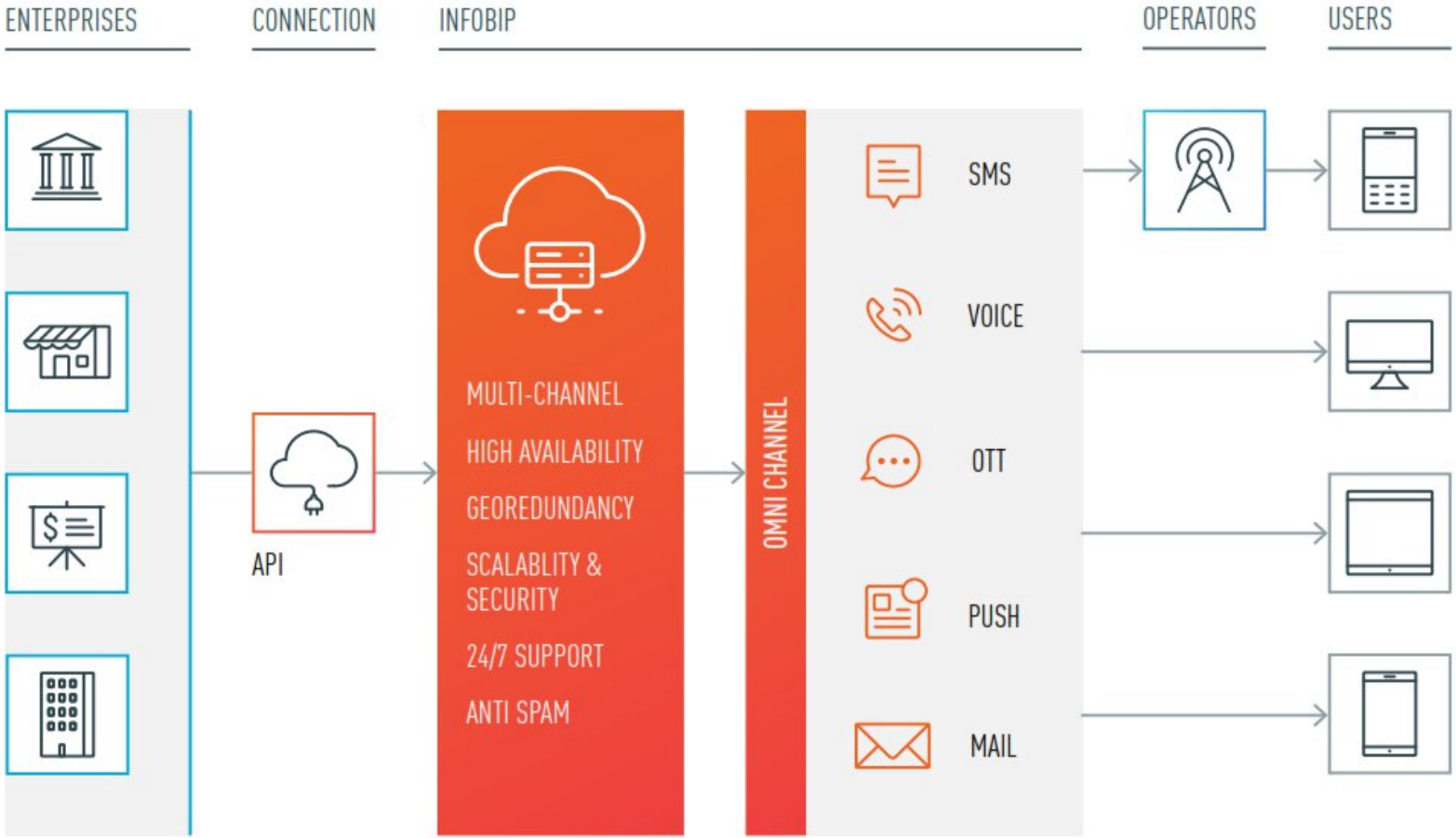
Элегантное интеграционное тестирование зоопарка микросервисов с помощью Testcontainers и JUnit 5 на примере глобальной SMS-платформы

Андрей Маркелов - Infobip


Microservices
LIGHTYEAR
Microservices Everywhere


ENTERPRISES
CONNECTION
INFOBIP
OPERATORS
USERS
API
MULTI-CHANNEL
HIGH AVAILABILITY
GEOREDUNDANCY
SCALABLITY & SECURITY
24/7 SUPPORT
ANTI SPAM
OMNI CHANNEL
SMS
VOICE
OTT
PUSH
MAIL

IT-инфраструктура

- Микросервисная архитектура
- 9 дата-центров по всему миру
- 500+ уникальных сервисов в проде
- Взаимодействие с 4млрд+ телефонов

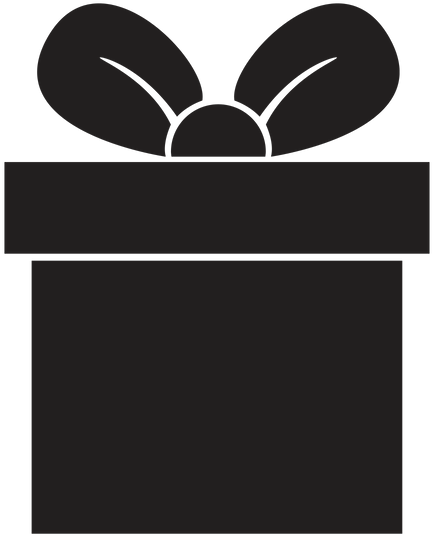
IT-инфраструктура

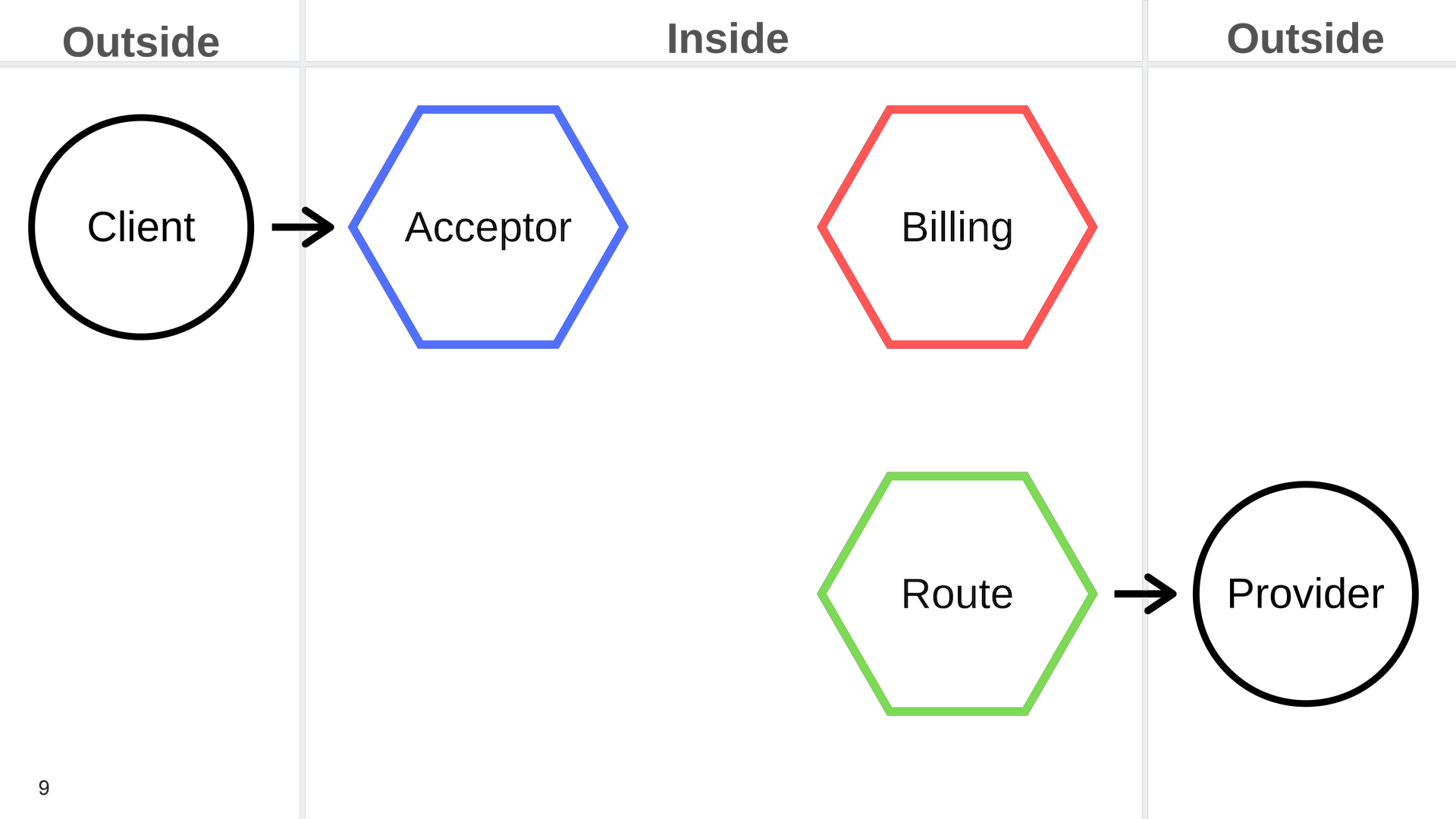
- Микросервисная архитектура
- 9 дата-центров по всему миру
- 500+ уникальных сервисов в проде
- Взаимодействие с 4млрд+ телефонов
- 30+ релизов в день

