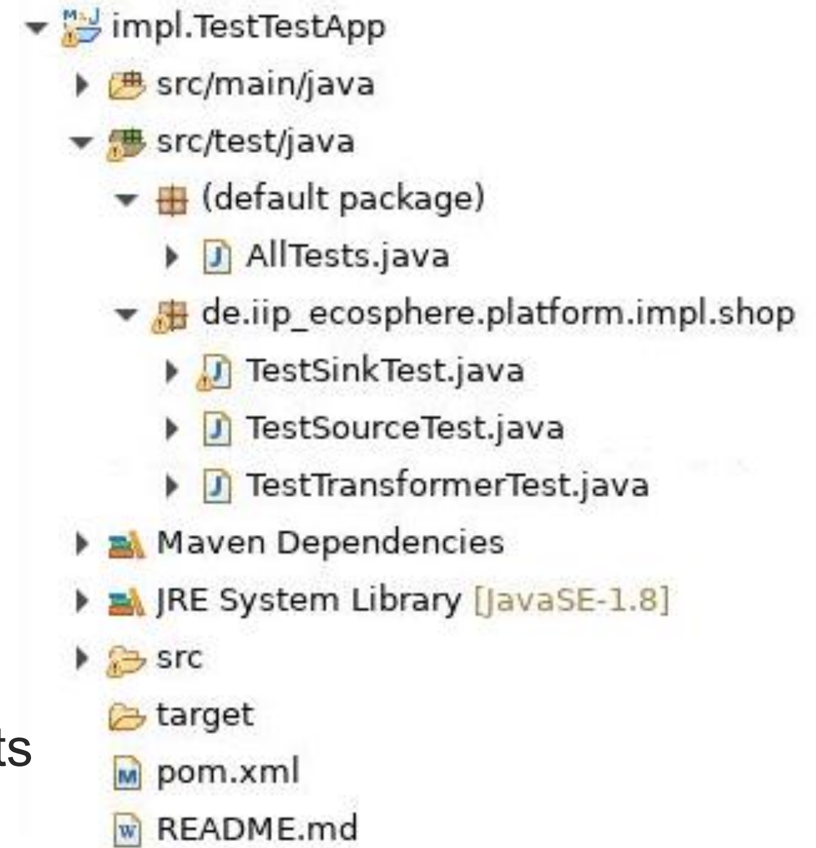
```python
class TestPyServive(unittest.TestCase):

    def test_inputTest(self):

        #Change name to correct .json, access rawData for values i.e. rawData["InputType"]["Value1"]
        rawData = ""
        try :
            with open ("../resources/testData-PyService.json", "r") as f:
                rawData = json.load(f)
        except:
            print("You need to edit the template files in /src/test/resources to be correct json")
```



© IIP-Ecosphere Konsortium

Running Unit Tests

- Manual
  - Import the project separately into Eclipse
  - Run a single Java JUnit Test e.g. “*TestSinkTest.java*”
  - Run all Java tests by running the “*AllTests.java*”
  - Run a single Python Unit Test with “*mvn test -Dpython.test.test=<name>*”
  - Run all Unit Tests including Python with “*mvn test*”
- On install
  - When running “*mvn install*” in the template project the tests are executed



© IIP-Ecosphere Konsortium

Output of Unit Tests

- Running “*mvn test*”

```
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running de.iip_ecosphere.platform.impl.shop.TestSinkTest
Output: 0.0
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.015 s - in de.iip_ecosphere.platform.impl.shop.TestSinkTest
[INFO] Running de.iip_ecosphere.platform.impl.shop.TestSourceTest
14:22:21.696 [main] INFO  d.i.p.s.resources.ResourceLoader - Registered resource resolver Classloader
14:22:21.707 [main] INFO  d.i.p.s.resources.ResourceLoader - Registered resource resolver Spring BOOT-INF/classes resolver
14:22:21.723 [main] INFO  d.i.p.s.resources.ResourceLoader - Loading resource 'transformed_test_data.csv' via Classloader
>>>NewInputImpl[airTemp=298.5,procTemp=309.5,rotSpe=1483,toolWear=92,torq=44.8,type=1]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.369 s - in de.iip_ecosphere.platform.impl.shop.TestSourceTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0
```


© IIP-Ecosphere Konsortium

Table of Contents

- Prerequisites
- Introduction
- Generated Artifacts
- Creating Unit Tests
- **Utilising Generated Tests**

