

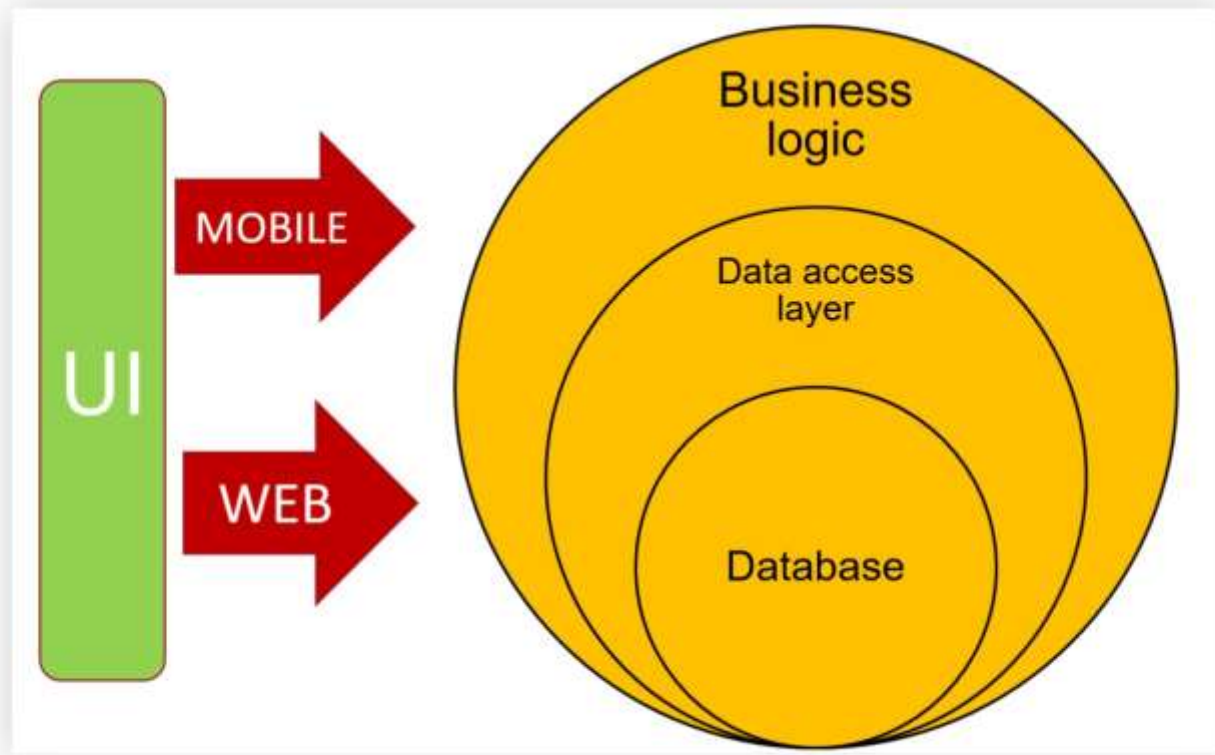


**Practical contract testing
with Spring Cloud Contract**

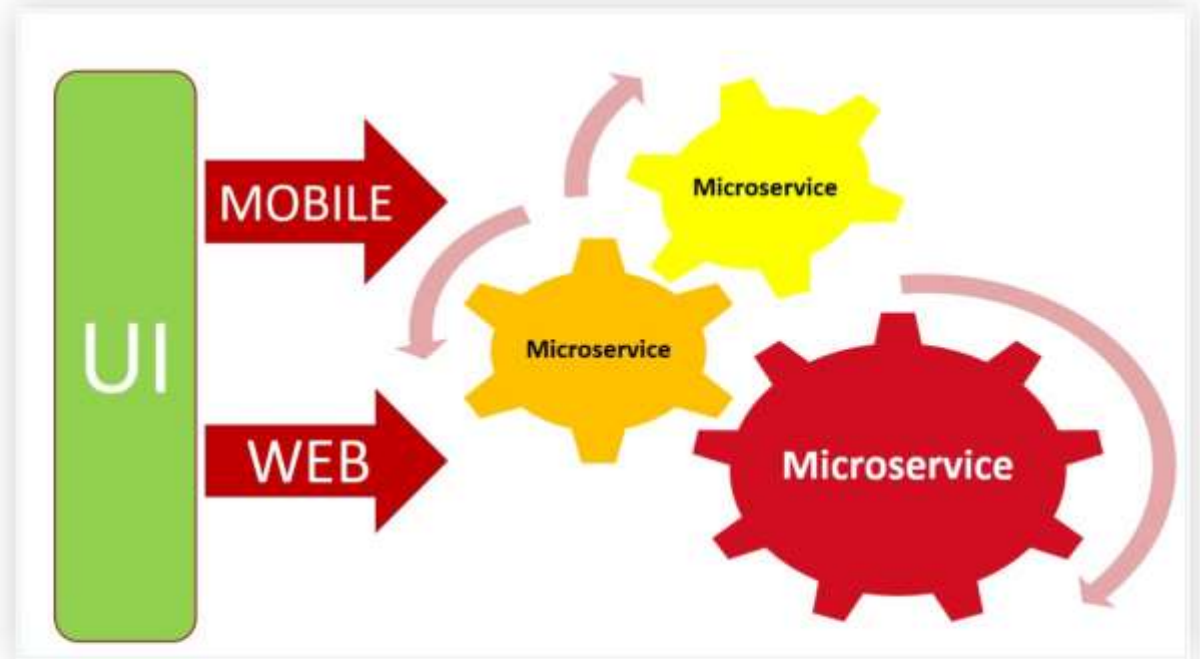
A close-up photograph of a disassembled hard drive. The central spindle and the top surface of a platter are visible. The drive is resting on a dark, textured surface. A semi-transparent grey rectangular box is overlaid on the center of the image, containing the text "Testing is not easy" in white, bold, sans-serif font.