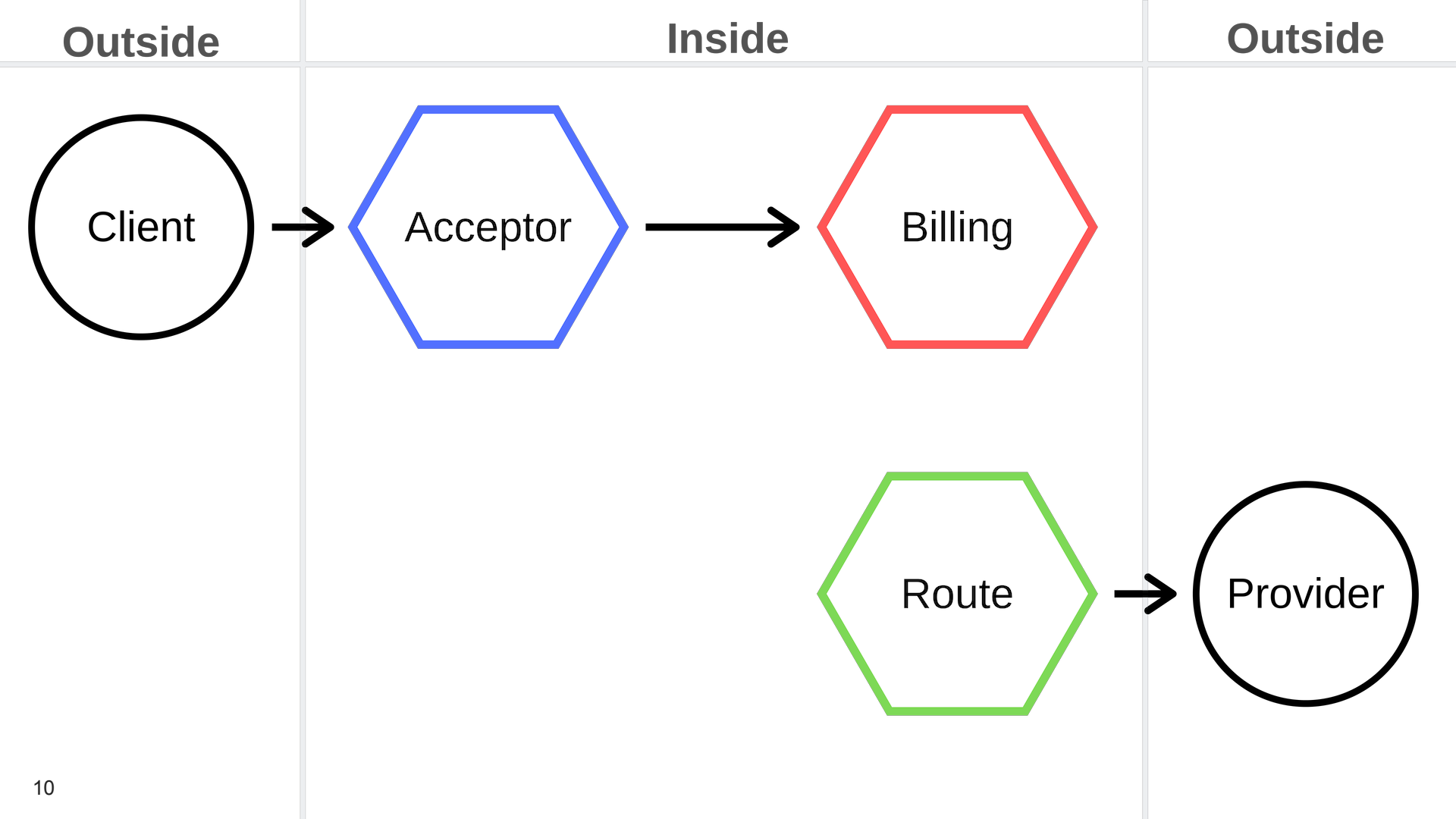
Проблема

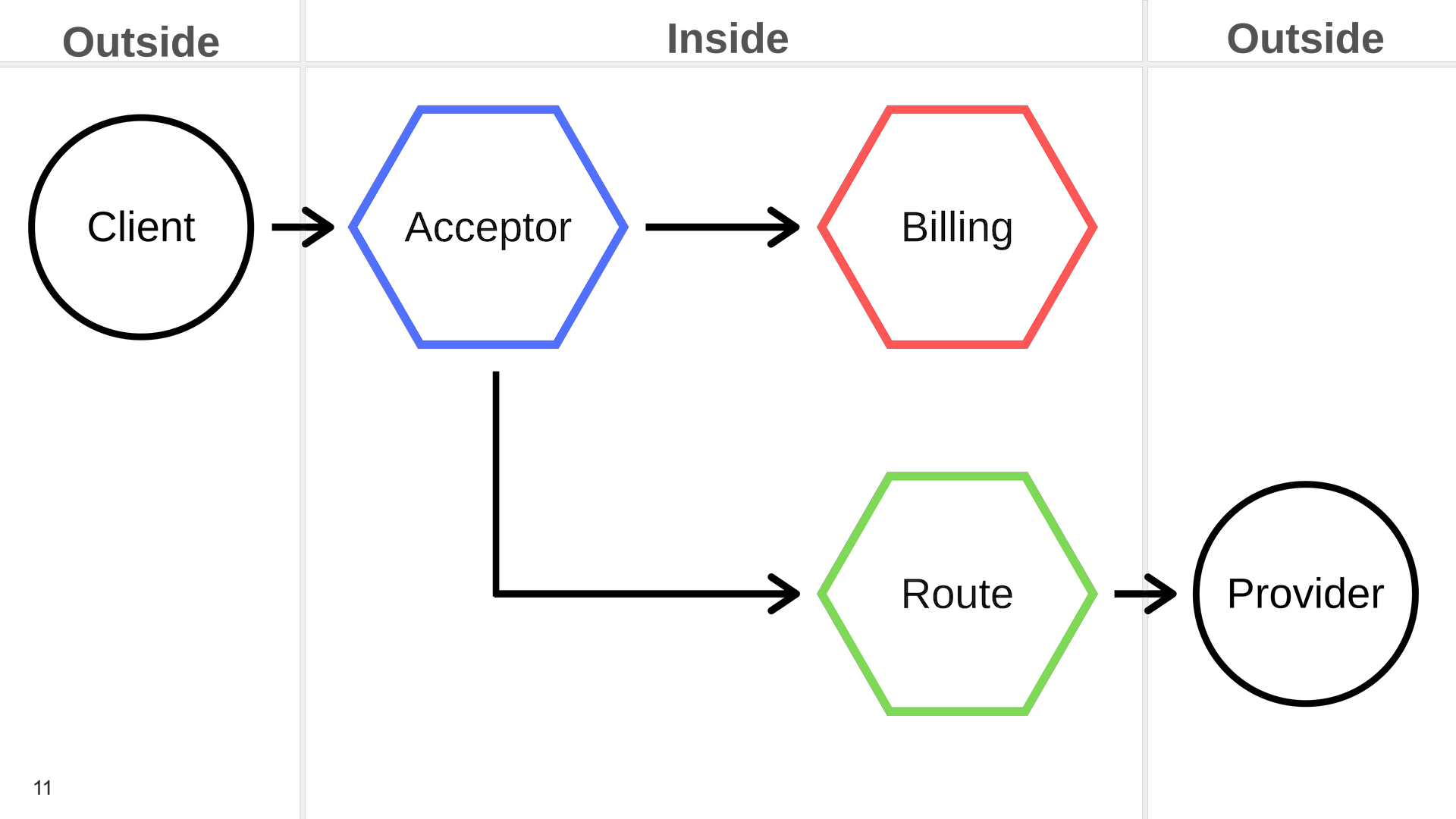
Проверить, что деплой новой версии одного из сервисов не сломает платформу

Базовый сценарий

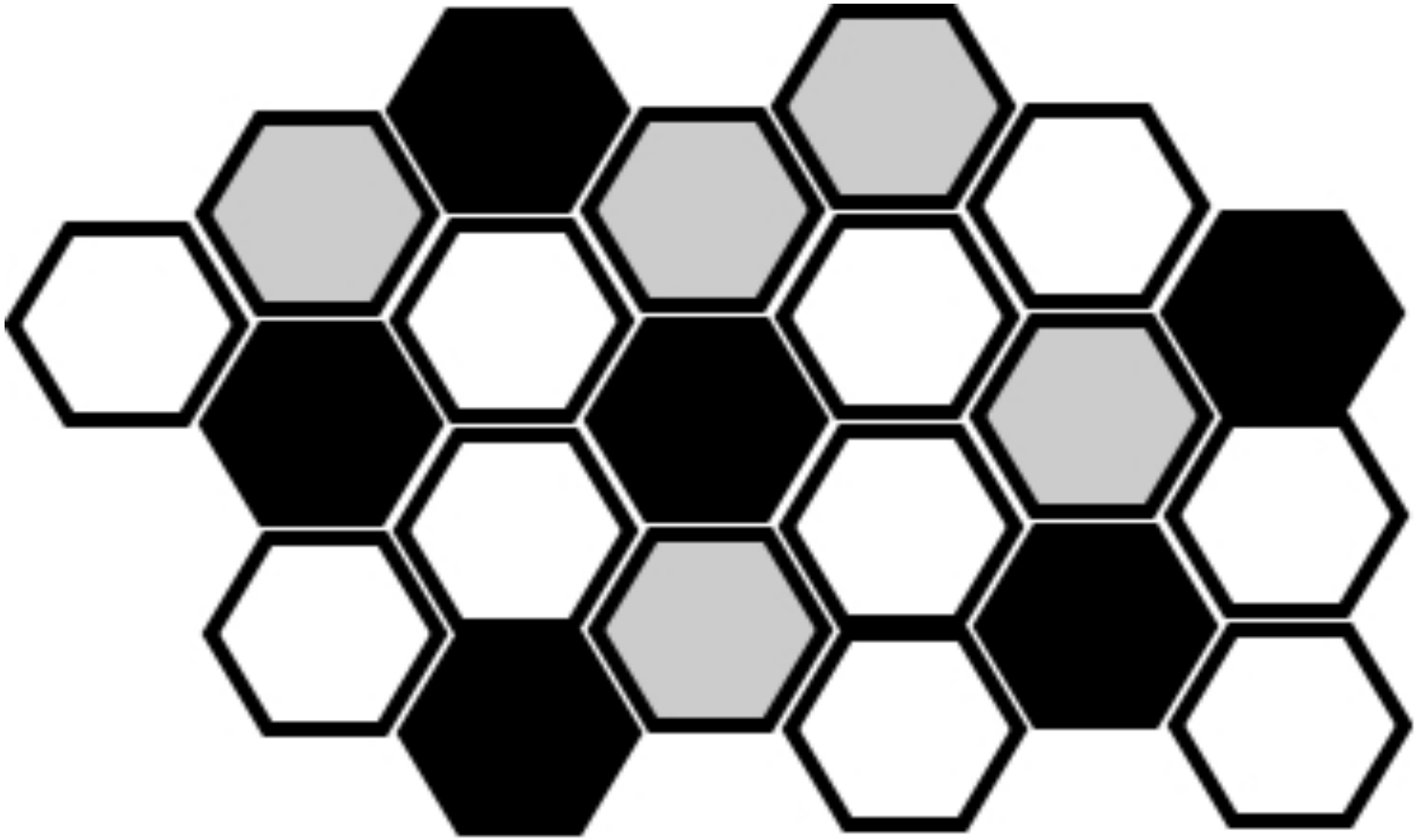
Outside	Inside	Outside
Client →		→ Provider


