# Consumer driven Contracts

- Introduced by Ian Robinson 2006
- Practice of using published contracts to assert expectations between consumers and producers while preserving loose coupling between services

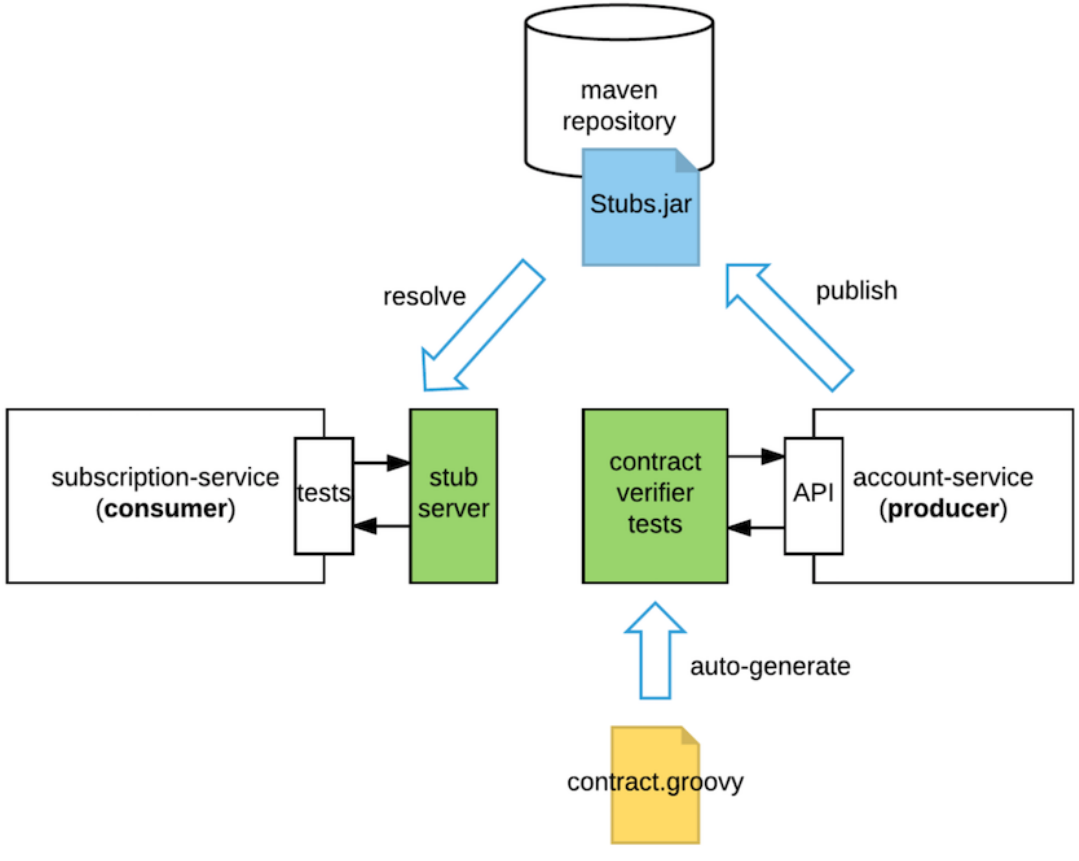
# Central Premise of CDC-T

- CDC-T T for Testing
- allow producers and consumers to publish stubs in the form of consumer driven contacts
- the contract describes a call interface for a service
- no runtime sharing of libraries

# The process

1. producer publishes stubs by defining and integration-tests that uses the contract
2. consumers mock the producer by downloading the versioned stubs from a shared location
3. producers share stubs through a contract definition but not share types or client libraries