Testing is not easy

Monolith architecture



Microservices architecture

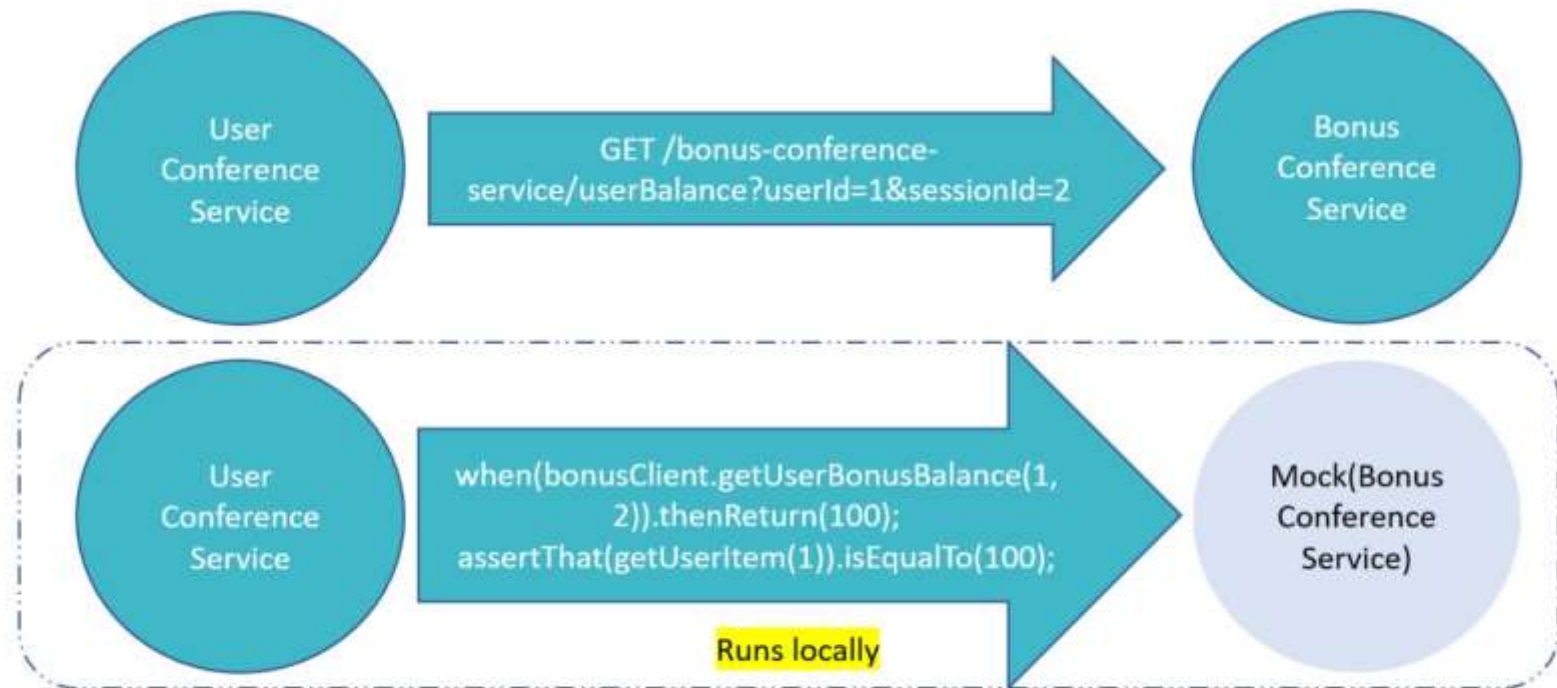


A row of slot machines in a casino. The focus is on the first machine, which has a purple and gold theme and is titled "EASY MONEY". The machine's screen displays a grid of numbers: 5, 20, 25, 10, 50, 100, 40, 25, 15, and 5. Above the screen, there are several circular signs, including one that says "EASY MONEY" and another that says "BARCELONA". The background is a dimly lit casino floor with other slot machines and a person sitting at a table in the distance.