Outside
Inside
Outside
Client
Acceptor
Billing
Route
Provider


Outside
Inside
Outside
Client
Acceptor
Billing
Route
Provider

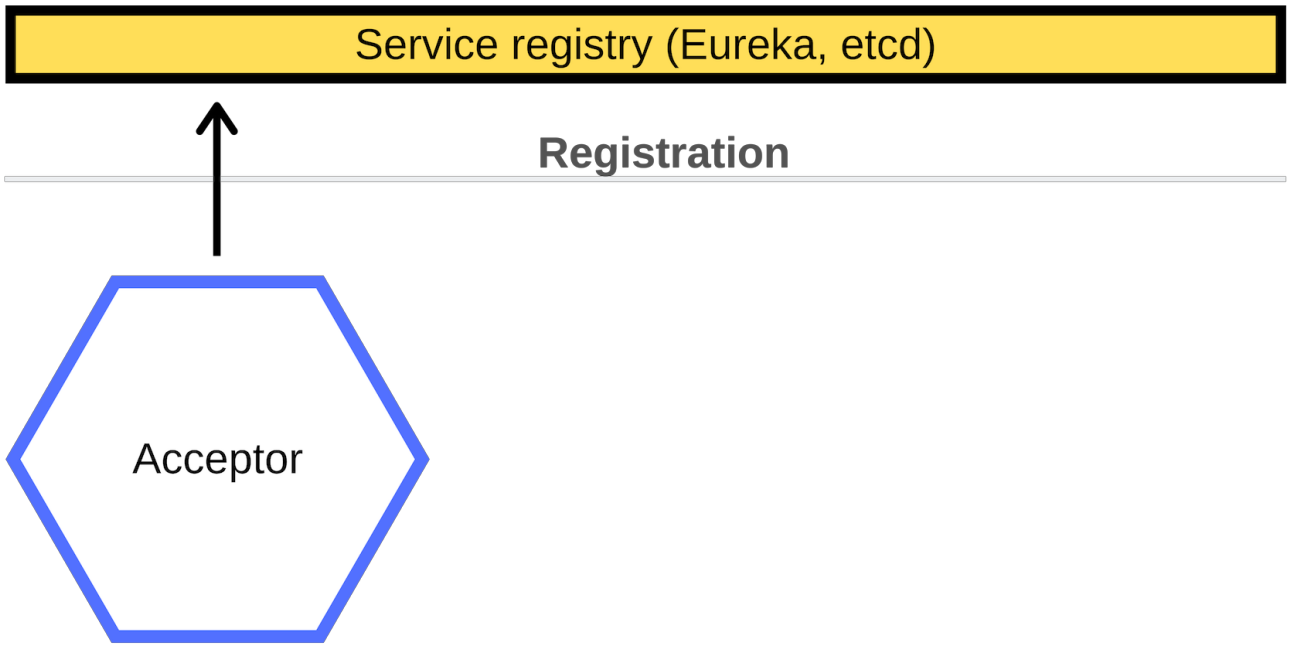

Outside
Inside
Outside
Client
Acceptor
Billing
Route
Provider

Архитектура микросервисов

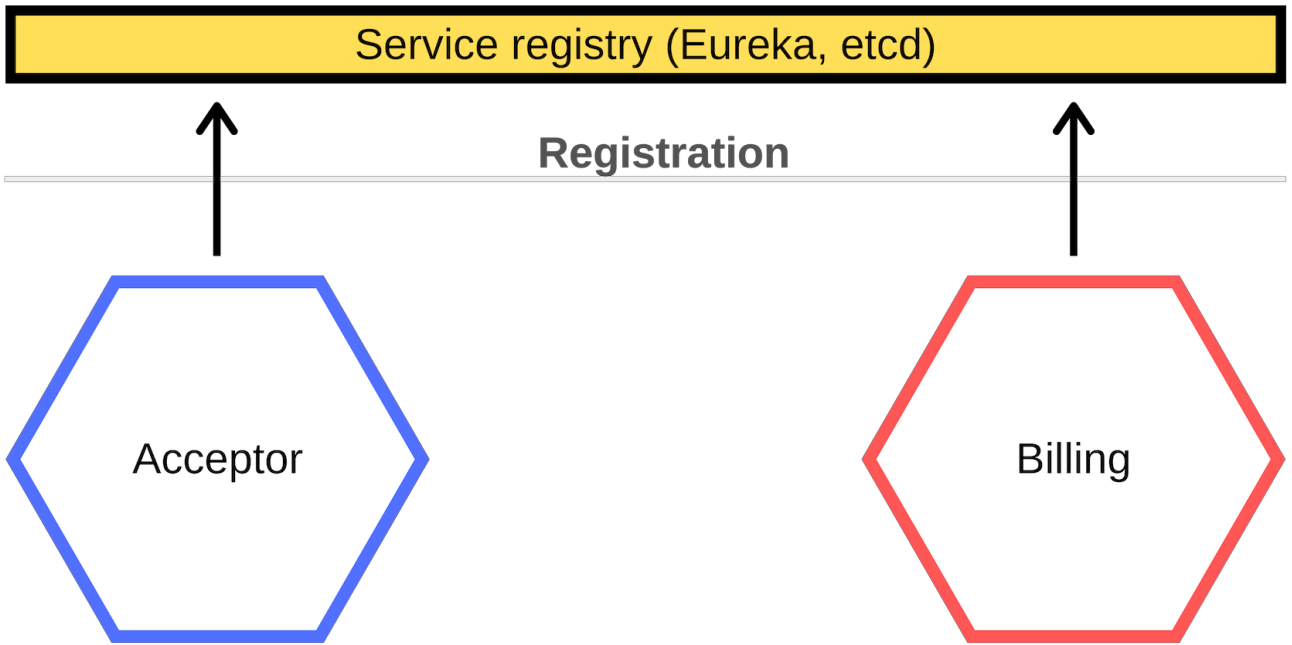


Регистрация

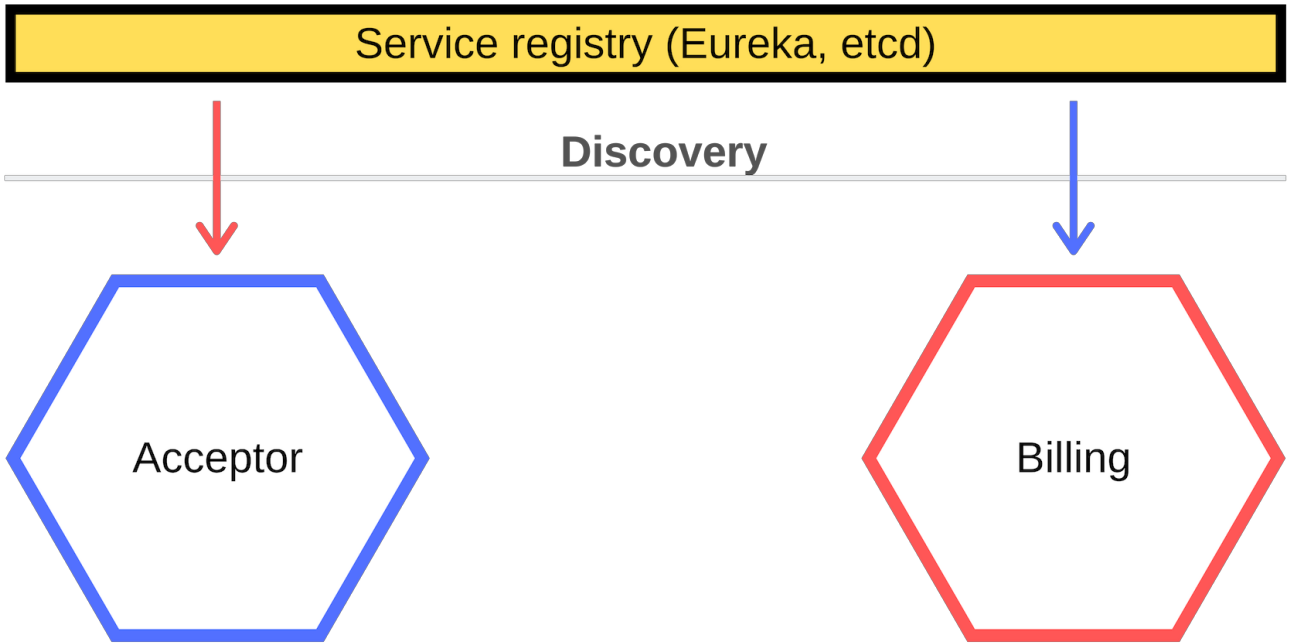
Регистрация



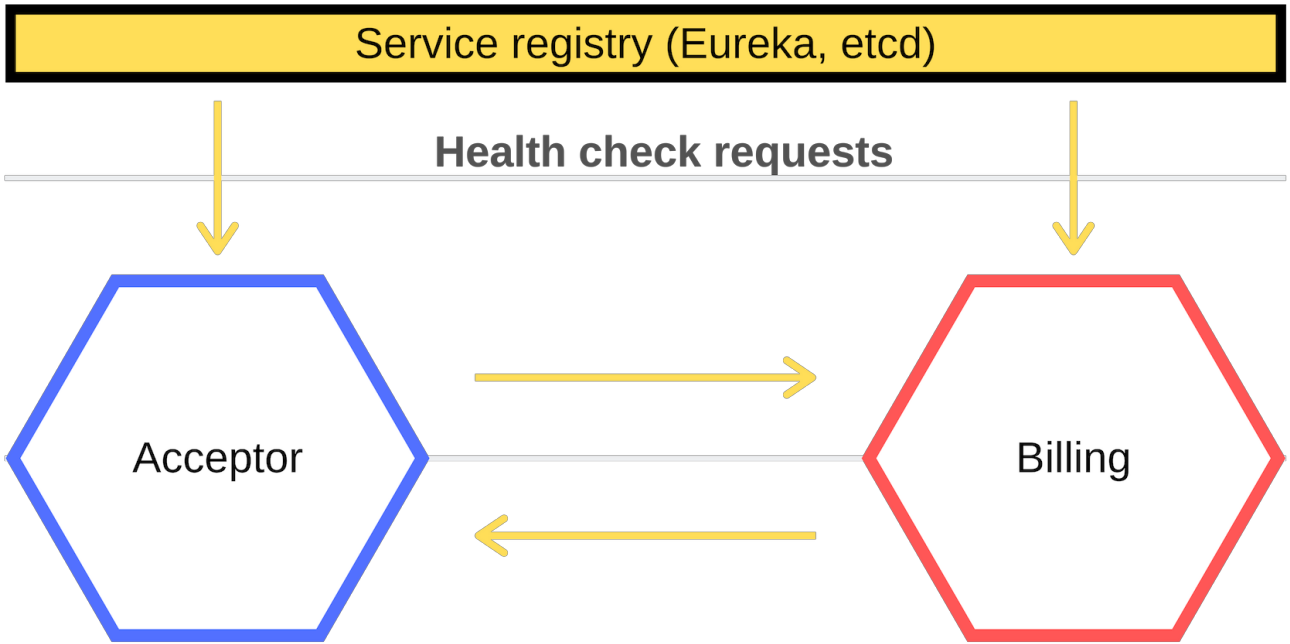
Регистрация



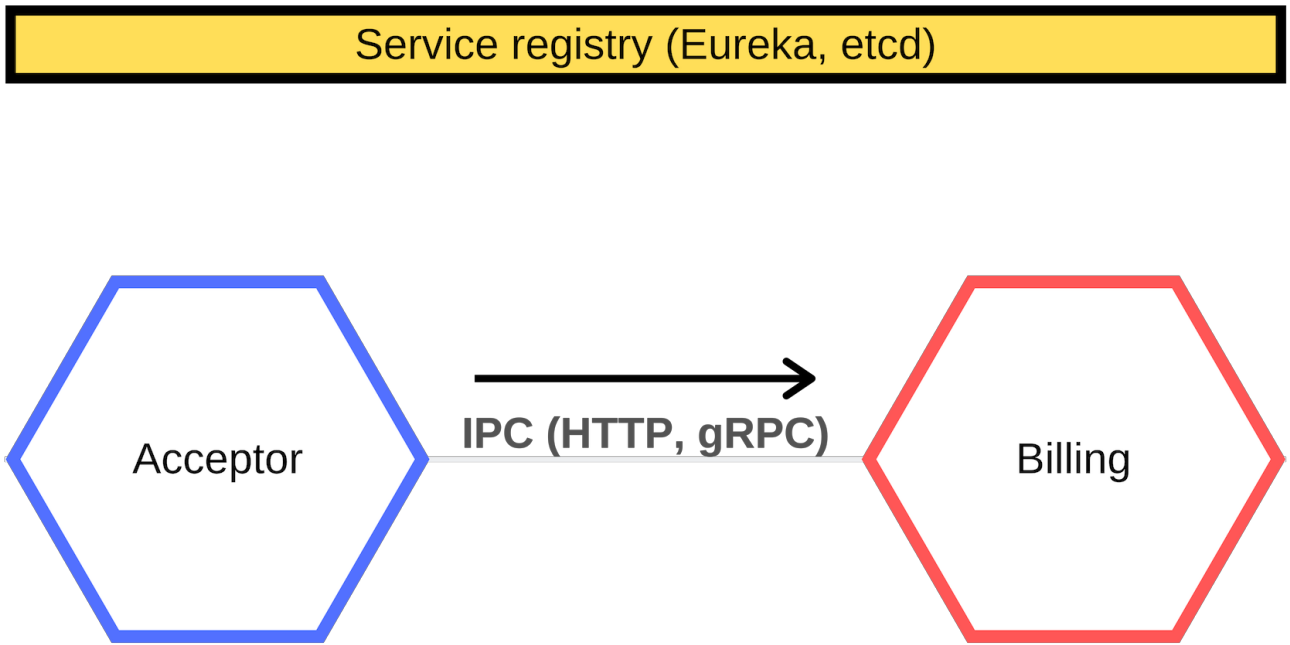
Обнаружение



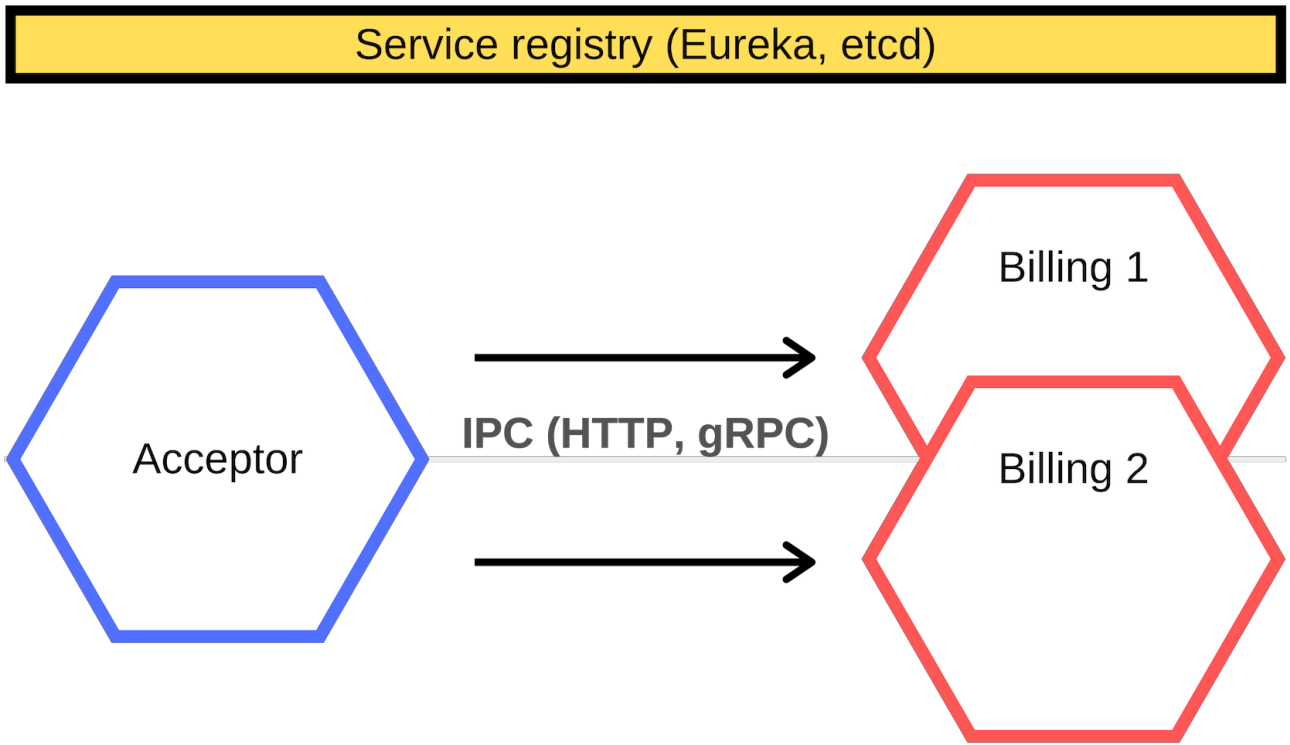
Health checks



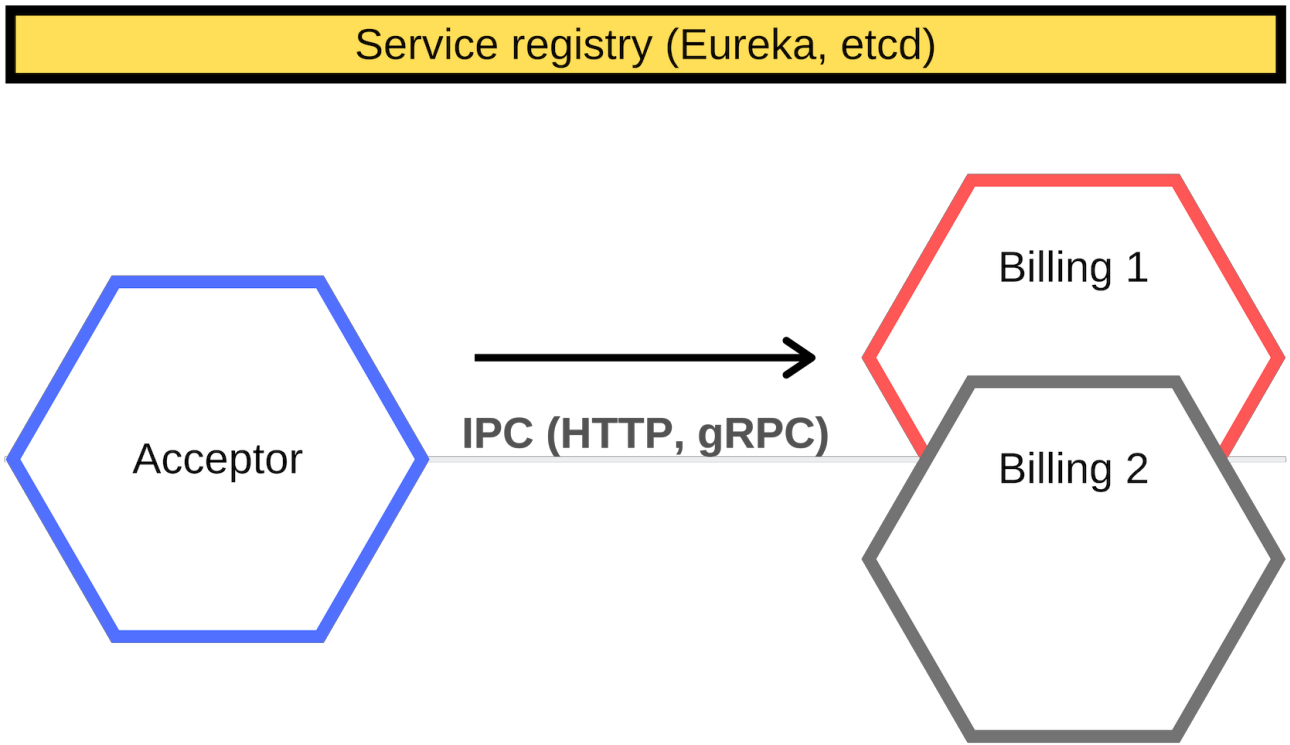
Direct invocation



Client-side balancing



Fault-tolerance



Business facing

Support programming

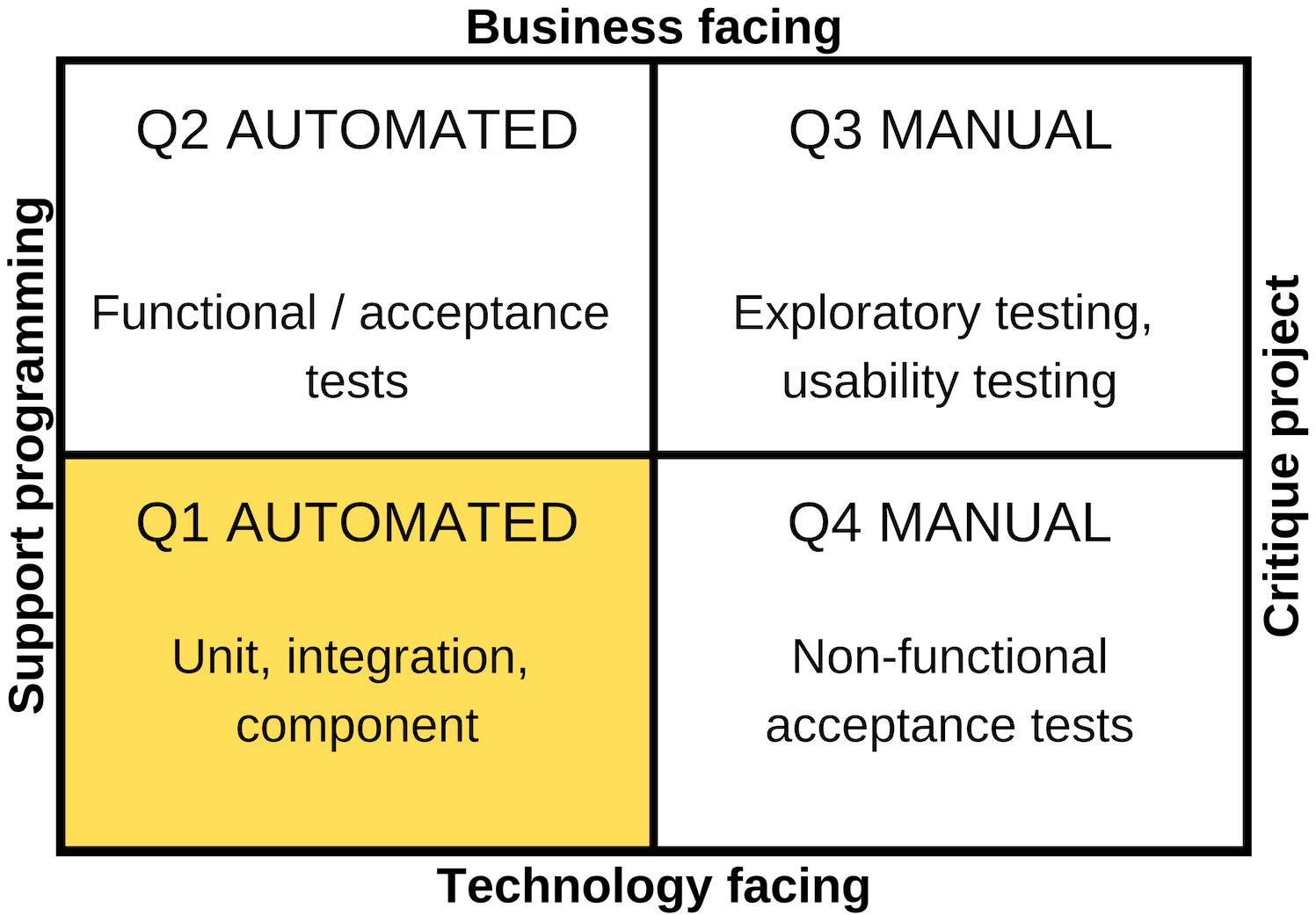
Q2 AUTOMATED	Q3 MANUAL
Functional / acceptance tests	Exploratory testing, usability testing
Q1 AUTOMATED	**Q4 MANUAL**