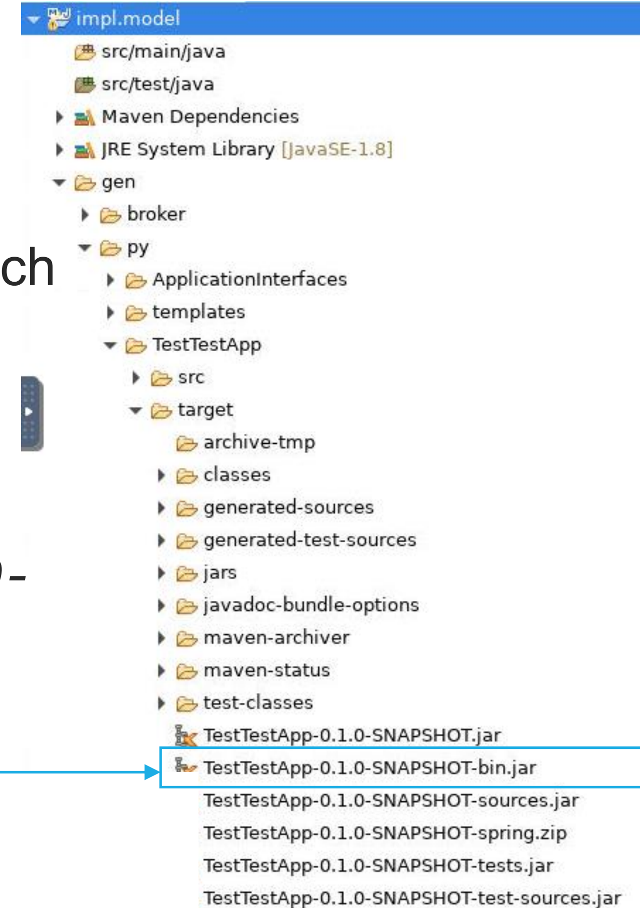
Using generated Tests (1)

- The templates also create basic tests to confirm functionally of the services in the context of the platform
  - They are only possible AFTER the application is build thus unit testing is still valuable
  - Each service which is taking in data get a generated .json as mentioned before
    - The tests are run via *“mvn exec:java@<serviceTestName>”*
      - The names can be found in the *pom* of the services

```
<execution>
    <id>PyServiceTest</id>
    <configuration>
        <mainClass>iip.nodes.PyServiceTest</mainClass>
        <classpathFilenameExclusions>transport.amqp-0.5.0-SNAPSHOT.jar</classpathF
        <classpathScope>test</classpathScope>
    </configuration>
</execution>
<execution>
    <id>SinkTest</id>
    <configuration>
        <mainClass>iip.nodes.SinkTest</mainClass>
        <classpathFilenameExclusions>transport.amqp-0.5.0-SNAPSHOT.jar</classpathF
        <classpathScope>test</classpathScope>
    </configuration>
</execution>
```


© IIP-Ecosphere Konsortium

Using generated Tests (2)



- We can test the whole App before deploying it to the platform
- Add the `-bin.jar` to the execution of "*App*" in the template project which you imported
- Location of the needed file
- Example: *../impl.model/gen/py/TestTestApp/target/TestTestApp-0.1.0-SNAPSHOT-bin.jar*
- Run: "*mvn exec:java@App*" for the test

```
<execution>
    <id>App</id>
    <configuration>
        <mainClass>de.iip_ecosphere.platform.examples.SpringStartup</mainClass>
        <arguments>
            <argument>&lt;put relative location of integrated, generated application here&gt;</argument>
        </arguments>
        <classpathScope>test</classpathScope>
    </configuration>
</execution>
```


© IIP-Ecosphere Konsortium

Using generated Tests (3)

- Tested function (example SinkTest)

```
@Override
public void processNewOutput(NewOutput data) {
    System.out.println("Output: " + data.getResult());
}
```

- Tested output

```
14:30:09.566 [iip.nodes.SinkTest.main()] INFO  d.i.p.s.e.DataMapper$BaseMappingConsumer - Test data: SinkTest.DataUnit[newOutput=NewOutputImpl[result=3.0]]
received in Sink: NewOutputImpl[result=3.0]
Output: 3.0
received in Sink: NewOutputImpl[result=3.0]
Output: 3.0
14:30:10.424 [iip.nodes.SinkTest.main()] INFO  d.i.p.s.e.DataMapper$BaseMappingConsumer - Test data: SinkTest.DataUnit[newOutput=NewOutputImpl[result=3.0]]
received in Sink: NewOutputImpl[result=3.0]
Output: 3.0
received in Sink: NewOutputImpl[result=3.0]
```

From: *TraceKind::SYSOUT* in .ivml

© IIP-Ecosphere Konsortium

Summary

- What we learned
    - How to write our own unit tests for Python and for Java
    - How to run our own unit test for Python and for Java
    - How use generated tests for each service
    - How to use generated tests for the whole app
- How to go on
    - Deploying the application to the platform

© IIP-Ecosphere Konsortium