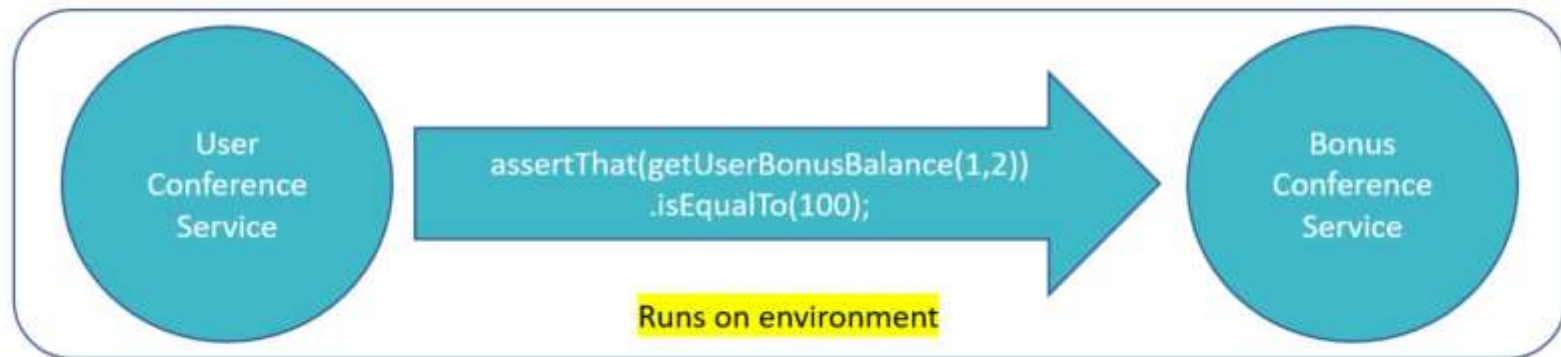
Let's play!



Unit test



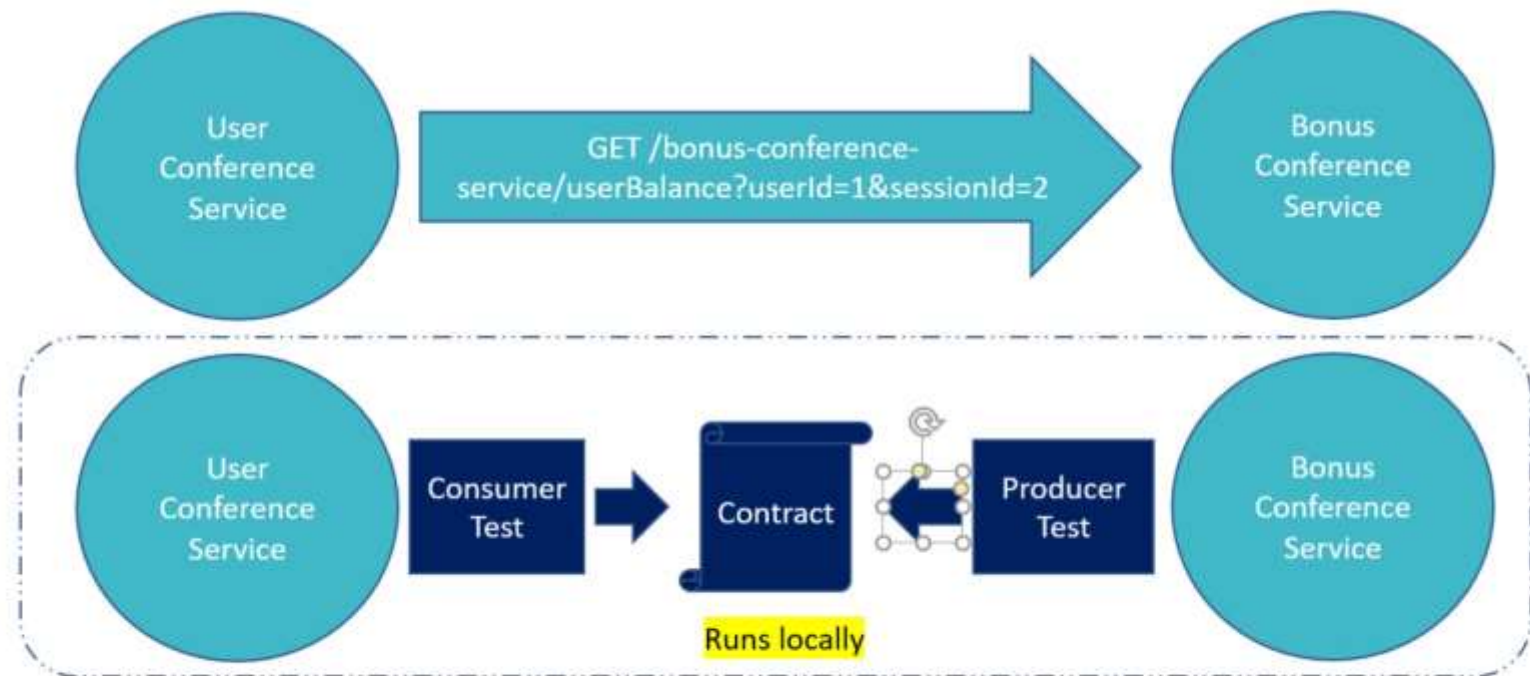
API test



What should I choose?