Unit, integration, component	Non-functional acceptance tests

Critique project

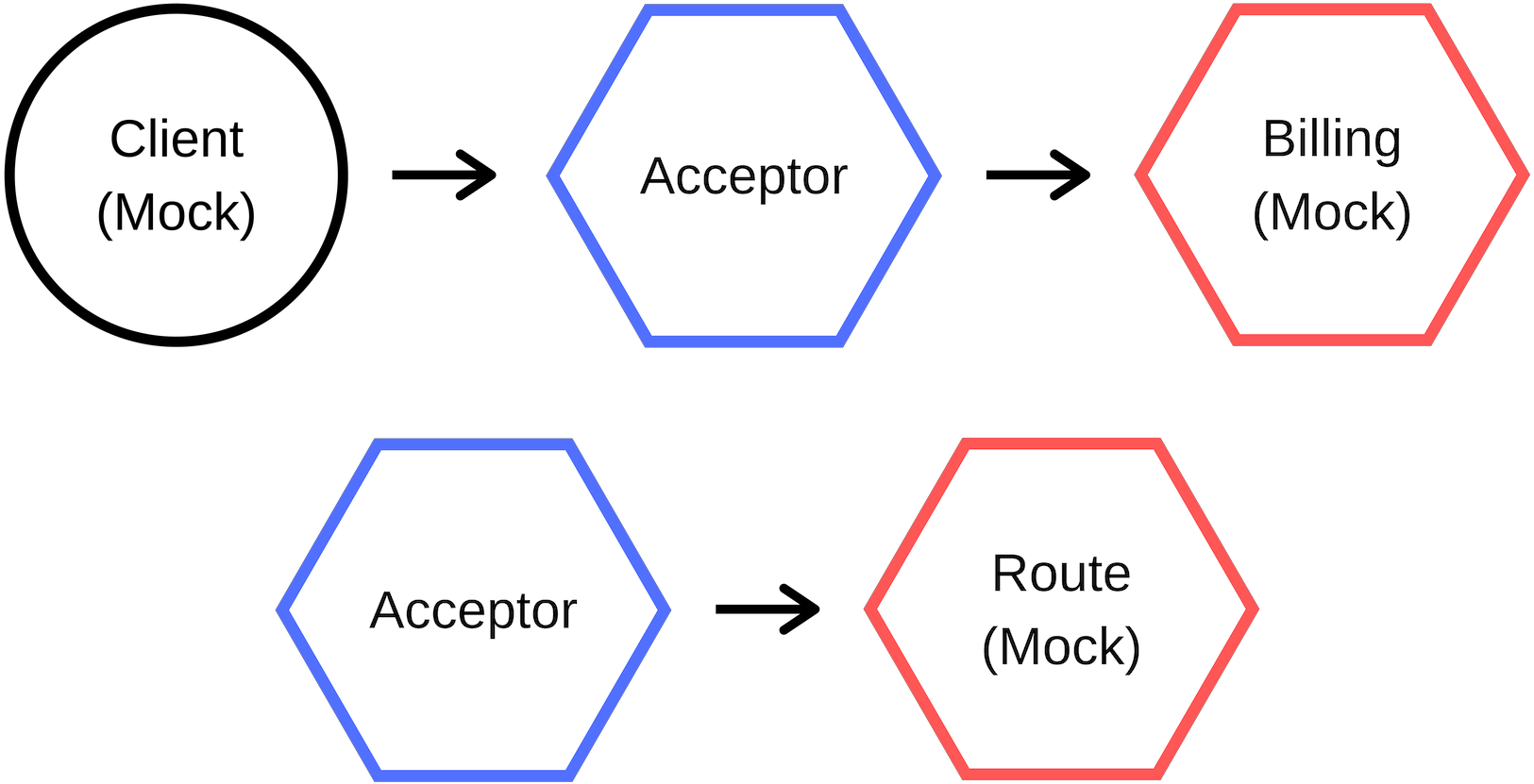
Technology facing

- Q1: поддерживающие код/технологические: юнит- и интеграционные тесты
- Q2: поддерживающие код/из бизнес-требований: компонентное и сквозное тестирование
- Q3: оценивающие проект/из бизнес-требований: юзабилити и исследовательское тестирование
- Q4: оценивающие проект/технологические: нефункциональное тестирование

http://www.exampler.com/old-blog/2003/08/21/#agile-testing-project-1


Business facing
Q2 AUTOMATED
Functional / acceptance tests
Q3 MANUAL
Exploratory testing, usability testing
Q1 AUTOMATED
Unit, integration, component
Q4 MANUAL
Non-functional acceptance tests
Support programming
Critique project
Technology facing

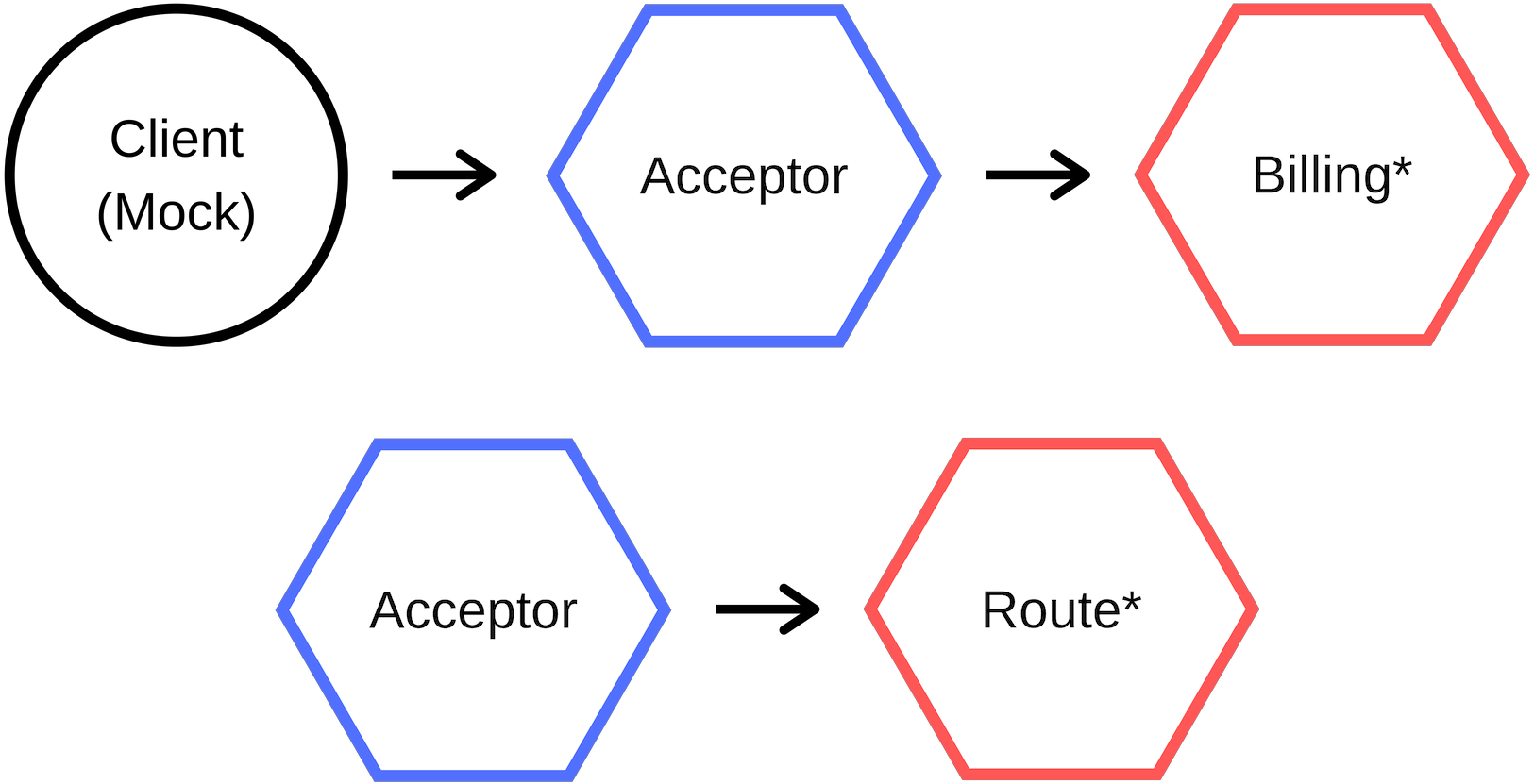
Unit (mocked) tests


Client
(Mock)
Acceptor
Billing
(Mock)
Acceptor
Route
(Mock)