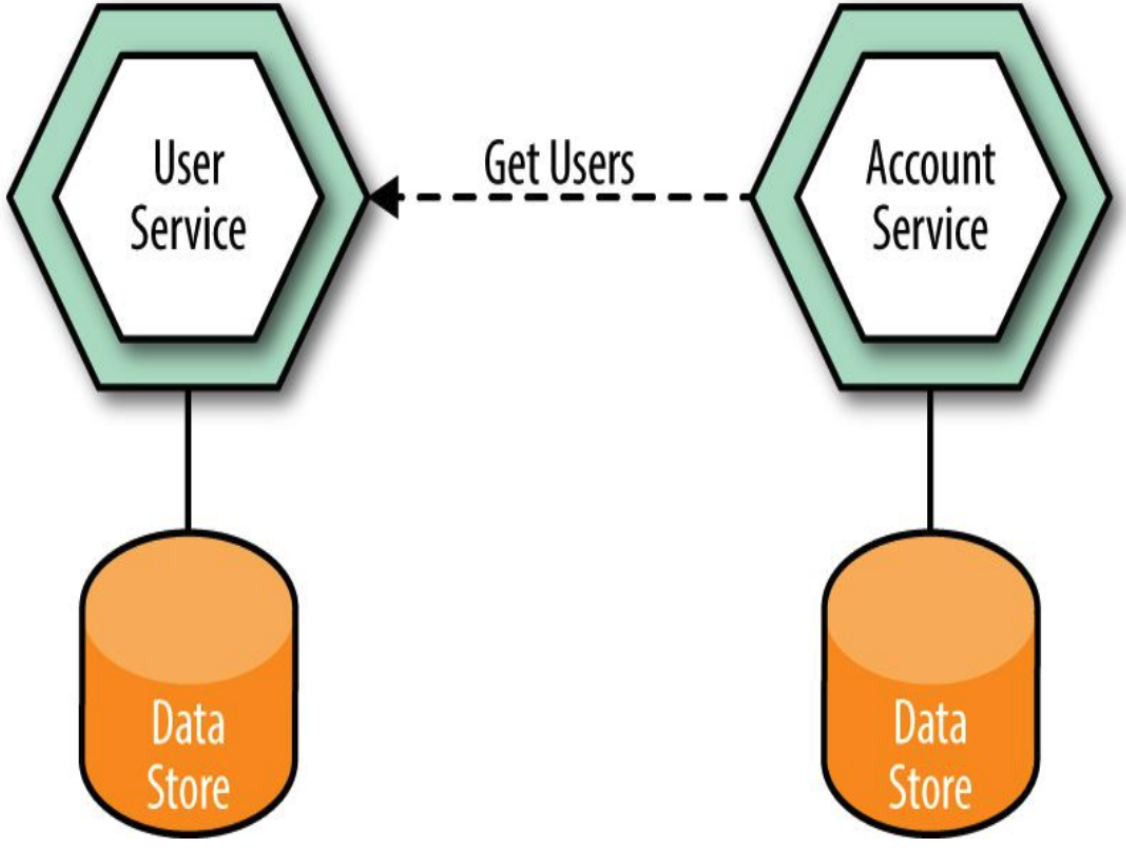
# Process in picture



# Spring Cloud Contract

- supports the ability to:
  - to publish
  - to simulate
  - and mock
- remote services via stubs

# Two microservices



# About the UserService

- Essential Code:

```
@Service
public class UserService {
 @Autowired
 private final UserRepository userRepository;

 public User getUserByPrincipal(Principal principal) {
  return Optional.ofNullable(principal)
   .map(p -> userRepository.findUserByUsername(p.getName()))
   .orElse(null);
 }
}
```

# About the UserController

- The usercontroller class will be called by the AccountService

```
@RestController
@RequestMapping(path = "/uaa/v1")
public class UserController {

 @Autowired private UserService us;

 @Autowired private AuthService as;

 @RequestMapping(path = "/me")
 public ResponseEntity<User> me(Principal p)  {
  return Optional.ofNullable(as.getAuthenticatedUser(p))
   .map(q -> ResponseEntity.ok().body(us.getUserByPrincipal(q)))
   .orElse(new ResponseEntity<User>(HttpStatus.UNAUTHORIZED));
 }
}
```