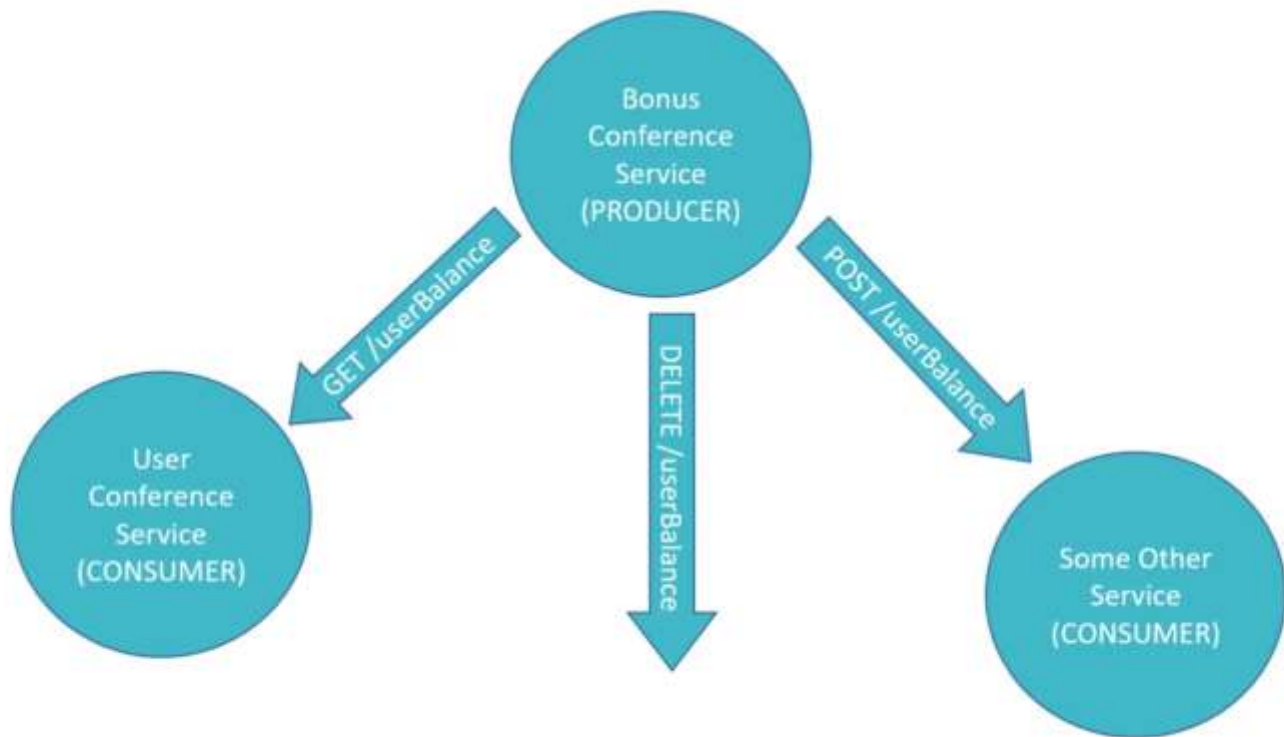
Contract testing to the rescue!