Unit (mocked) tests

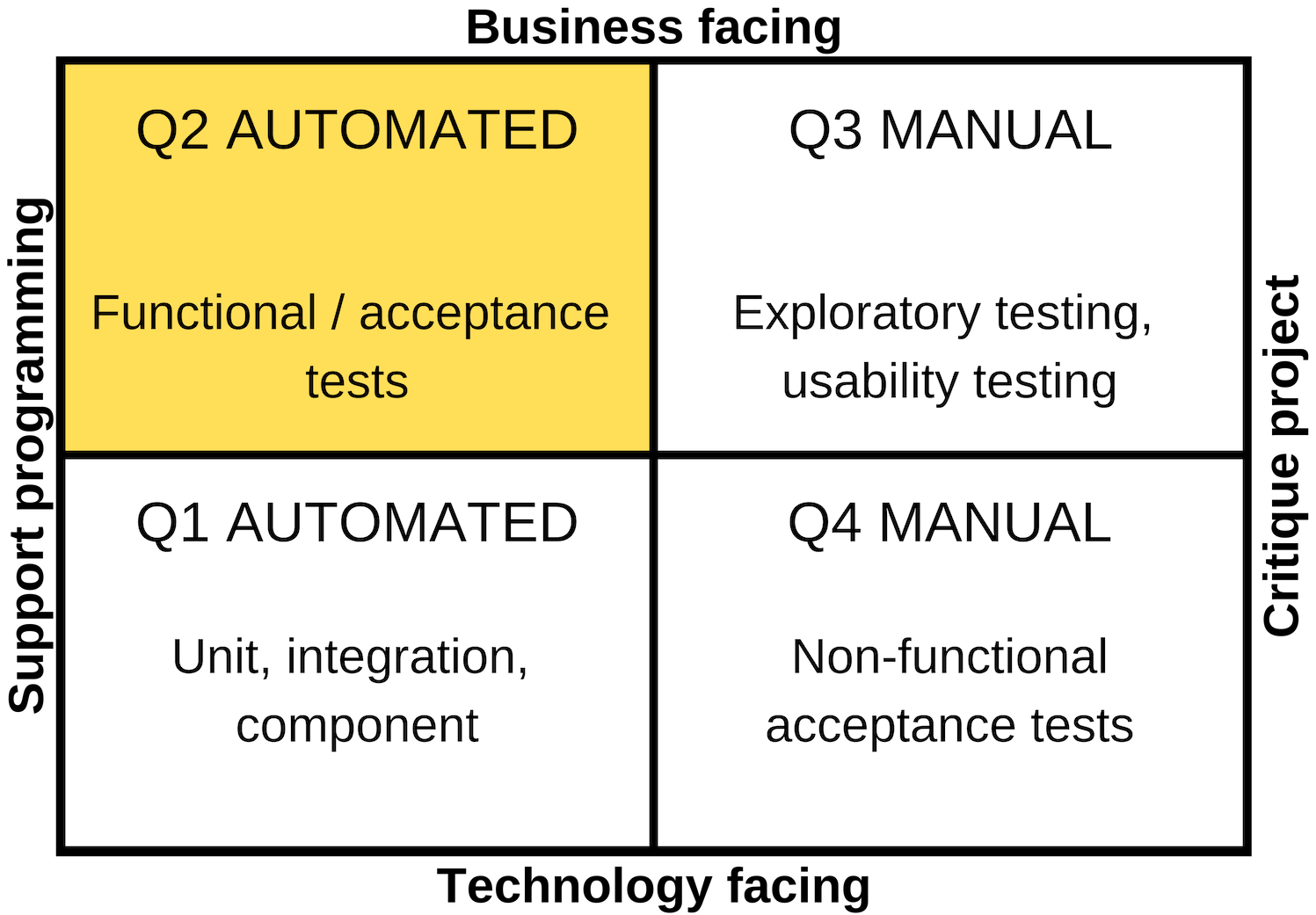
+ Дешево
+ Быстро
+ Надежно
– Нельзя доверять

Integration (component) tests


Client
(Mock)
Acceptor
Billing*
Acceptor
Route*

Integration (component) tests

– Дорого
– Медленно
– Ненадежно
± Можно доверять


Business facing
Q2 AUTOMATED
Functional / acceptance tests
Q3 MANUAL
Exploratory testing, usability testing
Q1 AUTOMATED
Unit, integration, component
Q4 MANUAL
Non-functional acceptance tests
Support programming
Critique project
Technology facing

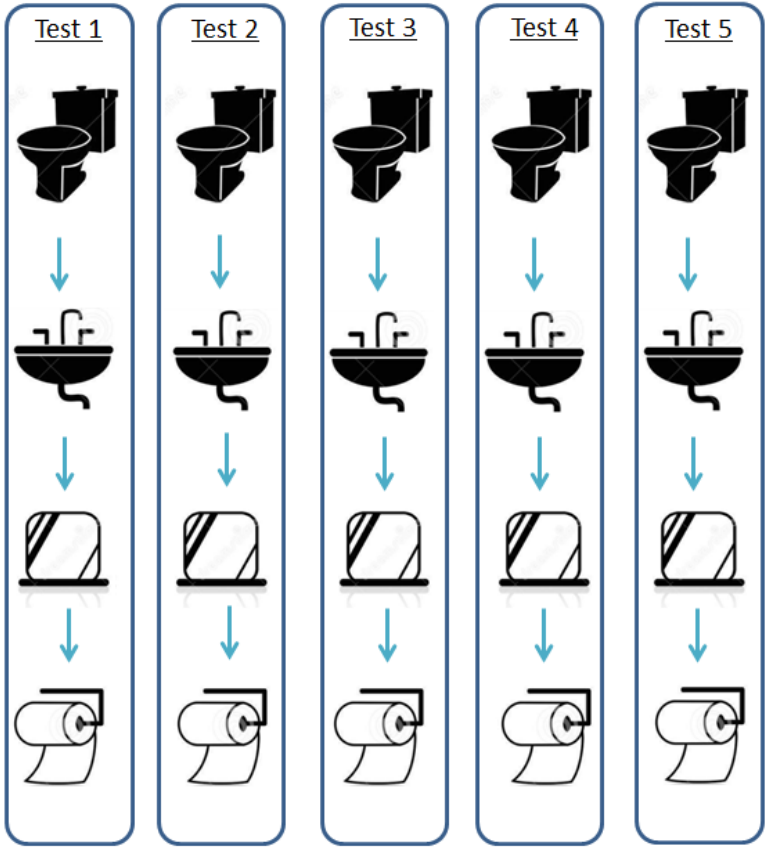
End-to-end tests

Test 1

Test 2

Test 3

Test 4

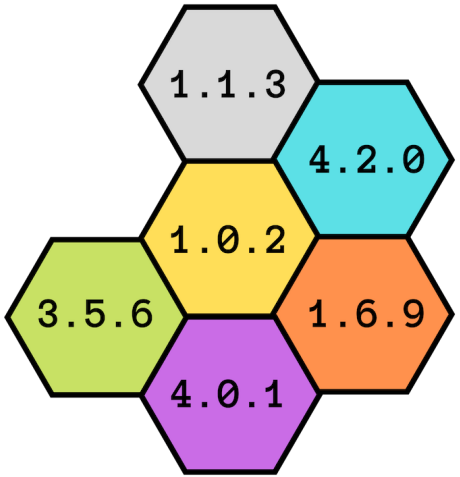

Test 1
Test 2
Test 3
Test 4
Test 5

Стейдж?

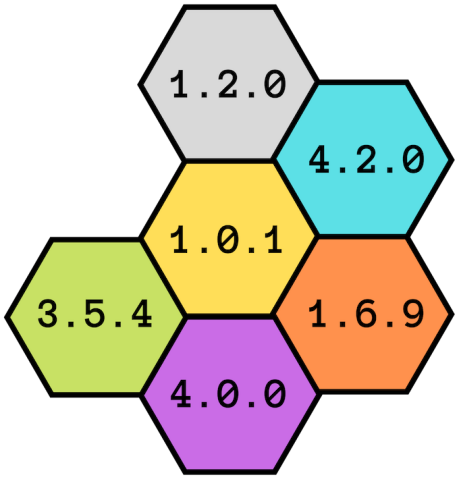
Слишком долго

Разные версии

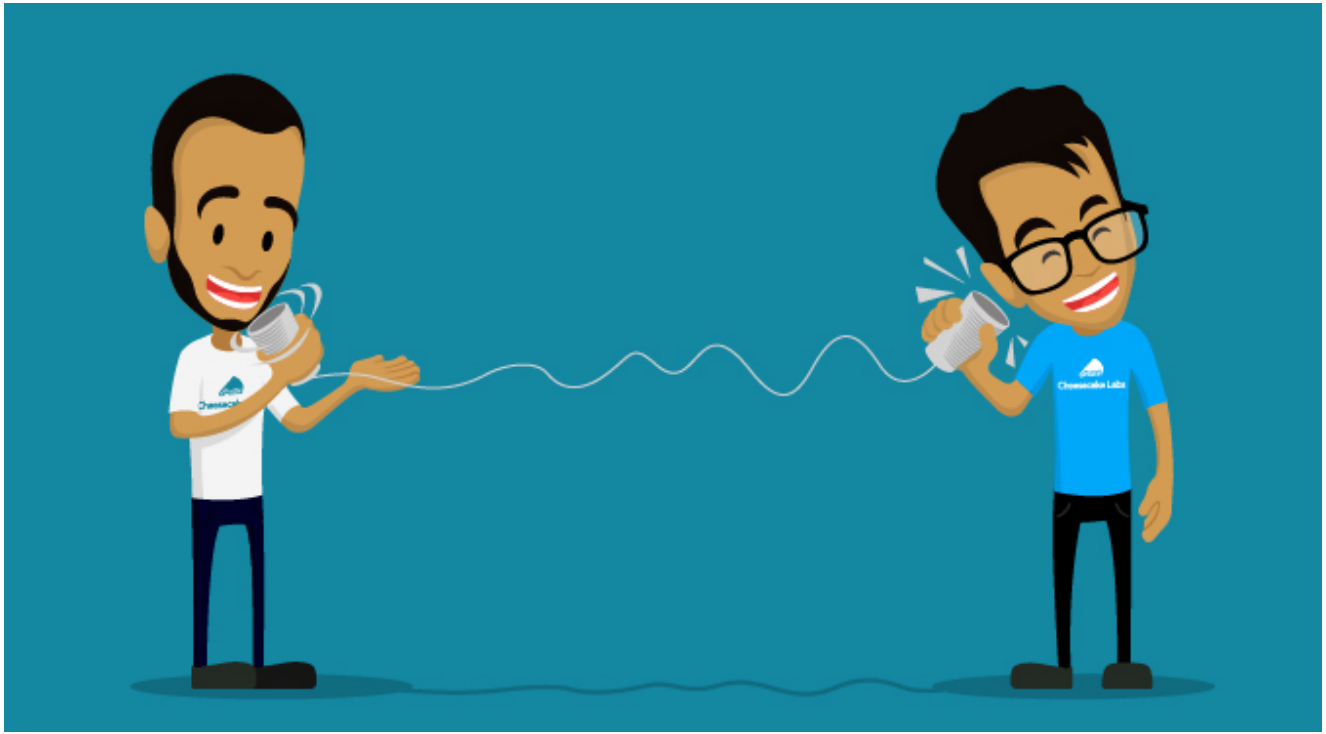
Stage



Production



Распределенные команды



Задача

- Запускать реальные сервисы
- Управлять версиями
- Возможность дебажить
- Интеграция с CI/CD

Варианты

- Bash
- Maven, Gradle
- Docker-compose
- Что-то еще

Минусы Bash, Maven и Gradle

Тяжело поддерживать скрипты

```
wait_for_port() {
    tt=0
    while ! echo hello | nc -w 1 127.0.0.1 $1; do
      sleep 0.1
      let tt=tt+1
      if [ $tt -gt $(($2*10)) ]; then
        exit 1
      fi
    done
}
```

581
votes
6
answers

Q: How to use double or single brackets, parentheses, curly braces

I am confused by the usage of brackets, parentheses, curly **braces** in **Bash**, as well as the difference between their double or single forms. Is there a clear explanation? …

bash syntax

asked Feb 2 '10 by Tim

494
votes
6
answers

Q: When do we need curly braces around shell variables?

