Lab 8

Part 1

First start zipkin with the following command:

docker run -d -p 9411:9411 openzipkin/zipkin

In the API Gateway, ProductService and the StockService that you wrote in lab 7 add the following libraries

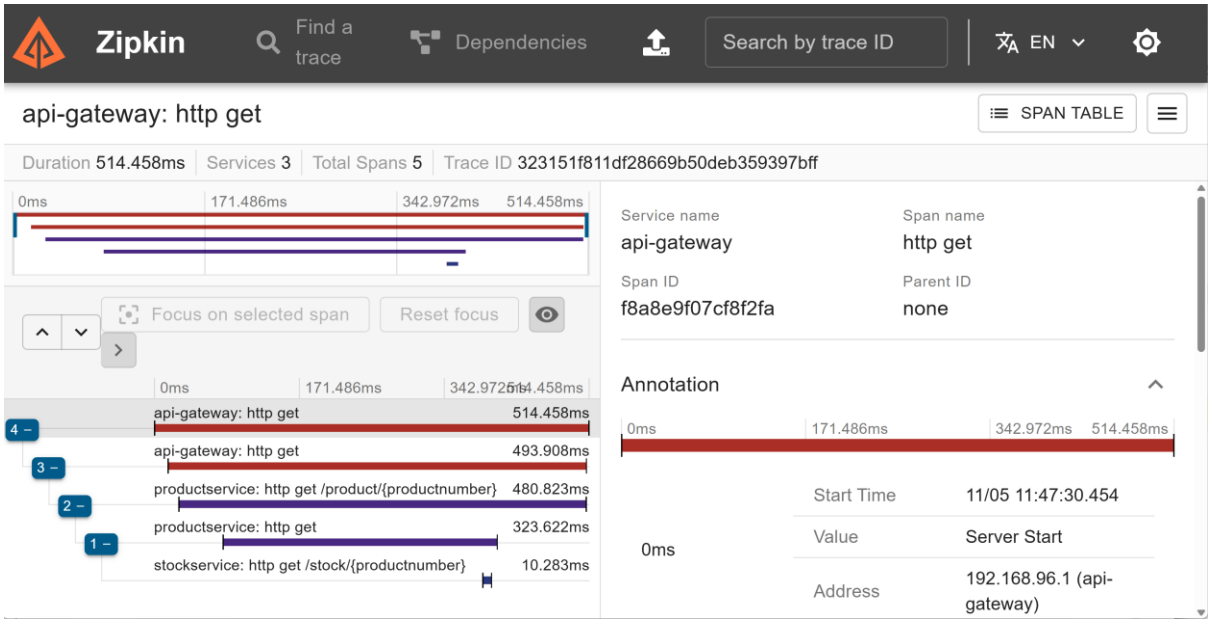
```
<dependency>
   <groupId>io.micrometer</groupId>
   <artifactId>micrometer-tracing-bridge-otel</artifactId>
</dependency>
<dependency>
   <groupId>io.github.openfeign</groupId>
   <artifactId>feign-micrometer</artifactId>
   <version>13.6</version>
</dependency>
<dependency>
   <groupId>io.opentelemetry</groupId>
   <artifactId>opentelemetry-exporter-zipkin</artifactId>
</dependency>
```