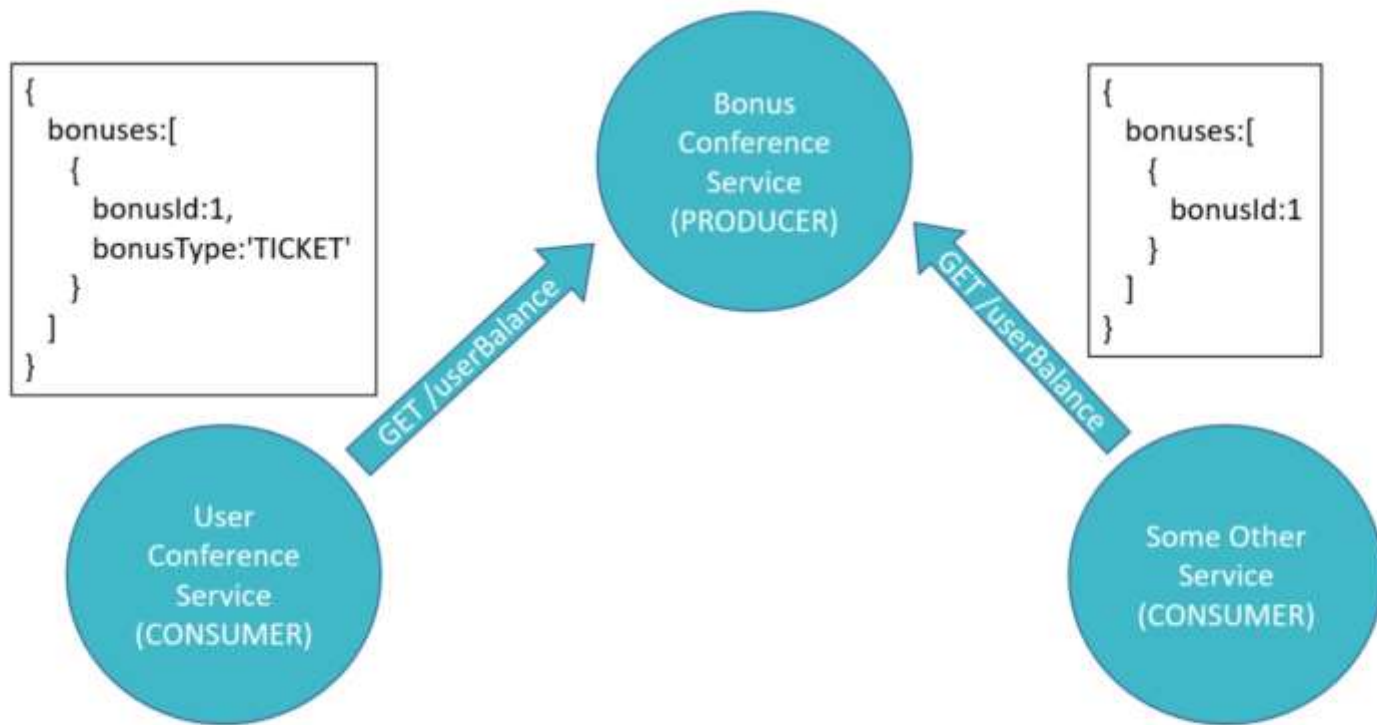
Why contract testing

ASPECT	UNIT	CONTRACT	API
WAY TO EXECUTE	LOCAL	LOCAL	ENVIRONMENT
FEEDBACK TIME	FAST	MEDIUM	SLOW
SETUP COMPLEXITY	LOW	MEDIUM	HIGH
CHANGE TOLERANCE	NO	YES	YES
CHECK FUNCTIONALITY	YES	NO	YES
CROSS-TEAM COLLABORATION	NO	YES	YES

Producer-driven contract testing



Consumer-driven contract testing



Toolset in Java



Spring Cloud
Contract

PACT 

A first-person perspective shot of someone relaxing in a striped hammock. The person's feet, wearing black and white sneakers, are visible in the foreground. A small brown dog is sitting on the floor next to the hammock, looking towards the camera. The background shows a window with patterned curtains and a wooden chair. A semi-transparent grey box with white text is overlaid in the center of the image.

Get some REST

REST: CONTRACT

```
Contract.make {
  description( description: ""Should return bonuses for specific conference user"" )
  request {
    method method: 'GET'
    urlPath( $(consumer( clientValue: '/bonus-conference-service/userBalance' ),
              producer( clientValue: '/userBalance' ))
            ) {
      queryParameters {
        parameter 'userId': $(consumer(regex( regex: '\\d+' )),
                                producer( clientValue: 777 ))
        parameter 'sessionId': $(consumer(regex( regex: '\\d+' )),
                                   producer( clientValue: 777 ))
      }
    }
  }
}
```


REST: CONTRACT

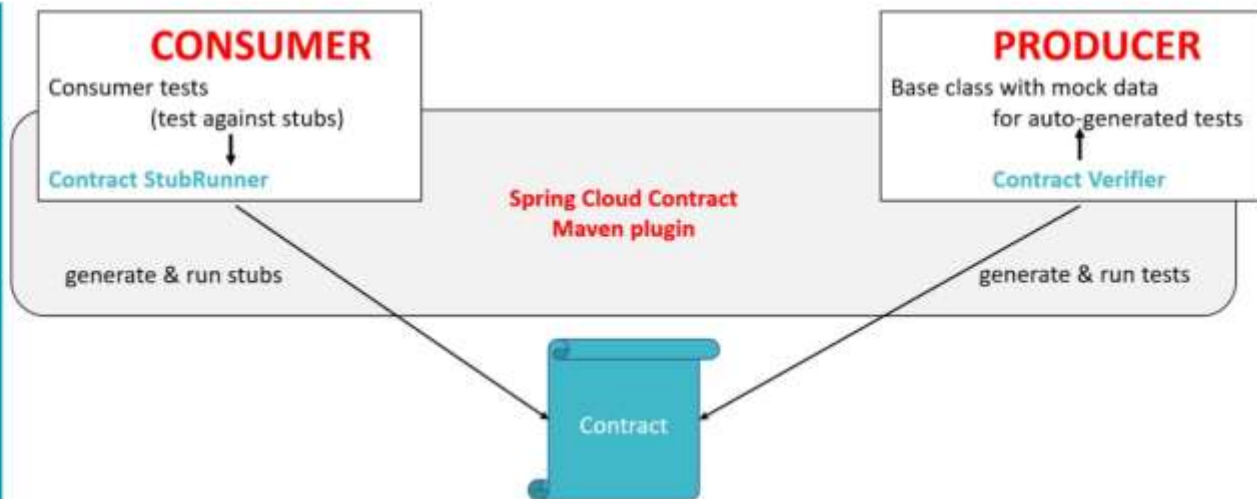
```
response {  
  status status: 200  
  body(  
    "bonuses": [[  
      "bonusId" : $(consumer( clientValue: 1L),  
        producer(positiveInt())),  
      "bonusType": $(consumer( clientValue: "TICKET"),  
        producer(anyNonEmptyString()))  
    ],  
  )  
  headers {  
    contentType(applicationJsonUtf8())  
  }  
}
```