In shell scripts, when do we use {} when expanding variables? For example, I have seen the following: var=10 # Declare variable echo "${var}" # One use of the variable echo "$var" # Anothe …

bash shell syntax curly-braces

asked Jan 5 '12 by New User

507
votes
11
answers

Q: Syntax for a single-line Bash infinite while loop

I am having trouble coming up with the right combination of semicolons and/or **braces**. I'd like to do this, but as a one-liner from the command line: while [1] do foo sleep 2 done …

bash loops while-loop

asked Aug 17 '09 by Brian Deacon

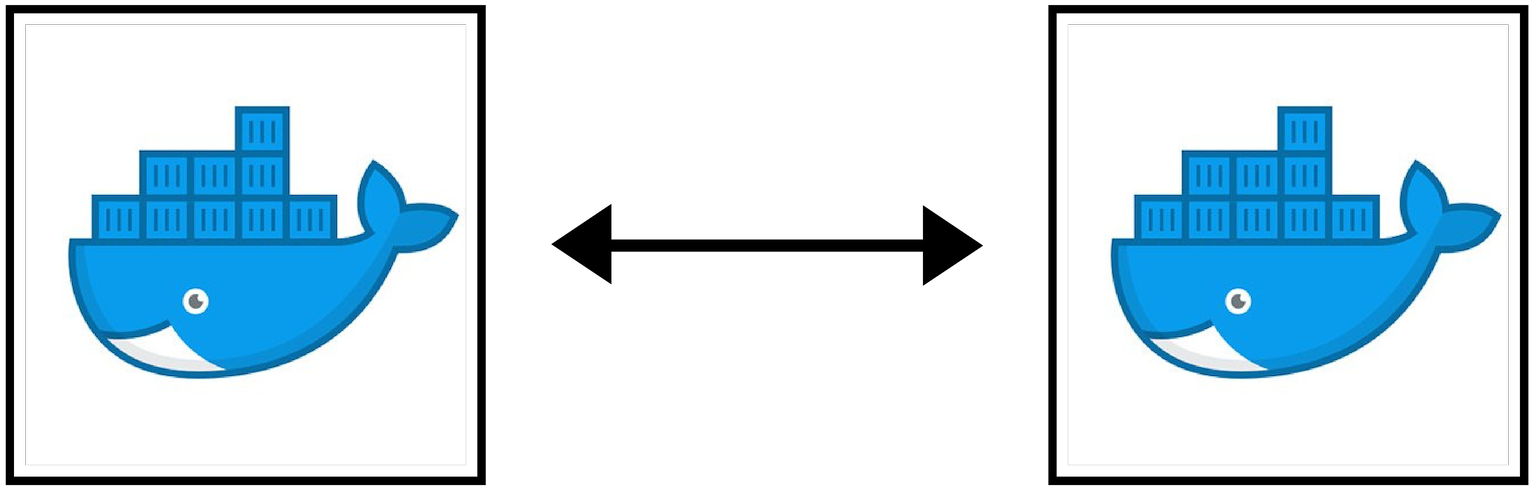
Тяжело поддерживать скрипты

```
wait_for_port() {
    tt=0
    while ! echo hello | nc -w 1 127.0.0.1 $1; do
      sleep 0.1
      let tt=tt+1
      if [ $tt -gt $(($2*10)) ]; then
        exit 1
      fi
    done
}
```

Нет гарантий остановки контейнеров по окончанию тестов

```
for dockerid in `docker ps -q`; do
    docker kill $dockerid
done
```

Работа с сетью



Работа с сетью

```
docker network create testnetwork
docker run -itd --name mysql \
    --network=testnetwork mysql:latest
docker run -itd --name route \
    --network=testnetwork andreymarkelov/route:latest
docker run -itd --name acceptor \
    --network=testnetwork andreymarkelov/acceptor:latest
```

Минусы docker-compose

Все равно требуется вспомогательный код

```
version: "2"
services:
  route:
    image: andreymarkelov/route:${ROUTE_VERSION}
    ports:
      - "${ROUTE_PORT}:8080"
  acceptor:
    image: andreymarkelov/acceptor:${ACCEPTOR_VERSION}
    ports:
      - "${ACCEPTOR_VERSION}:8080"
```

Порядок запуска

```
version: "2"
services:
  simple-sd:
    image: andreymarkelov/simple-sd
    ports:
      - "8761:8080"
  acceptor:
    image: andreymarkelov/acceptor
    ports:
      - "8081:8080"
```

Порядок запуска

```
version: "2"
services:
  simple-sd: (1)
    image: andreymarkelov/simple-sd
    ports:
      - "8761:8080"
  acceptor: (2)
    image: andreymarkelov/acceptor
    ports:
      - "8081:8080"
```

Порядок запуска

```
version: "2"
services:
  simple-sd:
    image: andreymarkelov/simple-sd
    ports:
      - "8761:8761"
  acceptor:
    image: andreymarkelov/acceptor
    ports:
      - "8081:8081"
    depends_on:
      - simple-sd
        condition: service_healty_script
```

Порядок запуска

```
version: "3"
services:
  simple-sd:
    image: andreymarkelov/simple-sd
    ports:
      - "8761:8761"
  acceptor:
    image: andreymarkelov/acceptor
    ports:
      - "8081:8081"
    depends_on:
      - simple-sd
    command: ["./wait-for-it.sh", "simple-sd:5432", "--", "java", "-jar", "acceptor.jar"]
```

Опять нет гарантий остановки контейнеров по окончанию тестов

```
for dockerid in `docker ps -q`; do
    docker kill $dockerid
done
```

Что-то еще

- Testcontainers
- JUnit 5
- MockServer

Testcontainers

Обнаружение докера

Testcontainers will try to connect to a Docker daemon using the following strategies in order:

- Environment variables:
  - DOCKER_HOST
  - DOCKER_TLS_VERIFY
  - DOCKER_CERT_PATH
- Defaults:
  - DOCKER_HOST=https://localhost:2376
  - DOCKER_TLS_VERIFY=1
  - DOCKER_CERT_PATH=~/.docker

If Docker Machine is installed, the docker machine environment for the first machine found. Docker Machine needs to be on the PATH for this to succeed.
If you're going to run your tests inside a container, please read Running inside a Docker container first.

Контейнер как объект

```
public class BillingContainer extends GenericContainer<BillingContainer> {
    public BillingContainer(int port) {
        super("andreymarkelov/billing:latest");
        addExposedPorts(port);
        addEnv("EUREKASERVER_PORT", "8761");
        addEnv("EUREKASERVER_URI", "http://simple-sd:8761/eureka/");
    }
}

// use container
BillingContainer billingContainer = new BillingContainer(8080);
billingContainer.start();
```

Очистка контейнеров при остановке JVM

quay.io/testcontainers/ryuk:x.x.x

Проверка готовности контейнера с зависимым сервисом

```
redisContainer.waitingFor(new HostPortWaitStrategy());
redisContainer.start();
```

Проверка готовности контейнера с зависимым сервисом

```
billingContainer.waitingFor(new HttpWaitStrategy()
    .forPath("/health").forStatusCode(200));
billingContainer.start();
```

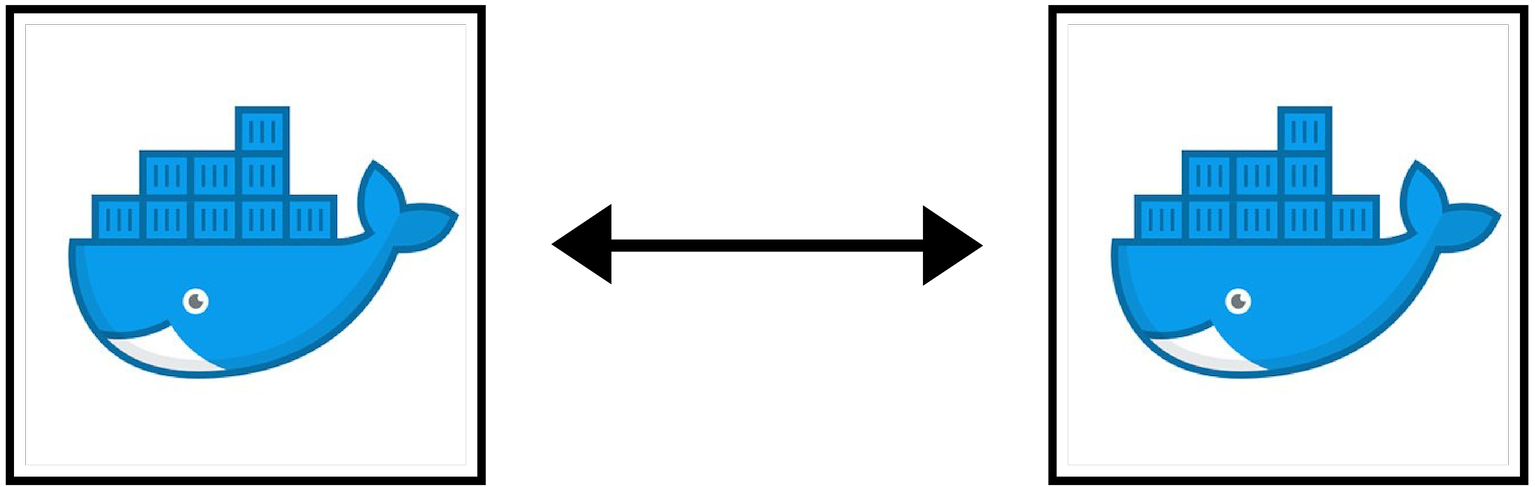
Управление логированием

```
routeContainer.withLogConsumer(
    new Slf4jLogConsumer(LoggerFactory.getLogger(" --- route --- "))
);
sdContainer.withLogConsumer(
    new Slf4jLogConsumer(LoggerFactory.getLogger(" --- simple-sd --- "))
);
```

Управление логированием

```
--- simple-sd --- | STDOUT: 2018-11-28 15:48:26.931  INFO 9 --- [nio-8761-exec-6]
c.n.e.registry.AbstractInstanceRegistry  : Registered instance BILLING/
9a902d9f6aa5:billing:8082 with status UP (replication=true)
 --- route --- | STDOUT: Starting the Route
 --- route --- | STDOUT: Waiting for the Simple Service Discovery to start on port 8761
```

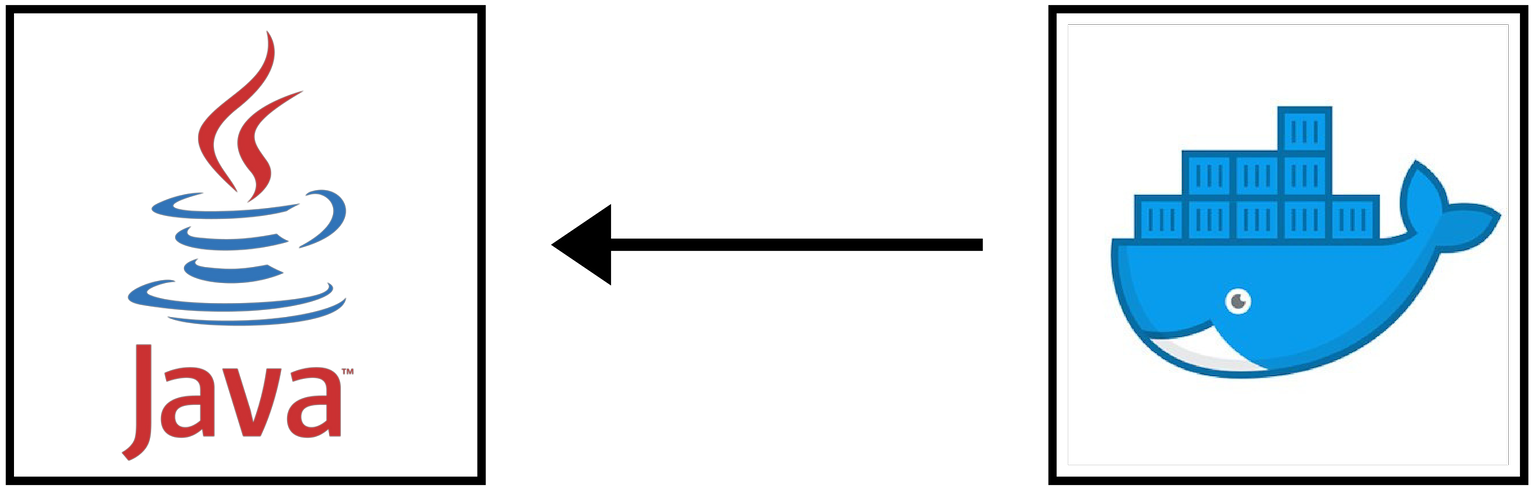
Работа с сетью



Работа с сетью

```
Network network = Network.newNetwork();

redisContainer.withNetwork(network);
redisContainer.withNetworkAliases("redis");

acceptorContainer.withNetwork(network);
acceptorContainer.withNetworkAliases("acceptor");
acceptorContainer.addEnv("REDIS_HOSTNAME", "redis");
acceptorContainer.addEnv("REDIS_PORT", "6379");
```

Работа с сетью



Доступ из контейнера к порту теста

```
HttpServer server = new HttpServer();
server.start();
Testcontainers.exposeHostPorts(server.getAddress().getPort());
// some code here
container.start();
String response = container.execInContainer(
    "wget",
    "-O",
    "-",
    "http://host.testcontainers.internal:" + server.getPort()
).getStdout();
```

JUnit 5

До JUnit 5

```
abstract class AbstractTest {
    // some code here
}

class BaseTest extends AbstractTest {
    // some code here
}

class SingleExecutionTest extends BaseTest {
    // some code here
}

class MultipleExecutionTest extends BaseTest {
    // some code here
}

class SingleExecutionWithVoiceTest extends SingleExecutionTest {
    // some code here
}
```

JUnit 5 (extentions)

```
@ExtendWith(MyExtension.class)
class SomeTests {
    // [... tests using MyExtension ...]
}
```

JUnit 5 (extentions)

```
public class BillingExtention implements BeforeAllCallback, AfterAllCallback {
    private BillingContainer billingCnt = new BillingContainer(8082);

    @Override
    public void afterAll(ExtensionContext context) throws Exception {
        billingCnt.stop();
    }

    @Override
    public void beforeAll(ExtensionContext context) throws Exception {
        billingCnt.start();
    }
}
```

JUnit 5 (extentions)

```
@ExtendWith(BillingExtention.class)
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
public @interface Billing {}
```

JUnit 5 (extentions)

```
@Billing
class MyBillingTest {
    @Test
    void checkBillingTest() {
        // container already started
    }
}
```

А как же сеть?

```
public class BillingExtention implements BeforeAllCallback, AfterAllCallback {
    private BillingContainer billingCnt = new BillingContainer(8082);

    @Override
    public void afterAll(ExtensionContext context) throws Exception {
        billingCnt.stop();
    }

    @Override
    public void beforeAll(ExtensionContext context) throws Exception {
        billingCnt.start();
    }
}
```