# Create a CDC Test

- create CDC-Test for /uaa/v1/me endpoint
- spring cloud contract to publish specification
- published as maven artifact
- UserService will publish specification that passed all unit tests written for the UserController

# The UserController Test

```
@RunWith(SpringRunner.class)
@WebMvcTest(UserController.class)
public class UserControllerTest {
 @Autowired private MockMvc mvc;
 @MockBean private UserService us;
 @MockBean private AuthService as;

 @Test public void getUserShouldReturnUser(){
  String content = "{\"username\": \"user\", \"Thom\"}";
  given(this.us.getUserByPrincipal(new PrincipalImpl("user")))
   .willReturn(new User("user", "Thom"));
  given(this.as.getAuthenticatedUser(null)).willReturn(
   new PrincipalImpl("user"));
  this.mvc.perform(get("/uaa/v1/me").accept(APPLICATION_JSON))
   .andExpect(status().isOk()).andExpect(content().json(content))
 }
```

# The Stub definition

- create a spring cloud contract stub definition
- defines a consumer driven test for fetching authenticated users
- stub definitions are used to describe the mocked (remote) service ***behaviour*** of a consumer-driven test
- behaviours of producer are also called ***expectations***
  - because they are expectations set by service producer exposed to consumer

# Groovy DSL

- each stub definition file covers a single method
- file is off .groovy
- Spring Cloud Cloud uses a groovy DSL

# shouldReturnUser.groovy

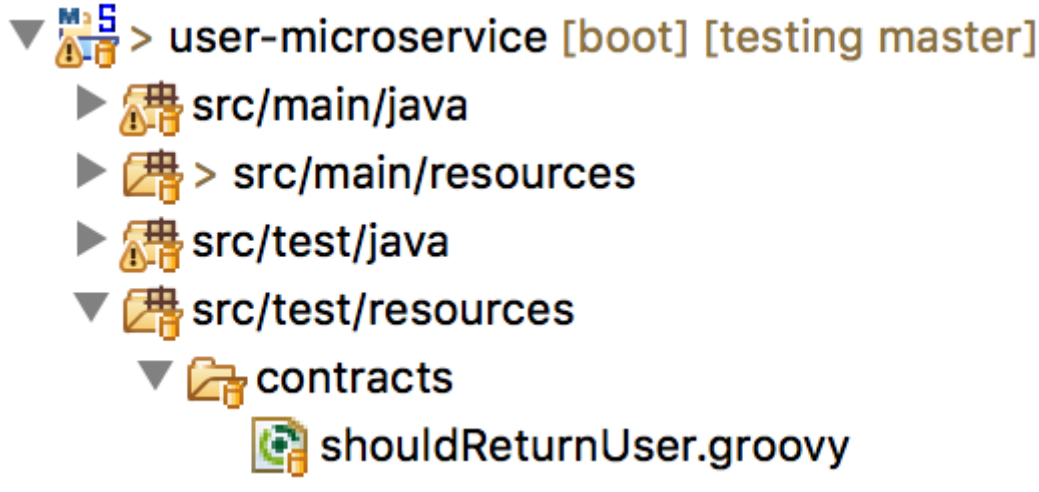
```
package contracts

org.springframework.cloud.contract.spec.Contract.make {
    request {
        method 'GET'
        url '/uaa/v1/me'
        headers {
            header('Content-Type': consumer(regex('application/*json*')))
        }
    }
    response {
        status 200
        body([
                username    : value(producer(regex('[A-Za-z0-9]+'))),
                firstName   : value(producer(regex('[A-Za-z]+')))
        ])
        headers {
            header('Content-Type': value(
                    producer('application/json;charset=UTF-8'),
                    consumer('application/json;charset=UTF-8'))
            )
        }
    }
}
```