REST: CONTRACT

```
Contract.make {
  description( description: """Should return bonuses for specific conference user""")
  request {
    method method: 'GET'
    urlPath( ${consumer( defaultValue: '/bonus-conference-service/userBalance' },
              producer( defaultValue: '/userBalance' ))
    } {
      queryParams {
        parameter 'userId': ${consumer( regex( regex: '\\d+' ),
                                         producer( defaultValue: '777' ))}
        parameter 'sessionId': ${consumer( regex( regex: '\\d+' ),
                                           producer( defaultValue: '777' ))}
      }
    }
  }
  response {
    status status: 200
    body {
      "bonuses": [
        {
          "bonusId" : ${consumer( defaultValue: 1L },
                               producer( positiveInt( )))},
          "bonusType": ${consumer( defaultValue: "TICKET" ),
                       producer( anyNonEmptyString( ))}
        }
      ]
    }
    headers {
      contentType( applicationJsonUtf8( ))
    }
  }
}
```

- Groovy / YAML / Kotlin DSLs
- Verify only format of request and response
- Use regular expressions

HOW SPRING CLOUD CONTRACT WORKS



REST: CONSUMER TEST

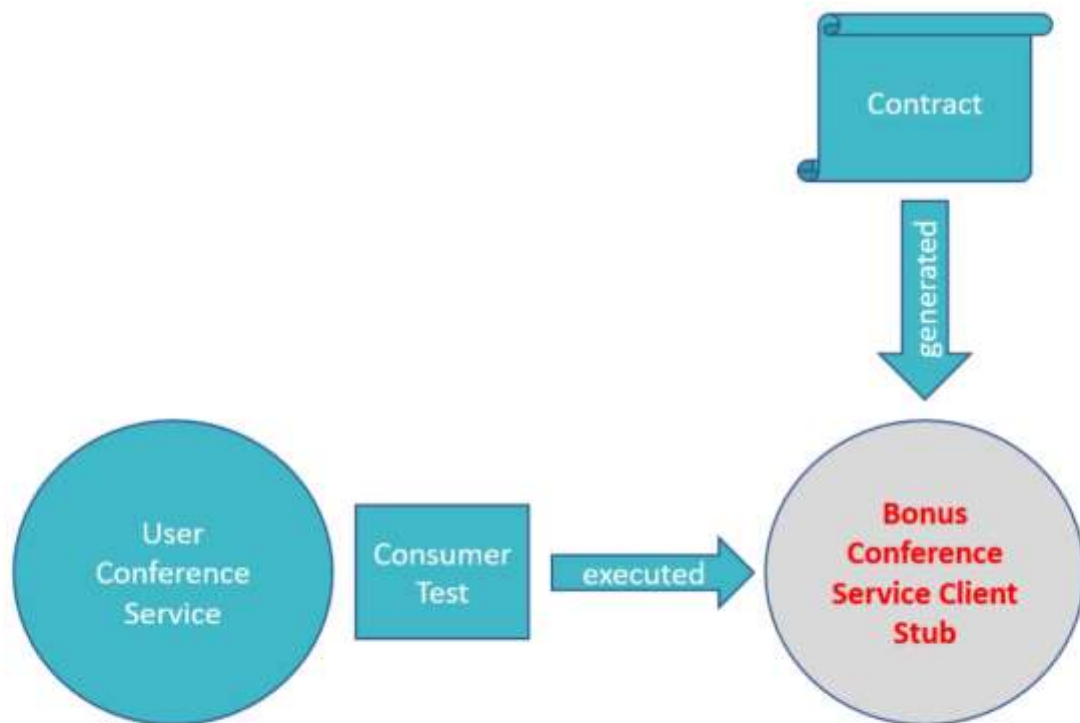
```
@ClassRule
public static StubRunnerRule rule = new StubRunnerRule()
    .repoRoot("http://linktoartifactory.com")
    .downloadStub(ivyNotation: "com.yourcompany.contracts:bonus-conference-service+:stubs")
    .withPort(STUB_RUNNER_PORT)
    .withConsumerName("user-conference-service")
    .withStubPerConsumer(true)
    .stubsMode(StubRunnerProperties.StubsMode.REMOTE);

@Autowired
private BonusConferenceClient bonusConferenceClient;
```