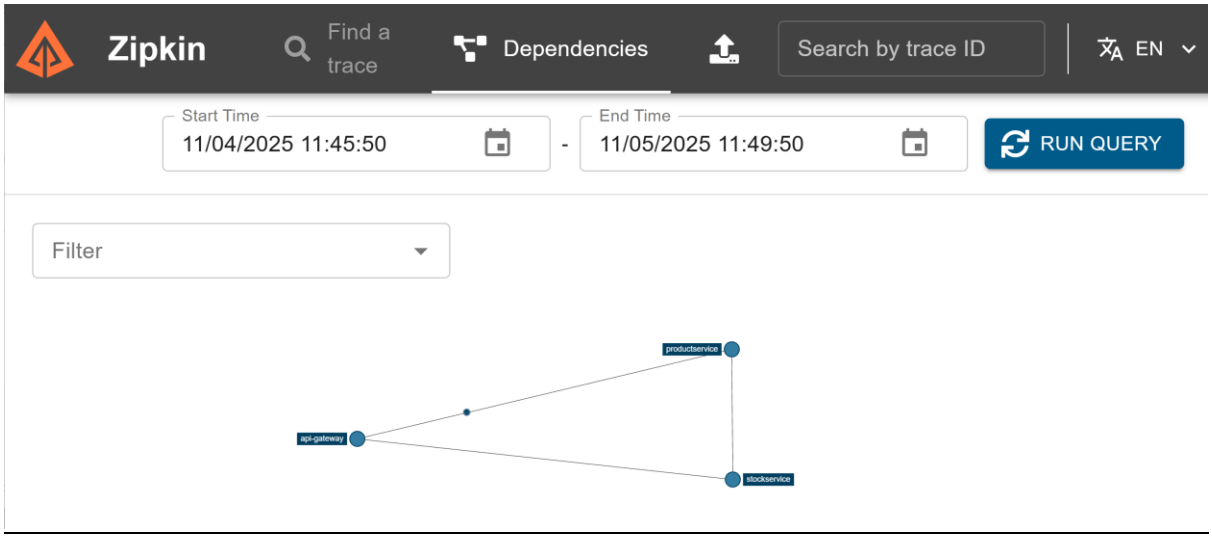
Then add the necessary configuration in **application.yml**

Run all applications

Then open the zipkin console on http://localhost:9411/zipkin

You can click the search button, then see the traces between the services, and you can see the dependencies between the services.





Part 2

Add the circuit breaker to the remote call from the ProductService to the StockService. Test its working.

Part 3

Given is the project **EvenoddApplication**. This project contains the following controller:

```
@RestController
public class EvenOddController {

    @GetMapping("/validate")
    public String isNumberPrime(@RequestParam("number") Integer number) {
        return number % 2 == 0 ? "Even" : "Odd";
    }
}
```

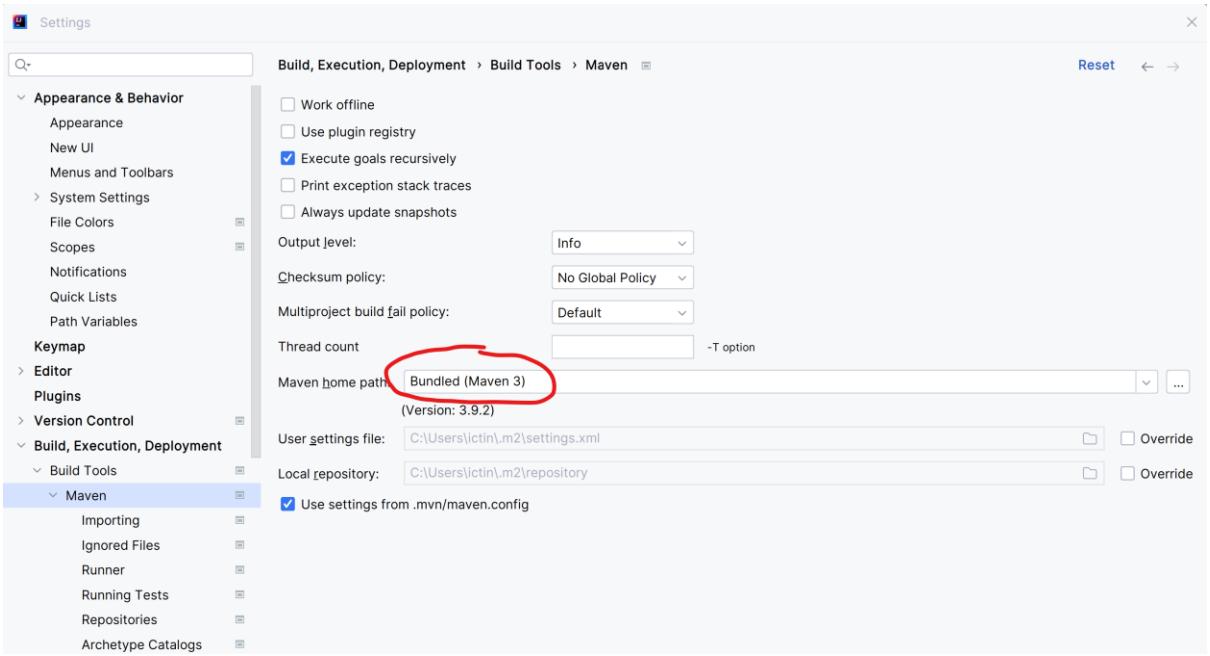
And the following contracts (in test/resources/contracts):

```
import org.springframework.cloud.contract.spec.Contract
Contract.make {
      description "should return even when number input is even"
      request{
            method GET()
            url("/validate") {
                  queryParameters {
                        parameter("number", "2")
                  }
            }
      }
      response {
            body("Even")
            status 200
      }
```