Контекстное хранилище

```
public class BillingExtention implements BeforeAllCallback, AfterAllCallback {
    private static final String NETWORK = "networkExtension";

    private BillingContainer billingContainer = new BillingContainer(8082);

    @Override
    public void afterAll(ExtensionContext context) throws Exception {
        billingContainer.setNetwork(getOrCreateNetwork(context));
        billingContainer.withNetworkAliases("billing");
        billingContainer.stop();
    }

    @Override
    public void beforeAll(ExtensionContext context) throws Exception {
        billingContainer.start();
    }

    public static Network getOrCreateNetwork(ExtensionContext context) {
        Store store = context.getStore(
            ExtensionContext.Namespace.create(Network.class, context.getUniqueId())
        );
        return Network.class.cast(store.getOrComputeIfAbsent(NETWORK, k -> Network.newNetwork()));
    }
}
```

Депенденси инжекшен

```
@Acceptor
class MyAcceptorTest {
    @Test
    void checkAcceptorTest(AcceptorClient aClient) {
        // use aClient here
    }
}
```

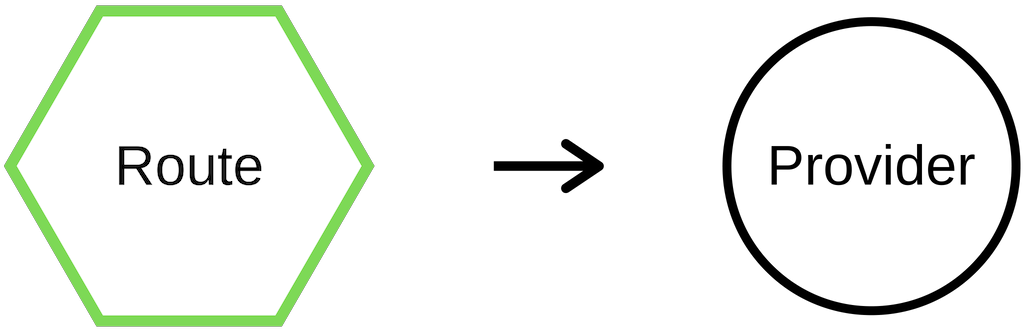
Депенденси инжекшен

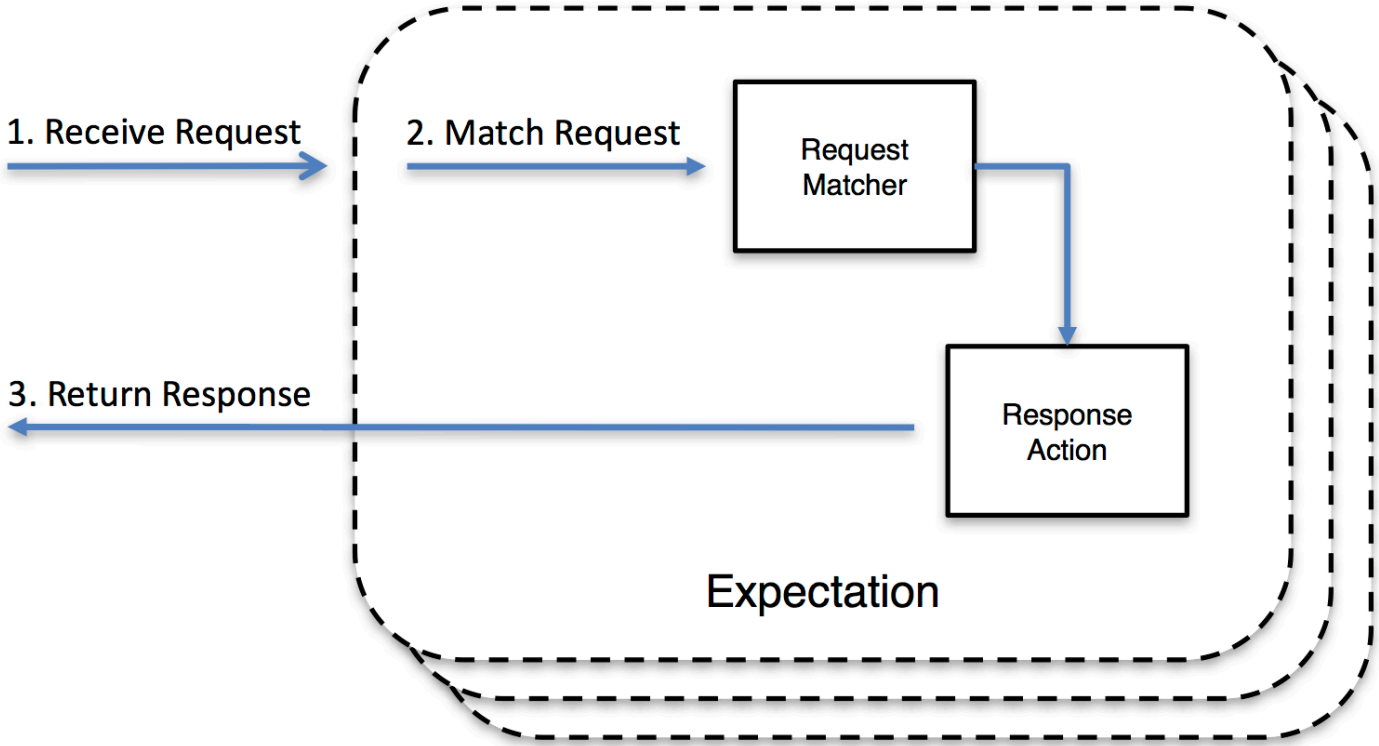
```
public class AcceptorExtention implements BeforeAllCallback, AfterAllCallback, ParameterResolver {
    private static AcceptorContainer acceptorCtn = new AcceptorContainer(NetworkHolder.network(), 8081);

    @Override
    public void afterAll(ExtensionContext context) {
        acceptorCtn.stop();
    }

    @Override
    public void beforeAll(ExtensionContext context) {
        acceptorCtn.start();
    }

    @Override
    public boolean supportsParameter(ParameterContext parameterContext, ExtensionContext extensionContext) {
        return parameterContext.getParameter().getType().equals(AcceptorClient.class);
    }

    @Override
    public Object resolveParameter(ParameterContext parameterContext, ExtensionContext extensionContext) {
        return new AcceptorClient("localhost", acceptorContainer.getFirstMappedPort());
    }
}
```

MockServer

MockServer can be used for mocking any system you integrate with via HTTP or HTTPS (i.e. services, web sites, etc).

http://www.mock-server.com


Route
Provider

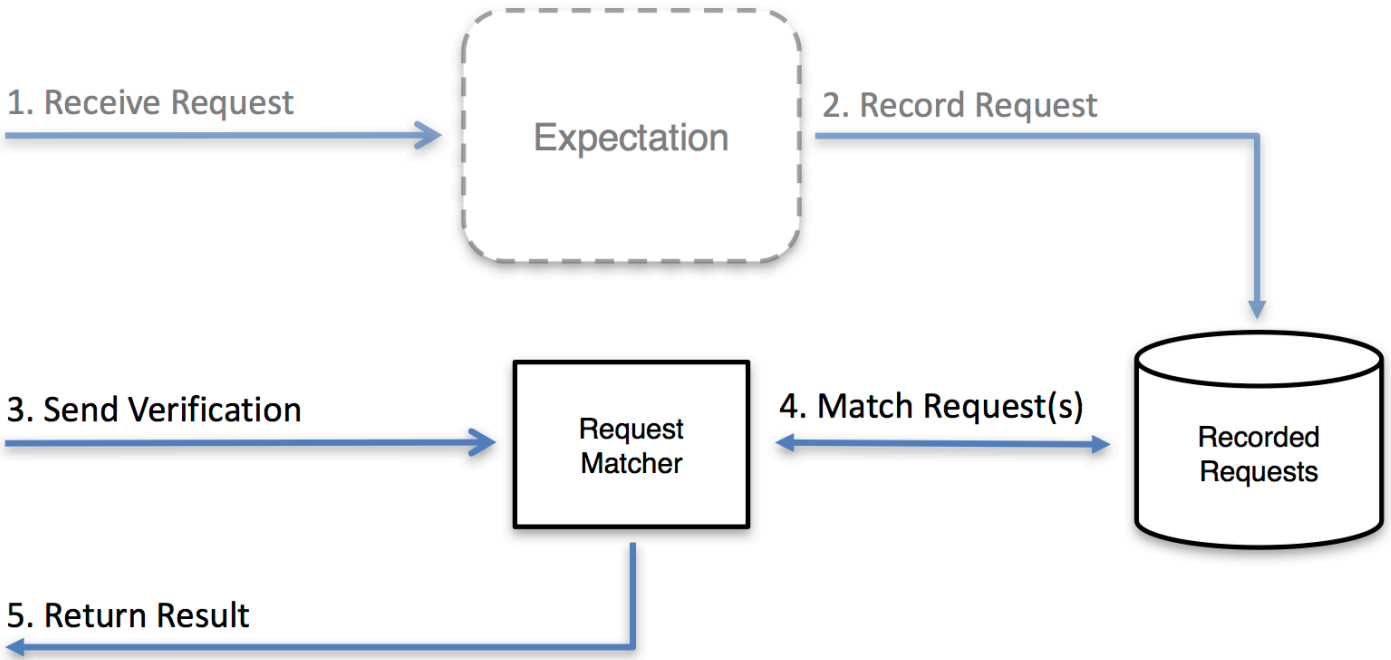

1. Receive Request
2. Match Request
Request Matcher
3. Return Response
Response Action
Expectation

Expectation

```
PUT http://localhost:1080/mockserver/expectation
{
    "httpRequest": {
        "path": "/sendsms"
    },
    "httpResponse" : {
        "statusCode": 200,
        "body": "{\"messageId\": \"0123456789\", \"state\": 1}"
    }
}
```

Expectation

```
MockServerClient client = new MockServerClient("localhost", 1080);
client.when(HttpRequest.request().withPath("/sendsms"))
    .respond(HttpResponse.response()
       .withBody("{\"messageId\": \"123\", \"state\": 1}")
      .withStatusCode(200));
```

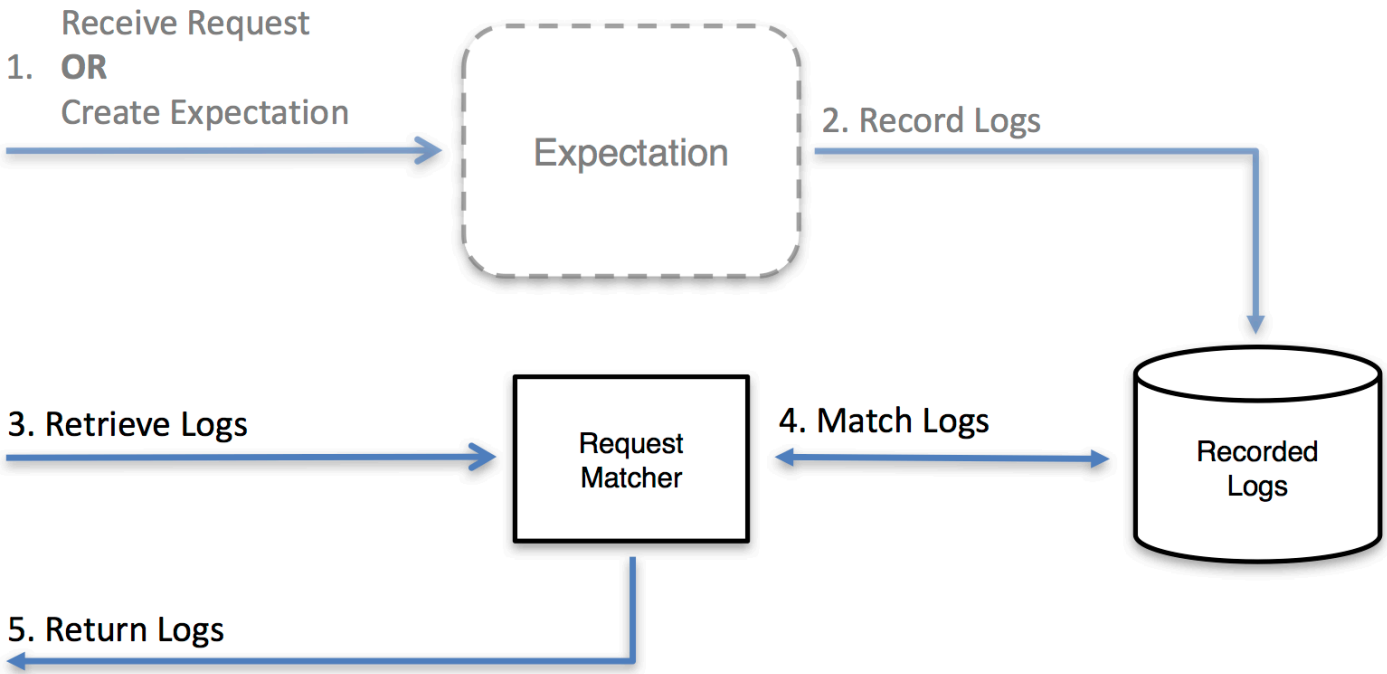

1. Receive Request
Expectation
2. Record Request
3. Send Verification
Request Matcher
4. Match Request(s)
Recorded Requests
5. Return Result

Verification

```
PUT http://localhost:1080/mockserver/verify
{
    "httpRequest": {
        "path": "/sendsms"
    },
    "times": {
        "count": 1,
        "exact": true
    }
}
```

Verification

```
MockServerClient client = new MockServerClient("localhost", 1080);
client.verify(
    HttpRequest.request().withPath("/sendsms"),
    VerificationTimes.once());
```


Receive Request
1. OR
Create Expectation
Expectation
2. Record Logs
3. Retrieve Logs
Request Matcher
4. Match Logs
Recorded Logs
5. Return Logs

Recorded

```
PUT http://localhost:1080/mockserver/retrieve?type=REQUESTS
{
    "path": "/sendsms"
}
```

Recorded

```
MockServerClient client = new MockServerClient("localhost", 1080);
HttpRequest[] requests = client.retrieveRecordedRequests(
    HttpRequest.request().withPath("/sendsms"));
```

Как запускать?