# About the contract

- The contract consists of a request and response pair
- The value(consumer(…), producer(…)) are helper methods
- set matchers (regexp) or concrete values

# setup of project



# mvn install

- project must be configured to install 2 artifacts in repo
  1. the project
  2. the artifact containing the contract definition

# mvn configuration

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <executions>
                <execution>
                    <goals>
                        <goal>repackage</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
        <plugin>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-contract-maven-plugin</artifactId>
            <version>${spring-cloud-contract.version}</version>
            <extensions>true</extensions>
            <configuration>
                <baseClassForTests>demo.UserServiceBase</baseClassForTests>
                <basePackageForTests>demo</basePackageForTests>
            </configuration>
        </plugin>
    </plugins>
</build>
```

# mvn install

| | | |
|---|---|---|
| user-microservice | -- | Folder |
| 1.0.0-SNAPSHOT | -- | Folder |
| _remote.repositories | 256 bytes | Document |
| maven-metadata-local.xml | 909 bytes | XML text |
| user-microserv...SHOT-stubs.jar | 2 KB | Java JAR file |
| user-microserv...-SNAPSHOT.jar | 30,9 MB | Java JAR file |
| user-microserv...NAPSHOT.pom | 3 KB | TextEd...ument |
| maven-metadata-local.xml | 280 bytes | XML text |

# mvn output 1

## spring-cloud-contract-maven-plugin

- Generating server tests source code for Spring Cloud Contract Verifier contract verification
- Will use contracts provided in the folder contracts
- Using **UserServiceBase** as base class for test classes
- Creating new class file ContractVerifierTest.java

# Why a UserServiceBase?

- Spring Cloud Contract generates Test(s) that test rely on RestAssuredMockMvc
- Wire RestAssuredMockMvc to our environment with **XBase** class

# UserServiceBase

```
@RunWith(SpringRunner.class)
@WebMvcTest(UserController.class)
public class UserServiceBase {
 @MockBean  private UserService userService;
 @Before public void setup() {
  User actual = new User("user", "Thom");
  given(this.userService.getUserByPrincipal(
      new PrincipalImpl("user"))).willReturn(actual);

 RestAssuredMockMvc.standaloneSetup(
          new UserController(userService,authService));
 }
 public void assertThatRejectionReasonIsNull(Object rejectionReas
  assert rejectionReason == null;
 }
}
```

# mvn output 2

## spring-cloud-contract-maven-plugin

- Will use contracts provided in the folder contracts
- Copying Spring Cloud Contract Verifier contracts
- Converting from SC Contract Verifier contracts to WireMock stubs mappings
- WireMock stubs mappings directory: target/stubs/mappings
- Creating new json target/stubs/mappings/shouldReturnUser.json

# Run Stub in Consumer test

- stub runner complies to contract
- API is real - runs on actual port
- produces actual values

# Running test with Stub

```
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment = WebEnvironment.NONE)
@AutoConfigureStubRunner(
        ids = { "cnj:user-microservice:+:stubs:8081" },
        workOffline = true)
public class ConsumerDrivenTests {
 @Autowired private UserService service;

 @Test public void shouldReturnAuthenticatedUser() {
  User actual = service.getAuthenticatedUser();
  assertThat(actual).isNotNull();
  assertThat(actual.getUsername()).matches("[A-Za-z0-9]+");
  assertThat(actual.getFirstName()).matches("[A-Za-z]+");
 }
}
```

# About @AutoConfigureStubRunner

```
@AutoConfigureStubRunner(
        ids = { "user-microservice:+:stubs:8081" },
        workOffline = true)
```

- ids -> maven coordinate and port where stub runs
-
  - stands for version
- workOffline -> local maven repo

# WireMock

- WireMock is used
- About WireMock (from the website)
  - Mock your APIs for fast, robust and comprehensive testing
  - is a simulator for HTTP-based APIs
  - consider it a service virtualization tool or a mock server