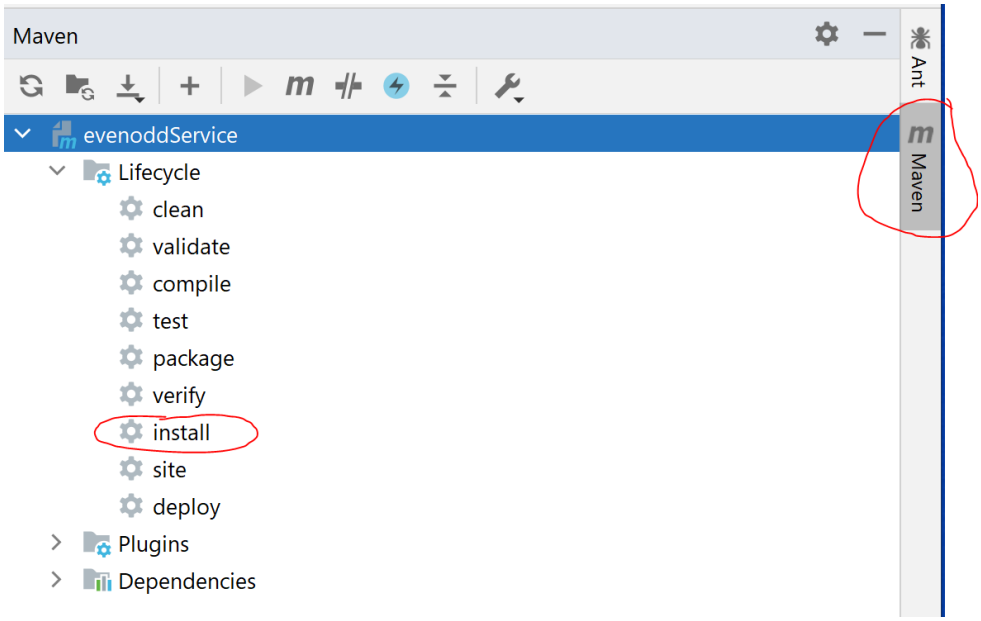
```
}
```

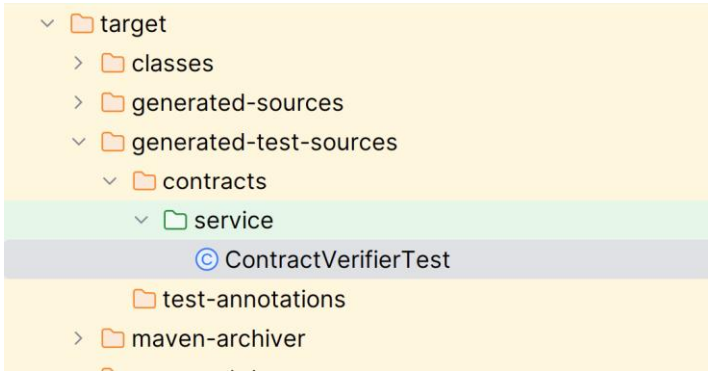
```
import org.springframework.cloud.contract.spec.Contract

Contract.make {
	description "should return odd when number input is odd"
	request {
		method GET()
		url("/validate") {
			queryParameters {
				parameter("number", "1")
			}
		}
	}
	response {
		body("Odd")
		status 200
	}
}
```