- Maven Plugin
- Java API
- Command line
- Deployable WAR
- Docker container

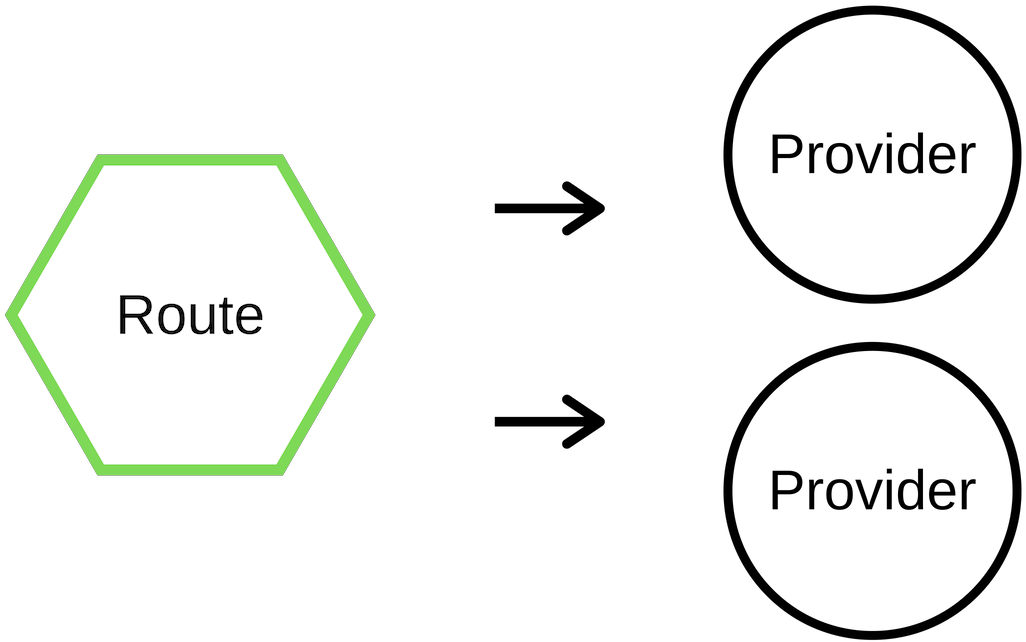
Как запускать?

- Maven Plugin
- Java API
- Command line
- Deployable WAR
- Docker container

Как запускать?

```
docker run -d -P jamesdbloom/mockserver
```

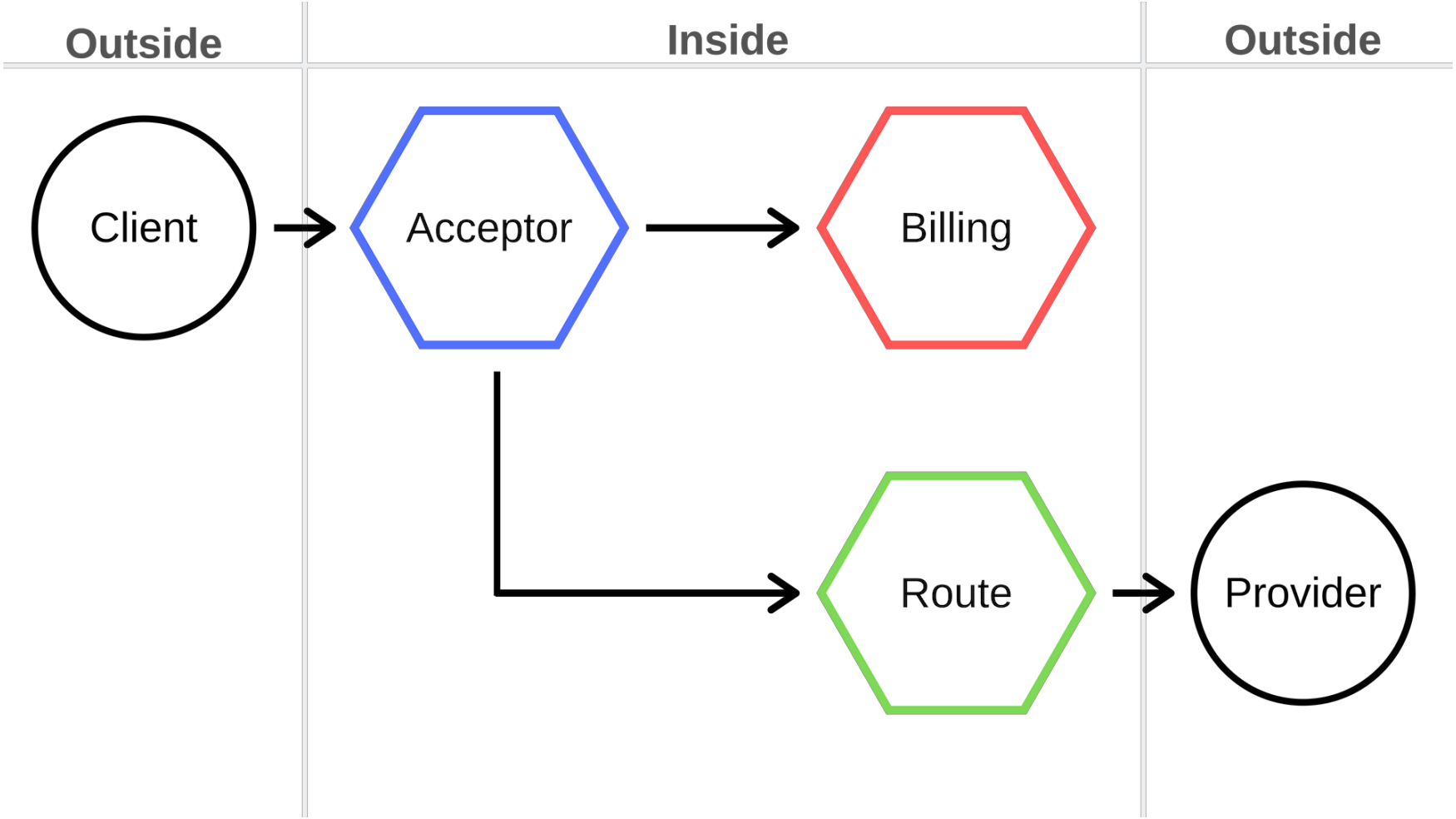
```
IMAGE                      PORTS
jamesdbloom/mockserver     0.0.0.0:32800->1080/tcp, 0.0.0.0:32799->1090/tcp
```


Route
Provider
Provider

Как запускать?

```
docker run -d -p 1080:1080 -p 1081:1081 jamesdbloom/mockserver \
/opt/mockserver/run_mockserver.sh -serverPort 1080,1081

IMAGE                     PORTS
jamesdbloom/mockserver    0.0.0.0:1080-1081->1080-1081/tcp, 1090/tcp
```

Test


Outside
Inside
Outside
Client
Acceptor
Billing
Route
Provider

Given-When-Then

Given Infobip API
When I send a message via REST API
Then the message should arrive to cell phone

```
class SendMessageTest {
    @Test @DisplayName("Full circle test")
    void testSuccessSend() {
    }
}
```

```
@MockServer @Redis @SimpleSd @Billing @Route @Acceptor
class SendMessageTest {
    @Test @DisplayName("Full circle test")
    void testSuccessSend() {
    }
}
```

```
@MockServer @Redis @SimpleSd @Billing @Route @Acceptor
class SendMessageTest {
    @Test @DisplayName("Full circle test")
    void testSuccessSend(AcceptorClient aClt) {
    }
}
```

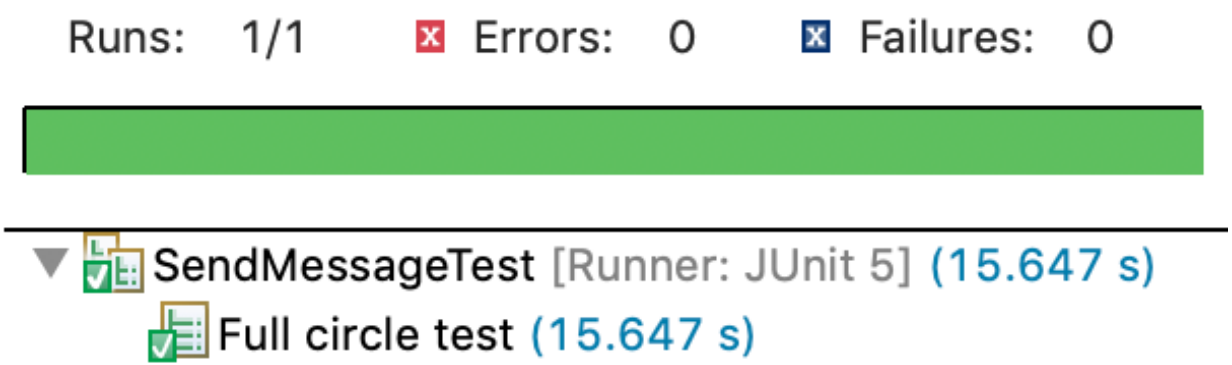
```
@MockServer @Redis @SimpleSd @Billing @Route @Acceptor
class SendMessageTest {
    @Test @DisplayName("Full circle test")
    void testSuccessSend(AcceptorClient aClt) {
        String resp = aClt.send(
                "Hi, how are you?", "Infobip", "79817442839", "andrey");
        assertTrue(resp.contains("123"));
    }
}
```

```
@MockServer @Redis @SimpleSd @Billing @Route @Acceptor
class SendMessageTest {
    @Test @DisplayName("Full circle test")
    void testSuccessSend(AcceptorClient aClt, MockServerClient mClt) {
        String resp = aClt.send(
                "Hi, how are you?", "Infobip", "79817442839", "andrey");
        assertTrue(resp.contains("123"));
    }
}
```

```
@MockServer @Redis @SimpleSd @Billing @Route @Acceptor
class SendMessageTest {
    @Test @DisplayName("Full circle test")
    void testSuccessSend(AcceptorClient aClt, MockServerClient mClt) {
        mClt.when(HttpRequest.request().withPath("/sendsms"))
            .respond(HttpResponse.response()
                .withBody("{\"messageId\": \"123\", \"state\": 1}")
                .withStatusCode(200));
        String resp = aClt.send(
                "Hi, how are you?", "Infobip", "79817442839", "andrey");
        assertTrue(resp.contains("123"));
        mClt.verify(
            HttpRequest.request().withPath("/sendsms"),
            VerificationTimes.once());
    }
}
```

```
@MockServer @Redis @SimpleSd @Billing @Route @Acceptor
class SendMessageTest {
    @Test @DisplayName("Full circle test")
    void testSuccessSend(AcceptorClient aClt, MockServerClient mClt) {
        mClt.when(HttpRequest.request().withPath("/sendsms"))
            .respond(HttpResponse.response()
                .withBody("{\"messageId\": \"123\", \"state\": 1}")
                .withStatusCode(200));
        String resp = aClt.send(
                "Hi, how are you?", "Infobip", "79817442839", "andrey");
        assertTrue(resp.contains("123"));
        mClt.verify(
            HttpRequest.request().withPath("/sendsms"),
            VerificationTimes.once());
    }
}
```

```
tests-demo andreymarkelov$ docker ps --format "table {{.Image}}\t{{.Ports}}"
IMAGE                                       PORTS
andreymarkelov/acceptor:latest              0.0.0.0:32840->8081/tcp
andreymarkelov/route:latest                 0.0.0.0:32839->8083/tcp
andreymarkelov/billing:latest               0.0.0.0:32838->8082/tcp
andreymarkelov/simple-sd:latest             0.0.0.0:32837->8761/tcp
redis:latest                                0.0.0.0:32836->6379/tcp
jamesdbloom/mockserver:mockserver-5.4.1     0.0.0.0:32835->1080/tcp, 0.0.0.0:32834->1090/tcp
quay.io/testcontainers/ryuk:0.2.2           0.0.0.0:32833->8080/tcp
```


Runs: 1/1
Errors: 0
Failures: 0
SendMessageTest [Runner: JUnit 5] (15.647 s)
Full circle test (15.647 s)

Выводы

- Можно проверить совместимость версий
- Можно использовать дебаг
- Микросервисы более независимы
- Все знают общую бизнес логику

Выводы

- Можно проверить совместимость версий
- Можно использовать дебаг
- Микросервисы более независимы
- Все знают общую бизнес логику
- Все равно этого недостаточно

Email: *andrey.v.markelov@gmail.com*
Github: *https://github.com/AndreyVMarkelov/pint-example*

Выводы

- Можно проверить совместимость версий
- Можно использовать дебаг
- Микросервисы более независимы
- Все знают общую бизнес логику
- Все равно этого недостаточно