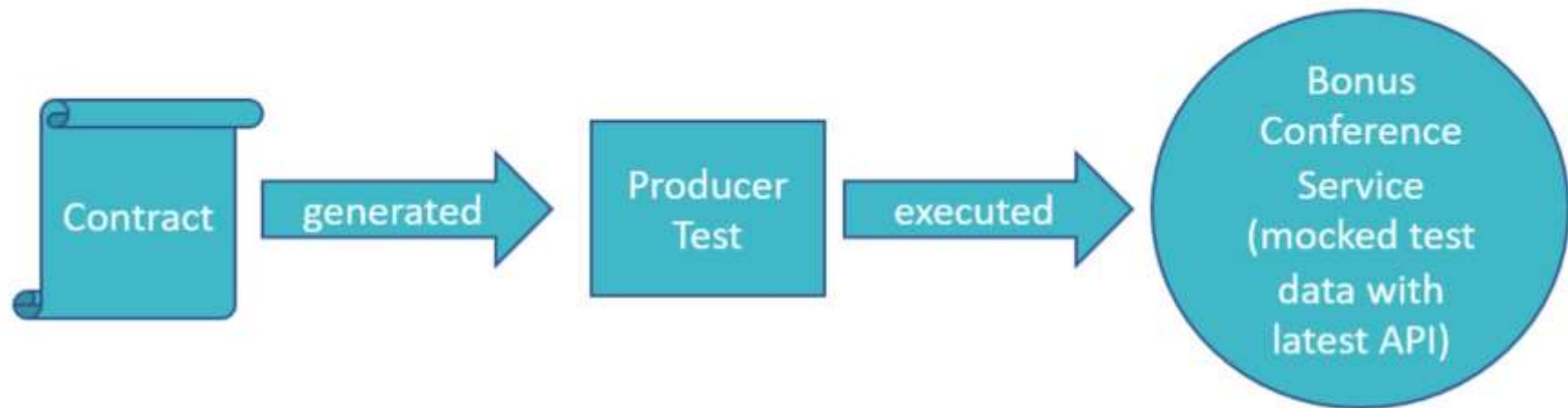
REST: CONSUMER TEST

```
@Test
public void shouldReturnConferenceBonusesForUser() {
    List<ConferenceBonusDto> userBonuses = bonusConferenceClient.getUserBonusBalance(USER_ID, SESSION_ID);
    assertThat(userBonuses).isNotEmpty().size().isNotEqualTo(0);
    ConferenceBonusDto bonus = userBonuses.get(0);
    assertThat(bonus.getBonusId()).isEqualTo(EXPECTED_BONUS_ID);
    assertThat(bonus.getBonusType()).isEqualTo(EXPECTED_BONUS_TYPE);
}
```

How consumer test works



How producer test works



REST: PRODUCER TEST

```
public class RestTest extends BalanceRestBase {

    @Test
    public void validate_shouldGetUserBonusBalance() throws Exception {
        // given:
        MockMvcRequestSpecification request = given();

        // when:
        ResponseOptions response = given().spec(request)
            .queryParams(s: "userId", ..objects: "777")
            .queryParams(s: "sessionId", ..objects: "777")
            .get(s: "/userBalance");

        // then:
        assertThat(response.statusCode()).isEqualTo(200);
        assertThat(response.header(s: "Content-Type")).matches("application/json;charset=UTF-8.*");
        // and:
        DocumentContext parsedJson = JsonPath.parse(response.getBody().asString());
        assertThatJson(parsedJson).field(["'bonusId'"]).isEqualTo(1);
        assertThatJson(parsedJson).field(["'bonusType'"]).isEqualTo("TICKET");
    }
}
```


REST: PRODUCER TEST

```
@WebMvcTest(controllers = {
    UserBalanceInternalController.class,
    ExceptionHandlerController.class
})
public abstract class BalanceRestBase extends BaseContractTest {
    private static final long USER_ID = 1L;
    private static final long SESSION_ID = 2L;

    @MockBean
    private UserBalanceService userBalanceService;

    @Before
    public void setUp() {
        UserBonus bonus = UserBonus.builder().bonusId(1L).bonusType("TICKET").build();

        when(userBalanceService.getUserBonusBalance(USER_ID, SESSION_ID))
            .thenReturn(Collections.singletonList(bonus));
    }
}
```

A glowing neon sign in the shape of a speech bubble, with a white neon outline and a dark purple interior. The sign is mounted on a dark, textured wall. A white rectangular banner is overlaid across the center of the sign, containing the text "What about messaging?".

What about messaging?

MESSAGING: CONTRACT

```
Contract.make {
  description( description: 'Should send a message in topic user_conference_balance_update')
  label label: 'userConferenceBalanceUpdate'
  input {
    triggeredBy( triggeredBy: 'userConferenceBalanceUpdate()' )
  }
  outputMessage {
    sentTo( sentTo: 'user_conference_balance_update' )

    body([
      userId      : $(consumer( clientValue: 111), producer(~/\d+/)),
      sessionId    : $(consumer( clientValue: 222), producer(~/\d+/)),
      action       : $(consumer( clientValue: "WIN_CHALLENGE"), producer(~/.+/))
    ])
  }
}
```

MESSAGING: CONSUMER TEST

```
@RunWith(SpringRunner.class)
@SpringBootTest(classes = BonusConsumerMessagingTest.MessageListener.class,
    webEnvironment = SpringBootTest.WebEnvironment.NONE
)
@AutoConfigureStubRunner
@TestPropertySource("/messaging-contract-test.properties")
public class BonusConsumerMessagingTest {
    private static final String BONUS_UPDATE_MESSAGE_LABEL = "userConferenceBalanceUpdate";

    @Autowired
    MessageListener messageListener;

    @Autowired
    StubTrigger stubTrigger;

    @Test
    public void shouldReceiveStampCardMessage() {
        BonusUpdateMessage expectedUpdateMessage = BonusUpdateMessage.builder()
            .userId(111L)
            .sessionId(222L)
            .action("WIN_CHALLENGE")
            .build();

        stubTrigger.trigger(BONUS_UPDATE_MESSAGE_LABEL);

        assertThat(messageListener.balanceUpdateMessage).isEqualTo(expectedUpdateMessage);
    }
}
```