Go to Settings and make sure Maven is configured with the **`Bundled Maven.`**

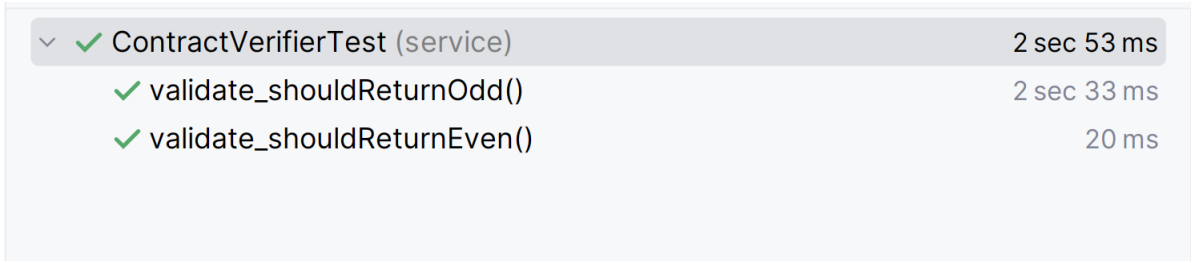


If you open the Maven tab on the right-hand side of IntelliJ and you run the **Maven install** command, then you see that a test class is created.





Right-click the directory **service** and select **"Mark Directory As" > "Test Sources Root".** This will tell IntelliJ that the files within this directory are intended for testing.
Then run the ContractVerifierTest class



In this project, add the following contract:

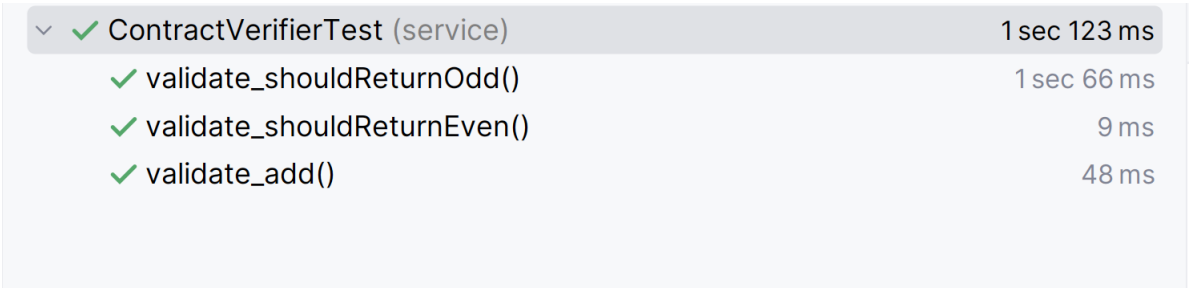
```
import org.springframework.cloud.contract.spec.Contract
Contract.make {
	description "add 2 numbers"
	request{
		method GET()
		url("/add") {
			queryParameters {
				parameter("value1", "2")
				parameter("value2", "5")
			}
		}
	}
	response {
		body("7")
		status 200
	}
}
```

Also write a new Controller class that implements this contract.

Modify the BaseTestClass as follows:

```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.MOCK)
@DirtiesContext
@AutoConfigureMessageVerifier
public class BaseTestClass {

    @Autowired
    private EvenOddController evenOddController;

    @Autowired
    private CalculatorController calculatorController;

    @BeforeEach
    public void setup() {
        StandaloneMockMvcBuilder standaloneMockMvcBuilder =

MockMvcBuilders.standaloneSetup(evenOddController,calculatorController);
        RestAssuredMockMvc.standaloneSetup(standaloneMockMvcBuilder);
    }
}
```

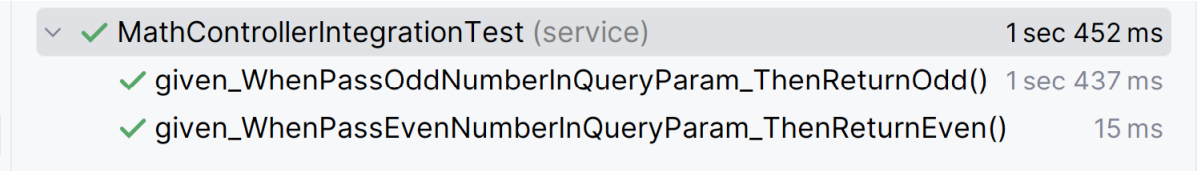
Run another Maven install, and see that the generated test class has another test method, and that this new test runs fine.



Also given is the **mathService** project that is the consumer of the evenoddService. The mathService contains the following test:

```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.MOCK)
@AutoConfigureMockMvc
@AutoConfigureJsonTesters
@AutoConfigureStubRunner(stubsMode = StubRunnerProperties.StubsMode.LOCAL,
        ids = "com.acme:evenoddService:+:stubs:8090")
public class MathControllerIntegrationTest {

          @Autowired
          private MockMvc mockMvc;

          @Test
          public void
given_WhenPassEvenNumberInQueryParam_ThenReturnEven() throws Exception {


mockMvc.perform(MockMvcRequestBuilders.get("/calculate?number=2")
                .contentType(MediaType.APPLICATION_JSON))
                .andExpect(status().isOk())
                .andExpect(content().string("Even"));
          }

          @Test
          public void given_WhenPassOddNumberInQueryParam_ThenReturnOdd()
throws Exception {


mockMvc.perform(MockMvcRequestBuilders.get("/calculate?number=1")
                .contentType(MediaType.APPLICATION_JSON))
                .andExpect(status().isOk())
                .andExpect(content().string("Odd"));
          }
}
```

This test uses the generated stubs from the evenoddService. When you run this test class as a JUnit test you see that the tests are correct.



Now we are going to modify the evenoddService. Change the contracts as follows:

```
import org.springframework.cloud.contract.spec.Contract
Contract.make {
	description "should return even when number input is even"
	request{
		method GET()
		url("/validate") {
			queryParameters {
				parameter("number1", "2")
				parameter("number2", "2")
			}
		}
	}
	response {
		body("Even")
		status 200
	}
}

import org.springframework.cloud.contract.spec.Contract

Contract.make {
	description "should return odd when number input is odd"
	request {
		method GET()
		url("/validate") {
			queryParameters {
				parameter("number1", "1")
				parameter("number2", "1")
			}
		}
	}
	response {
		body("Odd")
		status 200
	}
}
```

Run a Maven install on the POM file so that new stubs get generated. In the output we see that our local test fails.
Modify the controller so that the tests will pass.

```
	@GetMapping("/validate")
    public String isNumberPrime(@RequestParam("number1") Integer
                number1,@RequestParam("number2") Integer number2) {
        return number1 % 2 == 0  &&  number1 % 2 == 0 ? "Even" : "Odd";
    }
```

If we run the test in the mathService again, we see that the tests do not pass.
Update the mathService, so the tests pass again.

Part 4

Create a Config server and place all configuration of the applications from part 2 (API Gateway, ProductService and the StockService) in your github account. Test if everything still works.

Part 5

Suppose we have a microservice architecture with many different services. We also have an app that gets data from these different services. The app runs on both Android and IOS. The problem we face is that when we start this app, the first page on the app needs to show data that comes from 15 different microservices. The time to retrieve all this data from all 15 microservices takes 21 seconds, which is too slow and not acceptable.

The app should get all the required data as fast as possible. It should not take more than 2.5 seconds to retrieve all necessary data.

- Caching does not solve our problem because the data cannot be stale longer than 1 hour.
- Calling the services asynchronous is also not solving our problem because the calls we make to some services depend on the data we get back from other services

Explain how we can solve this problem?

What to hand in?

1. A zip file containing all services for part 1
2. A zip file containing all services for part 2
3. A zip file containing all services for part 3
4. A zip file containing all services for part 4
5. A PDF for part 5