MESSAGING: PRODUCER TEST

```
@RunWith(SpringRunner.class)
@SpringBootTest(classes = BonusMessagingProducerBase.ContractConfiguration.class)
@TestPropertySource("/messaging-contract-test.properties")
@AutoConfigureMessageVerifier
public abstract class BonusMessagingProducerBase {
    @Autowired
    private BonusMessagingProducerBase.ChannelDefinition channelDefinition;

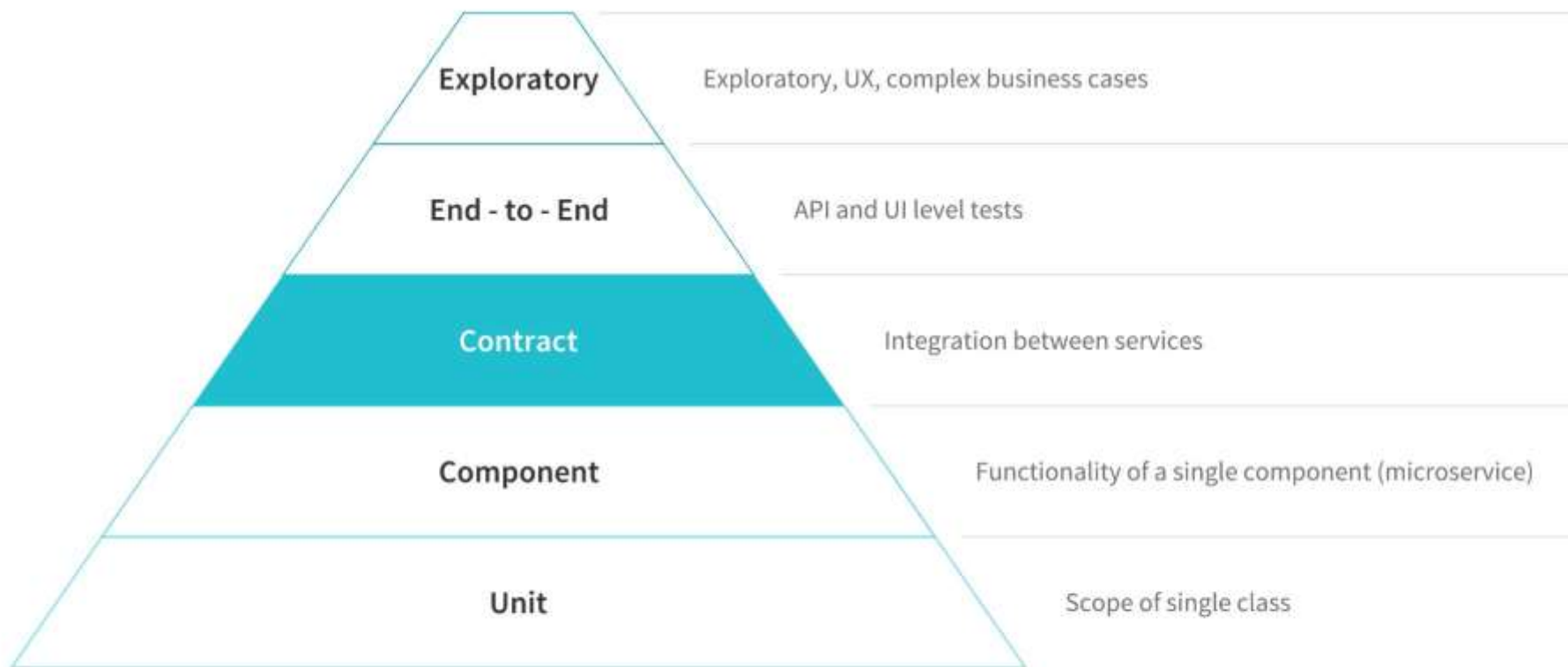
    @Autowired
    private MessagingServiceApi messagingServiceApi;

    public void userConferenceBalanceUpdate() {
        BonusUpdateMessage expectedUpdateMessage = BonusUpdateMessage.builder()
            .userId(111L)
            .sessionId(222L)
            .action("WIN_CHALLENGE")
            .build();
        channelDefinition.updateBonus().send(MessageBuilder.withPayload(expectedUpdateMessage).build());
    }
}
```

A silver revolver is positioned on the right side of the frame, lying on a bed of coarse, light-brown sand. Several brass bullet casing shells are scattered around the revolver, some lying horizontally and others at an angle. The revolver's cylinder is open, and its handle is visible. The sand is composed of small, irregular grains, creating a textured background. A semi-transparent text box is overlaid on the center of the image, containing the text "Is it a 'silver bullet?'".

Is it a “silver bullet?”

Test pyramid for microservices



Contract testing challenges



Technical training



Process changes



Additional technical setup

Biggest challenge



Communication!



Is it for me?

Why you may not need contract testing :)

- 1 Communication between teams is broken
- 2 Need to substitute functional testing with contracts
- 3 Unit and component testing is missing
- 4 “Fancy thing to try”

Conclusions



Contract testing brings value



Changes are hard



Choose wisely



Oleksandr Romanov

QA Automation Lead @ Playtika



al8xr



alex_roma_nov



www.linkedin.com/in/oleksandr-romanov



<http://alexromanov.github.io>



QUESTIONS?



THANK YOU!